

T.C.  
FIRAT ÜNİVERSİTESİ  
TEKNOLOJİ FAKÜLTESİ  
YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ

# **YMH325 MİKROİŞLEMCİLER VE PROGRAMLAMA**

(Ders Notları)

Prof. Dr. İbrahim TÜRKOĞLU

**ELAZIĞ – 2021**

# İÇİNDEKİLER

## BÖLÜM 1. GİRİŞ

## BÖLÜM 2. MİKROİŞLEMCİ VE MİKROBİLGİSAYARLAR

- 2.1. Bilgisayar Mimarisi
- 2.2. Mikroişlemciler ve Mikrobilgisayarlar
- 2.3. Mikroişlemci Yapıları: 8,16,32-bitlik Mikroişlemciler
- 2.4. Mikrobilgisayarların Gelişimi
- 2.5. Mikrobilgisayar Seçimi

## BÖLÜM 3. MİMARİLER

- 3.1. Mikrobilgisayar Tasarım Yapıları: Neuman Mimarisi, Harvard Mimarisi
- 3.2. Mikroişlemci Komut Tasarım Mimarileri: CISC, RISC, EPIC, DSP

## BÖLÜM 4. BAŞARIM ÖLÇÜTLERİ

- 4.1. Başarım Tanımı
- 4.2. Ölçme Koşulları ve Ölçme Birimleri
- 4.3. Başarım Ölçütleri: MIPS, MFLOPS, BENCHMARK

## BÖLÜM 5. MİKROİŞLEMCİ PROGRAMLAMA

- 5.1. MC6802 Mikroişlemcinin Yapısı ve Kayıtları
- 5.2. MC6802 ile Gerçekleştirilmiş Mikrobilgisayar
- 5.3. Assembly Dil Kuralları
- 5.4. MC6802 Mikroişlemcisinin Komut Kümesi ve Adresleme Modları
- 5.5. MC6802 Assembly Uygulamaları  
Program Yazma, Örnek Programlar, Alt Program, Kesme Alt Programı
- 5.6. Mikrobilgisayarlarda Giriş/Çıkış İşlemi: PIA, ACIA, SSDA, PTM (timer)

## BÖLÜM 6. MİKROBİLGİSAYAR PROGRAMLAMA

- 6.1. Mikrobilgisayarlar Kartları
- 6.2. Arduino Mikrobilgisayar Kartları
- 6.3. Arduino Mikrobilgisayar Kartlarında Program Yazma
- 6.4. C ile Arduino Programlama

## BÖLÜM 7. UYGULAMA GELİŞTİRME (*Laboratuvar*)

- Uygulama 1. Mikroişlemci Emülatör Programının Kullanımı
- Uygulama 2. Komut Kümesinin Kavratılması
- Uygulama 3. Adresleme Modları
- Uygulama 4. Program Geliştirme Uygulamaları-I
- Uygulama 5. Program Geliştirme Uygulamaları-II (copy, del)
- Uygulama 6. Program Geliştirme Uygulamaları-III
- Uygulama 7. Çevrimiçi Mikrobilgisayar Uygulaması Geliştirme (Circuits, Tinkercad) Programının Kullanımı
- Uygulama 8. Dijital Giriş / Çıkış Uygulaması-I (led kontrolü)
- Uygulama 9. Dijital Giriş / Çıkış Uygulaması-II (buton ile led kontrolü)
- Uygulama 10. Dijital Giriş / Çıkış Uygulaması-III (RGB led kontrolü)
- Uygulama 11. Dijital Giriş / Çıkış Uygulaması-III (buton ile RGB led kontrolü)
- Uygulama 12. Sensör Uygulaması-I (sıcaklık ölçümü)
- Uygulama 13. Sensör Uygulaması-II (mesafe ölçümü)
- Uygulama 14. Sensör Uygulaması-III (sesli ikaz)

## BÖLÜM 8: PROJE (*Laboratuvar*)

Mikrobilgisayar kartı (arduino, raspberry, vs.) ile gömülü sistem uygulaması geliştirme

