PICmicro MCU C®

CCS C'de PIC Microchip
Programlamaya Giriş

Yazar : Nigel Gardner

İçindekiler

Giriş

Geçmiş
Neden C kullanılır?
PC Tabanlı vs. PICmicro®MCU Tabanlı Program Geliştirme
Ürün Geliştirme
Terminoloji
Deneme and Test Kodu
C Kodlama Standartları
Temeller

1 C Esasları

C Programının Yapısı
Bir C Programının Bileşenleri
#pragma
main()
#include
printf Fonksiyonu
Değişkenler
Sabitler
Yorumlar
Fonksiyonlar
C Anahtar Sözcükleri

2 Değişkenler

Veri Tipler
Değişken Tanımlama
Değişken Atama
Numaralama
typedef
Tip Dönüşümleri

3 Fonksiyonlar

Fonksiyonlar
Fonksiyon Prototipleri
Fonksiyon Argümanları Kullanımı
Geri Dönme Değerleri Fonksiyonları Kullanma
Klasik ve Modern Fonksiyon Bildirimleri

4 İşleçler

Aritmetik

Karşılaştırma
Mantıksal
Bit Bit
Arttırma ve Azaltma
İşlem Önceliği

5 Program Kontrol İfadeleri

If
If-else
? İşleçi
for Döngüsü
while Döngüsü
do-while Döngüsü
Yuvalanmış Program Kontrol İfadeleri
Break
Continue
Null
Return

6 Diziler / Stringler

Tek Boyutlu Diziler
Stringler
Çokboyutlu Diziler
Başlatma Dizileri
Stringlerin Dizileri

7 İşaretçiler

İşaretçi Temelleri
İşaretçiler ve Diziler
Fonksiyonlara İşaretçi Geçme

8 Yapılar / Bileşimler

Yapı Temelleri
Yapıların İşaretçileri
Yuvalanmış Yapılar
Bileşimleri Temeli
Bileşimlerin Yapıları

9 PICmicro®MCU'ya Özel C

Girişler ve Çıkışlar
C ve Assembler Karıştırma
Gelişmiş BIT Manipulasyonu

Zamanlayıcılar
A/D Dönüşümü
Veri İletişimleri
I2C İletişimleri
SPI İletişimleri
PWM
LCD Sürücü
Kesmeler
Include Kütüphaneleri
Ek Bilgi

Giriş

Neden C kullanılır?

C dili Dennis Ritchie ve Brian Kernighan tarafından 1970'lerde Bell laboratuarlarında geliştirilmiştir. Uygulama için kullanılan ilk platform bir Unix sistemi altında işleyen Pd11'dir.

C dilinin tanıtımından beri gelişti ve bilgisayar endüstrisinde geliştirme dili haline geldi. PC C++ ve diğer ANSI standartlarını kullanarak ekonomik geliştirme platformu haline geldi.

C dili ile yazılmış programları bir PC'den diğerine taşımak için çok az bir modifikasyon gerekir. Yani C dili taşınabilirdir. Bu, PC'ler ve anabilgisayarlar ile çalışırken kolaydır. Ancak mikrodenetleyiciler ve mikroişlemciler farklı kaynaklardan gelmektedirler. Çeşitli setup ve çevre kontrolleri mikrospesifik olurken anaprogram akışı değişmeden kalır. Bunun bir örneği PICmicro®MCU üzerindeki port yönlendirme kaydedicileridir.

Mikrodenetleyici uygulamalarında C'nin kullanımına; geniş program, geniş RAM hafızaları ve daha büyük işlem hızlarına gereksinim duyan üreticiler önayak olmuştur.

Bana iletilen bir örnek stopclock fonksiyonu için C'de 2/3 gün gerekirken Assembler'da bu sürenin 2 hafta olduğudur.

" Hemen bir C derleyicisi almalıyım " dediğinizi duyuyorum - Neden Assembler'da yazalım? Bu konu kod etkinliğini gündeme yeni getiriyor – Assembler'da yazılan bir program C versiyonun %80'i büyüklüğündedir. Büyük program hafızasına sahip bilgisayarlar için sorun yok.Ancak daha küçük cihazlar için çok etkin değil. Parayı ver ve PIC'i al.

PC Tabanlı vs. PICmicro®MCU Tabanlı Program Geliştirme

PC tabanlı ürünlerde geliştirme çalışmaları yapan mühendisler temel pre-wired donanım (mesela klavye, işlemci, hafıza, I/O, yazıcı ve görsel ekran (monitör)) lüksüne sahiptirler. Ürün geliştirme bundan sonra yazılımı oluşturmaya ve hataları ayıklamaya kalır.

PIC tabanlı tasarımla ilgilenenler girdi ve çıktı donanımı formunda dış dünyaya tüm arayüzleri yapmak durumundadırlar. Bir PIC programcısı " Merhaba Dünya " mesajını yazabilir ve derlemeden sonra gösterebilir. Mesajın ekranda görülebilmesi için PIC programcısı komutu uygulamak için PIC içinde bir RS232 arayüzü yapmak zorundadır ve geliştirme kartını PC'deki bir COM portuna bağlamak zorundadır.

" Niye Assembler kullanmakta tereddüt edelim " dediğinizi duyuyorum.Bende öyle diyorum. Bu son ürünün taşınabilirliği problemine dayanıyor. Bütün bir PC'yi 40 pinli bir DIL paketinde elde edebilseydik onu kullanırdık. Ancak bugünün minyatürleştirme endüstrisi henüz bu boyutlara ulaşmadı. PIC gibi mikrodenetleyicileri düşük maliyet ve taşınabilir uygulamalarından

dolayı kullanmaya devam edeceğiz. PIC tabanlı tasarımlar için geliştirme araçları geliştirmeciye grafik kütüphaneleri hariç PC tabanlı geliştirme ile aynı olanakları sunar.

Ürün Geliştirme

Ürün geliştirme şans ve tecrübenin bir kombinasyonudur. Çok basit görevlerin bazıları tüm ürüne göre kıyaslandığında geliştirme için çok fazla zaman gerektirir (Yani zaman kısıtlaması olduğu durumlarda dikkatli olun). Bir ürünün tasarlayacak kişi şunlara gereksinim duyar; zaman, huzur, sessizlik, mantık ve hepsinden önemlisi gereksinimleri anlamayı gerektirir. Herhangi bir geliştirmenin en kolay yolunu buldum; temiz bir kağıt ve fikir veya spesifikasyon ile başlamak.

Olası çözümleri oluşturmak ile başla; en basitini ve en güvenilirini bulmak için dene. Bu aşamada diğer fikirleri de bir kenara atma. Çünkü onların arasında güzel düşünceler olabilir.

İlk önce akış diyagramını, blok diyagramını, giriş-çıkış bağlantı planını veya diğer uygun çözümleri yap. Bütün I/O'ları konfigüre edilmiş bir bordu taklit edecek prototip kartı veya donanımı oluştur. I/O pinlerinin planını daha basitleştirmek için ileride yerlerini değiştirebileceğini unutma - genellikle yazılım da çok küçük konfigürasyon ile.

Daha sonra kodu yazmaya başla – test edilebilir bloklar halinde – ve aşama aşama programı oluştur. Bu bir seferde 200 satırlık kodu debug etmek zorunda kalmanızı önler.

Bu ilk projeniz ise – Ozaman basit tutun – Dev bir projeye başlamadan önce konuyu daha iyi anlamak için bir led'i bir butonla açıp kapayan bir devre deneyin.

Basit aşamalar halinde programı oluştur – ilerledikçe test et – akış diyagramını güncel tutmak için akış diyagramı üzerinde tekrar çalışın.

Fikir

Bir fikir sizin tarafınızdan EUREKA tarzında yada başkaları tarafından bir projenin gereksinimi olarak oluşturulabilir. Ana tema aynıdır.

Tasarım aşamasına başlamadan önce temel terminolojinin anlaşılması gerekir – yeni bir dil öğrenmek gibi – yani PICmicro®MCU PIC dili mikrodenetleyicisi tasarımı başlamadan tam olarak anlaşılmalıdır.

Şimdi genel terimlerle başlayalım; PIC dili ve mikroişlemci ,mikrodenetleyici temelli sistemler arasındaki farkla.

Terminoloji

Bazı basit terimlerle başlayalım.

Mikrodenetleyici ; plastik bir parça, metal ve arıtılmış kısımlar ki bunlar yazılımsız hiçbirşeye yaramazlar. Yazılım bir mikrodenetleyiciyi kontrol etmeye başladığında nerdeyse sınırsız uygulama alanına sahip olur.

I/O ; Girdi veya çıktı olarak konfigüre edilebilen dış dünyaya bağlantı pinidir. Mikrodenetleyiciye iletişim kontrol ve bilgi okumak için bir çok durumda gereklidir.

Yazılım ; Mikrodenetleyicinin işlemesi için gerekli bilgidir. Başarılı bir ürün yada uygulama için yazılımın hatasız ve bug'sız (bug : bir program üzerinde yazım hatasından kaynaklanan sistem açığı) olması gerekir. Yazılım C, Pascal veya Assembler gibi çeşitli dillerde yazılabilir.

Donanım ; Mikrodenetleyici, hafıza, arayüz bileşenleri, güç kaynağı, işaret koşullandırma devreleri ve kendisinden çalışmasını ve dış dünyaya bağlantı kurmasını sağlayan tüm bileşenler.

Donanıma bir başka bakış olarak özellikle çalışmadığında atabildiğimiz şeydir yaklaşımı getirilebilir.

Simulatör ; MPLAB® geliştirmeortamının, dahili operasyonunun bazısına erişime izin veren kendinde gömülü simülatörü vardır. Eğer siz olayların, ne zaman olduğunu bilirseniz bu, test etme dizaynlarınızı uzun yoludur. Eğer bir olay ' orası hakkında bazıyerler' olursa, siz simulator kısıtlayıcıyı bulabilirsiniz. Tüm iz , adım ve kusurları giderme tesisleri, bununla beraber , müsaittir. 16C5x gelişmi için başka bir ürün SIM ICE'dir - ICE özelliklerinin bir donanım simulatörü teklif bazısı fakat mal olunmuşun bir kesimindedir.

Devre Benzeticisi (Emilatörü); (ICEPIC yada PICmicro®MCU MASTER) Donatımın çok yararlı bir parçası, Microcontrollerin nerede oturduğu soket ve PC iniz arasına bağlandı. O, PC devam eden yazılıma olmak için imkan verir fakat devre tahtası sonunda bir Microcontroller'e benzer. ICE, o dünyanın dışıyla nasıl haberleştiğinin, mikronun içinde ve ne olduğunu gösterir, bir programa adım atmanıza izin verir.

Programlayıcı ; Bir ICE'nin yardımısız çalışması için ona izin veren mikrokontrollerin belleğinin dolabilmesine, programa imkan sağlayan birimdir. Onlar, tüm biçimlere ve büyüklüklere ve değişemlere girerler. Mikrochip'ten PICSTART PLUS and
PROMATE II her ikisi de seri porta bağlanır.

Kaynak Dosya ; Bir dilde yazılmış bir programı Assembler ve siz anlarsınız**.** Mikrodenetleyici onu anlamadan önce, Kaynak dosyası işlenmiş olmalı**.**

Assembler / Derleyici ; Bir nesne dosyası içindeki kaynak dosyayı çeviren bir yazılım paketidir.Hatala kontrolü dahilir, assembly işlemi yaparkenki oluşan hata bayrakları gibi bir programı hatalarını ayıklama özelliği kullanılır. MPASM en son assembler'dır.

Nesne Dosyası ; Bu, Assembler / Derleyici tarafından üretilen bir dosyadır ve fonksiyonlarını gerçekleştirebilmeyi anlayan ICE yada simulatör yada programlayıcının formundadır. Dosya uzantısı assemler direktifinde .OBJ yada .HEX tanımlanır.

Liste Dosyası ; Bu, Assembler / Derleyici tarafından üretilen bir dosyadır ve yazdığınız yorumlatı ve heksadesimal değerleri ile birlikte kaynak dosyasından tüm içerikleri kapsar. Bu, ne zaman program kusurlarını gidermek için yorucu kaynak dosyası kayıdıdan yazılımın içinde ne

olduğu sizin takip edenin bir daha muhteşem şansı varolduğu gibi incelemek için en yararlı dosyadır. Dosya uzantısı .LST'dir.

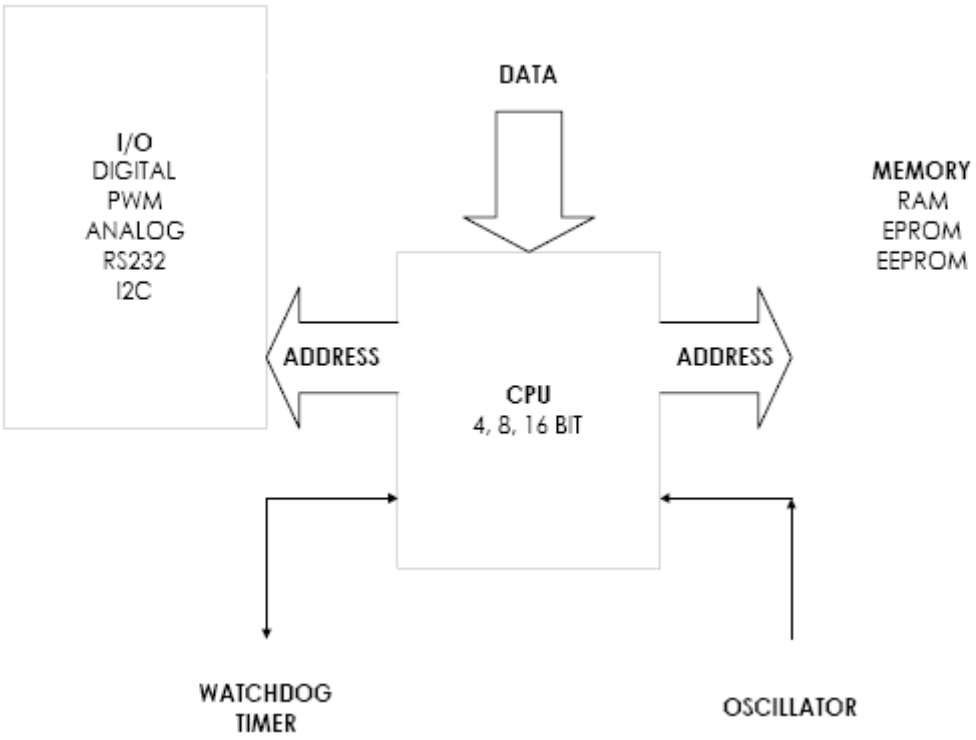
Diğer Dosya ; Hata dosyası bir hata listesi içerir fakat başlangıçları gibi hiçbir bildirim vermez. .COD dosyası emülatör tarafından kullanılır.

Bug'lar ; Hatalar sizin tarafınızdan doldurulan boşlukları yaratır. simpel typin errus'daki bu alanlar yazılım dili yazım hataları kullanımı içindir. Bu bug'ların çoğu derleyici tarafından bulunacaktır ve .LST dosyasında görülecektir, diğerleri arınmış ve deneme yanılma ile düzeltilmiştir.

Mikroişlemci

Bir mikroişlemci yada dijital bilgisayar üç temel bölümün oluşumudur: CPU, I/O ve Hafıza – bazı destek devre ilavesi ile.

Her bir bölüm tüm ıslıklar ve ziller temelindeki karmaşıklıkta değişebilir.



TİPİK MİKROİŞLEMCİ SİSTEMİ

Her bir dönüşümü alalım:

Giriş ve Çıkış (I/O) dijital, analog ve özel fonksiyonları kapsar ve dış dünya ile bağlantı kurmamızı sağlayan bölümdür.

Merkez işlemci birimi (CPU) sistemin kalbidir ve veri manipulasyonu ve hesaplamaları gerçekleştirirken 4,8,16 bit veri fotmatında çalışabilirler.

Hafıza RAM, ROM,EPROM, EEPROM yada bunların kombinasyonu olabilir ve veri ve program saklamada kullanılır.

Bir osilatör mikroişlemciyi sürmek için gerekmektedir. Onun fonksiyonu CPU içindekileri ve veri zamanlama, çıkışları hesaplama ve bu bilgi çıkışı vermeden öncedir. Osilatör farklı bileşenlerden yapılmış olabilir yada hazır yapılmış bir modül olabilir.

Mikro işlemci ile ilgili bulunan diğer devre watch dog timer'dır – sistem anahtarlamaya yardım etmeyi, aynı hat üzerinde bağlı devrelerin birisinin I/O 'sunu seçmeyi, chip numaralarını seçmek ve bağlantı kurmak için veri hatları ve adresler için buffer'lamayı sağlar.

Bir Mikroişlemci'den sistemin başlıca alanı CPU olan bir ürün gibi söz etmek normaldir. I/O ve hafıza aynı chiplerden düzenlenebilir ve doğru işlem yapabilmek için bir veri hattı, adres hattı ve adres çözümleme gerekmektedir.

Mikrodenetleyiciler

The PICmicro®MCU, diğer taraftan, bir Mikrodenetleyicidir ve aynı chip'te birleşmiş tüm CPU, hafıza, osilatör, watchdog ve I/O ya sahiptir. Bu, boşluğu, dizayn zamanı ve dış çevresel zamanlaması ve uyumluluk problemlerini kapsar fakat bazı hallerde, bir takım belleği büyüklüğüne dizayn ve I/O kabiliyetleri sınırlayabilir.

Mikrodenetleyicilerin PIC ailesi geniş bir I/O aralığı, hafıza ve geliştirme mühendisliğinin ençok karşılaşılan gerekli özel fonksiyonlarını sunar.

Bilgisayarların, mikroişlemcilerin ve mikrodenetleyicilerin dizaynının kütüphane araştırmasında birçok genel kitap bulacaksınız, öyleyse konu temel farklılıkların açıklamasından dolayı genişletilmiş veya buraya kopyalaması olmayacaktır.

Neden PIC kullanılır?

Kod Verimi ; PIC Harvard mimarisini temel alan 8 bit bir mikrodenetleyicidir – veri ve hafıza için ayrı dahili hatları bulunan anlamında. Üretilen iş oranı, her ikisi veriye eşzamanlı erişim nedeniyle ve program belleği o yüzden artılır. Genel mikrodenetleyicilere baktığımızda, her ikisi veri ve programa ele alıyor ve bir dahili hattı var. Bu PICmicro®MCU 'a karşılaştırınca, 2 'nin hiç olmazsa bir faktörü tarafından dolayı işlem yavaşlar.

Güvenlik ; Tüm talimatlar 12 bitlik veya 14 bitlik program belleği kelimesine göre olur. Yazılımın olası talimatları gibi, bir programın veri kısımının içine ve yorumcu veriyi çalıştırmak için atlama yoktur. Bu 8 bitlik hat kullanan bir Harvard mimarisine sahip mikrodenetleyicide ortaya çıkabilir.

Komut Seti ; 16C5x ailesi ve 16Cxx ailesinde yazılım düzenlemek için 33 komut öğrenmemiz gerekir.Her komut (CALL, GOTO ve bit set komutları -BTFSS, INCFSZ - haricinde) bir saykılda kullanılır.

Hız ; PIC, osilatör ve dahili saat yolu arasındaki 4 bağlantı tarafından ayrılır. Bu komut zamanın hesaplanmasını kolaylaştırır. Özellikle eğer 4 MHz'lik kristal kullanıyorsanız. Böylece her komut döngüsü 1us'de çalışır.PIC 20MHz kristal adımı saniyede 5 milyon komut ile çalıştıran hızlı bir mikrodur - Neredeyse 386SX 33! 'in hızının 2 katı.

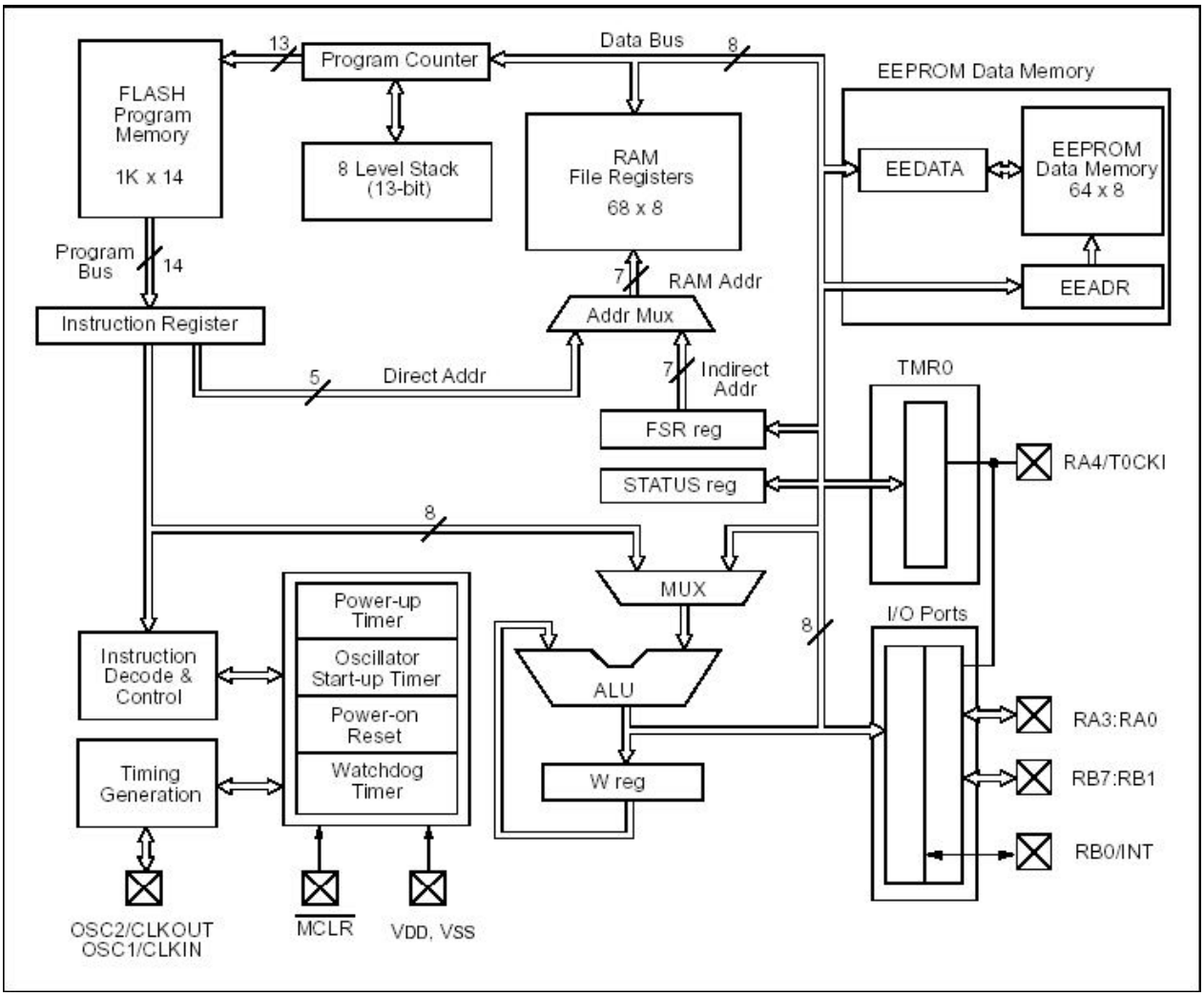
Statik İşlem ; PIC tam bir statik işlemcidir.Başka bir deyişle eğer saati durdurursanız bütün kaydedici içerikleri korunur. Pratikte yapmak yerine PIC'i uyku moduna alırdınız. Bu saati durdurur ve PIC içindeki çeşitli bayrakları kurar. Böylece uykudan önce hangi durumda olduğunu öğrenebilirseniz. Uyku modundayken PIC sadece bekleme akımı çeker ki bu bir mikro Amper'den daha az olabilir.

Sürme Yeteneği ; PIC yüksek bir çıktı sürme yeteneğine sahiptir ve LED'leri ve triyakları direk olarak sürebilirler. Herhangi bir I/O pini tüm cihaz için 25mA yada 100mA çekebilir.

Seçenekler ; Bir dizi hız, sıcaklık, paket I/O hatları, zamanlama fonksiyonlar, seri COM'lar, A/D ve hafıza kapasiteleri tüm ihtiyaçlarını karşılamak için PIC ailesi tarafından karşılanır.

Çok Yönlülük ; PIC çok yönlü bir mikrodur ve hacim olarak alanın önemli olduğu durumlarda birkaç lojik kapının yerini alabilecek kadar hacim bakımından ekonomiktir.

PIC FONKSİYON BLOK DİYAGRAMI



PIC16F84A(14Bit) BLOK DİYAGRAMI

Gizlilik ; PICmicro®MCU endüstride ki en iyi kod koruma olanaklarından birine sahiptir. Bir kere koruma biti programlandığında program hafızasının içeriği program kodunun tekrar inşa edilebilecek şekilde okunamaz.

Geliştirme ; PIC geliştirme ve üretim için OTP için pencereli şekilde (görsel anlamda) kullanıma sunulur.Geliştirme için ki araçlar okunabilir şekilde ve uğraştırıcı şekildedirler.

Kod Deneme ve Test Etme

C ile ayrıntılara inen bir programlama zorlayıcı bir durumdur. Devre emilatörü ve PIC için gerekli donanım projenin geliştirme aşamasında engelleyicidir. Bu diskte sunulan C derleyicisi internetten alınmıştır ve kod öğrenimi için bir test araç olarak bu kitaba dahil edilmiştir. I/O çıkışlarını kontrol edebilen PIC'e özel C derleyicilerine başlamadan önce temel kod örnekleri ve fonksiyonları denenebilir, test edilebilir ve görülebilirler.

C Kodlama Standartları

Program yazma ev inşa etmek gibidir. Eğer temeli güvenli ise geri kalanı üzerine inşa edilebilir (çıkılabilir). Eğer temel zayıf ise kod daha sonraki bir aşamada çöker. Aşağıdaki tavsiyeler C++ standart dökümanından alınmış ve PIC için uyarlanmıştır.

İsimler – Fonksiyonlarına Uydur

- İsimler programlamanın özüdür.
- Bu nedenle işlevine ve program içinde ne yaptığına uygun bir isim bul.
- Okunabilirliği arttırmak için birleşik isimler kullan.

ErrorCheck , ERRORCHECK'den daha kolaydır.

İsimleri okunabilirliği arttırmak için türü belirten küçük harf ile birleştir.

g	Global	**gLog**;
r	Reference	**rStatus();**
s	Static	**sValueIn;**

Parantezler {}

Parantezler veya köşeli parantezler klasik Unix tarzında ;

```
if (koşul) {
…….
}
```

yada okunması için daha kolay bir yöntemle kullanılabilir :

```
if (koşul)
{
…….
}
```

Sekmeler ve Girintili Yazma

Sekme yerine boşluk kullan çünkü 8'lik ayarlanmış tablolar sayfa genişliğini tüketir. Sadece yazılımın okunabilirliği için içeriden başla, ayrıca sekme ayarlarının bir editörden diğerine farklılık göstermesi kodun taşınabilirliğini azaltır.

Satır Uzunluğu

Monitörler ve yazıcılar arasındaki uyumluluk için satır uzunluğunu 78 karakter tut.

LC Formatı

Önceki if'in karşılamadığı koşulları yakalamak için else kullan:

```
if (koşul)
{}
else if (koşul)
{}
else
{
…….. /* Kapsanan bir şey değil, başka yakalamaz */
}
```

Koşul Formatı

Derleyicinin izin verdiği durumlarda her zaman bir eşitlik – eşitsizlik karşılaştırılmasının sol tarafında sabit kullan. Eğer bir '=' atılırsa derleyici sizin için hatayı bulur. Aynı zamanda değer öncelikli bir yere sahiptir.

```
if ( 6 == ErrorNum) …
```

Tüm Değişkenlere ilk değer ver

Bütün değişkenleri kayan yada rasgele koşullardan korumak için bilinen bir değer ata.

```
int a=6, b=0;
```

Yorumlar

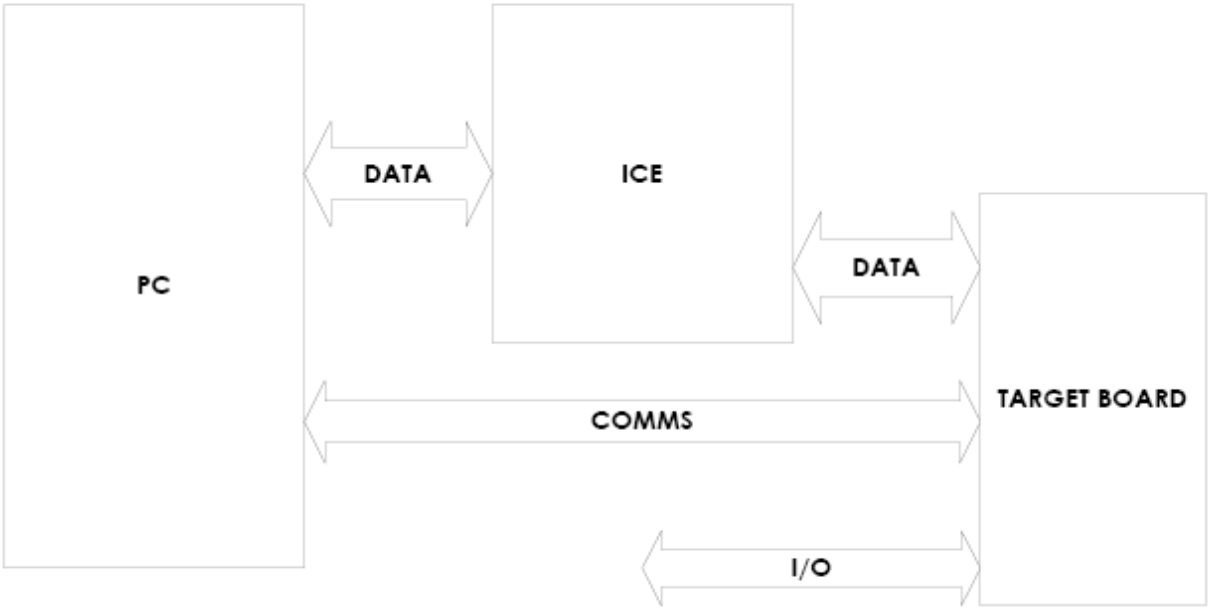
Yorumlar yazdığınız hikayenin diğer kısmını oluşturur. Siz bugün programınızın nasıl çalıştığını biliyorsunuz ancak 2 ay yada 2 yıl sonra siz yada başkası bunu hatırlayacak mı? Daha fazla çalışmanın gerektiği, gelişmelerin gerektiği yada debug'larının gerektiği yerlerde yorum kullan.

Temeller

Her bir programı bir başlangıcı vardır. Microdenetleyicilerde bu başlangıç reset vektörüdür. 14 bit core'da (16Cxx ailesi) reset 00h dır. 12 bit core'da (PIC 16Cx ve 12C50x) reset hafızada daha yüksek noktadadır. – 1FFh, 3FFh, 7FFh.

Bitme noktası program eğer bir kere çalıştırılırsa durdurulacağı noktadır. İletişimi ayarlayan rutin baud oranıdır. Trafik ışığı gibi diğer programlar başa dönecektir.Basiç C gibi yüksek seviyeli programlarda kullanılan en yaygın örneklerden biri ekrana "Merhaba Dünya" yazdırmaktır.

Eğer kılavye,işlemci,ekran bir PC'de bağlıysa C ve PC kullanmak kolaydır. Hook programı çavre cihazlarına bağlamak için temel hookların yerleştirilmesi gerekir. PICmicro®MCU veya herhangi bir mikroişlemci /mikrodanetleyici, C sistem için bir program geliştirirken sadece yazılım hooklarına değil mikroyu dış dünyaya bağlamak için fiziksel donanımada ihtiyaç duyulur. Böyle bir sistem aşağıda gösterilmektedir.



Böyle bir Layout kullanmak temel I/O'ların, COM'ların geliştirilmesi,test edilmesi ve debog edilmesine olanak sağlar. ICE'ın kullanımı (gerekli olmasa bile) geliştirme maliyetlerini ve mühendislerin baş ağrısını artırır. Bir projenin başlangıcındaki yatırım fazla görünebilir. Ancak geliştirmede kazanılan zaman ve debug daha sonra bunu telafi eder.

Bir tasarımı geliştirmek için kullanılan donanımı el yapımı PCB, protoboard veya hazır geliştirme kartı (bizim sunduğumuz PICmicro®MCU milenyum boardın kapsadığı gibi) olabilir.Milenyum boardın başlangıç için gerekli tüm temel donanımlara sahip iken ilk outlay'i minimumda tutar.

Aşagıdaki donanımı hangi formatta istiyorsanız toplayın. Size bir PIC programlayıcısı gerekecektir. C derleyicisine ek olarak minimal outlay'daki PIC Start Plus gibi.

Mühendisler PIC öğretirken kullandoğı basit bir programdüğmeye bas LED'i yakmaktır. Basit bir kodla başla 2000 satırlık bir şeyle değil.

Assembler'de bir örnek

```
main    btfss    porta,switch    ;test for switch closure
        got      main            ;loop until pressed
        bsf      portb,led       ;turn on led
lp1     btfsc    porta,switch    ;test for switch open
```

```
        goto    lp1     ;loop until released
        bcf     portb,led   ;turn off led
        goto    main    ;loop back to start
```

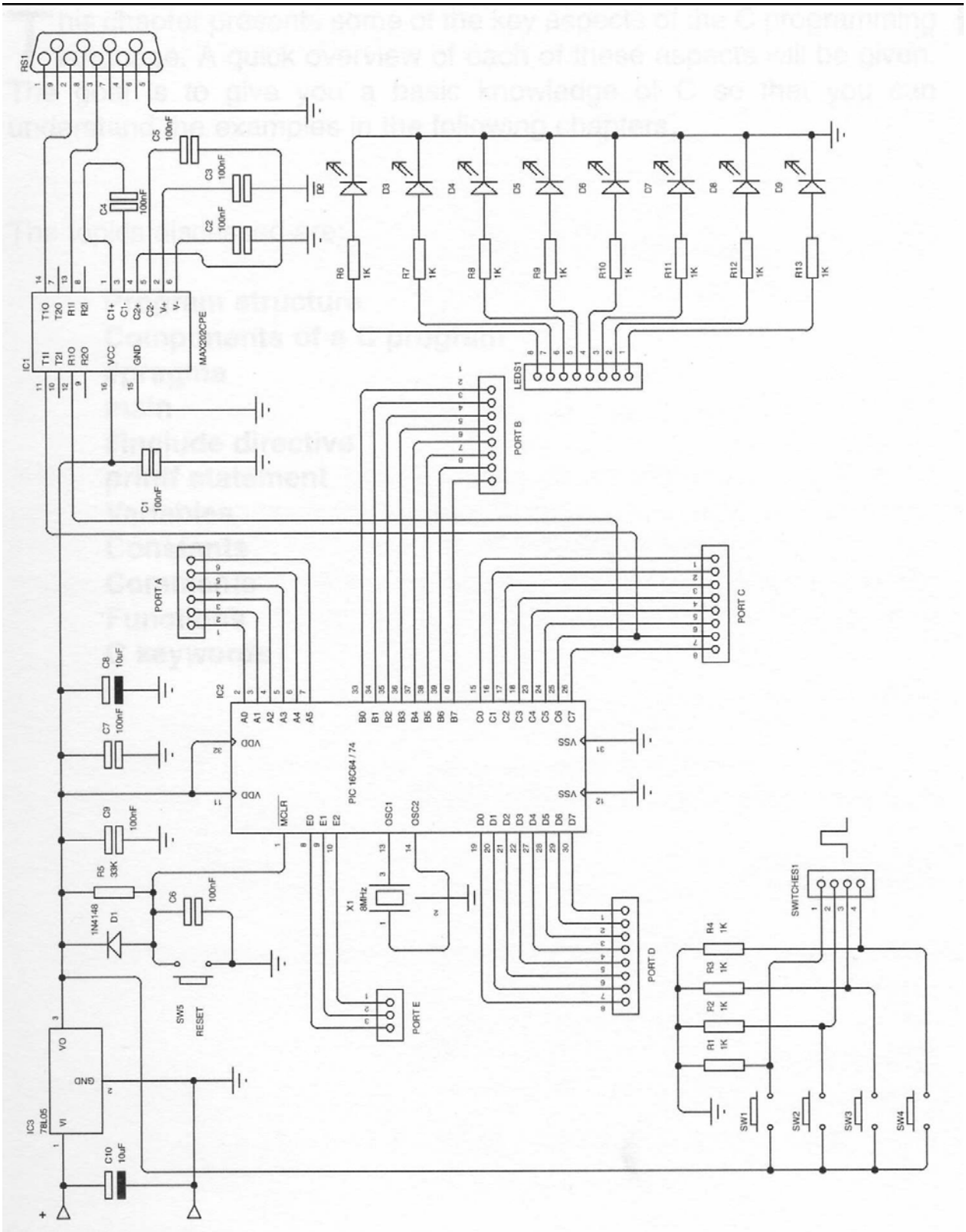
Bunu C'de yazalım

```
main()
{
    set_tris_b(0x00); //set port b as outputs
    while(true)
    {
        if (input(PIN_A0)) //test for switch closure
            output_high(PIN_B0); //if closed turn on led
        else
            output_low(PIN_B0); //if open turn off led
    }
}
```

Assemble olduğunda buna benzer.

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

Görebileceğimiz gibi derlenmiş versiyon hafızada daha çok kelimeye sahiptir. C de 14 Assemle'de 9 , bu kodda adil bir örnek değil ama program büyüdükçe C kodu daha etkin olur.



C ESASLARI

Bu bölüm C programlama dilinin ana yönlerinin bazılarını takdim etmektedir.Bu yönlerinin her birinin hızlı bir bakışı verilecektir. Amac ; ileriki bölümlerdeki örnekleri anlayabilir bir temel C bilgisi vermektir.

Ele alınan konular;

Program yapısı
Bir C programının parçaları
#pragma
main
#inculude direktifi
printf ifadesi
Degişkenler
Değişmeyenler
Yorumlar
Fonksiyonlar
C ana kelimeleri

1.1 C PROGRAMLARININ YAPISI

Tüm C programı ön işlemci direktifleri, bildirileri, tanımları, ifadeleri, değimleri ve fonksiyonları içermektedir.

Önişlemci Direktifi

Bir önişlemci direktifi C ön işlemcisinde bir komuttur(bir programı derlerken ki ilk adımı otomatik çağıran komut).İki en çok ortak önişlemci direktifleri **#define** direktifi, belirlenmiş tanıtıcı için metni değiştiren, ve #include direktifi, bir program içinde harici bir dosya metnini içerir.

Bildiri

Bir bildiri değişkenlerin öznitelikleri ve isimleri, fonksiyonları, ve programda kullanılan metinleri saptar. Genel değişkenler dış fonksiyonları bildirir ve dosyanın sonundaki bildirinin sonundan görülürler. Bir yerel değişken bir fonksiyon içerisinde bildirilir ve fonksiyonun sonundaki bildirinin sonundan görülebilir.

Tanım

Bir tanım bir fonksiyon veya değişkenin içeriklerini tanımlar. Bir tanım aynı zaman da fonksiyonlar ve değişkenler için gerekli yeri ayırır.

İfade

Bir ifade tek bir değer veren operatörler ve işleçlerin bir kombinasyonudur.

Deyim

Deyimler bir C programında program işletiminin düzenini yada akışını kontrol eder.

Fonksiyon

Bir fonksiyon bildirilerin, tanımların, ifadelerin ve spesifik görevi gerçekleştiren deyimlerin bir toplamıdır. Fonksiyonlar C de yuvalanmış olmayabilir.

main Fonksiyonu

Tüm C programları program işletiminin başladığı yerde **main** fonksiyonunu içermelidir. Main fonksiyonunun kapsamı programın başlangıç ve bitiş noktasını tanımlamaktır.

Örnek: Genel C program yapısı

```
#include <stdio.h>              /* önişlemci direktifi */
#define PI 3.142                /* C başlık dosyasını içerir  */
float area;                     /* genel  bildiri*/
int square (int r);               /* protatip bildiri*/
main()
{                               /* ana fonksiyonun başlangıcı */
        int radius_squared;     /* yerel bildiri */
        int radius = 3;         /* bildiri ve başlangıç durumuna getirme */
        radius_squared = square (radius);
```

```
                                    /* fonksiyona bir değer atama  */
        area = PI * radius_squared;
                                     /* atama değimi */
        printf("Area is %6.4f square units\n",area);
}                                    /* ana fonksiyon ve programın sonu */
square(int r)                        /* fonksiyon başı */
{
        int r_squared;               /* buradaki bilinen bildiriler*/
                                    /* sadece kareye  */
        r_squared = r * r;
        return(r_squared);          /* deyim çağırma değerine dönüş */
}
```

1.2 BİR C PROGRAMININ PARÇALARI

Tüm C programları fonksiyonlar ve deyimler gibi esas parçaları kapsar. Deyimler işlemleri gerçekleştiren programın parçalarıdır. Tüm C programları bir veya daha fazla fonksiyon içerir. Fonksiyonlar alt programlar, bir yada daha fazla değim içerenlerin her biri ve programın diğer parçalarıdır. Programları yazarken girintili yazma boşluk bırakma yorumlar okunabilir şekilde olmalıdır. Bir sonra ki örnek C programının gerekli parçalarının bazılarını gösterir.

```
#include <stdio.h>
/* ilk C programı */
main()
{
        printf("Merhaba Dünya!");
}
```

#include <stdio.h > deyimi program içinde ki ‘stdio.h’ kaynak dosyasından derlediği anlamına gelir. Uzantısı .h olanlar header dosyasıdır. Bir başlık dosyası programda kullanılan standart fonksiyonlar hakkında bilgi içerir. Başlık dosyası olan stdio.h dosyası STandarD Input and Output anlamına gelir ve giriş ve çıkış fonksiyonlarının çoğunu içerir.

/* ilk C programım / C’de bir yorumdur. Yorumlar /* dan öncegelir ve */ ile sonlanır. Yeni sitil yorumlar // ile başlar ve satır sonuna kadar gider. Yorumlar derleyici tarafından yok sayılırlar ve bu yüzden derlene kodun uzunluğu veya hızına etki etmezler.

Tüm C programları bir **main()** fonksiyonuna sahip olmalıdır. Bu program içerisindeki giriş noktasıdır. Tüm fonksiyonlar aşağıdaki biçimde olmalıdır.

```
Fonksiyon isim()
{
        kod
}
```

Bir fonksiyondaki değimler sırasıyla yürütülürler. { ile başlar } ile sonlandırılırlar.

{ ve }işaretleri sırasıyla C deki kod bloklarının başlangıcını ve bitişini gösterir.

Son olarak **printf("Merhaba dünya!");** deyimi; mevcut tipik bir C değimidir. Neredeyse tüm C deyimleri bir noktalı virgül (;) ile sonlanır.

Bu yüzden bir satırdaki deyimlerin numaralarında veya bir hat içinde deyimlerin durumlarına etki etmez.

Tüm deyimlerin sonlarındaki noktalı virgül (;) derleyiciye deyimin sonuna ulaştığı ve bir sonraki deyimden ayrılmasını bildirir. Başarısızlık genellikle sonraki bir satırdaki bir hata ile oluşur. **if** deyimi birleşik bir deyimdir ve birleşik deyimin sonunda **;** olması gerekir.

```
if (Bu doğru)
   Bu fonksiyonu yap();
```

1.3 #PRAGMA

Pragma komutu derleme anındaki parçalı bir işlemi gerçekleştirmeyi derleyiciye okutur. Dosya formatı yaratırken yada PICmicro®MUC kullanımında belirtildiği gibi

```
#pragma device PIC16C54
```

CCS C'de pragma seçmelidir böyle kabul edilir.

1.4 main()

Her program sadece bir kez kullanılan bir **main** programına sahip olmalıdır. Hiçbir parametre takip eden () içinde bulunmamalıdır. **Void** anahtar sözcüğü parametrelerin olmadığı (ve) arasında görülebilir. main gibi sınıflanmış bir fonksiyondur, tüm ikili arasında olması gereken kodlardır.

```
main()
{
    program yapısı
}
```

1.5 #include

Başlık dosyası (bir .h uzantısında belirtilen) özel bir PIC için PICmicro®MUC hafızalarının yeri yada fonksiyon dönüşlerinin ve fonksiyon kabullerinin ne anlamda olduğunu içeren kütüphane fonksiyonları hakkında bilgi içerir.

```
#include <16C54H>
```

Bu bilgi derleyici tarafından tüm donanım özellikleri ve kaynak programlarla birlikte bağlantı kurma da kullanılır.

```
#include <16c71.h>
#include <ctype.h>
#use rs232(baud=9600,xmit=PIN_B0,rcv=PIN_B1)
main()
```

```
{
    printf("Karakter giriniz:");
    while(DOĞRU)
        putc(toupper(getc()));
}
```

PIN_B0 ve **PIN_B1** tanımları 16C71.H başlık dosyasında bulunmaktadır. **Toupper** fonksiyonu **CTYPE.H** başlık dosyasında bulunmaktadır. Bu başlık dosyalarının ikisi de programda kullanılmak zorundadır.

Kullandığımız fonksiyonlar hakkında derleyicinin gerekli bilgiyi derleyicinin sahip olmasını sağlar. **printf** ve **putc**. Gibi giriş çıkış fonksiyonları için başlık dosyaları çoğu C derleyicilerine gereken nottur. Bunlar bir ayırıcı başlık dosyası gerektirmeyen ve **#use rs232** yolu ile çekilen PICmicro®MUC için yaratıcı fonksiyonlardır.

#include <thisfile.h>

Dosya için dosya yöneticilerini içeriğine bakılmasın ön işlemciye anlatır.

#include "thatfile.h"

Muhtemelen **#include** direktifinin sonunda noktalı virgül (;) yoktur. Bunun sebebi **#include** direktifinin bir C deyimi olmamasıdır, fakat onun yerine derleyiciye bir ön işlemci direktifi denilebilir. **include** dosyasının tümü derleme aşamasında kaynak dosyasının içerisine konulmuştur.

1.6 printf Fonksiyonu

Printf fonksiyonu programcıya yazılabilir bilgi göndermeye izin veren standart bir kütüphane fonksiyonudur. Bir **printf** deyiminin genel biçimi;

printf("control_string", argument_list);

Bir **control_string** ; her son da çift alıntılı bir stringdir. Bu string içinde, özel sembollerin belirteçleri bir % işareti ile belirtilir. **control_string** her zaman **printf()** fonksiyonu içinde olmalıdır. Biçim string'in de biçim belirteçleri yoksa bir argument-list gerekmeyebilir. argument-list değişkenlerin ve sabitlerin bileşimi olabilirler. Sonraki iki örnek **printf()** deyimlerinin bir sabit ve bir değişken sonrası kullanımını gösterir.

```
printf("Mrehaba dünya!");
printf("Microchip® is #%d!",1);
```

Biçim belirteçleri (%d) veri gösteriminin tipine bağlıdır. Aşağıdaki tabloda C'deki biçim belirteçlerinin tamamı gösterilmektedir.

Biçim belirteçleri printf()
%c single character

%d signed decimal interger
%f floating point (decimal notation – must include)
%e floating point (exponential or scientific notation)
%u unsigned decimal integer
%x unsigned hexadecimal integer (lower case)
%X unsigned hexadecimal integer (upper case)
l prefix used with %d, %u, %x to specify long integer

NOT: Bir O dan sonra, bir biçim stringe içindeki %a karakteri baştaki sıfırların yazılması mecburdur. Numaranın ardından kalıp alanının genişliği belirtilir.

printf("The Hex of decimal 12 is %02x\n",12);

Bu şekilde de gösterilir.
Ondalık 12 nin on altılık değeri OC'dir.

Boşluk dizisi

\n yeni satır	**\t yatay boşluk**
\r satır başına dönüş	**\f biçimleme**
\' tek alıntı	**\" çift alıntı**
\\ ters slash	**%% yüzde işareti**
\? Soru işareti	**\b geri boşluk**
\0 nurasgele karakter	**\v dikey boşluk**
\xhhh heks kodu hhh koyar	

Biçim belirtmeleri % [bayraklar] [genişlik] [küsurat], bir önceki örnekteki satır ;

printf("Area is %6.4f square units\n",area);

Farklı şekilde çıkışı portunda da gösterilebilir. Çıkış bununla birlikte kendi çıkış fonksiyonunu tanımladığın hiçbir şeye etki etmez.
Örneğin;

```
void lcd_putc(char c)
{
   // çıkışa bir kod ver
   // karakteri LCD ye gönder
}
printf(lcd_putc, "değer %u dur", değer);
```

1.7 Değişkenler

Bir değişken belli bir hafıza yeri için bir isimdir. Bu hafıza yeri değişkenin nasıl bilindiğine göre değişen çeşitli değerler tutabilir. C de tüm değişkenler kullanılmadan önce belirtilmek zorundadır. Bir deyişken bildirimi derleyicinin ne tip değişken kullandığını anlatır. Tüm deyişken bildirimleri C de deyimdirler ve bu yüzden (;) notalı virgül ile sonlandırılmalıdırlar.

C nin desteklediği bazı temel veri tipleri **char**, **int**, **float**, ve **long** dur. Bu değişken belirtimi için genel biçim;

tip değişken_isim;

Değişken belirtimine bir örnek **char ch;** dir Derleyici bu deyimi ch değişkeninin bir char(karakter) gibi belirtildiği şeklinde yorumlar.

1.8 Constants

Sabitler program tarafından değiştirilemeyen sabit değerlerdir. Örneğin 25 bir sabittir. Tam sayı sabitleri kesirli parçalı olmayacak şekilde belirtilir. -100 yada 40 gibi. Tam sayı olmayan sabitler rakamın küsuratını ayırmak için ondalık nokta gerekir.465,75 sayısı bir noktalı sabittir. Karakter sabitleri 'A' veya &' gibi tek ayraçlar içine konulurlar.

Derleyici programınızda bir sabit ile karşılaştığın da, ne çeşit sabit olduğuna karar vermek zorundadır. C derleyicisi, tutacağı en küçük bağdaşabilir veri tipinin içine sabit uydurur. 15 bir **int** ve 64000 bir **unsigned** (işaretsiz) dir. Bir sabit **#define** deyimi kullanılarak belirtilebilir.

#define <etiket> değer

<etiket> programınızın başından sonuna kadar kullanacağınız isim tanımlar, **value** (etiket) atadığımız değerdir.

```
#define DOĞRU 1
#define pi 3.14159265359
```

C size heksedesimal ve oktal biçimleri belirtmeye izin verir. Heksedesimal sabitler 'ox' ön eki almak orundadırlar. Örneğin 0xA4 geçerli bir heksedesimal sabittir. Nümerik sabitlere ek olarak, C, string sabitleri destekler. String sabitleri çift tırnak içine konulan karakterler dizisidir.

Sabitler ön işlemci olarak adlandırılan kısımda kod derlenmeden önce metin kısımlarını gerçekleştiren **#define** ile tanımlanır.

ile başlayan direktifler ön işlemci direktifleri olarak adlandırılır. Örneğin;

```
#define YAŞLI_DEĞİL (YAŞ<65)
.
.
.
.
.
if YAŞLI_DEĞİL
printf("GENÇ");
```

#define verisi hafızada saklanmaz, derleme anında çözülür. **ROM** da saklı sabitler, değişken bildiriminde **const** anahtar sözcüğünü kullanır.

char const id[5]={“1234”};

Biz string tutmaya 5 yer kullanırız, çünkü string (10) karakter ile sonlandırılır.

1.9 Comments

Yorumlar, kaynak kodun işlevini ve anlamını belgelemeye kullanılırlar. Tüm yorumlar derleyici tarafından yok sayılırlar.Bir yorum, bir C anahtar sözcüğü ortası, fonksiyon isim yada değişken ismi için hariç programın hiçbir yerinde bulunmaz. Yorumlar çok uzun satırlar olabilir ve geçici olarak kodun bir satırını kopyalayacak şekilde kullanılabilir.

Yorumlar iki biçime sahiptir. İlk biçimi tüm C derleyicileri tarafından kullanılır ve ;

/* Bu bir yorumdur */

İkinci biçim ise çoğu C derleyicisi tarafından desteklenmektedir ve ;

// Bu bir yorumdur

ALIŞTIRMA : Aşağıdaki yorumların hangisi geçerlidir?

/* Benim yorumum çok kısa */

/* Benim yorumum çok çok çok çok çok çok çok çok çok çok uzun ve geçerlidir*/

/* Bu yorum /* tam */ görünüyor fakat geçerli değil */

1.10 Fonksiyonlar

Fonksiyonlar bir C programının temel yapı taşlarıdır. Tüm C programları en azından bir fonksiyon içerir, **main()**. Yazacağınız çoğu fonksiyonlar bir çok fonksiyon içerecektir. Bir çok fonksiyonu bulunan C programı için biçim;

```
main()
{
   fonksiyon1()
   {
   }
   fonksiyon2()
   {
   }
}
```

main(), program işlerken çağrılan ilk fonksiyondur. Diğer fonksiyonlar **function1()** ve **function2()**, programdaki hiçbir fonksiyon ile çağrılmazlar.

Genel **main()** diğer hiçbir fonksiyon tarafından çağrılmaz, bununla birlikte, C'de hiç sınırlama yoktur.
Aşağıdaki C iki fonksiyonlu bir örnektir.

```
main()
{
    printf("I ");
    fonksiyon1();
    printf("c.");
}
function1()
{
    printf("benzer ");
}
```

Kendi fonksiyonlarınızı yazarken, bir fonksiyon { ile kapatıldıysa bu fonksiyonun bitimi için bir hatırlatıcıdır, fonksiyon asıl çağırıldığı noktadan sonra program bir satır kodu işlemeye başlayacaktır. 3.1 de görebilirsiniz.

1.11 Makrolar

#define önceki bölümü örneklerle açıklayacak kadar güçlü bir direktiftir. C daha güçlü şekilde parametreler üretmeyi tanımlamaya izin verir. Parametreler kullanıldığında bir makro gibi çağrılır. Makrolar okunabilirliği artırmak yada yazmayı korumada kullanılır. Örnek bir makro :

```
#define var(x,v) unsigned int x=v;
var(a,1)
var(b,2)
var(c,3)
```

benzer şekilde :

```
unsigned int a=1;
unsigned int b=2;
unsigned int c=3;
```

İfadeler bölümünü okuduktan sonra daha anlaşılabilir bir diğer örnek :

```
#define MAX(A,B) (A>B)?A:B)
z=MAX(,y); // z daha büyük  x veya y değeri içerecek
```

1.12 Koşul derleme

C, derleme zaman ayarları tabanlı derlemelerden doşlana veya içeren kod bölümlerine izin veren bazı özel ön işlemci direktiflerine sahiptir. Aşağıdaki örneği düşünelim ;

```
#define HW_VERSION 5
#if HW_VERSION>3
output_high(PIN_B0);
#else
output_high(PIN_B0);
#endif
```

Yukarıda HW_VERSİYONU'nun ayarlarına bağlı olan sadece tek satır derlenecektir. Bir dosyada **#if** in düzinelercesi olabilir ve bir sabit değiştirirerek farklı donanım versiyonu için benzer kod derlenebilir. Kod, program çalışırken değer biçen normal **#if** den farklı derlendiği zaman **#if** değer biçer ve bitirir.

Bir ID **#defined** ise **#ifdef** tümüyle görmeyi kontrol eder.
Örnek :

```
#define DEBUG
#ifdef DEBUG
printf("ENTERING FUCT X");
#endif
```

Bu örnekte ; bir satırında çıkarma yada yorumlama ile programdaki tüm debug satırları derlemeden elenebilir.

1.13 Donanım Uyumluluğu

Derleyici donanım hakkında bilgi sahibi olması gerekmektedir ki kodu doğru bir şekilde derleyebilsin. Tipik bir program başlangıcı aşağıdaki gibidir.

```
#include <16c74.h>
#fuses hs,nowdt
#use delay(clock=8000000)
```

İlk satır, pin adları gibi özel #define araçları içermektedir. İkinci satır PICmicro®MCU sigortalarını set eder. Mesela yüksek hızlı osilatör (hs) ve bekleme zamanlayıcısı kapalı (nowdt) dir. Son satır derleyiciye osilatör hızının ne olduğunu anlatır. Aşağıda bazı diğer örnek satırlar vardır :

```
#use rs232(buad=9600,xmit=PIN_C6,rcv=PIN_C7)
#use i2c(master,scl=PIN_B6,sda=PIN_B7)
```

Bu kitaptaki örnek program donanım tanımlama satırlarını göstermez. Bunlar derlemeyi ve bu programları RCW yapmayı gerektirir.

Ek olarak, C değişkenleri donanım kaydedicilerine haritalanabilir ve oluşturulabilir. Bu değişkenler bitler yada baytlar olabilir. Onlar tanımlandıktan sonra programda diğer hiçbir değişken ile kullanılmaz. Örnekler :

```
#bit carry=3.0
#byte portb=6
#byte intcon=11
```

1.14 C Anahtar Sözcükleri

ANSI C standardı C dilinde kullanmak için 32 anahtar sözcük tanımlanır. C'de, belirli kelimeler döngülerde kullanılmak için yada veri tiplerini tanımlamayı derleyici tarafından kullanmak için saklanmıştır. Tüm C anahtar sözcükleri, onları derleyici tanıması için lowercase içinde olmalıdır. Tipik olarak, çoğu C derleyicileri işlemci mimarisinin avantajını kullanan birkaç anahtar sözcük ekleyecektir.

Aşağıda değişken isimleri olarak kullanılan, rezerve edilmiş anahtar sözcüklerin bir listesi vardır.

auto	**double**	**int**	**struct**
break	**else**	**long**	**switch**
case	**enum**	**register**	**typedef**
char	**extern**	**return**	**union**
const	**float**	**short**	**unsigned**
continue	**for**	**signed**	**void**
default	**goto**	**sizeof**	**volatile**
do	**if**	**static**	**while**

ALIŞTIRMA:

1-) İsminizi ekrana yazan bir program yazınız.

2-) Yıl olarak adlandırılan bir tam sayı belirten bir program yazınız. Bu değişkene şimdiki yılın değeri verilsin ve sonra, bir **printf()** değimi kullanılarak, ekrana yılın değerini göstersin. Program sonunda bunu görmelisiniz: **Bu yıl 1998 dir**.

Değişkenler

C dilinin önemli bir noktası da veriyi nasıl depoladığıdır. Bu konuda C dilinin veri depolanmasında değişkenleri nasıl kullandığını daha yakından inceleyeceğiz.