## BÖLÜM 1. GİRİŞ

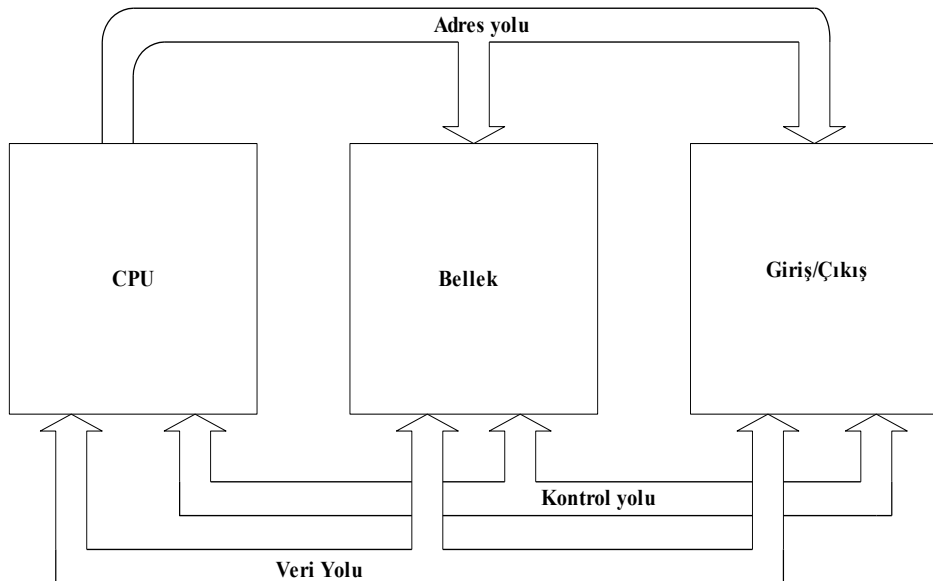
Günümüzde hızla gelişen teknoloji bilgisayarla kontrol edilen cihazları bizlere çok yaklaştırdı. Öyle ki günlük hayatımızda sıkça kullandığımız bir çok elektronik cihaz (cep telefonu, faks, oyuncaklar, elektrikli süpürge, bulaşık makinası, vs.) artık çok küçük sayısal bilgisayarlarla (mikro denetleyiciler) işlev kazandırılabilir. Benzer işler, ilk zamanlarda mikroişlemci tabanlı bilgisayar kartları ile yapılabilir. Mikroişlemci ile bir cihazı kontrol etme işlemi Giriş/Çıkış ve hafıza elemanı gibi ek birimlere ihtiyaç duyar. Böylesi bir tasarım kolay olmamakla birlikte, maliyet ve programlama açısından da dezavantajlara sahiptir. İşte mikrodenetleyiciler bu sorunları ortadan kaldırmak ve bir çok fonksiyonu tek bir entegrede toplamak üzere tasarlanmış olup, günümüzde hemen hemen bir çok elektronik cihazda farklı tipleri bulunmaktadır.

Mikroişlemci, saklı bir komut dizisini ardışıl olarak yerine getirerek veri kabul edebilen ve bunları işleyebilen sayısal bir elektronik eleman olarak tanımlanabilir. Mikroişlemci temelde mantık kapıları, flip-floplar, sayıcı ve saklayıcılar gibi standart sayısal devrelerden oluşur.

Genel olarak bilgisayar ile iki şekilde ilgilenilir :

1. Yazılım (Software) : Bilgisayarın fiziksel parçalarını işler hale getiren bileşenlerdir.
2. Donanım (Hardware) : Bilgisayarı oluşturan fiziksel parçaların tümüdür.

Her ikisi de birbirinin tamamlayıcısıdır. Birisi olmazsa diğeri de olmaz. Sistem öncelikli olarak tasarlanırken önce sistemi meydana getirecek elemanlar ,yani donanım parçaları göz önüne alınır. Daha sonra yazılım bu yapıya bakılarak yazılır. Yazılım, donanımın hangi yönetime göre nasıl çalışacağını gösteren bir sanal uygulamadır. Hangi zamanda hangi elemanın devreye girerek üzerindeki bilgiyi işlemesini sağlamaktadır. Basit bir bilgisayarın ana elemanları Şekil 1.1.'de görülmektedir. Tüm sayısal bilgisayarlar şekilde gösterilen elemanlara sahiptirler. Bunların dışındaki eleman ya da cihazlar seçimlidir.