Bu konuda tartışılacak konular

veri tipleri
bildirimler
atamalar
veri tiplerinin sınırları
tip dönüşümleri

2.1 Veri Tipleri

C programlama dili beş temel veri tipini ve dört tip dönüştürücüsünü desteklemektedir. Aşağıdaki tablo temel veri tiplerinin ve tip dönüştürücülerinin anlamlarını açıklamaktadır.

Tip	Açıklaması	Kullanım şekli
karakter	karakter verisi	char
tam sayı	işaretsiz tam sayılar	int
reel sayı	reel sayılar	float
çift doğruluklu sayı	çift doğruluk değerindeki sayılar	double
boş	değer yok	void
işaretli	pozitif ya da negatif sayı	signed
işaretsiz	sadece pozitif sayılar	unsigned
uzun sayılar	daha geniş bir aralıktaki sayılar	long
kısa sayılar	daha dar bir aralıktaki sayılar	short

Her bir veri tipi kullanılan dönüştürücüye de bağlı olmak üzere belirli bir değer aralığını ifade etmektedir. Aşağıdaki bir sonraki tablo ise temel veri tipleri ile dönüştürücülerin olası tüm kombinasyonları için oluşabilecek değer aralıklarını göstermektedir.

Tip	Bit uzunluğu	Değer aralığı
short	1	0 veya 1
short int	1	0 veya 1
int	8	0 ile 255 arası
char	8	0 ile 255 arası
unsigned	8	0 ile 255 arası
unsigned int	8	0 ile 255 arası
signed	8	-128 ile 127 arası
signed int	8	-128 ile 127 arası
long	16	0 ile 65536 arası
long int	16	0 ile 65536 arası
signed long	16	-32768 ile 32767 arası
float	32	3.4E-38 ile 3.4E+38 arası

NOT: Veri tipleri ve değer aralıkları için farklı C derleyici belgelerini okuyunuz.

C dili unsigned int, short int ve long int veri tipleri için kısa yazımlı bir notasyonu da desteklemektedir. Bu ifadeler yerine int ibaresi hiç kullanılmadan, kısaca unsigned, short ve long ifadeleri kullanılabilir. CPU'nun aritmetik işlemleri daha kolay yapabilmesini sağlamak için C dili negatif değerleri 2'ye tümleyen biçiminde saklar. Bir sayının 2'ye tümleyenini bulmak için her biti tersleyip sonuca 1 eklemek yeterlidir. Örneğin, işaretli 29 sayısının 2'ye tümleyenini bulmak için:
00011101 = 29

```
11100010        tüm bitleri tersle
       1   =    1 ekle
11100011        -29
```

Yine bir örnek olarak uzun sayı tipinde olan 12000 değerini a değişkenine atayalım. 12000'in onaltılık sistemdeki karşılığı 2EE0 değeridir. Aşağıdaki kod bloğu düşük değerlikli kelimeyi (E0), 11h yazmacına, yüksek değerlikli kelimeyi (2E) ise 12h yazmacına yazacaktır.

```
long a = 12000;
main()
{
        0007:  MOVLW    E0
        0008:  MOVWF    11
        0009:  MOVLW    2E
        000A:  MOVWF    12
}
```

ALIŞTIRMA:

1. Aşağıdaki ifadeyi farklı bir şekilde yazın

```
long int i;
```

2. İşaretli bir sayı ile işaretsiz bir sayının farkını anlamk için aşağıdaki programı yazın. İşaretsiz bir tamsayı olan 35000 değeri işaretli tamsayı formatında -30536 şeklinde yazılmaktadır.

```
main()
{
        int i;                  /* işaretli tamsayı */
        unsigned int u;         /* işaretsiz tamsayı */
        u=35000;
        i=u;
        printf("%d %u\n",i,u);
}
```

2.2 Değişken bildirimi

Değişkenler temel olarak iki yerde bildirilebilirler: fonksiyonun içinde ya da dışında ve tanımlandıkları yere göre de sırasıyla yerel ya da global değişken adını alırlar. Değişkenler aşağıda gösterildiği gibi tanımlanırlar:

```
tip     değişken_adı
```

Bu ifadede tip C dili tarafından geçerli bir veri tipi, değişken_adı ise değişkene vereceğimiz adı temsil etmektedir.

Yerel değişkenler (fonksiyonun içinde bildirilenler) sadece tanımladıkları fonksiyon bloğu içerisinde yer alan ifadelerce kullanılabilirler.

Yerel bir değişkenin değerine bildirildikleri fonksiyonun dışındaki fonksiyonlar içinde yer alan ifadelerce erişilemez. Burada hatırlanması gereken en önemli konu yerel değişkenlerin bellekte fonksiyon başlatılırken yaratılıp, fonksiyon sonlandığında silindikleridir. Aynı zamanda yerel değişkenlerin bildirimi fonksiyonun başlangıcında, ifadelerden önce yapılmalıdır.

Farklı fonksiyon bloklarının içerisinde yer alan yerel değişkenlerin aynı ismi almasına izin verilmektedir. Aşağıdaki örneği ele alalım:

```
void f2 (void)
{
        int count;
        for (count=0; count<10;count++)
                printf("%d \n",count);
}

f1()
{
        int count;
        for (count=0;count<10;count++)
                f2();
}

main()
{
        f1();
        return 0;
}
```

Bu program 0'da 9'a kadar olan sayıları ekrana on kez yazdıracaktır. Programın işleyişi her iki fonksiyonun içinde yer alan count değişken adından etkilenmeyecektir.

Öte yandan global değişkenler pek çok fonksiyon tarafından kullanılabilirler. Global değişkenlerin kullanılacakları tüm fonksiyonlardan önce bildirilmeleri gerekmektedir. En önemlisi, programın işleyişi sona eresiye kadar bellekteki yerlerinden silinmezler.

Aşağıdaki örnek global değişkenlerin nasıl kullanıldığına örnektir:

```
int max;
f1()
{
        int i;
        for (i=0;i<max;i++)
                printf("%d \n",i);
}
```

```
main()
{
        max=10;
        f1();
        return 0;
}
```

Bu örnekte hem main() hem de f1(), max değişkenine referans vermektedir. main() fonksiyonu max değişkenine değer atamakta ve f1() fonksiyonu da max değerini for döngüsünde kullanmaktadır.

ALIŞTIRMA:

1. Yerel ve global değişken arasındaki ana farklılıklar nelerdir?
2. Hem yerel hem de global değişkenler C dili içerisinde aynı ismi alabilirler. Aşağıdaki programı yazın.

```
int count;

f1()
{
        int count;
        count=100;
        printf("f1() fonksiyonundaki count degeri: %d\n", count);
}

main()
{
        count=10;
        f1();
        printf("main() fonksiyonundaki count degeri: %d\n",count);
        return 0;

}
```

main() içerisinde count değişkenine yapılan referans global bir değişkeni belirtmektedir. f1() içerisinde ise count yerel değişkeni global değişken değerini önüne geçerek kendi değerini kullandırtmaktadır.

2.3 Değişkene değer atanması

Şu ana kadar sadece değişkenlerin nasıl bildirildiğini gördük, fakat hiç gerçekten değer atamadık. Değişkenlere değer atanması işlemi aslında oldukça basittir:

```
değişken_adı = değer;
```

Değişkene değer atanması da bir C dili ifadesi olduğundan sonuna noktalı virgül konmalıdır. 100 değerinin bir tamsayı değişkenine atanmasına dair örnek aşağıda verilmiştir:

```
count = 100;
```

Burada 100 değerine sabit adı verilmektedir.

Değişken değerleri bildirimin yapıldığı yerde atanabilir. Bu yöntem özellikle programda kullandığınız değişkenlerin ilk değerlerinin bilinmesi için güvenli ve basit bir metoddur. Örneğin:

```
int a=10, b=0, c=0x23;
```

C dili pek çok farklı tipte sabit içermektedir. Karakter tipi bir sabit 'M' örneğinde olduğu gibi harfin tırnak işareti içine yazılması ile yazılır.

Tamsayı tipinde bir değişkene atama yapılırken tam sayılar yazılır. Reel sayı tipindeki bir değişkene ise mutlaka ondalık işareti olan nokta kullanılarak atama yapılmalıdır. Örneğin C dilinde 100 değerinin reel bir sayı olduğunu belirtmek için 100.0 yazılmalıdır.

Bir değişken aynı zamanda başka bir değişkenin değerini de alabilir. Aşağıdaki program bu kullanımı örneklemektedir:

```
main()
{
        int i;
        int j;
        i=0;
        j=i;
}
```

ALIŞTIRMA:

1. count adında bir tamsayı değişkenin tanımlandığı bir program yazın. count değişkenine 100 değerini atayın ve printf() komutu ile değeri ekrana yazdırın. Programın çıktısı aşağıdaki gibi olmalıdır:

```
count degiskeninin degeri : 100
```

2. Değişken adları ch, f ve d olmak üzere sırasıyla char, float ve double tipinde üç değişkenin tanımlandığı bir program yazınız. Karakter tipindeki değişkene 'R' değerini, reel sayı tipindeki değişkene 50.5 değerini ve çift doğruluklu sayı tipindeki değişkene de 156.007 değerini atayın. Bu değişkenlerin değerlerini sırasıyla ekrana yazdırın. Programın ekrandaki görüntüsü aşağıdaki gibi olmalıdır:

```
ch degeri: R
f degeri: 50.5
d degeri: 156.007
```

2.4 Sıralı listeler (Enumeration)

C dilinde adlandırılmış tamsayı sabitleriyle yaratılmış listeler kullanmak mümkündür. Bu bildirime sıralama adı verilir. Sıralama ile yaratılmış sabitler tamsayıların kullanılmasına ihtiyaç duyulan her yerde kullanılabilirler. Bir sıralama bildiriminin genel yapısı aşağıdaki gibidir:

```
enum ad {sıralı liste} değişken(ler);
```

Değişken listesi sıralamanın seçeneksel bir elemanıdır. Sıralama değişkenleri sadece sıralı listede belirtilmiş olan değerleri kullanabilirler. Örneğin aşağıdaki ifadede:

```
enum renk_tipi {kirmizi, yesil, sari} renk;
```

renk değişkenine program içerisinde sadece kirmizi, yesil ya da sari değeri atanabilir. (Örnek: renk=kirmizi;)

Derleyici ilk girişin değeri 0 olmak üzere sıralı listedeki her elemana tamsayı değerler atayacaktır. Her bir elemanın değeri bir önceki elemandan bir büyüktür. Bu durumda yukarıdaki örnek için kirmizi'nin değeri 0, yesil'in değeri 1 ve sari'nin değeri 2 olacaktır.Varsayılan değer olarak atanan bu değerlerin üzerine istenildiği taktirde başka değerler de atanabilir. Bu örnekte de bu yöntem gösterilmiştir:

```
enum renk_tipi {kirmizi, yesil=9, sari} renk;
```

Bu durumda ise kirmizi'ya 0, yesil'e 9, sari'ya ise 10 değeri atanır.

Sıralı liste bir kez tanımlandığı taktirde listenin adı programın ilerleyen noktalarında yeni değişkenler yaratmak üzere kullanılabilir. Örneğin benim_rengim değişkeni renk_tipi sıralı listesi ile aşağıdaki şekilde tanımlanır.

```
enum   renk_tipi benim_rengim;
```

Bir değişken aynı zamanda başka bir değerle de sınanabilir:

```
if (renk==meyve)
        // yapilacak islemler
```

Aslında en temelde sıralı listeler kodun yazımını kolaylaştırır. Bir değeri bir değişkene atamak yerine değerin anlamını açıklamak için sıralı listeler kullanılabilir.

ALIŞTIRMA:

1. PIC17CXX ailesi için bir sıralı liste yaratın.
2. Banknot halinde bulunan paralara ait bir sıralı liste yaratın.
3. Aşağıdaki kod parçası doğru mudur? Neden?

```
enum {PIC16C51, PIC16C52, PIC16C53} cihaz;
```

```
cihaz=PIC16C52;
printf("Ilk bulunan PIC cihazi: %s\n",cihaz);
```

2.5 typedef komutu

typedef ifadesi ile varolan tipler aracılığı ile yeni tipler yaratılarak tanımlanır. Yapısı aşağıdaki gibidir:

```
typedef eski_isim yeni_isim
```

Bu noktadan sonra yeni yaratılan adla değişken bildirimi yapılabilir. Örneğin, aşağıdaki program bloğu işaretli char tipi değişken yerine yeni yaratılmış olan smallint tipi bir değişken kullanmaktadır.

```
typedef signed char smallint;
main()
{
        smallint i;
        for (i=0;i<10;i++)
                printf("%d \n",i);
}
```

typedef ifadesini kullanırken iki nokta akıldan çıkmamalıdır. İlki, typedef ifadesi varolan orijinal ismi ya da tipi yok etmez; önceki örnekte signed char tipi hala daha geçerli ve var olan bir tiptir. İkincisi ise birden fazla typedef ifadesi ile aynı orijinal tipi temel alarak çok sayıda yeni adda tip yaratılabilir.

typedef ifadesi genellikle iki sebepten dolayı kullanılır. Birincisi taşınabilir programlar yazmak. Eğer yazdığınız program hem 16-bitlik hem de 32-bitlik tamsayı tipini kullanan makinelerde çalışacaksa sadece 16-bitlik tamsayı tipinin kullanıldığından emin olmak istersiniz. Program 16-bitlik bir makine için yazılırken

```
typedef int myint;
```

şeklinde yazılan ifade ile tüm tamsayılar 16-bitlik myint tipinde tanımlanmış olur. Bundan sonra 32-bitlik işlemciler için programı derlemeden önce yukarıdaki satır aşağıdaki hale çevrilmelidir.

```
typedef short int myint;
```

Bu sayede myint olarak bildirilen tüm değişkenler 16-bitlik olmuş olur.

typedef ifadesinin kullanılmasının ikinci nedeni ise koddaki okunabilirliği arttırmaktır. Eğer kodunuz sayma işlemi için pek çok değişken kullanıyorsa typedef ifadesi kullanılarak bütün değişkenlerin aynı tipten bildirilmesini sağlayabilirsiniz.

```
typedef int sayac;
```

Bu sayede sizin yazdığınız kodu okuyan başka bir kişi sayac olarak bildirilen değişkenin program içerisinde sayaç olarak kullanıldığını anlayabilir.

ALIŞTIRMA:

1. unsigned long tipi değişken için UL adında yeni bir tip yaratın. Bir program içerisinde bu tanımı gerçekleştirdikten sonra bu yeni tipe ait bildirimler yapın ve bu değerleri ekrana yazdırın.
2. Aşağıdaki kod parçası geçerli ve doğru mudur?

```
typedef int yukseklik;
typedef yukseklik uzunluk;
typedef uzunluk derinlik;
derinlik d;
```

2.6 Tip dönüşümleri

C dili programcıya farklı veri tiplerini bir ifade içerisinde hep beraber kullanabilme olanağı sunar. Örnek olarak aşağıdaki kod bloğunu inceleyin:

```
char ch='0';
int i=15;
float f=25.6;
double sonuc;

sonuc=ch*i/f;
```

Farklı veri tiplerinin idaresi, derleyiciye aradaki farkları ayırt edebilmesi için çok katı kurallar eşliğinde verilmiştir. Kuralların ilk kısmı tip yükseltmesidir. C derleyicisi bir ifadeyi işletirken char ya da short int tipindeki bir değişkeni otomatik olarak int tipi değişkene çevirir. Tip yükseltmesi sadece ifade işletilirken geçerli olup değişkenin kendisi aslında bellekte daha fazla yer kaplamaz.

Otomatik olarak yapılan tip yükseltmelerine değindiğimize göre artık biliyoruz ki C derleyicisi programı işletirken tüm değişkenleri olası en yüksek değişken tipine yükseltir. Bu işlem belirli bir sistem çerçevesinde gerçekleşmektedir. Aşağıdaki algoritma tip dönüşümlerini göstermektedir:

```
EĞER            ilk ifade long double tipinde ise
BU DURUMDA      ikinci ifade de long double tipine çevrilir
EĞER            ilk ifade double tipinde ise
BU DURUMDA      ikinci ifade de double tipine çevrilir
EĞER            ilk ifade float tipinde ise
BU DURUMDA      ikinci ifade de float tipine çevrilir
EĞER            ilk ifade unsigned long tipinde ise
BU DURUMDA      ikinci ifade de unsigned long tipine çevrilir
EĞER            ilk ifade long tipinde ise
BU DURUMDA      ikinci ifade de long tipine çevrilir
EĞER            ilk ifade unsigned tipinde ise
BU DURUMDA      ikinci ifade de unsigned tipine çevrilir
```

Şu koşullar altında bir önceki örneğe dönerek ne tür yükseltmeler ve dönüşümler yapıldığını görelim. İlk olarak ch değişkeni int tipine yükseltilir. Yapılan ilk işlem ch*i işlemi olduğundan ve halihazırda ch, int tipine yükseltildiğinden yeni bir dönüşüm yapılmaz ve değer hesaplanır. Bir sonraki işlem ch*i çarpımının f değişkenine bölümüdür. Algoritmaya göre eğer bir ifade float tipinde ise diğer ifade de float tipine çevrilmelidir. Bu nedenle ch*i çarpımından elde edilen değer float tipie çevrilir ve elde edilen sonuç f değişkenine bölünür. Son olarak float tipinde elde edilen ch*i/f değeri double tipinde olan sonuc değişkeninde saklanabilmek üzere double tipine dönüştürülür.

Bazı durumlarda derleyicinin yapacağı tip dönüşümlerine güvenmek yerine kendiniz aşağıda belirtilen şekilde tip dönüşümü yapabilirsiniz:

```
(tip) deger
```

Bu işleme tip dönüşümü denir ve değişkende geçici bir değişikliğe yol açar. Bu ifadede type geçerli bir C dili tipi ve deger değişkeni de geçerli bir sabit ya da değişken olmalıdır. Aşağıdaki kod parçası reel bir sayının tamsayı kısmının nasıl yazdırılabileceğini göstermektedir:

```
float f;
f=100.2;
printf("%d \n",(int)f);
```

Yukarıdaki kod parçası işletildiğine ekrana 100 değeri yazılacaktır. Eğer 8-bitlik iki tamsayının çarpımı yine 8-bitlik bir değişkende saklanacak ve bu çarpım değeri long tipinde bir değişkene atanmadan hemen öncesine kadar yine 8-bit olarak saklanacaktır.

```
int a=250, b=10;
long c;
c=a*b;
```

Yukarıdaki işlemin sonucu yanlış bir şekilde 196 olarak hesaplanacaktır. Zira sonucu doğru alabilmek için uzun sayı tipinde bir değişkene ihtiyacımız vardır. Bu yüzden en azından bir değerin işlemden önce long olarak tanımlanması ya da tip dönüşümü yapılması gerekmektedir.

```
c=(long) a*b;
```

Yukarıdaki işlemin sonucu ise bize doğru sonuç olan 2500'ü verecektir çünkü işlem yapılmadan önce daha geniş bir değer aralığına sahip olan long tipine dönüşüm yapılmıştır.

2.7 Değişken saklama sınıfı

C dilinde her değişken ve fonksiyonun tip ve saklama sınıfı olmak üzere iki niteliği vardır. Tip dediğimiz şey daha önce belirttiğimiz char, int vb. yapılardır. Bunun yanında automatic (otomatik), external (harici), static (durağan), ve register (yazmaç) olmak üzere dört adet de saklama sınıfı mevcuttur. Bu sınıfların C dilinde yazımları aşağıdaki gibidir:

```
auto        extern      static      register
```

Auto

Herhangi bir fonksiyon içerisinde bildirilen değişkenler varsayılan nitelik olarak auto alırlar. Bu yüzden

```
{
        char c;
        int a, b, c;
}
```

ifadesi

```
{
        auto char c;
        auto int a, b, c;
}
```

ifadesi ile aynı anlamı taşımaktadır.

Ne zaman bir kod bloğu yazılsa derleyici kod dahilinde belirtilen değişenler için hafıza (RAM)'de belirli yer ayırır. Bu yerel kod bloğu için kullanılan RAM adresleri başka kod blokları tarafından da kullanılabilir.

```
main()
{
        char c=0;
        int a=1, b=3, c=5;
        0007:  CLRF     0E-0Eh yazmacı c değerine ayrılır
        0008:  MOVLW    01-w'ye 1 yükle
        0009:  MOVWF    0F-a değişkenine atanan yazmacı w değeri ile yükle
        000A:  MOVLW    03-w'ye 3 yükle
        000B:  MOVWF    10-b değişkenine atanan yazmacı w değeri ile yükle
        000C:  MOVLW    05-w'ye 5 yükle
        000D:  MOVWF    11-c değişkenine atanan yazmacı w değeri ile yükle
}
```

Extern

extern anahtar kelimesi harici bir bağlantısı söz konusu olan bir değişken ya da fonksiyon tanımlar. Bunun anlamı fonksiyon ya da değişkenin tanımlandığı dosya dışındaki dosyalar için de görülebilir olmasıdır. CCS C dili içerisinde herhangi bir fonksiyonu yoktur.

Static

static değişken sınıfı aksi belirtilmediği müddetçe global olarak erişilebilen ve ilk değer olarak 0 değeri atanmış olan aktif değişkenler yaratır.

```
void test()
{
        char x, y, z;
        static int sayac=4;
        printf("sayac= %d \n",++count);
}
```

Yukarıdaki örnekte sayac değişkeni bir kere tanımlanmakta ve sıfırlanmakta olup fonksiyonun her çağrımında değeri artarak ekrana yazdırılmaktadır.

Register

register değişken sınıfı, sıklıkla kullanılan değişkenlerin yüksek hızda çalışan bellek birimlerinde saklanmasının avantaj getirdiği büyük ölçekli sistemler için geliştirilen yazılımlar için yaratılmıştır. Sadece derleyiciye bilgi amaçlı kullanılabilir. CCS C dilinde etkin bir kullanımı yoktur.

Fonksiyonlar

Fonksiyonlar, C dilin temel yapı bloklarıdır. Tüm ifadeler, fonksiyonların içinde olmalı. Bir fonksiyondan bir tartışmayı nasıl alınacağı nasıl fonksiyonlara geçiş tartışmalar vereceği biz diskler olduğu bu kısımda.
Bu kısımda tartışılan konular:

Fonksiyonlara tartışmalar geçme
Fonksiyonlardan tartışmalar geri dönme
Prototstringleri işleyin
Klasik ve modern fonksiyon tanımları

3.1 Fonksiyonlar

Önceki kısımlarda , biz, ana bir programdan çağırılmış olan fonksiyonların çok örneğini gördük. Örneğin

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

Realitede , bu program, bir hata veya , , bir uyarıyı en azından üretmelidir.Sebep, o fonksiyon **f1()**dir. İfade edilmiş olmalı veya o kullanılan fonksiyon önce, tanımladır. sadece, değişkenlerden oluşur.

Siz, zaten standart bir C fonksiyonu kullanıyorsanızın, fonksiyon hakkında derleyici bilgilendirilen programınızın başında başlık dosyası bulunmalıdır. Eğer siz bir veya da ha fazla fonksiyon kullanıyorsanız bu hatayı düzeltmek için iki yol, vardır. Birisi, sonraki kısımda anlatılan fonksiyon prototstringlerini kullanmaktır. Diğeri, bu gibi programınızı yeniden düzenlemektir:

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

Bir hata, oluşmuştur. çünkü fonksiyon **f1()** olmayacaktır. çünkü **main()**'da çağırılan komuttan önce, tanımlanmalıdır.

3.2 Fonksiyon Özellikleri

Fonksiyonda döndürülen ne tstring değer olduğunu derleyiceye uyarlanmasında kullanılan iki metot vardır.

Genel formu:

type function_name();

Örneğin , ifade **sum()** derleyicisi o fonksiyonu söyleyecektir. **sum()'da** dönüşler bir tam sayı olur. Bir fonksiyonun dönüş değeri hakkında derleyiciyi bilgilendirmek için gereken yol, fonksiyon prototibidir.Bir fonksiyon prototibin değerini değil yalnızca, fonksiyonun dönüş değerini verir , fakat, fonksiyonun, kabul ettiği numara ve tartışmaların tstringini tanımlar. Prototstring, fonksiyon yapısıyla tam uyuşmalıdır.

Prototstringler, bir fonksiyona ve fonksiyon tanımı geçen bilgisayar programcısına argüman arasında herhangi bir talimata aykırı tstring dönmeleri bildirme tarafından programda bug'ları tanımlamak için yardım eder. Eğer fonksiyon tanımında belirttiği gibi bir fonksiyona gönderilen argüman numarası olursa o hatta raporlar Aynı değildir.

Bir fonksiyon prototibi için genel format, burada gösterilir:

type function_name(type var1, type var2, type var3);

Örnek üzerinde her bir değişkenin tstringi farklı olabilir. Bir fonksiyon prototibinin bir örneği, bu programı içine alır. Fonksiyon, uzunluk tarafından , genişlik ve yükseklik tanımlanan hacimi hesaplar.

```
int volume(int s1, int s2, int s3);
void main()
{
        int vol;
        vol = volume(5,7,12);
        printf("volume: %d\n",vol);
}
        int volume(int s1, int s2, int s3)
{
        return s1*s2*s3;
}
```

Dönüşün, bir sabidin yerine bir deyim veya değişkeni kullandığı, farkına varın.

Prototstringlerin önemi, bizim, şimdiye kadar kullandığımız küçük programlar ile anlaşılamaz. Bununla beraber hataların gidermedeki prototstringlerin önemi programların büyüklüğü çizgilerin çok binine birkaç çizgiden büyüdüğü gibi, açıktır.

Alıştırma

1. **1.)** Hatalar fonksiyon hacimine dört parametreyi göndermek için derleyici tarafından , programın üzerinde değişmeyi nasıl yakaladığı göstermek:

vol = volume(5,7,12,15)

2.) Aşağıdaki program doğrumu? Niçin değil?

```
double myfunc(void)

void main()
{
        printf("%f\n",myfunc(10.2));
}
        double myfunc(double num)
{
                return num/2.0;
}
```

3.3 Void

Bir istisna bir fonksiyonun veya dış gecen herhangi bir parametreleri yok değilken, Bu fonksiyon ilân edilen olur:

void nothing (void)

Bunun bir örneği;

```
double pi(void)                 //fonksiyonun tanımlanması
{                                       //atamadan geç
        return 3.1415926536;            //fakat pi ile döndür
}
main()
{
        double pi_val;
        pi_val = pi();                  //pi değerini çağır
        printf("%d\n",pi_val);
}
```

3.4 Fonksiyon Argümanlarının Kullanımı

Bir fonksiyon argümanı fonsiyon çağırıldığında fonksiyona atanan bir değerdir.Fonksiyonlara atanan bazı argümanları sıfırdan izin verir.Bir fonksiyonun argüman sayıları derleyiciiye bağlı kabul edilebilir,fakat bir fonksiyonun ANSI Standartları en az 31 argüman kabul edilebilir.

Bir fonksiyon tanımlandığında, özel değişkenler parametreleri almayı tanımlamalıdır.Bu özel değişkenler biçimsel parametreler gibi tanımlanır.Parametreler fonksiyon isimini takip eden parantezler arasında tanımlanır.Örneğin, fonksiyon çağırıldığında fonksiyona gönderilen iki tamsayının toplamını aşağıda hesaplar ve gösterir.

```
        void sum(int a, int b)
        {
                printf("%d\n",a+b);
        }
```

Bir programda fonksiyonu nasıl çağırabileceğimize bir örnek;

```
void sum(int a, int b); //This is a function prototype
main()
{
        sum(1,10);
        sum(15,6);
        sum(100,25);
}
void sum(int a, int b)
{
        printf("%d\n",a+b);
}
```

sum() fonksiyonu çağırıldığında, derleyici a ve b değişkenşleri içindeki herbir argümanı kopyalar.Bu ,a ve b biçimsel parametre değişkenleri ve argümanlarını **(1,10,15,6,100,25)** çağıran fonksiyona değer atamayı hatırlatmak önemlidir.

Fonksiyonlar iki yolla argüman atayabilirler.Birinci yol " değer ile çağırma " denebilir.Bu metod fonksiyonun biçimsel parametresi içindeki bir argümanın değerini kopyalayabilir.Herhangi bir değişme çağırma rutinin asıl değerine etki etmeyen biçimsel parametreleri oluşturmaz.İkinci yol " referans ile çağırma " denebilir.Bu metod da, argümanın adresi fonksiyonun biçimsel parametresi içine kopyalanır.Bu fonksiyon içinde,biçimsel parametreler çağırma rutinindeki aktif değişkenlere ulaşılmasında kullanılır.Bu değişikliklerin anlamı biçimsel parametre kullanımı ile değişenlere etki edebilir.İşeratçiler bölümünde bu konuyu tartışacağız.Şimdilik, bir fonksiyona argüman atadığımızda sadece değer tarafından çağırılmayı kullanacağız.

Alıştırma:

1. Bir değer argümanını alan ve ekrana bir değer yazdıran bir program yazınız.
2. Bu programdaki yanlışlık nedir?

```
print_it(int num)
{
        printf("%d\n",num);
}

main()
{
```

```
        print_it(156.7);
}
```

3.5 Değerleri Döndüren Fonksiyonların Kullanımı

C'deki herhangi bir fonksiyon çağırma rutinine bir değer döndürtebilir.Tipik olarak,fonksiyon bir eşitliğin sağ tarafına (=) işareti koyulur.dönen değer daha önceki ifadelerde kullanıldığından tekrar gerek duyulmaz,fakat bir **printf()** deyiminde kullanılabilir.Bir değeri döndüren bir fonksiyonu derleyiciye anlatmak için genel format:

```
fonksiyonismi_tipi(biçimsel parametreler)
{
        <deyimler>
        dönen değer;
}
```

type, fonksiyonun dönen değerinin veri tipini özellikle belirttiği yerlerdir.bir fonksiyon bir diziyi ayıran herhengi bir veri tipini döndürebilir.Eğer veri tipi belirtilmemişse, C derleyicisi fonksiyonun bir tamsayı çevirdiğini varsayar. Fonksiyonun bir değer döndürmüyorsa,ANSI C standardı **void'**e dönen fonksiyonu belirtir. Bu açık bir şekilde fonksiyonun bir değer döndürmediğini derleyiciye anlatır.Bu örnek değer döndüren bir fonksiyonun kullanıldığı tipik bir örnektir:

```
#include <math.h>

main()
{
        double result;
        result = sqrt(16.0);
        printf("%f\n",result);
}
```

Bu fonksiyon kayan bir sayıyı döndüren **sqrt()** fonksiyonunu çağırır. Bu sayı **result** değişkenine atandı.Başlık dosyası **math.h**'nin içerdiğine dikkat edin çünkü derleyici tarafından kullanıldığna dair bilgi içermektedir.

Atayabileceği değişken veri tipi ile fonksiyonun döndürdüğü veri tipini uyuşturmak önemlidir.Benzeri fonksiyonlara gönderdiğiniz argümanlar için de geçerlidir.

Öyleyse, Bir fonksiyondan bir değere nasıl dönebilirsiniz? Genel formu:

```
return değişken_ismi;
```

değişken_ismi bir sabit, değişken veya herhangi bir değere dönme gibi veri tipine sahip geçerli C deyimidir. Aşağıdaki örnek fonksiyonların tiplerini göstermektedir:

```
func();
```

```
sum(int a, int b);
main()
{
        int num;
        num = func();
        printf("%d\n", num);
        num = sum(5,127);
        printf("%d\n",num);
}
func()
{
        return 6;
}
sum(int a, int b)
{
        int result;
        result = a + b;
        return result;
}
```

Not alınacak önemli birşey de, bir dönme deyimi ile karşı karşıya gelme,fonksiyon acilen çağırma rutinine döner.Dönme işleminin olmamasından sonra herhengi bir deyim yoktur.Bir fonksiyonun dönüş değeri bir deyimde kullanmaı veya bir değişken atamayı gerektirmez,bununla birlikte,eğer kaybedilen değer değil ise.

Alıştırma :

1. 1 ile 100 arasında bir tamsayı kabul eden ve karesini döndüren bir fonksiyon yazın.
2. Aşağıdaki fonksiyonun yanlışı nedir?

```
main()
{
        double result;
        result = f1();
        printf("%f\n",result);
}
int f1()
{
        return 60;
}
```

3.6 Klasik ve modern fonksiyon tanımları

C 'ın orijinal versiyonu, resmi parametre tanımının farklı bir yöntemini kullandı. Bu klasik form aşağıda gösterilir:

```
type function_name(var1,var2,...varn)
type var1;
type var2;
.
type varn;
{
<statements>
}
```

Tanımın , iki ayrı şekilde tanımlı olduğunun farkına varın. Parametrelerin yalnızca adları, iç parantezi içerilir. Parantez verisi tstringlerinin dışında ve resmi parametre adları, belirtilir.

Modern form , , bizim, önceki örneklerde kullanıyor olduğumuz, formu verir.

```
type function_name(type var 1,...type var n)
```

Fonksiyon tanımının bu tstringinde , her iki veri tstringleri ve resmi parametre adları, parantez arasında belirtilir. ANSI C standart, fonksiyon tanımlarının her ikisinide hesaba katar.

Amaç, daha eski C programlarıyla uyumluluğu sürdürmektir. Harfi harfine var. Eğer siz kodun bir parçasında klasik formu görürseniz, endişelenme; C derleyiciniz, onu ele alan, yapabilen olur. İlerleme yazı kodu iken, modern formu kullanmalısınız.

Alıştırma:
1. Bir fonksiyon prototibi ve, onu kullanıyor olan yararları nedir?
2. Modern forma fonksiyon tanımları için klasik bir formu kullanıyor olan bu programa çevirir.

```
void main(void)
{
printf("area = %d\n", area(10,15));
}
area(1,w)
int 1,w
{
return 1*w;
}
```

3.7 Sabit String Atama

Çünkü PICmicro®MCU'un, ROM erişimde sınırlamaları var , sabit stringleri, olağan tavırda işlemeden gecilen olmaz. CCS C derleyici, standart dışı bir tavırda bu duruma ele alır.

O zaman eğer bir sabit stringi gecilen olsaydının, stringte her karakter için çağırılan yalnızca bir karakter parametresi fonksiyonu izin veren bir fonksiyona. Örneğin:

```
void lcd_putc(char c)
{
.....
}
        lcd_putc("abcd");

        Aynı şekilde;

        lcd_putc("a");
        lcd_putc("b");
        lcd_putc("c");
        lcd_putc("d");
```

4.1 Aritmetik Operatörler

C dili toplama,çıkarma,çarpma,bölme,modül aritetik operatörlerini tayin eder.

+ toplama
- çıkartma
*** çarpma**
/ bölme
% modül
+, -, * ve / operatörleri herhangi bir veri tipiyle kullanılmış olabilir.Modül operatörü sadece integer larda kullanılır.Modül operatörü bölümden elde edilen kalanı verir.
Bu operatörün virgülden sonraki sayılarda bi anlamı yoktur.
Bu operatör iki yol kullanır,birincisi bir çıkartma operatörü olur,ikincisi bir numaranın işretine ters çevirir.
Aşagıdaki örnekler bu iki yolu açıklar.
a = a – b ; çıkartma
a = -a ; a'yı tersleme
Aritmetik operatörler bazı sabitlerin birleşimi ve/veya değişiminde kullanılırlar.
Örneğin;
result = count –163;
C aritmetik operatörlerini kullanıyor olunca, size bazı kısayollar verir.
Önceki örneklerin birisi olan; **a = a – b;** a -=b; şeklinde yazılmış olabilr.Bu metod +,-.* ve / operatörlerinde kullanılır.

a*=b eşiti **a=a*b** dir.
a/=b **a=a/b**

a+=b **a=a+b**
a-=b **a=a-b**
a%=b **a=a%b**
a<<=b **a=a<<b**
a>>=b **a=a>>b**
a&=b **a=a&b**
a|=b **a=a|b**
a^=b **a=a^b**

C kodunu ve assemble kodunu aritmetik işlemleri karşılaştıracak olursak;

int a,b,c,;

a = b + c;

0007: MOVF 0F,W ; **b yi yükle**
0008: ADDWF 10,W ; **c'ye b'yi ekle**
0009: MOVWF 0E ; **a'nın içine kaydet**

0007: MOVF 0F,W ; **b yi yükle**
0008: MOVWF 0E ; **a'nın içine kaydet**
0009: MOVF 10,W ; **c'yi yükle**
000A: SUBWF 0E,F ; **a'dan çıkar**

Problemlerin çözümü için assemblerın anlasılır biçimde olması önemlidir.

a = b;

0007: MOVF 0F,W ; **b'yi yükle**
0008: MOVWF 0E ; **a'nın içine kaydet**

eğer

a==b;

0007: MOVF 0F,W ; **b'yi yükle**
0008: SUBWF 0E,F ; **a'dan çıkar**
0009: BTFSC 03,2 ; **sıfırmı test et**
000A: GOTO 00D ;

4.2 İlişkisel Operatörler

İlişkisel operatörler iki değer arasında karşılaştırma yapan operatörlerdir.bu nedenle bu operatörlere karşılaştırma operatörleride denir.

> büyüktür operatörü
>= büyüktür veya eşittir operatörü
< küçüktür operatörü

<= küçüktür veya eşittir operatörü
== eşit eşittir operatörü
!= eşit değildir operatörü

İlişkisel operatörler hakkında dikkat edilecek şey, bir kıyasın o sonucu
Her zaman 0 mi veya 1 mi dir.
Aşağıdaki örnekler ilişkisel operatörlerle bazı deyimleri gösterir.
var > 15; eğer var 15 ten küçük veya eşitse, sonuç 0 dır.
(yanlış)
var != 15; eğer var 15 ten büyük veya küçükse,sonuç 1 dir.
(doğru)

ÖRNEKLER:

1)Aşagıdaki örnege benzer değişik ilişkisel operatörler yazın.

count != 0
2)Aşagıdaki ifade ne zaman dogru yada yanlış olur? Niçin?

4.3 Mantıksal Operatörler
Mantıksal operatörler, temel mantıksal operasyonlar ve , veya , ve değili destekler.
Tekrar bu operatörler operandlarını dogru(true) ya da yanlış(false) olarak yorumladıktan sonra işleme sokarlar.

		VE	VEYA	DEĞİL
p	**q**	**p&&q**	**p||q**	**!p**
0	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**1**
1	**0**	**0**	**1**	**0**
1	**1**	**1**	**1**	**0**

Mantıksal ve ilişkisel operatörlerin işleme soktukları operandların ve işlem sonunda ürettikleri değerler int türünden **1**(dogru) ya da **0**(yanlış) sayılarıdır.
Aşagıdaki örnekte bu iki operatörün birleşimi grülmektedir:

count>max || ! (max==57) && var>=0

İliskisel ve mantıksal operatörler farklı C bölümünde kullanılmaktadır.

ÖRNEKLER:

1)İlişkisel ve mantıksal operatörün birlikte kullanıldıgı ifadeler yazınız.
count == 0

resutl <= 5

4.4 Kaydırma Operatörleri

C numaralarda bit bit işlem yapan altı özel operatör içerir.Bu kaydırma operatörleri sadece int ve char data tipinde kullanılır.

Bu kaydırma operatörleri:

& Bit VE operatörü
| Bit VEYA operatörü
^ Bit özel veya operatörü
~ Bit değil operatörü
>> Sağa öteleme operatörü
<< Sola öteleme operatörü

Kaydırma operatölerinin genel formatı aşğıdadır:

Değişken << Öteleme sayısı (sola kaydırma)

Değişken >> Öteleme sayısı (sağa kaydırma)

C'de sola ve sağa öteleme yapmak için iki ayrı bit operatörü bulunur.Öteleme işlemleri özellikle bitlerin sayı içerisindeki pozisyonlarını değiştirmek amacıyla kullanılmaktadır.C'deki iki öteleleme operatörü de iki operand alır.Diğer bit işlemlerinde olduğu gibi öteleme işlemlerinde işleme giren operandlar değişmez,işlem sonucunu başka bir değişkene atanması gerekir.

Sola öteleme işleminde bütün bitler öteleme sayısı kadar sola kaydırılır.Kaydırma işlemi sonunda en sağdaki bit sürekli sıfırla beslenir,en soldaki bit ise kaybolur.
Kaydırma operatörü ile ilgili örnekler:

```
      AND                          OR
      00000101 (5)                 00000101 (5)
&     00000110 (6)               | 00000110 (6)
      ----------                   ----------
      00000100 (4)                 00000111 (7)


       XOR                          NOT (ones compliment)
      00000101 (5)                ~ 00000101 (5)
^     00000110 (6)                  ----------
      ----------                    11111010 (250)
      00000011 (3)


      LEFT SHIFT                   RIGHT SHIFT
```

```
      00000101 (5)                        ~  00000101 (5)
      << 2                                   >> 2
      ---------------                        ---------------
=     00010100 (20)                          00000001 (1)
```

```
      a = b | c;
Basla
      0007: MOVF 0F,W ;          b'yi yükle
      0008: IORWF 10,W ;         inclusive or with c
      0009: MOVWF 0E ;           a'nın içine kaydet

      a = b & c;
Basla
      0007: MOVF 0F,W ;          b'yi yükle
      0008: ANDWF 10,W ;         fonksiyonu c ile VE le
      0009: MOVWF 0E ;           a'nın içine kaydet

      a = b >> 3;
Basla
      0007: MOVF 0F,W ;          b'yi yükle
      0008: MOVWF 0E ;           a'nın içine yükle
      0009: RRF 0E,F ;           içeriği ters çevir
      000A: RRF 0E,F ;           sağa
      000B: RRF 0E,F ;           üç kez
      000C: MOVLW 1F ;           içeriği maskele
      000D: ANDWF 0E,F ;         a ile f'i  VE le
```

```
      j = ~a;
Basla
      0007: MOVF 0F,W ;          b!yi yükle
      0008: MOVWF 0E ;           j'nin içine kaydet
      0009: COMF 0E,F ;          j'yi tersle
```

ÖRNEKLER:

1) **MSB bitini tersleyen program yazınız.**

2) **Char tipinde verileri ekranda binary olarak gösteren program yazınız.**

4.5 Arttırma ve Azaltma Operatörleri

Artırma ve azaltma operatörleri C'nin en çok kullanılan operandlarıdır.Tek operand alırlar;önek ya da sonek durumunda bulunabilirler.Artırma operatörü değişkenin içeriğini 1 artırmak için,eksiltme operatörüde 1 azaltmak için kullanılır.

a = a+1; **veya** **a = a-1;**

a++; veya ++a; ikiside birbirinin tamamen aynısıdır.Her ikiside a=a+1;anlamına gelir.

a--; veya --a; ikiside birbirinin tamamen aynısıdır.Her ikiside a=a-1;anlamına gelir.

```
int j, a = 3;
        0007: MOVLW 03
        0008: MOVWF 0F ;          registerı a' tahsis et

j = ++a;
        0009: INCF 0F,F ;         a = 4
        000A: MOVF 0F,W ;         w'nın içine yükle
        000B: MOVWF 0E ;          w'yı  j'nin içine depola

j = a++;
        000C: MOVF 0F,W ;         a değişkenini w'nın içine yükle
        000D: INCF 0F,F ;         a = 5
        000E: MOVWF 0E ;          j = 4
```

NOT:
Bu formatı kullanmayın

a = a++;

```
MOVF 0E,W ;          degişkeni  w'nın içine yükle
INCF 0E,F ;          değişkeni artır
MOVWF 0E;            artırılmış degeri önceki degere geri yükle
```

Aşagadaki örnek iki kullanımıda açıklamaktadır.

```
void main(void)
{
        int i,j;
           i = 10;
        j = i++;
        printf("i = %d, j = %d\n",i,j);
```

```
        i = 10;
        j = ++i;
        printf("i = %d, j = %d\n",i,j);
}
```

Birinci printf() ifadesinde i'ye 11 j'ye 10 yazılır.ikinci printf() ifadesinde i ve j'ye 11 yazılır.

Hepsinin karışımı

```
sum = a+b++          sum = a+b          b = b+1
sum = a+b--          sum = a+b          b = b-1
sum = a+ ++b         b = b+1            sum = a+b
sum = a+ -b          b = b-1            sum = a+b
```

ÖRNEK:

1)

```
void main(void)
{
        int a, b;
        a = 1;
        a = a+1;
        b = a;
        b = b-1;
        printf("a=%d, b=%d\n", a,b);
}
```

2)Programın sonunda a ve b degerleri ne olur?

```
a = 0;
b = 0;
a = ++a + b++;
a++;
b++;
b = -a + ++b;
```

4.6 Operatörlerde Üstünlük

C derleyicisinde hangi operatörün işlem yapma üstünlüğü olduguna bakılır.Örneğin programda a+b*c şeklinde bir ifadeye rastladınız,hangi operatör ilk olacak?

Operatörlerde öncelik	Örnek
1 () ++ --	(a+b)/c parantez
2 & * + - ~ ! ++ --	a=-b artı/eksi/değil/artma/azaltma

3	*** / %**	**a%b çarpma/bölme/mod**
4	**+ -**	**a+b toplama/cıkarma**
5	**<< >>**	**a=b>>c saga ve sola kaydırma**
6	**< > <= >=**	**a>=b büyük/küçük/eşittir**
7	**== !=**	**a= =b**
8	**&**	**a=b&c Bit ve**
9	**^**	**a=b^c Bit özel veya**
10	**|**	**a=b|c Bit veya**
11	**&&**	**a=b&&c LOjik ve**
12	**||**	**a=b||c Lojik veya**
13	**= *= /= %= += -= <<= >>=**	**a+=b atamalar**
$=	**^= |=**	

Bu operatörler bazen işimize yetmeyebilir ama üzülmeyin, sonradan yetecek.

ÖRNEKLER:

1.) Kodun sonunda a ve b degerleri ne olur?

```
int a=0,b=0;
a = 6 8+3b++;
b += -a*2+3*4;
```

BÖLÜM 5

5.1 if Komutu

if komutu koşullu bir komuttur.if dizisini oluşturan if koşulu gerçekleşince sonuç olarak koşulun belirttiği işlem gerçekleşir.Eğer sıfır değilse değer doğru ama sıfırsa değer yanlıştır. Basit formatı:

```
if   (koşul)              NOT: if koşulundan sonra ; kullanılmaz.
     ifade;
```

İfade mantıklı bir C ifadesi olmalıdır. if , ifadeyi değerlendirir eğer doğruysa koşulu gerçekleştirir , doğru değilse program koşulsuz devam eder. if ile ilgili basit bir örnek verelim:

```
if(num>0)
  printf("Numara positif\n");
```

Bu örnek ilişkili oparatörlerin programı kontrol eden koşullarla nasıl kullanıldığını göstermektedir.if koşulu kod dizilerini yürütmek içinde kullanılır.
Genel format:

```
if  (koşul)
{
   .
   ifade;
   .
}
```

Bağlar kod dizinsini kuşatmak için kullanılır.Eğer ifade doğru ise kodu bağların arasında gerçekleştirir. if ve kod dizisine örnek verelim;

```
if  (sayi<0)
{
     sayi= 0;
     printf("sayi dustu\n");
}
or  if (TestMode ==1)
{
  ….. parametreler yazılır
}
```

Diğer if komutu içinde kullanılan karşılaştırıcılar;

```
x == y      x ile y eşit
x != y      x ile y eşit değil
x  > y      x büyüktür y'den
```

x < y **x küçüktür y'den**
x <= y **x küçük eşittir y'ye**
x >= y **x büyük eşittir y'ye**
x && y **lojik AND**
x | | y **lojik OR**

Assemblera değiştirilmiş fonksiyon örneği;

```
int j, a=3;
    0007: MOVLW     03      ; a' yı 3'e yükle
    0008: MOVLF     0F
if (j ==2)
    0009: MOWLW    02       ; w'yı 2'ye yükle
    000A: SUBWF    0E,W     ; j ile test et
    000B: BTFSS    03,2     ; sıfırsa atla
    000C: GOTO     00F
{
    j = a;
   000D: MOVF      0F,W     ;sıfırsa o zaman
   000E: MOVWF     0E       ;j'nin içine a'yı yükle
}
```

ALIŞTIRMA:

1. Bu ifadele sonuçlarından hangisi doğru değerdir?

a. 0
b. 1
c. -1
d. 5*5<25
e. 1==1

2. Öyle bir fonksiyon yazın ki girilen sayının tek mi çift mi olduğunu seçebilsin.Eğer sayı çift ise 0 sonucunu versin ama sayı tek ise 1 sonucunu versin.

5.2 if – else Komutu

Eğer iki dizi komutunuz varsa ve koşula göre birini yürütmek istiyorsanız bunu nasıl yapardınız? İşte cevabı if- else komutudur. Eğer koşul doğru ise ilk ifade gerçekleşir, eğer koşul yanlış ise ikinci ifade gerçekleşir.if komutu genelde else ile birlikte kullanılır.if- else 'nin genel kullanımı;

```
if (koşul)

    ifade1;
```

```
        else

          ifade2;
```

Kod dizilerinde if-else formatı şöyledir;

```
    if (koşul)
 {
    .
    ifade;
    .
 }

else
 {
    .
    ifade;
    .
 }
```

Karışık sistemlerde yani birçok ifadenin bulunduğu sistemlerde if else seçici olarak tek bir ifadeyi gerçekleştirebilir.Aşağıda küçük bir örnek bulunmaktadır;

```
  if (num<0)
     printf("Numara negatif.\n");

  else
     printf("Numara pozitif.\n");
```

Else ilave etmek bize iki yönde karar yolu sağlamaktadır.if- else aynı zamanda birden çok koşullarda da kolaylık sağlamakta ve uygunluk göstermektedir. Bunun genel kullanımı;

```
  if (koşul1)

{
 .
  ifade(s)
 .
}


 else if(koşul2)
{
 .
 ifade(s)
```

```
  .
 }

 else
{
  .
  ifade(s)
  .
}
```

Burada farklı koşullar değerlendirilir ve hangi koşul sağlanıyorsa, o koşulun kod dizisi seçilir.Bunu daha iyi açıklamak için aşağıda basit bir örnek verelim;

```
 if (num ==1)
    printf (" 1 sayısı\n");

 else if (num ==2)
    printf("2 sayısı\n");

 else if  (num==3);
    printf("3 sayısı\n");
 else
    printf("sayı yok\n");
```

NOT: if koşulunda tek çift kıyaslamalara dikkat edilmelidir. Koşul içinde çift kıyaslama kullanılır (&& , == ,||). Eğer bu komutlarda tek kıyaslama kullanılırsa fonksiyon bunu reddedebilir. Bu genel olarak yapılan bir hatadır ve compile ederken anlaşılmayabilir ama operasyon başarısız olur.

ALIŞTIRMA:

1.Aşağıdaki kod parçaları doğru yazılmış mıdır?

```
  if (sayi>20)

     printf ("sayi 20'den buyuktur");
     sayi- ;
 }
```

2.Öyle bir program yazın ki her iki lira da, 5lira da, 10lira da, 20lira da, 50 lira da yazdırsın.Sadece geçerli değişken değerleri 1,5,10,25,50 ve 100.

5.3 ? Oparatörü

? komutu genellikle if – else yapısında kullanılır.Şu şekilde;

(ifade1) ? (ifade2) : (ifade3) ;

ifade? geçerli C dizisidir. İlk olarak ifade1 değerlendirilir ve eğer sonuç doğru ise sonra ifade2 değerlendirilir. Eğer sonuç yanlış ise ifade3 değerlendirilir. Alttaki bununla ilgili örnek bulunmaktadır;

```
int i,j;
i = j;
i ? j=0 : j=1;   or   j=i ? 0:1;
```

i sıfırdan farklı veya 1 olana kadar j=0 koşulu gerçekleşir. Bundan farklı if koşulunda ? oparatörü değere döndürür.Örnek olarak;

i = (i<20) ? i+1 : 10;

20 ve üstü değilse artacağım sonra 10 olarak belirlenecek.

5.4 for Döngüsü

C'de kullanılan üç döngü çeşidinden biri for döngüsüdür. Eğer tekrarlanması gereken komut yada komutlar varsa bunu for döngüsüyle kolayca yapabiliriz. Kullandığımız bu for döngüsü diğer programlama dillerinde kullanılan (BASIC veya Pascal gibi) for döngüsüyle benzerlik göstermektedir.Genel kullanımı şöyledir;

for (kurulum değeri ; test ; artış)

Kurulum değeri sayıcı değişkenin ilk değeridir. Bu sayıcı değişkeni for döngüsünden önce tanımlanmalıdır ve bu kısım for döngüsünde başlangıçta ve sadece bir kere kullanılır.Test bölümünde öncelikli her uygulama değerlendirilir.Bu bölüm daha çok döngü sayıcı değerinin doğru yada yanlış olma durumunu test eder. Eğer test yanlış çıkarsa döngü biter ve normal programa devam edilir.Artış bölümü ise sayıcı değerinin artış durumunu gösterir.

For döngüsüne bir örnek;

```
void main(void)
{
   int i;
   for(i=0; i<10 ; i++)
         printf(" %d " ,i)
       printf("bitti");
}
```

Bu program 0'dan 9'a kadar olan sayıları ekrana yazdırmaktadır. İlk olarak sayıcı değeri olan i'ye sıfır değeri atanmaktadır. Daha sonra bu değer bir artar ve test bölümünde doğru mu yanlış mı olduğuna bakılır yani 10'dan küçük olup olmama durumuna bakılır.Eğer doğruysa bu değer ekrana yazdırılır.Her ekrana yazdırmadan sonra sayıcı değeri bir arttırılır ve bu 10'a kadar devam eder.Sayıcı değeri 10'a geldiği anda döngüden çıkılır ve ekranda bitti yazısı görünür.

Önceki durumda gördük ki her döngü bir şarta bağlıydı ve bu yüzden test yanlış çıkarsa döngü uygulanamamaktaydı. For döngüsünde sadece artış ile sınıflı değiliz.Aşağıda çeşitli tiplerde for döngüleri mevcuttur;

```
for  (sayi=100; sayi>0 ; sayi=sayi -1)
for  (sayi=0 ; sayi<50 ; sayi+=5 )
for  (sayi=1; sayi <10  &&  error == false ; sayi ++)
```

Örneğimiz assemblera çevirince neler olduğuna bakalım;

```
int h,a;
for (h=0 ; h!=10 ; h++ )

   0007: CLRF      0E      ; h'yi temizle
   0008: MOVLW     0A      ; 10'u yükle
   0009: SUBWF     0E,W    ; h'den çıkar
   000A: BTFSC     03,2    ; o için test et
   000B: GOTO      00F     ; eğer i=10'sa döngüden çık
a++ ;
   000C: INCF      0F,F    ; a'yı arttır
   000D: INCF      0E,F    ; h'yi arttır
   000E: GOTO      008     ; tekrar döngü
```

ALIŞTIRMA:

1. Aşağıda belirtilen for döngüleri neler yapar?

```
for(i=1;  ; i++)
for(sayi=1; sayi ; sayi ++)
```

2. Girilen bütün sayıların faktörünü alabilen bir program yazınız.

5.5 while Döngüsü

C'deki bir diğer döngü while döngüsüdür.İfade doğru iken yani koşul sağlandığında while döngüsü komut dizinini tekrarlar.Aşağıda while döngü formatını görmektesiniz;

```
   while (koşul)
      ifade;
```

veya

```
   while(koşul)
  {
    ifade;
  }
```

Burada koşul mantıklı bir C ifadesi olmalıdır.Koşul değeri öncelikle tüm kod dizininde kontrol edilir.Bunun anlamı eğer koşul sağlanmıyorsa koşul doğru olduğunda yapılması gereken kod dizini gerçekleşmemektedir.While döngüsüne bir örnek;

```
  #include <16C74.H>
  #use  RS232  (Baud=9600, xmit-pit_c6, RCV=pin_c7)

  void main(void)
{

   char ch;
  printf("Give me a q\n");
  ch=getch();
  while(ch!='q')
        ch= getch ();
  printf("got a q!\n");
}
```

Yukarda ki kod dizinine baktığımızda ilk ifade de klavyeden harf girilir ve klavyeden girilen bu harf program tarafından okunarak değerlendirilir. Girilen karakter değeri q değilse program klavyeden diğer karakter bekler.Bir kere klavyeden q alındığı zaman printf işleme girer ve program sonlanır.

ALIŞTIRMA:

1. Aşağıdaki while kod dizileri nasıl çalışmaktadır? Açıklayınız.

a.
```
while(i<10)
{
 printf("%d ", i );
 i++;
```

b.
```
while(1)
    printf("%d", i++);
```

2.Öyle bir program yazın ki ch=getch; komutunu kullansın ve klavyeden girilen değerleri ekrana yazdırsın.Dönüş ile karşılaşıldığında program sonlansın.

5.6 do-while Döngüsü

C dilindeki son döngümüz do döngüsüdür. Burada do ve while beraber kullanılıştır. Örnek olarak;

```
   do
{
    ifadeler
}

   while(koşul)
```

Bu durumda koşul doğru olduğu sürece ifadeler gerçekleşecektir. Burada koşul yine makul bir C ifadesi olmalıdır. Bir örnekle do while döngüsünü gösterelim;

```
 #include <16C74.H>
 #use  RS232 (Baud=9600, xmit-pin_c6 , RCV=pin_c7)

   void main(void)
{
  char ch;
  do
  {
    ch=getch( );
  }

  while (ch != 'q');
  printf("Got a q!\n");
  }
```

ALIŞTIRMA:

1. 5.5 bölümündeki 1. alıştırmadaki a ve b şıklarını do-while döngüsünü kullanarak yazınız.
2. Aynı şekilde 2. alıştırmadaki programı do-while döngüsünü kullanarak yazınız.

5.7 Program Kontrol İfadelerinin İç İçe Oluşturulması

İç içe farklı döngüler oluşturulabilmektedir.ANSI C standartları bize 15'e kadar içi içe döngü kullanmamızı sağlamaktadır.Aşağıda bununla ilgili bir örnek görebilirsiniz;

```
    i=0;
```

```
    while(i < 10)
 {
     for(j=0; j<10; j++)
         printf("%d", i*10+j);
     i++;
 }
```

Bu rutin 00'dan 99'a kadar olan sayıları ekrana yazdırır.

ALIŞTIRMA:

1. Öyle bir program yazın ki klavyeden girilen karakterlerin (getch();) ASCII karşılıklarını ekrana yazdırsın.Mesela D harfi girildiğinde D harfinin ASCII karşılığı olan 68 ekranda görünsün.Q harfi girildiğinde ise program sonlansın.