Şekil 1.1: Genel Bilgisayar yapısı

Bilgisayarı oluşturan bu sistemdeki elamanlar; mikroişlemci (CPU), bellek ve giriş/çıkış (G/Ç) birimleridir. Mikroişlemcinin işleyeceği komutlar ve veriler geçici veya kalıcı belleklerde tutulmaktadır. Bilgiyi oluşturan komut ve veriler bellekte karmaşık veya farklı alanlarda tutulabilir. Yazan kişinin karakterini veya seçtiği yolu gösteren çeşitli algoritmalarından meydana gelen program işlemciyi kullanarak verilerin işlenmesini sağlar. Bilginin işlenmesi sırasında ortaya çıkabilecek ara değerler ,en sonunda sonuçlar bellekte bir yerde depolanmak zorundadır.Bütün bu yapılan işlemler bir hesaba dayanmaktadır. Bilgisayarın bilgiyi işlemedeki ana karar vericisi sistemin kalbi sayılan mikroişlemcidir. CPU tarafından gerçekleştirilen iki temel işlem vardır.Birincisi komutların yorumlanarak doğru bir sırada gerçekleşmesini sağlayan kontrol işlevi, diğeri toplama, çıkarma vb özel matematik ve mantık işlemlerinin gerçekleştirilmesini sağlayan icra işlevidir.

Bilgisayarda çalıştırılan yazılımlar kendi aralarında ikiye ayrılır. Bunlar, programcı tarafından yüksek düzeyde yazılan programlardır ki insanlar tarafından anlaşılabilir düzeydedir ve bu yazılan programların makine tarafından anlaşılmasını sağlayan bağdaştırıcı (interface) yazılımlardır ki işletim sistemi (OS) olarak anılırlar. Mikroişlemci mantıksal 0 ve 1 esasına göre çalıştığından,verilen komutların da bu esasa dayanması gerekmektedir. Kısaca sayısal bilgisayarların kullandığı doğal dile makine dili denir. Programcı tarafından yüksek düzeyde yazılan programlar ancak yine insanlar tarafından anlaşılabilir. Bu programların makine tarafından anlaşılabilmesi için derleyici,yorumlayıcı ve assembler gibi aracı programların kullanılması gerekir. Demek ki, yazılım denildiğinde akla, işletim sistemi, üst düzey diller vasıtasıyla yazılan çeşitli uygulama programları gelir. Bu diller;

- Yüksek seviyeli diller
- Orta seviyeli diller
- Düşük seviyeli diller

olmak üzere üç sınıfa ayrılabilir. Bu yüksek, orta, düşük kelimelerinin anlamı donanımın yazılıma ne kadar yakın olduğunu gösterir.

Yüksek seviyeli dillerin kontrol sistemlerinde kullanımı zordur. Yüksek seviyeli bir dilde yazılan program derleyici tarafından derlendiğinde bilgisayar bunu düşük seviyeli dile (makina diline) çevirerek anlar. Orta seviyeli dillerin (assembly) kontrol sistemlerinde kullanımı uygundur.

Assembly dilini kullanırken donanımı bilmemiz zorunludur. Örneğin Intel 8085 ve Motorola 6800 mikroişlemcilerinin assembly dilleri farklıdır. Çünkü donanımları farklıdır. Orta seviyeli diller makina dilinde, yani ikili sayı sistemi ile program yazma zor ve zahmetli bir iştir. Bunun için makina dilinin komutlar şeklinde verilmesini sağlayan assembly diller geliştirilmiştir. Assembly

dilinde program yazmak makina diline göre daha kolay ve anlaşılırdır. Fakat fazla miktarda komut içerir. Bunun için anlama ve kullanımı belli bir zaman alır.

Assembly makinaya yönelik dillerdir. Programcı kullandığı bilgisayarın donanımını ve adresleme tekniklerini çok iyi bilmelidir. Assembly programları standart değildir. Aynı model olmayan her mikroişlemcinin kendine özgü assembly dili vardır. Programcı bu dille makinayla en basit şekilde iletişim kurar. Assembly dilinde yazılan her program bellekte saklanırken veya işlenirken 0 veya 1'ler formuna çevrilmeye gerek duyar. Bu çevirme işi programcı tarafından üretici firmanın databook kitabına bakılarak elle veya bir assembler (Assembly derleyicisi) yardımıyla yapılır.