5.8 break Komutu

break komutu döngü içinde herhangi bir noktada döngüden çıkılmasını sağlar.Break komutu ile devam edilen komut dizisi atlanabilir.Döngü break komutu ile karşılaştığı zaman program döngüden sonraki komut dizisine atlar.
Örnek olarak:

```
   void main(void)
 {
     int i;
     for(i=0;i<50 ; i++)
      {
            printf("%d ",i );
            if (i==15)
                 break;
     } }
```

Bu program 0 – 15 arasındaki sayıları ekrana yazdırır.Break komutu bütün C döngülerinde kullanılabilir.

ALIŞTIRMA:

1. Aşağıdaki döngü nasıl çalışır?

```
  for(i=0;1;i++)
{
 printf("Microchip is great!");
 if (getch ( ) =='q')
 break;
}
```

2.Üç tane program yazın ve her biri birer C döngüsü içersin.Sayıcı devam etsin ama anahtara basıldığında döngüden çıkılsın.Anahtar fonksiyonu olarak kbhit() kullanılabilir.khbit() anahtara basıldığında 1, çekildiğinde 0 vermektdir. khbit()
header dosyası olarak conio.h ‘ı kullanmaktadır.

5.9 continue Komutu

Farzedelim ki döngü içinde belirli bir durum meydana geliyor ve döngü bitmeden döngü sonuna atlamak istiyorsun. C bunu continue komutu ile sağlamaktadır. Program bu komutla karşılaşınca, bütün komutları atlayarak döngünün koşulunu kontrol eder. Örnek olarak:

```
#include <16C74.H>
  void main(void)
 {
        int i;
        for (i=0; i<100;i++)
      {
        continue;
        printf("%d ",i);
      }
 }
```

Bu programda printf komutu hiç uygulanamaz. Program printf’ i atlayarak i<100’ü değerlendirir ve aynı zamanda i’yi arttırır.
Continue komutu while ve do-while döngüleri için programı direkt olarak değişkenin test edildiği yani koşul satırına götürür.Continue komutu döngü uygulama bölümünde artışı sağlar ve şartlı koşul sağlanır.

5.10 switch Komutu

if komutu iki alternatif arasında seçim yapmak için iyi bir seçimdi ama alternatif fazla olduğu zaman kullanışsız bir komut olmaktadır. Bu durumu switch komutunu kullanarak çözebiliriz. Switch çok sayıdaki if- else ile eşdeğerdir.
Switch komutunun genel kullanımı şöyledir:

```
 switch ( degisken)
{
     case sabit1:
          ifade (s);
          break;

    case sabit2:
          ifade (s);
          break;
```

```
    case sabitN:
          ifade (s);
          break;
     default:
          ifade(s);
}
```

İfade ardı ardına tamsayı veya karakterlere göre kontrol edilir.Eğer benzer bulunursa komut gövdesi break'a kadar gerçekleştirilir.Eğer benzer bulunamazsa ifade default ile gerçekleştirilir.default isteğe bağlı bir komuttur.
Switch komutuna bir örnek verelim:

```
main()
{
  char ch;
  for( ;; )
  {
     ch=getch();
     if(ch=='x')

 return 0;
switch(ch)
{
   case '0' :
      printf("Pazar\n");
      break;
   case '1' :
      printf("Pazartesi\n");
      break;
   case '2':
      printf("Sali\n");
      break;
   case '3' :
      printf("Carsamba\n");
      break;
   case '4':
      printf("Persembe\n");
      break;
   case '5' :
      printf("Cuma\n");
      break;
   case '6':
      printf("Cumartesi\n");
      break;
   default:
      printf("Gecersiz Giris\n");
```

```
}
}
}
```

Bu örnekte 1'den 7'ye kadar numaralar okunacaktır.Eğer bu numaraların dışında bir numara girilirse Gecersiz Giris ifadesi görülür.Bu değerlerden biri girilirse karşılığı olan gün ekranda görülecektir.

Diğer bir örnekde karakterlere numaralar verilmekte ve bunlar LCD de görünmektedirler.DIP switch ve her yoldaki ayarlar okunur, sonra diğer bitlerden ayrılır daha sonra çağrılan rutine ayrılır.

```
byte cp1_sw_get()                //her yoldaki karakterler
{
byte cp1;
cp1=portd & 0b01110000;       //bilinmeyen değer maskelenir
switch(cp1)                     //şimdi değer çözülür
{
case 0x00: cp1 = 8; break;
case 0x10: cp1 = 16; break;
case 0x20: cp1 = 20; break;
case 0x30: cp1 = 28; break;
default: cp1 = 40; break;
76
}
return(cp1);                      //önceki değere git ve rutini çağır
}
```

ANSI standartlarına göre bu C derleyicisi en az 257 olaya kadar desteklemektedir.İki olay aynı switchlerde aynı değerleri alamaz.switchler iç içe olabilirler.İç ve dış switchler uyuşmazlık göstermezler.ANSI derleyicisi en az 15 bölüme kadar switch komutunu desteklemektedir. Aşağıda iç içe switch komutları görülmektedir.

```
switch (a)
{
case 1:
switch (b)
{
case 0:
printf("b yanlis");
break;
case 1:
```

```
printf("b dogru");
break;
}
break;
case 2:
.
.
```

break komutu switch ifadesinin içinde isteğe bağlı olarak kullanılabilir. Bunun anlamı iki ifadenin aynı kodları paylaşmasıdır.Bunu bir örnekle açıklarsak.

```
void main(void)
{
int a=6,b=3;
char ch;
printf("A = ilave\n");
printf("S = eksiltme\n");
printf("M = carpım\n");
printf("D = bolme\n");
printf("secimi gir:\n");
ch=getch();
switch (ch)
{
case 'S':
b=-b;
case 'A':
printf("\t\t%d",a+b);

break;
case 'M':
printf("\t\t%d",a*b);
break;
case 'D':
printf("\t\t%d",a/b);
break;
default:
printf("\t\tSay ne?");
}
}
```

ALIŞTIRMA:

1. Aşağıdaki kod dizisinde yanlışı bulunuz.

```
float f;
switch(f)
{
```

case 10.05:
.
.

2. switch komutunu kullanarak para değerini yazdırın.Para değeri para değişkenin içinde tutulsun.
3. switch komutunu kullanmanın if – else'e göre avantajları nelerdir.

5.11 null Komutu

null komutu sadece noktalı virgül komutu içermektedir. Gerekli olan yerlerde kullanılır ve null uygulamaya geçince hiçbir şey olmaz sadece bir saykıllık gecikme sağlar.

do, for, if ve while gibi komutlar program gövdesinde görebileceğimiz ve programı uygulayan komutlardır. null komutu bu programlarda rahatlama sağlamaktadır.

```
for(i=0;i<10;i++)
   ;
```

Döngü ifadesinin for [i++]=0 sırası 10 sayı unsurunun 0. sırasıdır. İfadenin gövdesi bir null'dur ta ki ilave komutlar gerekli olana kadar.

5.12 return Komutu

return komutu fonksiyonda uygulamaya son verir ve çağrılan rutine döndürür. Gerekli ise değer çağrılan fonksiyonu döndürebilir. Eğer bir atlandıysa dönülen değer tanımsız olur.Eğer çağrılan fonksiyon dönüş içermiyorsa kontrol hala önceki fonksiyondadır.Eğer dönüş değeri gerekli değilse ,fonksiyon void dönüş tipine bildirir.

```
GetValue(c)
int c;
{
c++;
return c;
}
void GetNothing(c)
int c;
{
c++;
return;
}
main()
{
int x;
x = GetValue();
GetNothing();
}
```

Diziler ve Stringler

Bu bölümde diziler ve stringleri ele alacağız. Bir dizi, aynı veri türündeki değişkenlerle ilgili basit bir listedir.

Bir string boşlukla sonlandırılmış bir karakter dizisi olarak tanımlanır ve ayrıca en çok kullanılan bir boyutlu dizi olarak bilinir.

Bu bölümde tartışılacak konular:

Diziler
Stringler
Bir boyutlu diziler
Çok boyutlu diziler
Oluşturma

6.1 Bir Boyutlu Diziler

Bir dizi benzer tipten bir takım değişkenleri bir grup altında toplamak için kullanılır. Dizideki tekil bir değişken dizi elemanı olarak isimlendirilir. Bu ilişkili veri gruplarını tutmanın basit bir yoludur.

Bir boyutlu dizileri tanımlamanın genel formu:

```
type  var_name [size] ;
```

şeklindedir.

Burada **type** C veri tipindendir, **var_name** dizinin ismi ve **size** de dizinin kaç elemanlı olacağını belirtir. Örnek olarak, eğer 50 elemanlı bir dizi tanımlayacaksak aşağıdaki ifadeyi kullanacağız.

```
int  height[50] ;
```

Bir dizi tanımlandığında, eğer dizi 50 elemanlı ise son eleman 49. indekstedir. Yukarıdaki örneği kullanarak, height dizisinin 25. elemanına 60 değerini vermek istiyoruz diyelim. Aşağıda bunun nasıl yapıldığı görülüyor:

```
height[24] = 60 ;
```

C bir boyutlu dizileri bitişik hafıza bölgelerinde tutar. Birinci eleman en düşük adrestedir. Eğer takip eden kod kısmı çalıştırılırsa…

```
int  num[10] ;
int i ;
```

```
for(i=0 ; i<10 ; i++)
    0007 :   CLRF      18          ; i' yi sil
    0008 :   MOVLW     0A
    0009 :   SUBWF     18 , W      ; 10' dan küçükse test et
    000A :   BTFSC     03 , 0
    000B :   GOTO      013
num[i] = i ;
    000C :  MOVLW      0E          ; num kısmını yüklemeye başlar
    000D :  ADDWF      18 , W
    000E :   MOVWF     04
    000F :   MOVF      18 , W
    0010 :   MOVWF     00
    0011 :   INCF      18 , F
    0012 :   GOTO      008
```

i dizisi hafızada aşağıdaki şekilde görünür.

Eleman	1	2	3	4	5	6	7	8	9	10
	0	1	2	3	4	5	6	7	8	9

Herhangi bir dizi elemanını bir sabit veya değişken olarak her yerde kullanabilirsiniz. Aşağıdaki örnek basit şekilde indeks karesini dizi elemanına atar ve sonra elemanları ekrana yazar.

```
#include <16c74.h>
void main(void)
{
        int num[10] ;
        int i ;
        for(i=0 ; i<10 ; i++)
            num[i] = i * i ;
        for(i=0 ; i<10 ; i++)
            printf("%d ", num[i]) ;
}
```

Eğer elinizde 10 elemanlı bir dizi varsa ve kazara 11. elemana kadar yazarsanız ne olur? C' nin dizi indeksleri için sınır kontrolleri yoktur. Bu yüzden dizide belirtmeden bir elemana okuma, yazma yapabilirsiniz. Bununla birlikte, bu genelde feci sonuçlar doğuracaktır. Çoğu kez programın düzgün çalışmamasına ve bazen de bilgisayarın düzgün çalışmamasına sebep olacaktır.

C aşağıda görülen şekilde olduğu gibi bir dizi değerini başka bir diziye atamaya izin vermez:

```
int a[10] , b[10] ;
.
.
a=b ;
```

Yukarıdaki örnek doğru değildir. Eğer bir dizinin içeriğini diğer bir diziye kopyalamak istiyorsanız, bunu birinci dizideki elemanları teker teker ikinci diziye kopyalayarak yapmalısınız. Aşağıdaki örnekte her bir dizinin 20 elemanlı olduğu farz edilip, a[ ] dizisinin içeriğinin b[ ] dizisine nasıl kopyalanacağı gösteriliyor.

```
for(i=0 ; i<20 ; i++)
b[i] = a[i] ;
```

UYGULAMA:

1. Belirtilen kod parçasında yanlış olan nedir?

```
int i ;
char  count[10] ;
for(i=0 ; i<100 ; i++)
        count[i] = getch( ) ;
```

2. **getch( )** kullanarak klavyeden girilen ilk on karakteri okuyan programı yazınız. Program karakterlerin eşleşip eşleşmediğini rapor olarak verecek.

6.2 Stringler

En çok rastlanan bir boyutlu dizi stringtir. C string veri tipinde bir yapıya sahip değildir. Onun yerine stringlerin bir boyutlu dizi şeklinde kullanımını destekler. Bir string bir sıfır olarak tanımlanmıştır.Bütün string türleri boşlukla sonlandığı için fazladan bir tane boşluk elemanı eklemek gerekir. Bu son eleman boşluğu tutacaktır. Tüm string sabitleri otomatik olarak C derleyicisi tarafından boşlukla sonlandırılır.

Klavye kullanarak programınızın içerisine bir stringi nasıl dahil edersiniz?
gets (str) fonksiyonu satır başı ile karşılaşana kadar klavyeden karakter okuyacaktır. Okuduğu string karakterlerini str ile belirtilen dizide saklayacak. Str dizisinin uzunluğunu, klavyeden girilen karakter sayısını ve boşluk elemanını da dikkate alarak, bu sayıya eşit veya daha büyük olacak şekilde tanımlamayı unutmamalısınız.

gets( ) fonksiyonunun nasıl kullanıldığını bir örnekle açıklamaya çalışalım.

```
void  main(void)
{
        char star[80] ;
        int i ;
        printf("Bir string giriniz (80 karakterden az) : \n") ;
        gets(str) ;
        for(i=0 ; str[i] ; i++)
            printf("%c" , str[i]) ;
        printf("\n%s" , str) ;
}
```

Burada görüyoruz ki string iki şekilde ekrana yazdırılabilir**: %c** yi kullanarak bir dizinin karakterleri olarak veya **%s** i kullanarak bir string şeklinde.

UYGULAMA:

1. Bu programda yanlış olan nedir? (**strcpy** fonksiyonu ikinci argümanı birinci argümanın içine kopyalar.)

```
#include  <string.h>
void  main(void)
{
        char str[10] ;
        strcpy(str,  "Motorola who?") ;
        printf(str) ;
}
```

2. Öyle bir program yazın ki ekrandan string karakterlerini okusun ve onları ters çevrilmiş düzende ekrana yazsın.

6.3 Çok Boyutlu Diziler

C bir boyutlu dizilerle sınırlı değildir. C' de iki veya daha çok boyutlu dizi tanımlanabilir. Örneğin 5x5 elemanlı **number** isimli bir tamsayı dizisini tanımlayıp kullanacaksanız:

```
int number[5][5] ;        ; 25 Ram bölgesi kullanır.
```

şeklinde bir tanımlama yapmanız gerekir.

Küme parantezleri setiyle ek boyutlar kolayca eklenebilir.Biz basit olması açısından iki boyutlu dizileri ele alacağız. Bir iki boyutlu dizinin en iyi gösterim şekli satır/sütun formatıdır. Bundan

dolayı iki boyutlu diziler soldan sağa doğru okunur. Aşağıdaki şekilde 5x5'lik bir dizinin grafik gösterimi veriliyor.
İki boyutlu dizileri bir boyutlu dizilere benzer şekilde kullanabiliriz. Örnek olarak, bir sonraki programda indeks çarpımları 5x4'lük diziye yüklenir ve daha sonra dizi içeriği satır ve sütun formatında ekranda gösterilir.

```
        void  main(void)
{
                int  array[5][4] ;
                int  i , j ;

                for(i=0 ; i<5 ; i++)
                    for(j=0 ; j<5 ; i++)
                        array[i][3] = i * j ;

        for(i=0 ; i<4 ; i++)
        {
                  for(j=0 ; j<4 ; j++)
                        printf("%d  ",array[i][j]) ;
                  printf("\n") ;
        }
}
```

Bu programın çıktısı aşağıdaki gibi olacaktır:

```
0       0       0       0
0       1       2       3
0       2       4       6
0       3       6       9
0       4       8       12
```

Gördüğünüz gibi çok boyutlu diziler kullanılırken her bir elemana erişmek için gerekli değişken sayısı artıyor. PICmicro MCU'nun hafıza haritasına bağlı olarak 100 veya 10 x10' luk diziler oluşturmak mümkün değildir. Buna rağmen 2 tane 50 elemanlı dizi oluşturulabilir.

UYGULAMA:

1. Öyle bir program yazın ki 3x3x3 bir diziye 1' den 27' ye kadar olan sayıları yüklesin ve sonra bu diziyi ekrana yazsın.

2. Birinci örnekteki programı kullanarak her bir satır ve sütunun toplamını yazdırın.

6.4 Dizileri Oluşturma

Yani gördünüz ki sadece tekil dizi elemanları değerleri tutabilir. C içinde bir metot vardır. Bu metot bir diziye bir değişkende olduğu gibi ilk değerini verir. Bir boyutlu dizilerin genel formu gösterilecek olursa:

type array_name[size] = { value_list } ;

value_list virgülle ayrılan sabitler listesidir. Bunlar dizi tipiyle uyumludur. Burada birinci sabit birinci elemanın içinde, ikinci sabit ikinci elemanın içinde olacak ve diğer sabitler de bu şekilde diğer elemanlara yerleşecek. Aşağıdaki örnekte 5 elemanlı tamsayı dizisi tanımı görülüyor.

int i[5] = {1 , 2 , 3 , 4 , 5 } ;

i[0] elemanı 1 **ve i[4]** elemanı 5 değerine sahip olacak.
Bir string iki metotla tanımlanabilir. İlki, her biri tekil karakterlerden oluşan bir liste şeklindedir. Bu tanıma uygun aşağıdaki örnek verilebilir:

char str[3] (‘ a ’ , ‘ b ’ , ‘ c ’) ;

İkinci metot aşağıda görüldüğü gibi bir string aktarma şeklindedir:

char name [5] = “ John ” ;

Stringte küme parantezi olmadığı göz önünde bulundurulmalıdır, stringler bu oluşturulan tiplerin içinde kullanılmadı, çünkü stringler C’ de bir boşlukla sonlanmalıdır. Derleyici otomatik olarak John’ un sonuna bir boşluk ekler.

Çok boyutlu diziler bir boyutlu dizilerle aynı yöntemlerle tanımlanır.
Örnekte 3x3 lük bir dizi tanımı görülüyor.

```
int num[3][3] = {  1,2,3,
                   4,5,6,
                   7,8,9 } ;
```

UYGULAMA:

1. Aşağıdaki belirtim doğru mudur?

   int count [3] = 10, 0 , 5.6 , 15.7 ;

2. Bir sayının karesi ve küpü için bir çevrim tablosuna sahip programı yaz. Her bir satırda sayının kendisi, karesi ve küpü bulunacak. 1’den 9’a kadar olan sayıların bilgisini tutmak

için 9x3'lük bir dizi oluşturun. **scanf("%d", &num) ;** ifadesi kullanıcıdan bir sayı girmesini bekler. Sonra o sayıyı, karesini ve küpünü ekrana yazar.

6.5 String Dizileri

String dizilere C' de çok rastlanır. Onlar diğer diziler gibi belirtilir ve tanımlanır. String dizilerinin kullanım şekli diğer dizilerden biraz farklıdır. Mesela aşağıdaki ifade neyi tanımlar?

char name[10][40] ;

Bu ifade **name** dizisinin 40 karakterlik 10 isim tuttuğunu belirtir. Bu tablodan bir stringe ulaşmak için, sadece ilk indeks kullanılır. Örnek olarak, bu dizideki 5. ismi yazdırmak için aşağıdaki ifade kullanılır.

printf ("%s" , names[4]) ;

İki boyutludan daha büyük boyutlu diziler için gösterim aynıdır. Buna örnek olarak **animal** dizisi verilirse:

char animals [5][4][80] ;

Belirli bir stringe ulaşmak için ilk iki boyutu kullanabilirsiniz. Örnek olarak **animal[2][1]** ifadesi ile 3. listedeki ikinci stringe erişilir.

UYGULAMA:

1. Öyle bir program yazın ki 0' dan 9'a kadar olan rakamlarda harfleri tutacak şekilde bir string tablosu oluştursun. Kullanıcı bir rakam girdiğinde program da o rakama karşılık gelen harfi göstersin.

6.6 String Fonksiyonları

Stringler programın içerisinde birkaç yöntemle kullanılabilir. **Strcpy** komutu kullanılarak bir karakter bir kaynaktan bir hedefe kopyalanabilir. Bunun sayesinde sabit bir string RAM' e yerleştirilebilir.

```
#include <string.h>                 //string kütüphanesi
functions
char string[10] ;                   //string dizisi tanımlandı
strcpy  (string,  "Hi There") ;     // stringe karakterleri yerleştirir.
```

PICmicro MCU'da işaretçiler ROM' u göstermez bu yüzden string fonksiyonlarının içine sabit bir string girilemez. Mesela **strlen("hi")** ifadesi doğru değildir.

Diğer örnekler:

```
char s1[10] , s2[10] ;

strcpy(s1,"abc") ;
strcpy(s2,"def") ;
strcat(s1,s2) ;
printf("%u", strlen(s1)) ;     //ekranda 6 görünecek
printf(s1) ;                   //abcdef yazacak

if(strcmp(s1,s2)!=0)
    printf("Eşleşme yok") ;
```

Kullanılabilir diğer string fonksiyonları aşağıdadır:

strcat İki stringi birleştirir.
strchr Bir karakterin stringte ilk kullanıldığı yere bakar.
strrchr Bir karekterin stringte en son geçtiği yeri bulur.
strcmp İki stringi karşılaştırır.
strcmp Karakterlerin küçük büyük olma durumunu göz önünde bulundurmadan iki stringi karşılaştırır.
strncpy Bir stringten diğerine belirli bir sayıda karakter kopyalar.
strlen Bir stringin uzunluğunu hesaplar.
strlwr . Büyük harfleri küçük harflere çevirir.
strpbrk İki stringin ilk eş karakterinin yerini bulur.
strstr Bir string parçasının stringin içinde ilk geçtiği yeri bulur.

Tanımlayacağınız string dizilerinin boyutunun, kullanacağınız stringlerin boyutuna eşit veya daha büyük olduğundan emin olunuz.

7-Pointerlar

Bu bölüm C'nin çok zahmetli ve çok önemli bir özelliğini olan pointerları kapsamaktadır.Bir Pointer en temelde bir nesnenin adresidir.Bu bölümde göreceğimiz bazı başlıklar:

Pointer temelleri
Pointerlar ve Düzenleyiciler
Fonksiyonlarda Geçen Pointerlar

7.1 Pointerlarla Tanışma

Bir pointer,bir hafıza yerinin adresini tutan bir hafıza yeridir.Örneğin,eğer bir pointerın değeri **a** ve bu **a**, **b**'nin tutulduğu yerin adresi ise, **a b**'yi işaret ediyordur.Yani **b** değerinin hafızada tutulduğu yerin adresi 100 ise, **a** değeri 100 demektir.
Bir pointer değişkeni genel olarak şöyle tanımlanır:

```
type *var_name;
```

type,C dilinde bir pointer atama komutudur.Bu komut, **var_name** ile belirtilen değişkenin tipini tayin eder.**var_name**'den önce gelen bir * işareti vardır.Bu işaret program derlenirken var_name değişkenini bir pointer olarak belirler.Bir pointer tamsayı değişkeni olarakta atanabilir.

```
int *ptr;
```

Pointerlerda * ve **&** olmak üzere iki özel operatör bulunur.Bir değişkenin bulunduğu yerin adresi **&** operatörü ile atanır. * operatörü ise değeri tekrar geri çağırmak istediğimizde kullanılır.Örneğin;

```
#include <16c74.h>
void main(void)
{
int *a,b;
b=6;
a=&b;
printf("%d",*a);
}
```

NOT:

Bir değişiklik yapılmadığı takdirde derleyici pointerlara 1 byte'lık yer ayırır.Bunun anlamı pointer 0-255 arası değer alabilir.Eğer geniş hafıza olarak atanırsa,pointera 2 byte(16-bit)'lik yer ayrılır.16 bit pointer seçmek için komut satırına,

```
#device *=16
```

yazılmalıdır.

ROM kodları genelde 16-bit ile haberleşir.

İlk ifade iki değer ile belirlenir.**a** bir tamsayı pointer, **b**'de bir tamsayıdır.Sonraki ifade de **b**'ye 6 değeri atanmıştır.Sonra **b**'nin adresi(**&b**) pointer değişkeni olan **a**'ya atanmıştır.Bu satır **b**'nin tutulduğu adresi **a**'ya okur.Son olarak **b**'nin değeri * operatörü kullanılarak **a** pointerından çağrılıp,ekrana yazdırılmıştır.Bu satır, **a**'nın işaret ettiği adresteki değeri ekrana yazar.Bu metotta, dolaylı yoldan pointerın tuttuğu adresten değer çağrılmıştır.Bir grafik örneği aşağıda görülmektedir.

Address: 100 102 104 106
Variable: i j k ptr
Content: 3 5 -1 102
int i, j, k;
int *ptr;

İlk olarak, **i** 3 ve **&i** 100(**i**'nin yeri)'dür.**ptr** 102 değerini içerir, ***ptr** ise 5(102 adresinin içindeki değer)'dir.

Bu şekilde bir pointer kullanarak, bir hafıza yerinin değerini atamak mümkündür. Örnek olarak, önceden yapılan programı bu yöntemle yapalım.

```
#include <16c74.h>
void main(void)
{
int *a,b;
a = &b;
*a=6;
printf("%d",b);
}
```

Bu programda, ilk olarak **b** değerinin adresini **a**'ya atadık,sonra **a**'yı kullanarak **b**'ye 6 değerini atadık.Bu satır (***a=6**), **a**'nın işaret ettiği hafıza yerine 6 değerini okudu. Önceki iki örnekte pointer kullanmak zorunlu değildir.Fakat pointerlar kullanılarak tanımlanmıştır.

EGZERSİZ:

1) 0'dan 9'a kadar **for** döngüsüyle bir program yazınız ve numaraları ekrana yazdırın.Bunu yaparken pointerları kullanınız.

7.2 Pointerlarda Kısıtlamalar

Genelde pointerlar diğer değişkenler gibi davranır.Bununla beraber birkaç kural ve istisnayı unutmamak gerekir. * ve **&** operatörlerine ilave olarak, dört adet operatör daha pointerlarda kullanılabilir.**+,++,-,--** .Sadece tamsayı niceliklerde, pointerlar için toplama ve çıkarma yapılabilir.

Bir pointerın değeri 1 arttırılırsa, bu pointer bir sonraki adres gözünü işaret eder. Eğer **p** pointerına 100 adresi atanır, sonra **p++** ifadesi uygulanırsa, tamsayılar 2 byte uzunluğunda olduğundan **p**'nin yeni değeri 102 olur.Eğer **p** bir **float** pointer ise 4 byte uzunluğunda

olduğundan 1 arttırıldığında 104 değerini alır.Pointerlar sadece aritmetik olarak belirlenebilir.Çünkü **char** tipi 1 byte'lıktır.

Pointerlar tamsayılarla çıkartılabilir yada toplanabilir.Aşağıda bir örnek gösterilmiştir.

```
int *p;
.
p = p+200;
```

Burada hedef, **p**'nin önceden işaret ettiği yerin 200 adres sonrasını işaret etmesini sağlamaktır.

Pointerlarda kendi aralarında toplama ve çıkarma işlemine tabi tutulabilir.Yalnız nesnenin yerini işaret eden pointerlarla çıkarma veya toplama yaparken çok dikkat edilmelidir.**ptr**'nin önceki değeri 1 ise, aşağıdaki ifade ile ilgili ne düşünüyorsunuz?

```
*p++;
```

Bu ifade ile varılmak istenen nokta **p**'nin işaret ettiği yeri 1 arttırmaktır.Bunu yapmak için aşağıdaki ifadeyi kullanın.

```
(*p)++;
```

Buradaki parantezin amacı bir arttırmadan önce **p**'nin bir pointer olduğunun belirtilmesi içindir. * operatörünün ++ operatöründen önce yapılması daha uygun olacaktır.Pointerlar ilişkisel operasyonda da kullanılabilirler.Bununla beraber, pointerlar kendi aralarında haberleşirken sadece birbirleriyle anlaşırlar, aynı nesneye yönelirler.

Pointerlar ROM'lara atanamazlar.Örneğin aşağıdaki kod parçası yanlıştır.

```
char const name[5] = "JOHN";
.
ptr=&name[0];
```

Bu değer **const** komutuyla ROM'a yerleştirilmiştir.

EGZERSİZ:

1) Aşağıdaki komutları yazıp atamaları yapın.**%p**'yi kullanıp her değişkenin değerini yazdırın.Tüm pointerları 1 arttırıp, tekrar yazdırın.Makinenizde ki her bir büyüklüğün değeri nedir?

```
char *cp,ch;
int *ip,i;
float *fp,f;
double *dp,d;
```

2) Aşağıdaki kod parçasında yanlış olan ne?

```
int *p,i;
p = &i;
```

```
p = p/2;
```

7.3 Pointerlar ve Düzenleyiciler

C'de pointerlar ve düzenleyiciler yakından alakalıdır ve birbirlerinin içinde kullanılabilirler.Bu ikisi arasındaki ilişki C'nin bir gücüdür.

Bir indeks olmaksızın bir düzenleyici ismi kullanıyorsanız, aslında düzenleyicinin başında bir pointer kullanıyorsunuzdur.Son bölümde, sadece geçtiğimiz dizinin ismi olan **gets()** fonksiyonunu kullandık.Aslında fonksiyonda dizinin ilk elemanı bir pointer olarak geçti.Önemli not : bir düzenleyici bir fonksiyonda geçtiği zaman, sadece ilk elaman pointera geçer.Bunlar sabit düzenleyici veya yapıyla birlikte yaratılamaz.

İndekssiz bir düzenleyici bir pointerdır.Başka bir pointera o değeri atayabilirsiniz.Pointerı aritmetik kullanan düzenleyici, erişim için size izin verir.Örneğin;

```
int a[5]={1,2,3,4,5};
void main(void)
{
int *p,i;
p=a;
for(i=0;i<5;i++)
printf("%d",*(p+i));
}
```

Bu kusursuz bir C programıdır.**printf()** ifadesinde ***(p+i)** kullandığımızı fark etmişsinizdir.İndeksin bir pointer gibi oluşuna şaşırabilirsiniz.Ayrıca bir pointerıda indeksleyebilirsiniz.Aşağıdaki program doğru bir programdır.

```
int a[5]={1,2,3,4,5};
void main(void)
{
int *p,i;
p=a;
for(i=0;i<5;i++)
printf("%d",p[i]);
}
```

Düzenleyiciyle işaret edildiğinde indekslenmiş bir pointerda bir şeyi hatırlamak gerekir.Düzenleyici ile işaretlenmiş pointerlar sadece ilk elamanı yada temel diziyi gösterir.Böyle bir durumda pointerlarla toplama yapmak yanlış olur.Bu yüzden **p++** ifadesi yukarıdaki program için yanlış bir kullanım olurdu.

Pointerlar ve düzenleyiciler karıştırılırsa tahmin edilmeyecek sonuçlar doğurabilir. Aşağıda bir örnek görülmektedir.Bu ikinci versiyon düzenleyicilerle pointerları karıştırmaz.

```
int *p;
int array[8];
```

```
p=array;
0007: MOVLW 0F ;düzenleyici başlangıcını yükle
0008: MOVWF 0E ;pointer
*p=3;
0009: MOVF 0E,W ;
000A: MOVWF 04 ;dolaylı hafızayı işaretle
000B: MOVLW 03 ;3'ü yükle
000C: MOVWF 00 ;ve işaretlenen yeri sakla
array[1]=4;
000D: MOVLW 04 ;4'ü yükle
000E: MOVWF 10 ;düzenleyicinin ilk komutu
int *p;
int array[8];
p=array;
0007: MOVLW 0F ;düzenleyici başlangıcını yükle
0008: MOVWF 0E ;pointer
p[1]=3;
0009: MOVLW 01 ;düzenleyici pozisyonunu yükle
000A: ADDWF 0E,W ;düzenleyici başlangıç pozisyonuna ekle
000B: MOVWF 04 ;düzenleyici pointerı yükle
000C: MOVLW 03 ;3'ü yükle
000D: MOVWF 00 ;konumu kaydet
*(array+1) = 4;
000E: MOVLW 10 ;düzenleyici pozisyonunu yükle
000F: MOVWF 04 ;işaretle
0010: MOVLW 04 ;4'ü yükle
0011: MOVWF 00 ;konumu kaydet
```

EGZERSİZ:

1) Kod parçası doğru mu?

```
int count[10];
.
count = count+2;
```

2. Kod parçası neyi gösterir?

```
int value[5]=(5,10,15,20,25);
int *p;
p = value;
printf("%d",*p+3);
```

7.4 Fonksiyonlara Geçen Pointerlar

Üçüncü bölümde, fonksiyonlara geçilebilen iki yoldan bahsettik, değerle çağıma ve referans ile çağırma.İkinci yöntem fonksiyona adresi geçer yada diğer kelimelerde bir pointer fonksiyona geçilir.Pointer kullanan değişkene bu noktada değişimler yapılır.

Diğer değişkenler gibi pointerlarda fonksiyonlara geçebilir.Aşağıdaki örnekte pointer kullanılan bir fonksiyonda bir dizi nasıl geçer gösterilmiştir.

```
#include <16c74.h>
void puts(char *p);
void main(void)
{
puts("Microchip is great!");
}
void puts(char *p)
{
while(*p)
{
printf("%c",*p);
p++;
}
printf("\n");
}
```

Bu örnekte, pointer **p**,dizinin ilk elamanı olan "M" 'yi işaret etmektedir.**while(*p)** ifadesi dizinin sonu olan null(**0**) ifadesini kontrol eder.Her döngüde, **p**'nin gösterdiği karakteri yazar.Dizide sonraki karaktere geçmek için **p**, 1 arttırılır.

Diğer bir örnekte aşağıdaki gibidir.

```
void IncBy10(int *n)
{
*n += 10;
}
void main(void)
{
int i=0;
IncBy10(i);
}
```

Okunabilirliği arttırmak için pointer çağrılıp tekrar yazılabilir.Örnek:

```
void Incby10(int & n)
{
n += 10;
}
void main(void)
{
int i=0;
Incby10(i);
}
```

EGZERSİZ:

1)Bir program yazın.İçinde **float** değişken geçen bir fonksiyon olsun.Fonksiyonda -1 değeri fonksiyon parametrelerini belirlesin.Sonra fonksiyondan **main()**'e dönün ve **float** tipi değişkenin değerini yazdırın.

2. Bir program yazın.İçinde **float** pointer geçen bir fonksiyon olsun.Fonksiyonda -1 değeri değişkenleri belirlesin.Sonra fonksiyondan **main()**'e dönün ve **float** tipi değişkenin değerini yazdırın.

Yapılar ve Birleşimler

Yapılar ve birleşimler C'nin en önemli iki tanımlama biçimidir.Yapılar değişik data tiplerine sahip olan değişken gruplarıdır.

Bu ünitede :

Yapı Temelleri
Yapıların İşaretçileri
Seri Yapılar
Birleşim Temelleri
Birleşim İşaretçileri

8.1 Yapılara Giriş

Bir yapı(struct) ortak bir isimle ulaşılabilen nesneler grubudur. Her bir nesne kendi data tiplerine sahip olup birbirlerinden farklı olabilirler.
C yapıları aşağıdaki yolla tanımlar:

```
struct tag-name
{
    type element1;
    type element2;
        .
    type elementn;
} variable-list;
```

struct anahtar kelimesi ile compilera bir yapı tanımlanacağı belirtilir.Yapı boyunca her type bir geçerli data tipidir. Bu tipler aynı olmak zorunda değildir. **tag_name** yapının ismidir. **Veriable_list** , **struct tag_name** tipindeki dataların sahip olduğu değişkenleri ifade eder. **Veriable_list** tercihlidir. Yapıların içindeki nesnelerin her biri alanlar veya üyeler olarak gösterir. Biz bunlara üyeler diyeceğiz

Genelde, yapıdaki bilgi mantıksal olarak ilişkilidir.
Örneğin;siz müşterilerinizin isim,adres ve telefon numaralarını saklamak için bir yapı kullanabilirsiniz. Takip eden örnek kütüphane bir kart kataloğu içindir.

```
struct catalog
{
    char author[40];
    char title[40];
    char pub[40];
    unsigned int data;
    unsigned char rev;
} card;
```

Bu örnekte yapını ismi **catalog'dur. Catalog** bir değişkenin ismi değil yapının ismidir. **Card** değişkeni **catalog** tipi yapının bir yapısı olarak ifade edilir.

Yapının herhangi bir üyesine ulaşmak için hem değişkenin ismi hemde üyenin ismini tanımlamanız gerekir. Bu isimler bir periyotla birbirinden ayrılır.
Örneğin; **catalog** yapısının bir üyesine ulaşmak için **card.rev='a'** kullanır. Burada **card** değişkenin ismi rev ise üyenin ismidir. Yapının elemanlarına ulaşmada operatör(işlemci) kullanılır. **Catalog** yapısının bir üyesini yazdırmak için :

```
printf("Author is %s\n",card.author);
```

Şimdi biz bir yapının nasıl tanımlanacağını ifade edeceğiz ve nasıl ulaşılabileceğini ve bir yapının hafızada nasıl görüleceğini biliyoruz.

author 40 byte
title 40 byte
pub 40 byte
date 2 byte
rev 1 byte

card yapısının bir üyesinin adresini öğrenmek istediğimizde **&card.date** kullanılır. Eğer yazanın (sunucunun) ismini yazdırmak istersen **printf("%s",card.pub) .** Başlıktaki özel elemana ulaşmak için döngüdeki 3. eleman gibi

card.title[2];

Başlıktaki birinci eleman 0 , ikinci eleman 1 ve üçüncü eleman 2 'dir.
Yapıyı bir kez tanımlandığında program her yerinde değişik yapı değişkenleri kullanılarak yaratılır.

struct tag-name var-list;

Örneğin, programda yapı kataloğu daha önceden tanımlanmış ise siz iki değişken daha aşağıdaki şekilde tanımlayabiliriz.

struct catalog book,list;

C'de yapıların dizileri (array) değişik data tiplerinde gösterilebilir.Bu örnek 50 elemanlı bir yapı sırasını gösteren **catalog** örneği.

struct catalog big[50];

Eğer özel bir yapıya sırasıyla ulaşmak isterseniz yapı değişkenini indexlememiz (işaretlememiz) gerekir **big[9].title.** Yapının big sırasına ulaşmak için hangi isimde üyeyi kullanılarak ulaşılır ?

big[9].title

Yapı fonksiyonlarada geçiş yapan bir fonksiyon değişik data tiplerindeki gibi return(dönüş) yapabiliriz. Bir yapıya değişik değerler işaretçi yardımıyla işaret edilir.
Takip eden yapı geçerlidir.

```
struct temp
{
    int a;
    float b;
    char c;
} var1,var2;
var1.a=37;
var2.b=53.65;
```

```
var2 = var1;
```

Bu yapı çalıştığında **var2** değişkeni **var1** değişkeni ile aynı içeriğe sahip olur.
Bu bir yapıyı başlatmanın bir örneğidir.

```
struct example
{
    char who[50];
    char ch;
    int i;
} var1[2]={"Rodger", 'Y',27,"Jack",'N',30};
```

Bir önemli not ; yapıdan fonksiyona geçerken bütün yapı ''değer çağırma''(call by value) metodu ile geçilir.Bunla beraber fonksiyondaki yapıların değişimi yapının çağırma rutuninin değerlerini etkilemez.Yapıdaki elemanlar fonksiyona geçme yöntemini etkilemezler.
PICmicro®'da kullanılarak kurulan LCD arayüzü şöyle olabilir örneğin;

```
struct cont_pins
{
   boolean en1; //bütün göstergeler için yetkilendir
    boolean en2; // 40x4 hatlı göstergeler için yetkilendir
    boolean rs; //register(kaydedici)  seç
    int data:4;
} cont;
#byte cont = 8; // d  portunu  control  et
```

Bu **cont_pins** yapısını kurar ve program yolu ile elde edilir.
NOT:
:4 o değer için 4 bit kullanılacağını gösterir.Bu durumda **D0 en1** ve **D3-6** data olur.

```
void LcdSendNibble(byte n)
{
cont.data=n; //present data
delay_cycles(1); //gecikme
cont.en1=1; //en1'i  1 yap
delay_us(2); // gecikme
cont.en1=0; //en1'i  0 yap
}
```

ÖRNEK:

1. Yapısı bir karakterli sırası (string) 40 karakterli bir program yazın.Bir karakteri klavyeden okusun ve **getch()** komutunu kullanarak kaydetsin. Bir **string**'i ve **getch()** komutu ile kaydet sonra değerleri yazdır.
2. Bu kodun hangi evresi yanlıştır.

```
struct type
```

```
    {
        int i;
        long l;
        char str[50];
    } s;
    .
    .
i = 10;
```

8.2 Yapı İşaretçileri

Bazen bir yapıya işaretçi yoluyla ulaşmak daha yararlıdır. Yapılara işaretçiler diğer data tiplerine işaretçilerle aynı yoldan gösterir. Örneğin takip eden bölüm kodu **p** değişkeni ve **q** yapı işaretçisini temp yapısıyla gösteren bir örnektir.

```
struct temp
{
     int i;
     char ch;
} p,q;
```

Temp yapısını bu tanımlaması kullanarak **q=&p** durumu tamamen geçerlidir. q p'yi gösterir. Burada gösterildiği gibi ok operatörü kullanılmalıdır.

```
q->i=1;
```

Bu durum p değişkenini ininci sayısına1 değerini atar.Ok işaretinin ,ok işaretinden hemen sonra negatif(-) işaret ve > işareti aralarında boşluk olmadan kullanıldığına dikkat edin.

C yapıdan fonksiyona geçtikten sonra büyük yapılar büyük data geçişlerinden dolayı program işlemesi hızını düşürür bu nedenle yapıdan fonksiyona işaretçi yoluyla geçilmesi daha kolaydır.

Not edilmesi gereken önemli bir şeyde :Bir yapı üyesine bir yapı değişkeni kullanılarak ulaşılmaya çalışıldığında bir periyot(dönem) kullan. Yapıdaki bir yapı üyesine işaretçi ile ulaşmaya çalıştığında ok işaretini kullanmak zorundasın.
Bu örnek bir yapıya bir işaretçinin nasıl kullanılacağını gösterir.

```
#include <16c74.h>
#include <string.h>
struct s_type
{
        int i;
        char str[80];
} s,*p;
void main(void)
{
        p=&s;
```

```
        s.i=10;
        p->i=10;
        strcpy(p->str,"I like structures");
        printf("%d %d %s",s.i,p->i,p->str);
}
```

İki satır **s.i=10** ve **p->i=10** birbirine eşittir.

ÖRNEK:
1. Bu kodun yapısı doğrumudur?

```
struct s_type
{
int a;
int b;
} s, *p;
void main(void)
{
p=&s;
p.a=100;
}
```

2. PICmicro® tipinde bir program yazarak 3. uzunlukta yapı işaretçisi oluştırun yapıları PIC16CXX ve PIC17CXX aletleri kullanarak yükleme ihtiyacı duyacaksınız. Kullanıcı klavyeden 1.2 ve 3 girişlerinde yazdırmak için hangi yapının kullanılacağını seçecek. Yapını formatı :

```
struct PIC
{
char name[20];
unsigned char progmem;
unsigned char datamem;
char feature[80];
};
```

8.3 Seri Yapıları

Şu ana kadar C data tiplerinden bir yapı çeşidi olan üyeleri sadece gördünüz bunla beraber yapıların üyeleri düğer yapılarda olabilir bunlara seri yapıları denir.Örneğin:

```
#define NUM_OF_PICS 25
struct PIC
{
        char name[40];
        unsigned char progmem;
        unsigned char datamem;
```

```
        char feature[80];
};
struct produtcs
{
        struct PIC devices[NUM_OF_PICS];
        char package_type[40];
        float cost;
} list1;
```

Bu yapı **devices** adında **PIC** yapı serisi paketin ismi olan bir string ve maliyeti olmak üzere 3 eleman üretir. Bu elemanlara **list1** değişkeni ile ulaşılabilir.

8.4 Birleşimlere Giriş

Bir bileşim(union) 2 veya 3 değişken tarafından paylaşılan tek bir hafıza yeridir. Bu değişkenle aynı hafıza yerinde değişik data tiplerinde olabilirler bunla beraber bir değişkeni bir defa kullanabilirsin. bir bileşim bir yapıya çok benzer bir birleşimin genel formatı şu şekildedir.

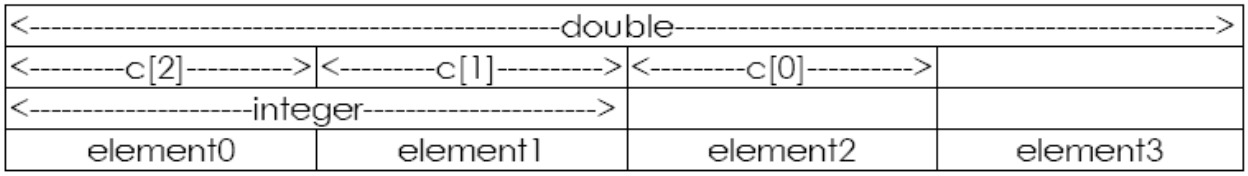
```
union tag-name
{
        type element1;
        type element2;
        .
        .
        type elementn;
} variable-list;
```

Yine **tag-name** bileşiğin adı ve **variable-list** değişkenleri ismi **tag-name** bileşiği oluşturan değişken tipidir. Bileşimlerle yapılar arasındaki fark bileşiğin her üyesi aynı data kullanmasıdır. Örneğin takip eden bileşim bir integer bir character ve double içerir.

```
union u_type
{
        int i;
        char c[3];
        double d;
} temp;
```

Bir bileşimin hafızadaki görünümü aşağıda gösterilmiştir.Bir önceki örneği bir bileşiğin değişimi için kullanacağız.

Integer 2 byte
Character 3 byte
Double 4 byte kullanır.



Bileşimlerin üyelerine ulaşmak yapılarla aynı olup periyod kullanılır. **temp.i** durumu **i** üyeli 2 baytlık integer'a ulaşmada **temp** ve **temp.d** bileşimi 4 baytlık double.d 'ye ulaşmada kullanılır.Bir bileşime işaretçi yoluyla ulaşmak isterseniz yapılardaki gibi ok işareti kullanmalısınız.

Bir bileşimin en büyük üyesine compiler'i taratma yapıldığında bileşimin büyüklüğüne dikkat etmek gerekir. Double 4 bayt uzunluğunda kabul edersek **temp** bileşkesi 4 bayt uzun uzunluğunda olmalıdır.
8-bit microkontroller'un seri portuna harici 12 bit A/D dönüştürücü(converter) bağlandığında bileşimi kullanmak güzel bir örnek . Microkontroller A/D'yi 2 bayt'da okur. Öyleyse bir bileşimini 2 **unsignedchars** olarak kurmalıyız ve üyeleri **signed short** olarak almalıyız.

```
union sample
{
      unsigned char bytes[2];
      signed short word;
}
```

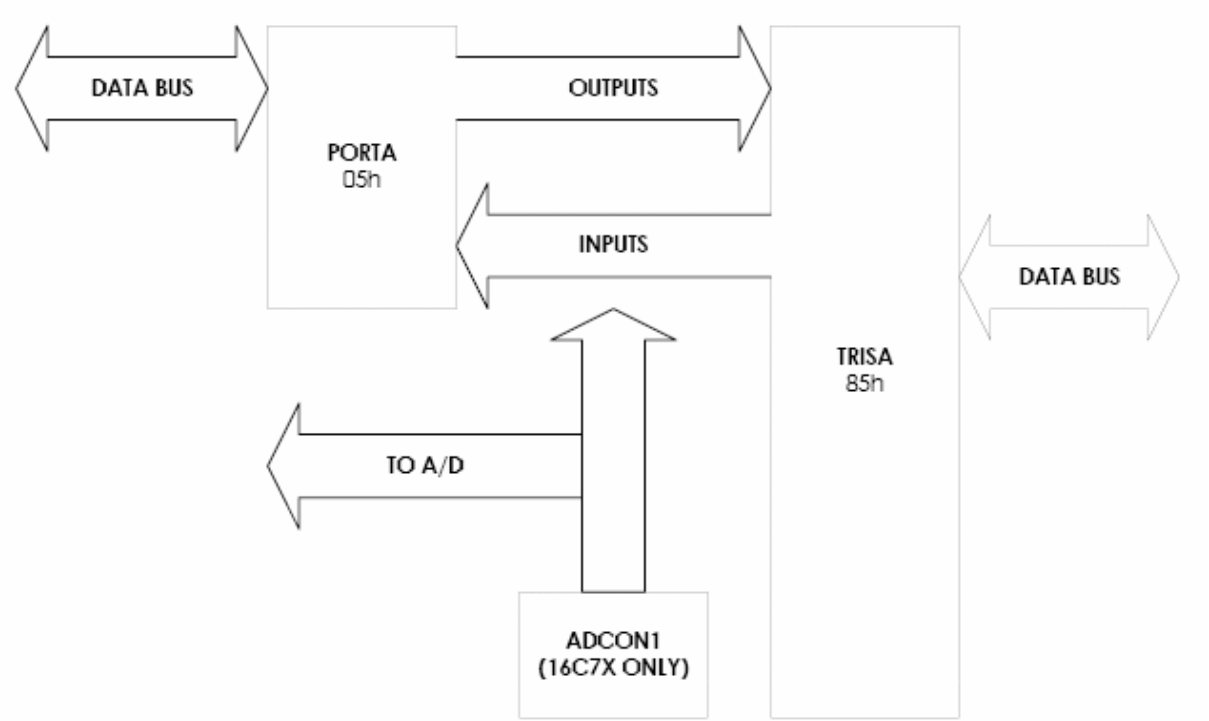
A/D'yi okumak istediğinde 2 byte'lık datayı A/D'den oku ve onları byte dizisinde sakla sonra 12 bit'lik örneği ne zaman kullanmak istersen 12 bit'lik sayıya ulaşmak için kelimeyi kullan.

Alıştırma:

1. Yapıyla bileşim arasındaki farklar nelerdir ? Benzerlikler nelerdir?
2. **long int** üye bileşimli ve 4 byte karakter sırası uzunluğunda bir program yazın. Program bir zaman sonra **long int 'i** her baytta ekranda göstersin.

9.1. GİRİŞLER VE ÇIKIŞLAR

PIC micro MCU'daki giriş çıkış portları 2 registerdan meydana gelmiştir.PORT VE PORT DIRECTION ve bunlar PORT A,B,C,D,E ve TRIS A,B,C,D,E olarak adlandırılmaktadır.Var olmaları tasarımda pic tarafından kullanılmalarına bağlıdır.8 bacaklı bir pic de tek gpıo registeri, tek tris registeri ve 6 I/O portu bulunur.16C74 de bulunan portlar ise PORT A,B,C,D ve E dir.33 adet I/O portu bulunur.aşağıda PORTA'nın blok diyagramı görülmektedir. B,C,D ve E PORTLARI benzerdir ama data sheet'te pic özellikleri için bakılması gerekir.



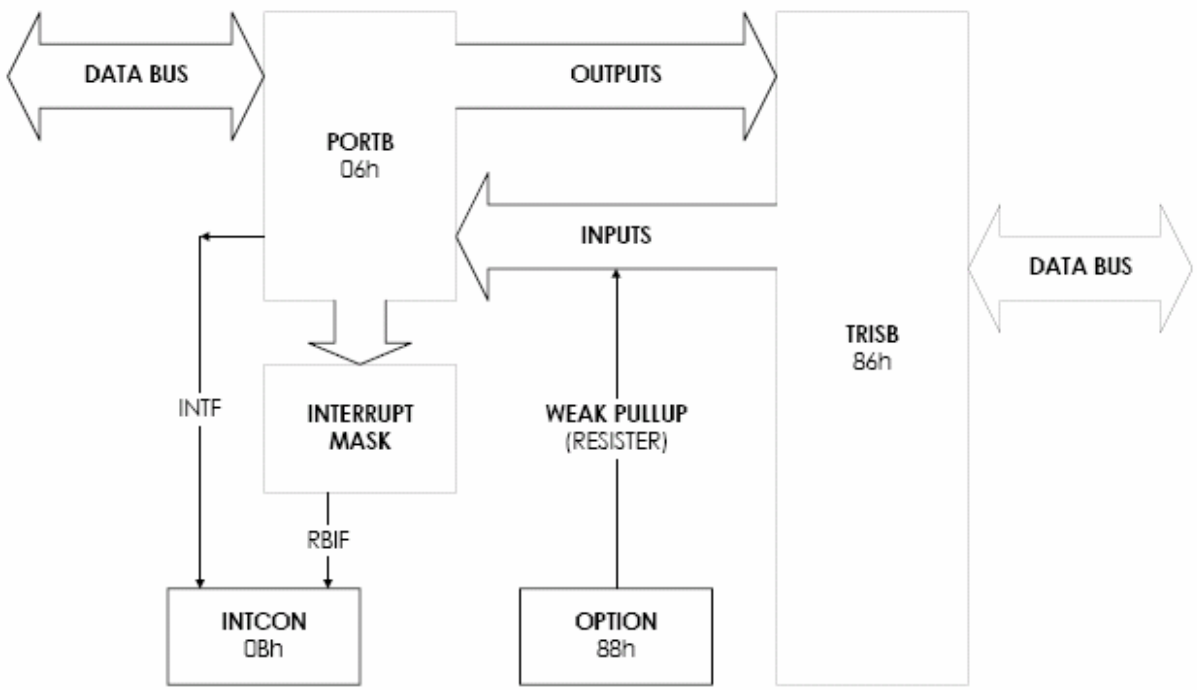
PIC çeşidine bağlı olarak Port A'nın 5 veya 6 ucu bulunmaktadır.ya girişlere yada çıkışlara göre düzenlenmektedir.port direction'nın düzenlenmesi Tris A registeri üzerinden yapılır. Girişler 1 ile çıkışlar 0 ile set edilir. Bacakların hepsi tipik olarak bacak başına 25ma kaynak ve akaç kapasitesine sahiptirler.

I/O uçlarına istisna açık kollektor çıkışına sahip olan A4 bacağıdır.

Sonuç olarak bacaktaki voltaj sevileri direnç ile yüksek bağlanmışsa Port A registerindaki biti tersler (port A.4 deki lojik 1 transistör üzerinden uyarılırsa ucu sıfıra çeker).
Assembler olarak bir örnek;

```
CLRF      PORTA          ;set outputs low
PAGE1                    ;select register page1
MOVLW     B'00000011'    ;A0,1 as inputs, A2-4 as outputs
MOVWF     PORTA          ;send W to port control register
PAGE0                    ;change back to register page 0
```

Veri MOVWF PORT üzerinden porta gönderilir ve bitler ya BSF yada BCF ile ayrı ayrı hareket ettirilebilir.Veri ya MOVFW PORTA yada BTFSS veya BTFSC ile port tarafından okunur.



PORT B nin BLOK DİYAGRAMI

PORT C benzerdir ama port B nin pull-up ve interrupt yeteneği yoktur.

Port C için diğer kullanılan I/O pinleri

C0	C1	C2	C3	C4	C5	C6	C7
I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O
T1OSO	T1OSI	CCP1	SCK	SDI	SDO	TX	RX
T1CKI	CCP2	PWM1	SCL	SDA		CK	DT
	PWM2						

C compiler sabit yolların numaralarına göre hızlı veya standart girişleri ve çıkışları interrupt(KESME) edebilir.standart moda, PORT DİRECTİON registerleri tüm I/O operasyonlarından önce kurulur. programa eklenen hatlar hızı yavaşlatır ama kodun güvenlik kısmını I/O hatlarının her zaman set edildiğini garanti ederek yoluna koyar.
Hızlı I/O kullanıcıya port direction'u set etmesini sağlar ve bu oradaki kalıntıları tanımlanmaz hale getirir.
Port B nin girişlerle set edilmesi ve değer içinde okunması ile ilgili bir örnek;

```
set_tris_b(0xff);       //make inputs
b_rate = portb;         //read in port
```

bit maskelemesi **&** eklenerek kolayca yapılır ve port isminden sonra örneğin maskelenmesi

b_rate = portb & 0b00000011; //mask out unwanted bits

değer **b_rate** içinde saklanmış sonra çağırılan fonksiyona döneceği değere kurulması için kullanılmıştır. Sonra bütün fonksiyon bazı anahtarların okuması için kullanılmıştır.

```
byte bd_sw_get() //baud rate selection
{
        byte b_rate;
        b_rate = portb & 0b00000011; //mask out unwanted
bits
        switch(b_rate)
        {
                case 0: set_uart_speed(1200);
                break;
                case 1: set_uart_speed(2400);
                break;
                case 2: set_uart_speed(4800);
                break;
                case 3: set_uart_speed(9600);
                break;
        }
}
```

PORT B kurulacağı zaman, port direction registerdan önce port durumlarının kurulması doğru olur.bu portu dışarıdan gelen kurulmadan önceki bilinmeyen durumlardan alıkoyar.register içindeki veya porttaki bit modelleri kurulacağı zaman,binary olarak çalışılır. Bu kaynak kodunu yazmada ve okumada size kolaylık sağlar.I/O portlardan ve I/O portlara verinin hareket ettirilmesi fonksiyonlardaki pek çok yapının kullanılmasıyla kolayca yapılır.burada bit seviye görülmektedir;

```
bit_set(variable, bit); //used to set a bit
bit_clear(variable, bit); //used to clear a bit
bit_test(variable, bit); //used to test a bit
```

yukarıdakiler değişkenlerde ve I/O da kullanılabilir.

```
        b = input(pin); //get the state or value of a pin
        output_bit(pin, value); //set a port pin to a specific
value
        output_float(pin); //set a pin to input or floating
mode
```

```
output_high(pin); //set an output to logic 1
output_low(pin); //set an output to logic 0
```

PORT B üzerindeki pull up'lar enable veya disable edilir.

```
set_tris_a(value)
```

Verilen porta göre girişlerin ve çıkışların kombinasyonunun set edilmesi, girişlerin 1 ile çıkışların 0 ile set edilmesiyle olur.