Tek tek komut kodu karşılığına bakılarak ikili komut kodları bulunuyorsa ve eğer program çok uzun veya tekrarlamalı ise ,kaynak programı amaç programa çevirmek çok zor ve hata yapma payı yüksek olacaktır. Bu gibi durumlarda iyi bir assembler programı kullanılmalıdır.

Bazen programcılar Assembly dili ile assembleri karıştırmaktadırlar. Assembly dili, konuşma dilinde emir şeklindeki cümleden özenle seçilerek alınmış ve sayısı genelde üç en fazla dört olabilen harflerden meydana gelen ve bir komut anlam ifade eden hatırlatıcıları içerir.

Assembly dilinde program yazmak makine dilinde yazmaktan daha kolay ve takibi daha basittir. Fakat bu programın belleğe konulmadan önce makine diline çevrilmesi gereklidir,işte bu işi assembler denilen (bir nevi paket programda denilen) çevirici program yapar.Bu çevirme işlemine kaynak programın amaç programa çevrilmesi denir.

Assembly dilinde yazılmış bir programın amaç programa çevirmede en çok kullanılan yöntem elle yapılan işlemdir. Bu yöntemde her satırdaki hatırlatıcıya karşılık gelen kodlar üretici firma tarafından yayınlanan databook'a bakılarak bulunur. Böylece amaç program bulunmuş olur.

### **Assembly Dilinin Dezavantajları**

- Assembly dilinde bir program yazmak için üzerinde çalışılan bilgisayarın özellikleri hakkında detaylı bilgi sahibi olunmalıdır. Mesela bunlar, bilgisayar mikroişlemcisinde bulunan kaydediciler ve sayısı, komut kümesi ve adresleme türleri gibi değişik özelliklerdir.
- Assembly dilinin diğer bir mahsuru elastiki olmamasıdır. Değişik firmalarca üretilen her mikroişlemcinin kendisine has bir programlama dili olmasıdır. Bundan dolayı bir mikroişlemci için yazılan bir assembly dilindeki program diğer bir mikroişlemcide çalışmayabilir.

### **Assembly Dilinin Avantajları**

- Assembly dilinde program yazarlar, donanımın çalışmasını çok iyi anlamak ve ona göre iyi programlar geliştirmek zorunda olduklarından kendilerine birçok kazanımlar sağlarlar. Yüksek düzeyli dillerde program yazarken bilgisayar donanımının görünmeyen bazı yanlarına assembly dilinde sahip olunur.
- Assembly dilinde yazılan programlar yüksek düzeyli dillerle yazılan programlara nazaran daha hızlı ve küçük boyutludur. Assembly dili, program büyüklüğünde ve çalışma hızında ideal optimizasyon sağlar.

**Düşük seviyeli diller** ise makina dilleridir. Yine makinaya özgü bir dildir. Bu dilde programlama çok zor, hata yapma oranı çok yüksek ve programı kontrol etme imkanı nerede ise yoktur.

Assembly ve makina diline uygun uygulamalar:

- Hesaplamalardan daha çok giriş/çıkış gerektiren uygulamalar
- Gerçek zaman denetimi ve uygulamaları
- Fazla veri işlemesi gerekmeyen uygulamalar
- Hızlılık istenen uygulamalar