Port direction registerları portun giriş olduğu her zaman düzenlenir ve önceki işlemciyi takip etmedikçe şunlar kullanılır;

```
#use fast_io(port)
#use fixed_io(port_outputs=pin, pin)
#use standard_io(port)
```

9.2. MİXİNG C AND ASSEMBLER

C programının içinde assembler kod isteyebilir. Nedenleri olarak kod yoğunluğu,zaman sınırlamaları veya basit hale getirmektir.Aşağıdaki örnekte program çağrıldığında d değerinin a ya eşitini bulmaktadır.

```
FindParity(type d)
{
        byte count;
        #asm
                movlw 8
                movwf count
                clrw
        loop:
                xorwf d,w
                rrf d,f
                decfsz count,f
                goto loop
                movwf _return_
        #endasm
}
main()
{
        byte a,d=7;
        a=FindParity(d);
}
```

Derlendiği zaman program şu hale gelmektedir.

```
FindParity(type d)
        {
        byte count;
        #asm
```

```
                0005: MOVLW 8
                0006: MOVWF count
                0007: CLRW
                0008: XORWF 27,W
                0009: RRF 27,F
                000A: DECFSZ 28,F
                000B: GOTO 008
        #endasm
                000C: MOVWF 21
                000E: GOTO 016
        }
main()
        {
                0011: MOVLW 07
                0012: MOVWF 26
byte a,d=7;
a=FindParity(d);
                0013: MOVF 26,W
                0014: MOVWF 27
                0015: GOTO 005
                0016: MOVF 21,W
                0017: MOVWF 25
        }
```

PİC 16Cxx Ailesine Uygun Hale Getirmek için;

Alan Tanımlama;
b: 8 bitlik file registeri içindeki bit adresi (0-7)
d: hedef seçimi; d=0 sonucu W'ya,d=1 ise sonucu file registerına kaydet. Assembler hedef olarak Wve f 'i kullanır.
f: file registerı adresi (0x00-0xFF)
k: sabit veri veya etiket;
W: accumulator
i: pointer kontrol tablosu (i=0 değişiklik yok, i=1 ise komuttan sonra değeri artır)

PIC 16CXX
Sabit İşleyen ve Kontrol Komutları

Hex	Mnemonic		Description	Function
3Ekk	ADDLW	K	Add literal to W	k + W >> W
39kk	ANDLW	k	AND literal and W	k .AND. W >> W
2kkk	CALL	k	Call subroutine	PC+1 >> TOS, k >> PC
0064	CLRWDT		Clear watchdog timer	0 >> WDT (and Prescaler, If assigned)
2kkk	GOTO	k	Goto address(k is 9 bits)	k >> PC(9 bits)
38kk	IORLW	k	Incl. OR literal and W	k .OR. W >> W
30kk	MOVLW	k	Move literal to W	k >> W
0009	RETFIE		Return from Interrupt	TOS >> PC, 1 >> GIE
34kk	RETLW	k	Return with literal in W	k >> W, TOS P >> C
0008	RETURN		Return from subroutine	TOS >> PC
0063	SLEEP		Go into Standby Mode	0 >> WDT, stop oscillator
3Ckk	SUBLW	k	Subtract W from literal	k – W >> W
3Akk	XORLW	k	Exclusive OR literal and W	k .XOR. W >> W

Byte Yönlendirmeli Komutlar

Hex	Mnemonic		Description	Function
07Ff	ADDWF	f, d	Add W and f	W + f >> d
05Ff	ANDWF	f, d	AND W and f	W .AND. f >> d
018F	CLRF	f	Clear f	0 >> f
0100	CLRW		Clear W	0 >> W
09Ff	COMF	f, d	Complement f	.NOT. f >> d
03Ff	DECF	f, d	Decrement f	f – 1 >> d
0BFf	DECFSZ	f, d	Decrement f, skip if zero	f – 1 >> d, skip if 0
0AFf	INCF	f, d	Increment f	f + 1 >> d
0FFf	INCFSZ	f, d	Increment f, skip if 0	f + 1 >> d, skip if 0
04Ff	IORWF	f, d	Inclusive OR W and f	W .OR. f >> d
08Ff	MOVF	f, d	Move f	f >> d
008F	MOVWF	f	Move W to f	W >> f
0000	NOP		No operation	
0DFf	RLF	f, d	Rotate left f	
0CFf	RRF	f, d	Rotate right f	
02Ff	SUBWF	f, d	Subtract W from f	f – W >> d
0EFf	SWAPF	f, d	Swap halves f	f(0:3) << f(4:7) >> d
06Ff	XORWF	f, d	Exclusive OR W and f	W .XOR. f >> d

Bit Yönlendirmeli Komutlar

Hex	Mnemonic		Description	Function
10Ff	BCF	f, b	Bit clear f	0 >> f(b)
14Ff	BSF	f, b	Bit set f	1 >> f(b)
18Ff	BTFSC	f, b	Bit test, skip if clear	skip if f(b) = 0
1CFf	BTFSS	f, b	Bit test, skip if set	skip if f(b) = 1

PIC 16C5X

Sabit İşleyen ve Kontrol Komutları

Hex	Mnemonic		Description	Function
Ekk	ANDLW	k	AND literal and W	k .AND. W >> W
9kk	CALL	k	Call subroutine	PC+1 >> TOS, k >> PC
004	CLRWDT		Clear watchdog timer	0 >> WDT (and Prescaler, If assigned)
Akk	GOTO	k	Goto address(k is 9 bits)	k >> PC(9 bits)
Dkk	IORLW	k	Incl. OR literal and W	k .OR. W >> W
Ckk	MOVLW	k	Move literal to W	k >> W
002	OPTION		Load OPTION Register	W >> OPTION Register
8kk	RETLW	k	Return with literal in W	k >> W, TOS P >> C
003	SLEEP		Go into Standby Mode	0 >> WDT, stop oscillator
00f f	TRIS	f	Tri-state port f	W >> I/O control register
Fkk	XORLW	k	Exclusive OR literal and W	k .XOR. W >> W

Byte Yönlendirmeli Komutlar

Hex	Mnemonic		Description	Function
1Cf	ADDWF	f, d	Add W and f	W + f >> d
14f	ANDWF	f, d	AND W and f	W .AND. f >> d
06f	CLRF	f	Clear f	0 >> f
040	CLRW		Clear W	0 >> W
24f	COMF	f, d	Complement f	.NOT. f >> d
0Cf	DECF	f, d	Decrement f	f – 1 >> d
2Cf	DECFSZ	f, d	Decrement f, skip if zero	f – 1 >> d, skip if 0
28f	INCF	f, d	Increment f	f + 1 >> d
3Cf	INCFSZ	f, d	Increment f, skip if 0	f + 1 >> d, skip if 0
10f	IORWF	f, d	Inclusive OR W and f	W .OR. f >> d
20f	MOVF	f, d	Move f	f >> d
02f	MOVWF	f	Move W to f	W >> f
000	NOP		No operation	
34f	RLF	f, d	Rotate left f	
30f	RRF	f, d	Rotate right f	
08f	SUBWF	f, d	Subtract W from f	f – W >> d
38f	SWAPF	f, d	Swap halves f	f(0:3) << f(4:7) >> d
18f	XORWF	f, d	Exclusive OR W and f	W .XOR. f >> d

Bit Yönlendirmeli Komutlar

Hex	Mnemonic		Description	Function
4bf	BCF	f, b	Bit clear f	0 >> f(b)
5bf	BSF	f, b	Bit set f	1 >> f(b)
6bf	BTFSC	f, b	Bit test, skip if clear	skip if f(b) = 0
7bf	BTFSS	f, b	Bit test, skip if set	skip if f(b) = 1

PIC 16C5X / PIC 16CXX

Bazı Özel Komutlar

Bu tür komutlar assembler tarafından ve programda bazı komutların yerine kullanılır. Makrolara benzeyen kısayolları vardır.

Mnemonic		Description	Assembly Code		Flag
ADDCF	f, d	Add Digit Carry to File	BTFSC	Status, Carry	
			INCF	f, d	Z
B	k	Branch	GOTO	k	
BC	k	Branch on Carry	BTFSC	Status, Carry	
			GOTO	k	
BDC	k	Branch on Digit Carry	BTFSC	Status, Digit Carry	
			GOTO	k	
BNC	k	Branch on No Carry	BTFSS	Status, Carry	
			GOTO	k	
BNDC	k	Branch on No Digit Carry	BTFSS	Status, Digit Carry	
			GOTO	k	
BZ	k	Branch on Zero	BTFSC	Status, Zero	
			GOTO	k	
BNZ	k	Branch on No Zero	BTFSS	Status, Zero	
			GOTO	k	
CLRC		Clear Carry	BCF	Status, Carry	
CLRDC		Clear Digit Carry	BCF	Status, Digit Carry	
CLRZ		Clear Zero	BCF	Status, Zero	
MOVWF	k	Move File to W	MOVF	f, W	Z
NEGF	f ,d	Negative File	COMF	f, f	
			INCF	f, d	Z
SETO		Set Carry	BSF	Status, Carry	
SETDC		Set Digit Carry	BSF	Status, Digit Carry	
SETZ		Set Zero	BSF	Status, Zero	
SKPC		Skip on Carry	BTFSS	Status, Carry	
SKPNC		Skip on No Carry	BTFSC	Status, Carry	
SKPDC		Skip on Digit Carry	BTFSS	Status, Digit Carry	
SKPNDC		Skip on No Digit Carry	BTFSC	Status, Digit Carry	
SKPZ		Skip on Zero	BTFSS	Status, Zero	
SKPNZ		Skip on No Zero	BTFSC	Status, Zero	
SUBCF	f, d	Subtract Carry from File	BTFSC	Status, Carry	
			DECF	f, d	Z
SUBDCF	f, d	Sub Digit Carry from File	BTFSC	Status, Digit Carry	
			DECF	f, d	Z
TSTF	f	Test File	MOVF	f, f	Z

9.3. İLERİ BİT KULLANIMI

CCS C Compiler, genelde PİC micro programaları için gerek duyulan bit kullanım fonksiyonlarının numarasına sahiptir.

bit_set, **bit_clear** ve **bit_test** basitçe biti değişken içinde set edebilir ve silebilir veya tek bitin durumunu test edebilir.Bitler düşük dereceli bite göre numaralandırılırlar (ilk sırada 0.bit, en yüksek değerde 7. bit bulunur).

Örneğin;

```
c='A';                    //c in binary is now 01000001
bit_test(c,5);  //c is now 01100001 or 'a'
if(bit_test(x,0))
        printf("X is odd");
else
        printf("X is even");
```

shift_left ve **shift_right** komutları bitin numarasına göre bir bit kaydırır. Buna ek olarak bu komutlar size bitin yerini değer olarak boşaltmanıza yardımcı olabilir.bu komutlar değerleri 0 yada 1 ise dışarıya kaydırıldığında geri dönderirler. Not;bu komutlar hafızadaki LSB düşük byte göze alırlar. İkinci parametre byteların numarası ve son parametre yeni bittir.

Örnek:

```
        int x[3] = {0b10010001, 0b00011100, 0b10000001};
        short bb; //x msb first is: 10000001, 00011100,
10010001
        bb = shift_left(x,sizeof(x),0);
                // x msb first is: 00000010, 00111001,
00100010
                //bb is 1
        bb = shift_left(x,sizeof(x),1);
                // x msb first is: 00000100, 01110010,
01000101
                //bb is 0
```

Not:Birinci parametre pointer'dır. Bu durumda x in bilinmesine kadar önceki pointer'dır. Basit değişken veya yapı kullanılmış ise & operatörü eklenmelidir.

```
long y;
struct { int a,b,c} z;
shitf_left(&y,2,0);
shitf_right(&z,3,0);
```

rotate_left ve **rotate_right** komutları da shift komutları gibi çalışır. Bit dışarı doğru kaydırılmışsa bir tarafı diğer tarafa kaydırır. Örneğin;

```
int x[3] = {0b10010001, 0b00011100, 0b10000001};
```

```
        //x msb first is: 10000001, 00011100, 10010001
rotate_left(x,sizeof(x));
        //x msb first is: 00000010, 00111001, 00100011
```

swap fonksiyonu düşük 4 biti yüksek 4 bitle değiştirir. Örneğin;

```
 int x;
x = 0b10010110
swap(x); //x is now 01101001
```

9.4. TIMERS

Bütün pic'lerde 8 bit timer vardır ve bazı piclerin fazladan 2 timeri vardır.

rtcc (timer0) = 8Bit

clock komutu veya harici kaynak tarafından yükseltilir.
Pre-scaler in yavaş artmasını sağlar.
Timer 0 ,255 ten 0 a düştüğü zaman kesme meydana gelir (16C5X hariç).

timer1 = 16Bit

clock komutu veya harici kaynak tarafından yükseltilir.
Pre-scaler in yavaş artmasını sağlar
Timer 1,65535 ten 0 a düştüğü zaman kesme meydana gelir
Capture moda, timer 1 bacaklar değişirse diğer register da kaydedilerek sayar.kesme yine meydana gelir.
Compare moda,sayma istenen değere ulaştığında bacak değişebilir ve kesme meydana gelir.
Bu timer PWM in bir kısmı parçası çalışır.

timer2 = 8Bit

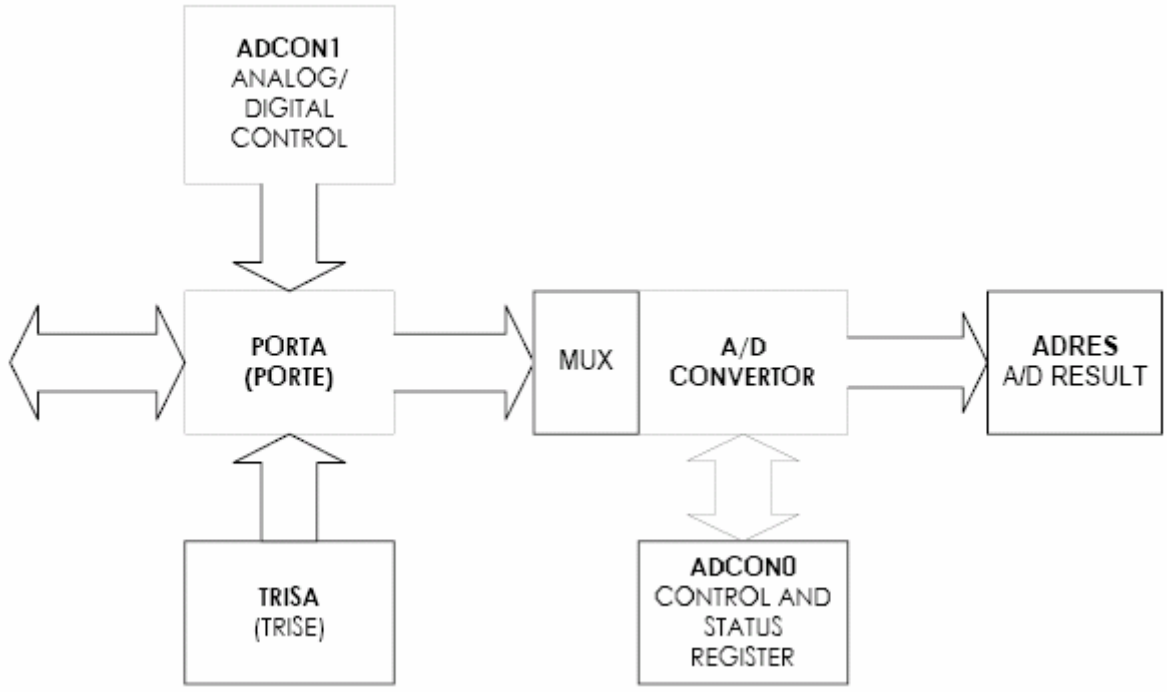
clock komutu veya harici kaynak tarafından yükseltilir.
Pre-scaler in yavaş artmasını sağlar
Timer 2,255 ten 0 a düştüğü zaman kesme meydana gelir
Kesme post-scaler tarafından kullanıldığında yavaşlayacaktır böylece kesme olmadan önce düşeceği belli bir numarayı isteyecektir.
Bu timer PWN in bir parçası olarak çalışır.
Aşağıdaki basit örnekte ne kadar sürede high durumana geçeceğini gösteren rtcc verilmiştir.

```
#include <16c74.h>
#fuses HS,NOWDT
#use delay(clock=1024000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
main() {
        int time;
        setup_counters(rtcc_internal, rtcc_div_256);
```

```
            //increments 1024000/4*256 times per second
            //or every millisecond
        while(!input(PIN_B0)); //wait for high
            set_rtcc(0);
        while(!input(PIN_B0)); //wait for low
            time = get_rtcc();
        printf("High time = %u ms.",time);
}
```

Aşağıdaki örnekte timer1 kullanılmış B0 pininin high seviyeye sürüldükten sonra C2 pinin ne kdar sürede high olduğu gösterilmiştir.

```
#include <16c74.h>
#fuses HS,NOWDT
#use delay(clock=8000000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
#bit capture_1 = 0x0c.2             //pir1 register
                                    //bit 2 = capture has taken place
main()
{
    long time;
    setup_timer1(t1_internal | t1_div_by_2);
                //Increments every 1 us
    setup_ccp1(ccp_capture_re);
                //configure CCP1 to capture rise
    capture_1=0;
    set_timer1(0);
    output_high(PIN_B0);
    while(!capture_1);
        time = ccp_1;
    printf("Reaction time = %1u us.",time);
}
```



9.5. A/D ÇEVİRİM

A/D 16C7x ve 12C67x'lerde 8 bitin rezolüsyonundadır. Bunun anlamı voltaj ölçüldüğünde 255 değer içinden biri olması gerekmektedir.eğer 5V kullanılacaksa, 5/255=19.6 mV0 dan 5V a olarak bulunur. Fakat, eğer ki referans voltajı 2.55 volt ise resolusyon 10mV olur ama dizi 0dan 2.55 e düşer.Diğer microchip parçalarında 10,11,12 ve 16 bit resolusyon vardır. Not:portlar için bir varsayım, bütün analog ve dijital kısımları analog yapmaktır.analog ve diijtal kullanımında I/O hattının kombinasyonu açısından bu not önemlidir.Aşağıdaki tablolar datasheet'lerden alınmıştır.

16C72/3				16C74 only			
A0, A1	**A2**	**A3**	**A5**	**E0**	**E1**	**E2**	**Vref**
A	A	A	A	A	A	A	Vdd
A	A	Vref	A	A	A	A	A3
A	D	A	A	D	D	D	Vref
D	D	Vref	A	D	D	D	A3
A	A	A	D	D	D	D	Vref
A	A	Vref	D	D	D	D	A3
D	D	D	D	D	D	D	---

16C71, 16C710, 16C711

A0, A1	A2	A3	Vref
A	A	A	Vdd
A	A	Vref	A3
A	D	D	Vdd
D	D	D	---

C de A/D nin setup ve operasyonları hazır library programlarla basit olarak yapılır.

set_adc_channel(0-7)

a/d çevrim için kanal seçimi

setup_adc(mode)

a/d dönüştürücüyü set etme
Modlar aşağıdaki gibidir:
adc_off, adc_clock_div_2, adc_clock_div_8, adc_clock_div_32,adc_clock_internal

setup_adc_ports(mix)

ADC bacaklarını analog,dijital veya combinasyonalrı şeklinde yükler.Sabitler **all_analog** ve **no_analog** bütün çipler için aynıdır.Diğer bazı sabitlere örnekler;
ra0_ra1_ra2_ra3_analog/a0_ra1_analog_ra3_ref

read_adc()

Analog dijital dünüştürücüden dijital veriyi okur.
Setup_adc ve Set_adc komutları çoğu kez bu komuttan önce çağırılır.

Örnek:

```
setup_adc(ALL_ANALOG); //sets porta to all analog inputs
set_adc_channel(1); //points a/d at channel 1
        delay_ms(5000); //waits 5 seconds
        value = read_adc(); //reads value
        printf("A/D value = %2x\n\r", value);//prints value
```

9.6. VERİ HABERLEŞMESİ /RS-232

Mühendislik hayatında PC ler, modemler arasındaki RS-232 haberleşmesi önemli bir yer tutmaktadır.Kendi başlarına yapılmış ürünler dış dünya ile haberleşmesi için ara birimlere ihtiyaç duyarlar. D konnektörü üzerinde bulunan 9 veya 25 pinin permütasyonu çok fazladır. En az ara birim 3 hat ile yapılabilir. Bunlar toprak hattı, alıcı hattı ve verici hattıdır.RS 232 ara birimi ile bağlantı yapılırken hangisinin veri haberleşme aletleri (DCE) hangisinin veri terminal aletleri (DTE) olduğunu bilmek önemlidir.

Konnektörler;

9 Pinli D Konnektörü;

Pin	Fonksiyon	Veri Yönü
1	Elde Kontrolü	
2	Alınan Veri	

3	Gönderilen Veri	O
4	Veri Terminali Hazır	O
5	Toprak	<>
6	Veri Kümesi Hazır	|
7	Gönderme İsteği	O
8	Göndermek için Temiz	|
9	Zil Belirleyici	|

25 Pinli D Konnektörü;

Pin	Fonksiyon	Veri Yönü
1	Koruma Toprağı	<>
2	Gönderilen Veri	O
3	Alınan Veri	|
4	Gönderme İsteği	O
5	Göndermek için Temiz	|
6	Veri Kümesi Hazır	|
7	Sinyal Toprağı	<>
20	Veri Terminali Hazır	O
22	Zil Belirleyici	|

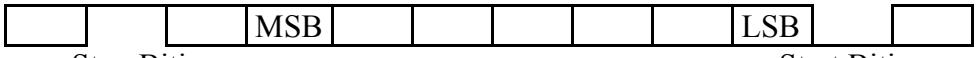
Diğer pinler basit bağlantı fonksiyonlarına sahip değildir.
Seri verileri uzak mesafelere göndermekte kullanılan modem, mikrodalga hatları gibi aletlere Data Communication Equipments (Veri Haberleşme Aletleri-DCE), seri verileri gönderen ve alan terminallere ve bilgisayarlara ise Data Terminal Equipments (Veri Terminal Aletleri-DTE) denir.
RS 232 sinyalleri için gerilim seviyeleri şu şekildedir;

Lojik 0-- +3V~ +25V
Lojik 1-- -3V~ -25V

Veri Formatları

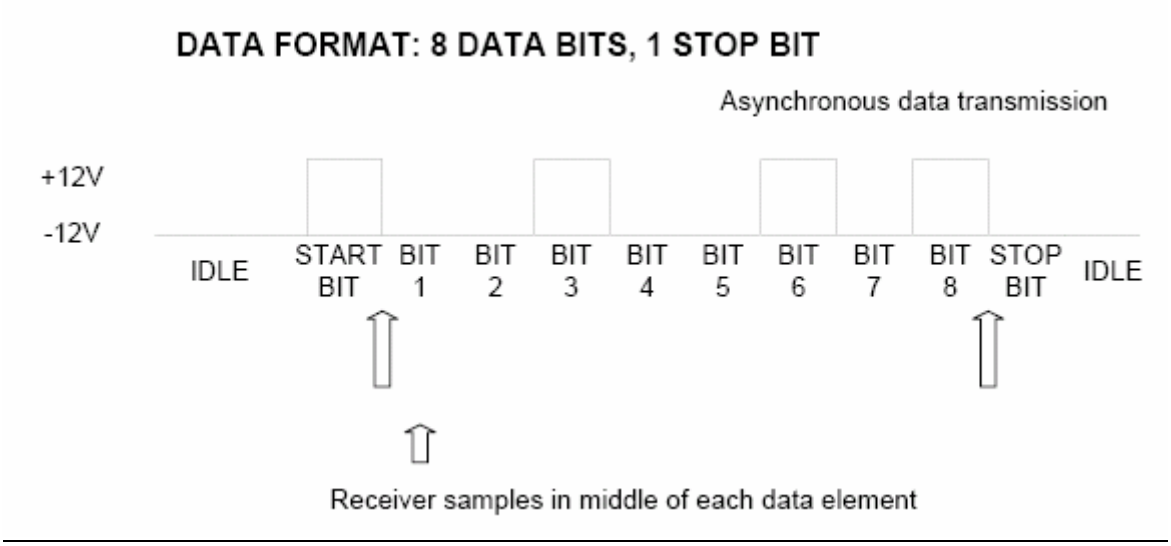
Başlangıç Biti	Her zaman 1 bit
Bitiş Biti	1 veya 2 bit
Veri Bitleri	7 veya 8 bit
Parity Biti	Hata tespiti istenmiyorsa hiçbir bit
	Hata tespiti isteniyorsa tek veya çift bitler

Asenkron Veri İletimi;



Stop Biti Start Biti

Asenkron Seri İletişim;
Asenkron iletişim belirgin özellikleri şunlardır;
-Transfer karakter bazında yapılır.
-Her bir veri transfer cihazının(karşılıklı uçlardaki) iletişim parametreleri benzer olmalıdır.(hız,kodlama seti vb.)
Bir asenkron karakter start biti,parity biti,veri bitleri ve stop bitlerinden oluşur.



Parity Bit(Eşlik Biti);

Transfer edilen karakterin karşı tarafa doğru gönderilip gönderilmediğini kontrol etmek için kullanılan bittir.Eğer alıcı alınan parity biti ile hesaplanan parity bitinin eşit olmadığını tespit ederse hata verir ve o andaki karakteri kabul etmez.Asenkron iletişimli modemlerde parity biti çeşitleri ODD,EVEN,MARK,SPACE yada NONE'dir.

ODD parity sisteminde tüm verinin içindeki 1'ler ile parity biti toplamı tektir.Mesela 8 bitlik '10101100' verisinin parity biti '1' olmalıdır. EVEN parity sisteminde tüm verinin içindeki 1' ler ile parity biti toplamı çifttir. Mesela 8 bitlik '10101100' verisinin parity biti '0' olmalıdır.

Parity sistemi hataları düzeltmez sadece hatanın oluştuğunu gösterir. Böylece bir hata oluştuğunda verinin tekrar yollanması için istek yapılır. PIC micro MCU'da parity testi yazılımsal olarak oluşturulmaktadır.

Baud, Bit/sn olarak ifade edilen bit iletişim hızı birimidir. Analog sinyaldeki değişim hızıdır. Örneğin bir bitin geçmesi için gereken zaman 3.33 msn ise baud oranı 1/3,33 msn veya 300 baud'dur.RS232, 15 metre mesafe için 20000 baud, 915 metre için 2000 baud hızlarıyla çalışır. Yükselme ve düşme zaman hızları, karışıklıkları önlemek açısından 30 V/µsn ile sınırlandırılmıştır.

ASCII DÖNÜŞÜM TABLOSU

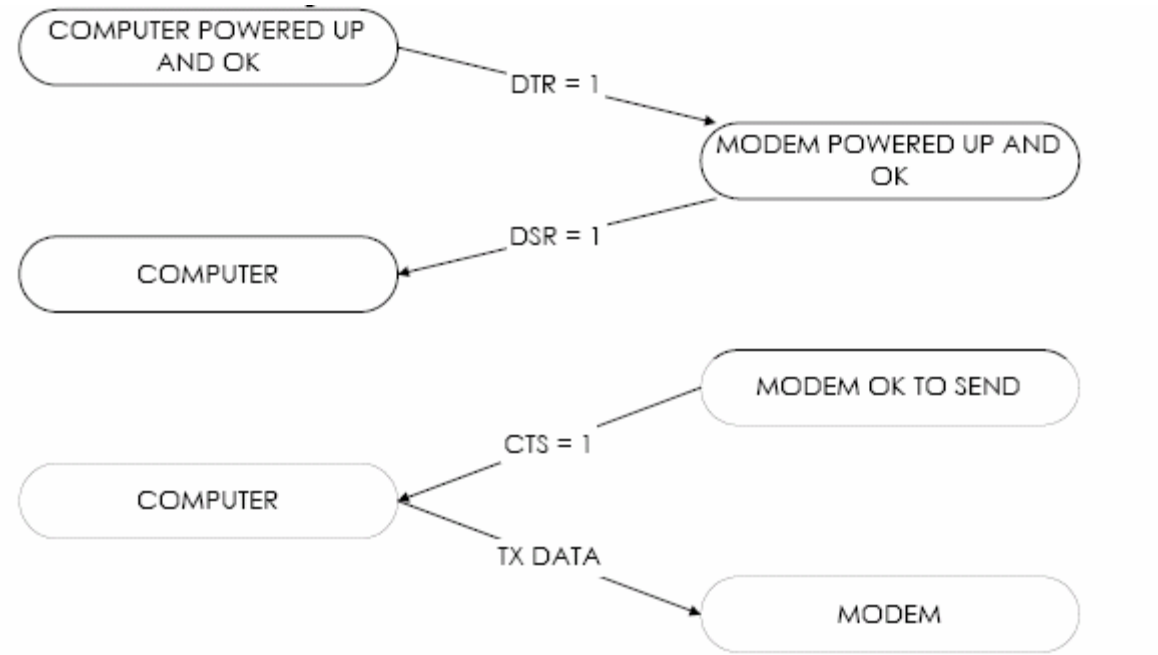
Control

HEX	lsb	msb bits	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
	0	0000	NUL	DLE	SP	0	@	P	-	p
^A	1	0001	SOH	DC1	!	1	A	Q	a	q
^B	2	0010	STX	DC2	"	2	B	R	b	r
^C	3	0011	ETX	DC3	#	3	C	S	c	s
^D	4	0100	EOT	DC4	$	4	D	T	d	t
^E	5	0101	ENQ	NAK	%	5	E	U	e	u
^F	6	0110	ACK	SYN	&	6	F	V	f	v
^G	7	0111	BEL	ETB	'	7	G	W	g	w
^H	8	1000	BS	CAN	(	8	H	X	h	x
^I	9	1001	HT	EM	)	9	I	Y	i	y
^J	A	1010	LF	SUB	*	:	J	Z	j	z
^K	B	1011	VT	ESC	+	;	K	[	k	{
^L	C	1100FF	FS	,	<	L	\	l		
^M	D	1101CR	GS	-	-	M	]	m	}	
^N	E	1110SO	RS	.	>	N	^	n	~	
^O	F	1111 SI	US	|	?	O	_	o	DEL	

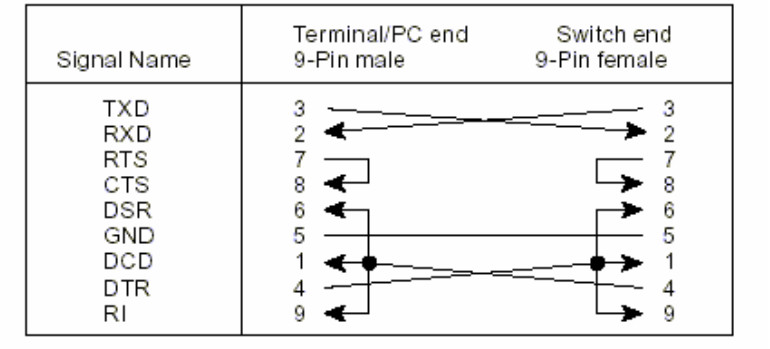
ASCII SEMBOLLERİNİN AÇILIMLARI

NUL	-	Null	DLE	-	Data Link Escape
SOH	-	Start of Heading	DC	-	Device Control
STX	-	Start of Text	EXT	-	End of Text
EOT	-	End of Transmission	ENQ	-	Enquiry
NAK	-	Negative Acknowledge	ACK	-	Acknowledge
SYN	-	Synchronous Idle	BEL	-	Bell
ETB	-	End Transmission Block	BS	-	Backspace
CAN	-	Cancel	HT	-	Horizontal Tab
EM	-	End of Medium	LF	-	Line Feed
SUB	-	Substitute	VT	-	Vertical Tab
ESC	-	Escape	FF	-	Form Feed
FS	-	File Separator	CR	-	Carriage Return
GS	-	Group Separator	SO	-	Shift Out
RS	-	Record Separator	SI	-	Shift In
US	-	Unit Separator	DEL	-	Delete
SP	-	Space (blank)			
DC1	-	Xon	DC3	-	Xoff

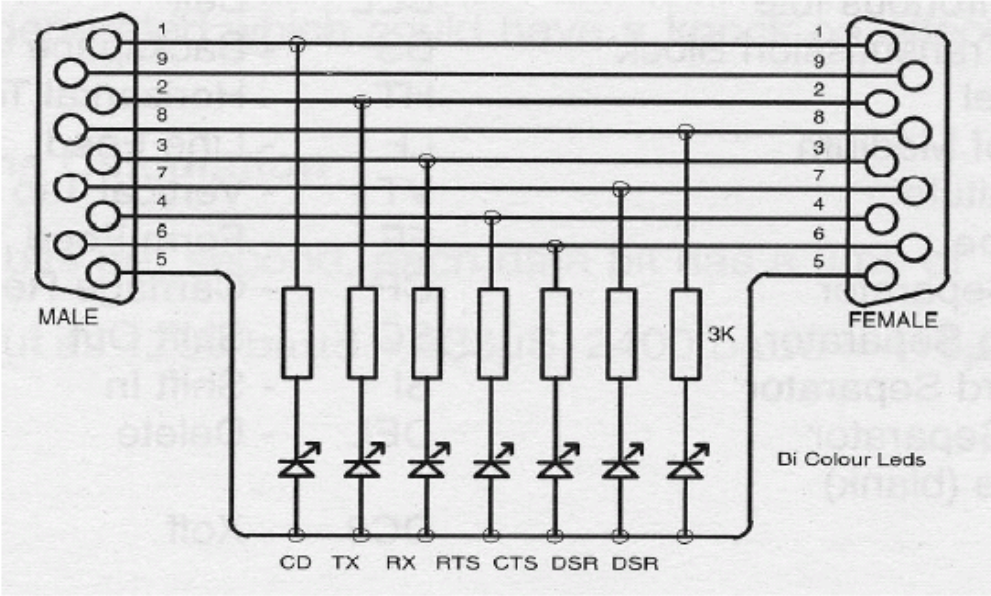
RS232 BAĞLANTI PROTOKOLÜ



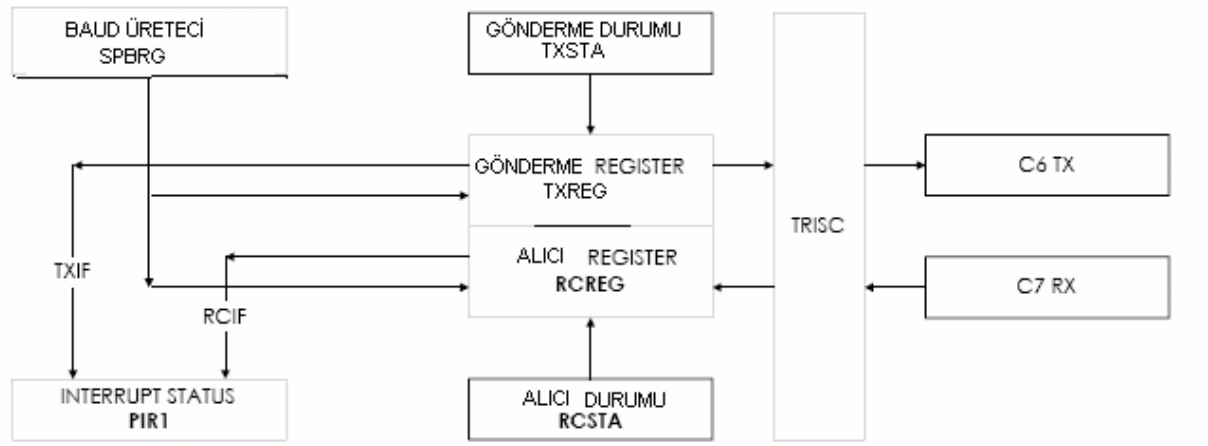
TİPİK MODEM BAĞLANTISI



RS232 TEST BAĞLANTISI



USART



Usart 3 moddan birinde çalışır. Senkron master, senkron slave ve asenkron. Asenkron modu, en çok kullanılan moddur. Bu modlarla pc'lere ve modemlere arabirim olmaktadır. Bundan başka usart A/D,D/A ve EEPROM'a arabirim olmaktadır.

Usart'ta kabul edilen veri formatı 8 veya 9 veri biti, none, odd veya even parity'dir. Parity sistemi yazılımda yapılmaktadır donanım fonksiyonu değildir. Alınan veri, framing ve overrun error olarak bilinen hatalar için kontrol edilir. Asenkron modda usart tam duplex haberleşmeyi sağlar, senkron modda ise yarı duplex haberleşmeyi sağlar. Önceden tanımlanan fonksiyonlar uygulamanın hızını arttırır;

```
#use fixed_io(c_outputs=pin_C6)   //speeds up port use
#use delay(Clock=4000000)         //clock frequency
#use rs232(baud=4800, xmit=PIN_C6, rcv=PIN_C7)
```

CCS compiler dahili olarak bulunan pic micro MCU'nun uartını kullanabilme yeteneğine sahiptir. Uart donanım olarak mevcutsa oluşturulan kod donanımı kullanacaktır. Donanım olarak mevcut değilse oluşturulan kod daha büyük olacaktır. Alıcı ve vericide ki interrupt dışında kod

donanım uart gibi davranır. Yazılım uart çıkıştaki veri seviyelerini terslendirme kabiliyetine sahiptir. Harici sürücü veya seviye kaydırıcıya gerek duymaz. Bu fonksiyon donanımda mevcut değildir. Bir pic micro MCU uygulamasının kodu başka bir uygulamaya rahatlıkla aktarılabilir.

C compiler haberleşme için hazır fonksiyonlara sahiptir, Örneğin;

`getc, getch, getchar`	RS232'nin rev pininden bir karakter alınırken bekletilmesini sağlar.
`gets(char *string)`	Taşıyıcı alınana kadar karakterin s string'ini okur. Karakterin en büyük uzunluğunu dizinin boyutu belirler.
`putc put char`	RS 232 xmit pinine tek karakter gönderir.
`puts(s)`	Besleme hattına string yollar.

kbhit () fonksiyonu bir karakterin hazır olup olmadığını saptar. Getc() fonksiyonunda karakterin beklemesi kbhit () fonksiyonu ile sağlanır. Aşağıdaki örnek bir karakter için 1,5 sn beklemektedir.

```
char timed_getc()
{
    long timeout;
    timeout_error=FALSE;
    timeout=0;
    while(!kbhit&&(++timeout<50000))  //1/2 second
       delay_us(10);
    if(kbhit)
       return(getc());
    else
    {
       timeout_error=TRUE;
       return(0);
    }
}
```

9.7. I²C HABERLEŞMESİ

I²C popüler bir bus ile donanım arasında iki-telli haberleşmedir. I²C bus, bir master ve herhangi bir numarada slave'e sahiptir. Master herhangi bir slave'den veri alabilir veya gönderebilir. Her slave belli bir adrese sahiptir.

I²C'nin telleri SCL ve SDA olarak adlandırılır. İkisini de pull-up direnci (1-10K) gerekmektedir. (+5 V)

İlk baytın LSB biti veri transferinin yönünü belirler. Veri transfer edilir ve alıcı her baytı doğrular. Alıcı SCL'yi azaltarak transfer hızını azaltabilir. Aşağıdaki örnek 10. adresteki slave'e 3 bayt yollar ve sonra 1 baytını okur;

```
#include <16C74.h>
#fuses XT, NOWDT
#use delay(clock=4000000)
#use I2C(master, SCL=PIN_B0, SDA=PIN_B1)
main()
{
    int data;
    I2C_START();
    I2C_WRTIE(10);
    I2C_WRTIE(1);
    I2C_WRTIE(2);
    I2C_WRTIE(3);
    I2C_STOP();
    I2C_START();
    I2C_WRTIE(11);
    data=I2C_READ();
    I2C_STOP();
}
```

9.8. SPI HABERLEŞMESİ

I^2C gibi, SPI 2 veya3 telli donanım veya standart aletlerindeki haberleşmedir.1 tel clock'u sağlar diğeri ise veri alır veya yollar. Eğer 3. tel kullanılıyorsa Enable veya Reset olarak kullanılır. Aşağıdaki kod 10 bitlik komut yollamaktadır. İlk olarak MSB biti yollanır.

```
        main()
        {
            long cmd;
            cmd=0x3e1;
            for(i=1;i<=6;i++)           //left justify cmd
                shift_left(cmd,3,0);
            output_high(PIN_B0);        //enable device
                      //send out 10 data bits each with a clock
pulse
            for(i=0;i<=10;++i)
            {
                output_bit(PIN_B1, shift_left(cmd,2,0));
                output_high(PIN_B2);    //B2 is the clock
                output_low(PIN_B2);
            }
                output_low(PIN_B1);     //disable device
                output_low(PIN_B0);
        }
```

Aşağıdaki C kodu 8 bitlik bir cevabı okur.

```
main()
{
    int data;
    output_high(PIN_B0);        //enable device
                          //send a clock pulse and
                          //read a data bit eight times
    for(i=0;i<=8;++i)
    {
        output_high(PIN_B2);
        output_low(PIN_B2);
        shift_left(&data,1,input(PIN_B1));
    }
        output_low(PIN_B0);     //disable device
}
```

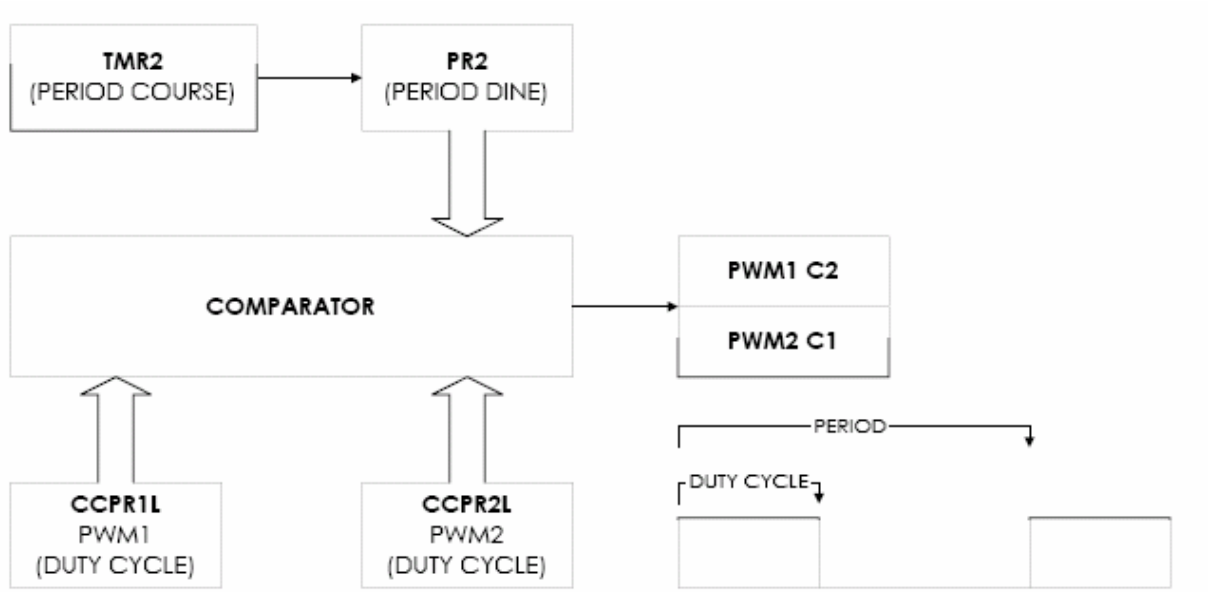
Son iki örnek PİC micro MCU'daki veriyi içeri ve dışarı kaydırır. Kolaylıkla bu kodlar başka aletler modifiye edilebilir. Yavaşlatıcı kısımlar gecikmeye ihtiyaç duyarlar.

Bazı Pic'lerde SPI donanım olarak mevcuttur. Aşağıdaki örnek donanımı yazma ve okumak için kullanılmaktadır.

```
main()
{
    int data;
    setup_spi(SPI_MASTER | SPI_H_TO_L | SPI_CLK_DIV_16);
    output_high(PIN_B0);
    spi_write(3);
    spi_write(0xE1);
    output_low(PIN_B0);
    output_high(PIN_B0);
    data=spi_read(0);
    output_low(PIN_B0);
}
```

Not: Donanım içerisinde yapılanlar kısıtlıdır. Mesela yukarıdaki kod 10 değil 16 bit yollar. Çoğu SPI aletleri ilk bit gelene kadar diğer bitleri reddeder.

9.9. PWM ÜRETECİ



PİC micro MCU'nun donanımdaki PWM üretecinin registerleri kurulur. Daha sonra sabit yazılım olmadan da PWM kendi başına devam edebilir.

PWM değerlerini hesaplanması basit bir tablo ile kolayca yapılabilmektedir.

Örnek:

PWM setup- frekans = 600Hz M/S oranı = 1/1

Ön seviyelendirme değeri= ((1/PWM frekansı)/ önseviyelendirme değeri*(4/OSC frekansı))-1

PWM rezolüsyon = (log(OSC frekansı/PWM frekansı))/log 2

Yukarıdaki örnek sonucunda, 4MHz'lik OSC ve /16 ön seviyelendirmeli 600 Hz lik PWM, 103,2 PR2 registerine yükler ve rezolusyonu 12,7 bittir.

```
setup_timer_2(mode, period, postscale)
```

ilk 2 harf timer 2 nin kısaltmasıdır

```
T2_DISABLED
T2_DIV_BY_1
T2_DIV_BY_4
T2_DIV_BY_16
```

set_pwm1_duty(value)

Bu kod pwm modeline 10 bit değerini yazar

set_pwm2_duty(value)

Eğer sadece 2 bit yollanırsa 2 LSB ler reddedilir.

set_ccp1(CCP_PWM), set ccp2(CCP_PWM)

bu fonksiyon pwm modelinin içinde ccp'yi oluşturur.

Örnek:

```
setup_ccp1(CCP_PWM);    //sets up for pwm
setup_timer_2(T2_DIV_BY_4, 140, 0);
                        //140 as offset, timer 4
set_pwm1_duty(70);
                  //50% duty cycle, i.e. half of 140
```

Eğer osilatör 4MHz olursa potansiyel rezolüsyonu 11 bit frekansı da 1.773KHz olur. Aşağıdaki örnek 2 tondan birini üretmektedir.Frekans aşağıdaki kod ile kurulur:

```
setup_timer_2(T2_DIV_BY_4, 100, 0);
```

duty oranı ise aşağıdaki kod ile kurulur:

```
set_pwm1_duty(50);
```

2 ton frekansı 2.475 KHz ve 996Hz'dir.Süre basit bir gecikme ile set edilir.

```
void sound_bell(byte y)
{
    setup_ccp1(CCP_PWM);    //configure CCP1 as a PWM
    if(y==2)
    {
        setup_timer_2(T2_DIV_BY_4, 100, 0);
        set_pwm1_duty(50);  //make this value half of tone
        delay_ms(200);      //0.2 second bell from
terminal
    }
    else
    {
        setup_timer_2(T2_DIV_BY_4, 250, 0);
        set_pwm1_duty(125); //make this value half of tone
        delay_ms(300);      //0.2 second bell from
terminal
    }
    set_pwm1_duty(0);
}
```

Not:PIC micro MCU'nun PWM donanımı olmadan yazılımı çalışamaz fakat usart'ta çalışır. Örneğin;

```
while(true)
{
   output_high(PIN_B0);
   delay_us(500);
   output_low(PIN_B0);
   delay_us(500);
}
```

bu kod 1Khz'lik kare dalga üretir.

9.10. LCD DİSPLAY SÜRÜCÜ

PIC micro MCU içinde dizayn aşamasında en önemli arayüzlerden bir tanesi Hitachi kontrolöre bağlı olan LCD displaydir. Operasyonun 2 modu vardır.4 ve 8 teller yazma/gecikme veya yazma/kontrol işlemleri display'e veri gönderilirken yapılır.Bu onun bir avantajıdır.

Tipik bir arayüz devresi aşağıda gösterildiği gibidir.PIC arayüzleri 4 bitlik moddadır.Register seçme(RS) display içerisindeki kontrol ve veri register banklarının yerini değiştirir ve the Enable(E) display içerisinde strobe verisi olarak kullanılır.

Kod şöyledir;

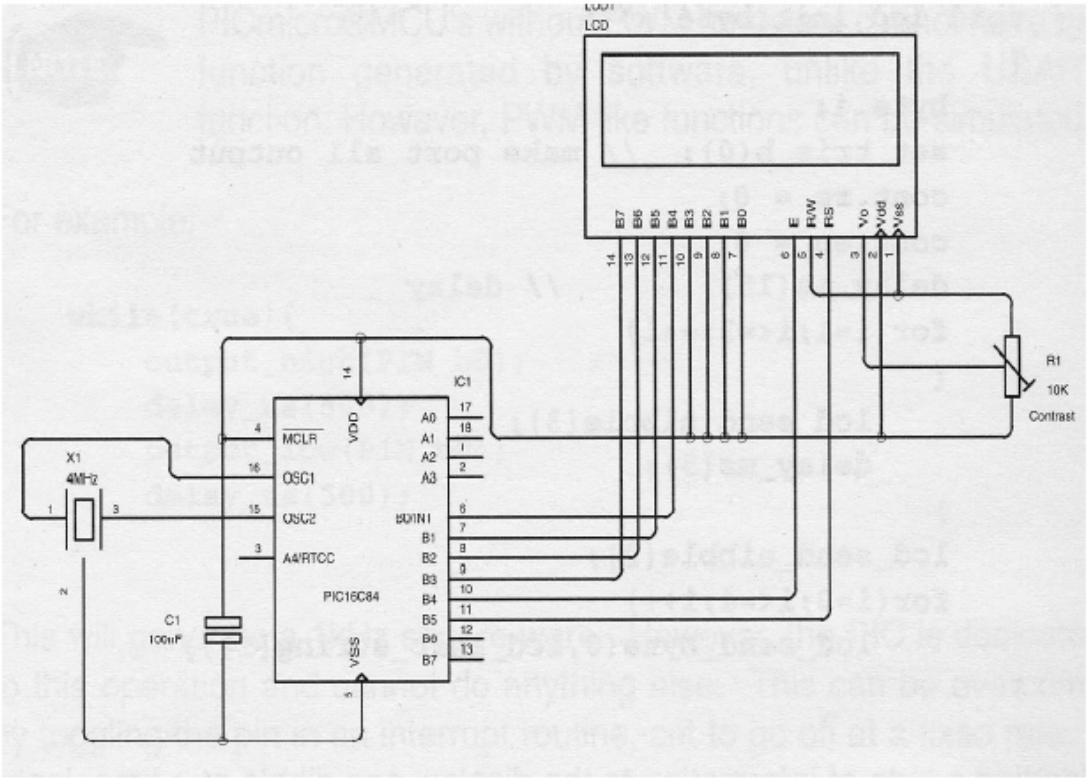
```
byte CONST LCD_INIT_STRING[5] =
                        {0x28, 0x06, 0x0c, 0x01, 0x80};
                        //send to the LCD to start it up
                        //see data sheet
void lcd_init(byte x)
 {
    byte i;
    set_tris_b(0);   //make port all output
    cont.rs = 0;
    cont.en = 0;
    delay_ms(15);
    for(i=1;i<=3;++i)
    {
       lcd_send_nibble(3);
       delay_ms(5);
    }
    lcd_send_nibble(2);
    for(i=0;i<=4;i++)
       lcd_send_byte(0,lcd_init_string[i]);
 }
```

Display'e bilginin baytı şu şekilde gönderilir;

```
       void lcd_send_byte(byte address, byte n)
       {
          cont.rs = address;
          delay_cycle(1);
          delay_cycle(1);
          cont.en = 0;
          lcd_send_nibble(n >> 4);   //shift upper nibble 4
places
          lcd_send_nibble(n & 0xf);  //mask off upper nibble
          delay_ms(10);
       }
```

lcd_send_nibble fonksiyonu strobe fonksiyonudur.

```
void lcd_send_nibble(byte n)
{
   lcd_data  = n;       //place data on port
   delay_cycle(1);      //delay 1 cycle
   cont.en = 1;         //set enable high
   delay_us(2);         //delay 2us
   cont.en = 0;         //set enable low
}
```



LCD displaylerden bahsederken kullanılan başka bir fonksiyon scroll'dur.Bu sağ/sol scroll inşası değildir.Ama yazılım olarak scroll up fonksiyonu vardır.Aşağıdaki örnekte 4 satırlık displayin en üstteki satırı en son gönderilir.

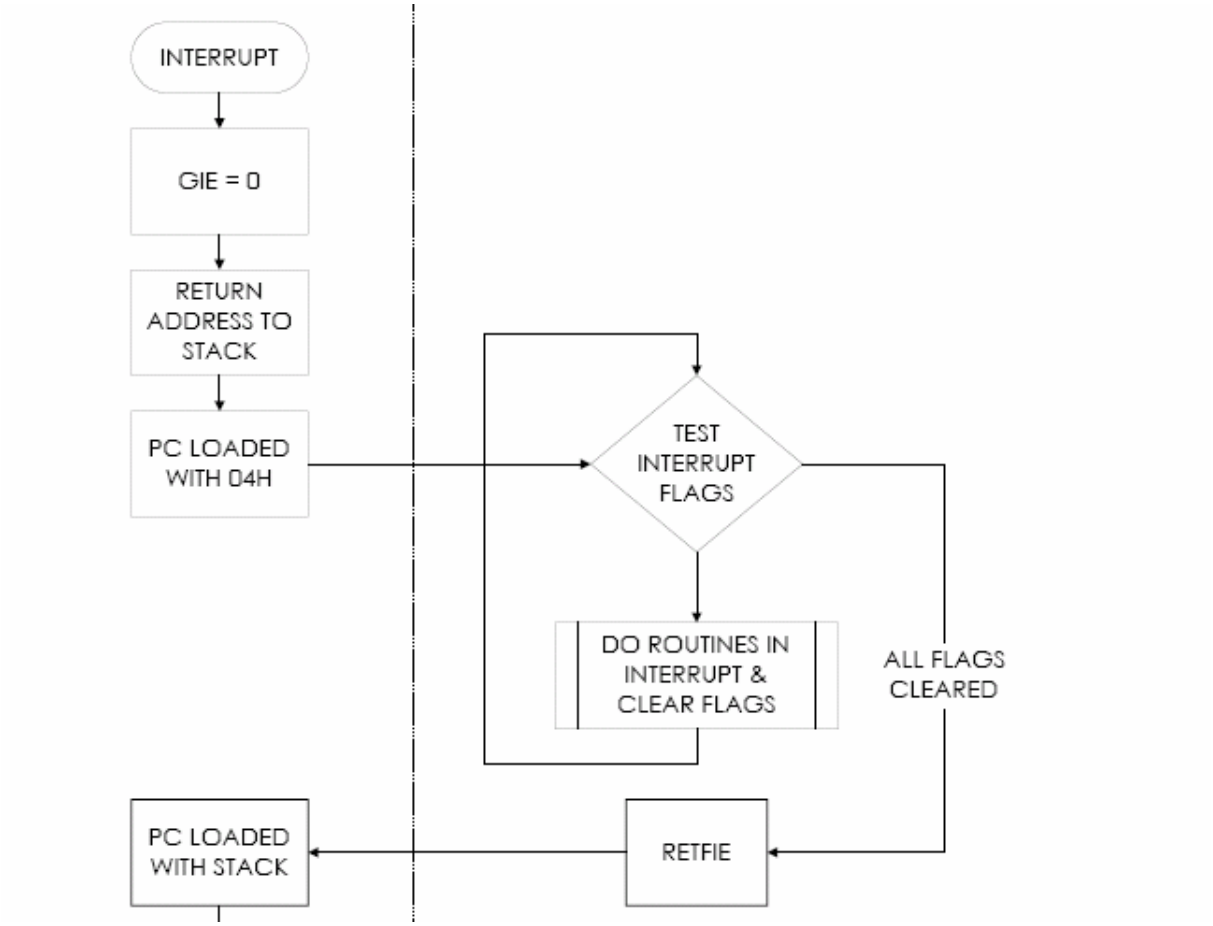
```
        byte Line1[0x10];       //setup arrays to hold data
        byte Line2[0x10];
        byte Line3[0x10];
        byte Line4[0x10];
        memcpy(Line1, Line2, 0x10);       //transfer data from
one
        memcpy(Line2, Line3, 0x10);       //array to the next
        memcpy(Line3, Line4, 0x10);       //and again
        memset(Line4, 0x20, 0x10);        //and clear bottom line
        lcd_same_line(0);                 //go to start of
display

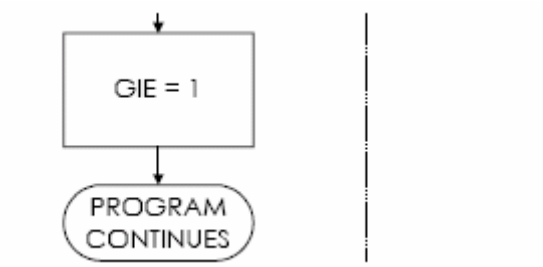
        for(i=0;i<==0x0f;++i)
        {
            k = Line1[i];               //send data from line1
            lcd_send_byte(1,k);         //memory array to display
        }       //repeat for other 3 lines
```

9.11. KESMELER

Kesmeler harici kaynaklardan veya PIC micro MCU içerisindeki kaynaklardan gelmektedir.Kesme meydana geldiğinde pic şu yolu izler.

DONANIM AKIŞI YAZILIM AKIŞI





Donanım kısmı pic micro MCU'nun kontrolündedir,yazılım ise tamamen kendi sorumluluğumuzdadır.Kesme meydana geldiğinde kaynak bir tane veya daha fazla kesme isterse ilk adım olarak test edilir ve belirlenir.

Hangi PIC çeşidinin dizaynda kullanılacağı kesme numarası ve tipine göre değişir.PIC16C5X serisinde kesme yoktur ve yazılım bu ürüne özeldir.Bazı kesme kaynakları gösterildiği gibidir.

#priority kesme priority'sini set eder.

```
#priority rtcc, rb, tmr0, portb
```

#int_globe kendi kesme işlemimizi oluştururken kullanılır.

#int_default Kesme bayraklarına bakarak yanlış kesmenin olup olmadığına bakar.

```
#int_rda          //enable usart receive interrupt
rs232_handler()  //interrupt driven data read and store
{
   b=getch();    //load character
   Buffer[Buff+1]=b;   //store character
   Buff++;             //increment pointer
}
enable_interrupts(level);
disable_interrupts(level);
```

Bu fonksiyonlar kesme bayraklarını kendileri set veya clear ederler böylece kesmeler program süresince meydana gelir veya gelmez.

```
ext_int_edge (edge);
```

Bu harici kesme meydana geldiğinde PORT B'nin 0. bitindeki gelen polarity'i seçer.

Bu örnek Usart üzerinden karakter alımını göstermektedir.Bu fonksiyon LCD display tarafından meydana getirilir,karakterler elle yapıldığından daha hızlı şekilde alınır.

```
#int_rda          //enable usart receive interrupt
rs232_handler()  //interrupt driven data read and store
{
   b=getch();    //load character
   Buffer[Buff+1]=b;   //store character
   Buff++;             //increment pointer
}
main()
{
   enable_interrupts(INT_RDA);
   enable_interrupts(GLOBAL);
   do { while(True);   }
```