## BÖLÜM 2. MİKROİŞLEMCİ VE MİKROBİLGİSAYARLAR

### 2.1. BİLGİSAYAR MİMARİSİ

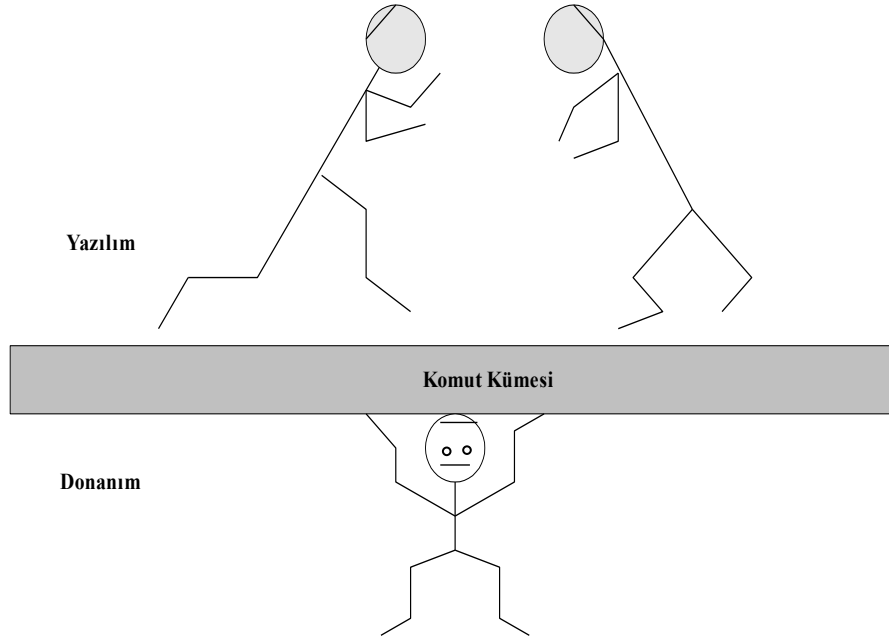
Bilgisayar mimarisi, komut kümesinin, donanım elemanlarının ve sistem organizasyonunun dahil olduğu bir bilgisayarın tasarımıdır. Mimari iki farklı yaklaşımla tanımlanmaktadır:

- ISA - Komut kümesi mimarisi
- HSA - Donanım sistem mimarisi

ISA, bir bilgisayarın hesaplama karakteristiklerini belirleyen komut kümesinin tasarımıdır.

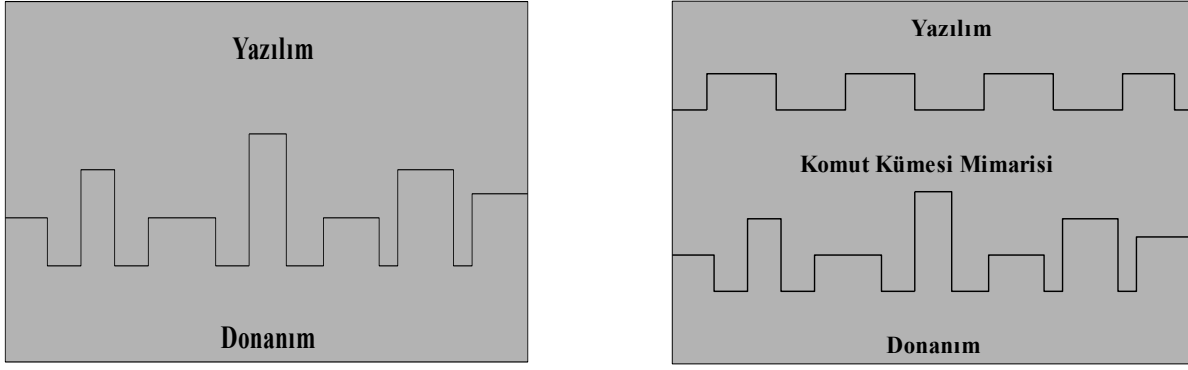
HSA; CPU, depolama ve G/Ç sistemlerinin dahil olduğu alt sistem ve bunların bağlantı şekilleridir.

Komut kümesinin yazılım ve donanımla ilişkisi Şekil 2.1.'de görülmektedir.



Şekil 2.1.: Komut kümesinin yazılım ve donanımla ilişkisi

Bilgisayar sistemlerinde bütün mesele, bu iki kavramı yerli yerine oturtmaktır. Mimari bir kavram olarak HSA'nın ne olduğu ve hangi elemanlardan meydana geldiği yukarıda açıklanmıştır. ISA ise programcının bu elemanlara yön verecek programı yazması durumunda nasıl bir kabul göreceğidir. Farklı şirketler tarafından üretilen farklı bilgisayarların fiyat/performans açısından elbette farklı mimarileri olabilir. Özel bilgisayar sistemleri (günümüzde bir çeşit oyun konsolları) için programcı kodlarını makinenin doğrudan özel donanımına göre yazmaktaydı. Böylece bir makine için yazılan program aynı firma tarafından üretilse bile, ne rekabet ettiği bir makinasında ne de diğer makinasında çalışabilmekteydi. Mesela, A makinası için yazılan bir oyun B makinasında veya C makinasında çalışmayacaktır. Programcı tarafından yazılan kodlar donanımı açma anahtarı olarak düşünülebilir (Şekil 2.2.).



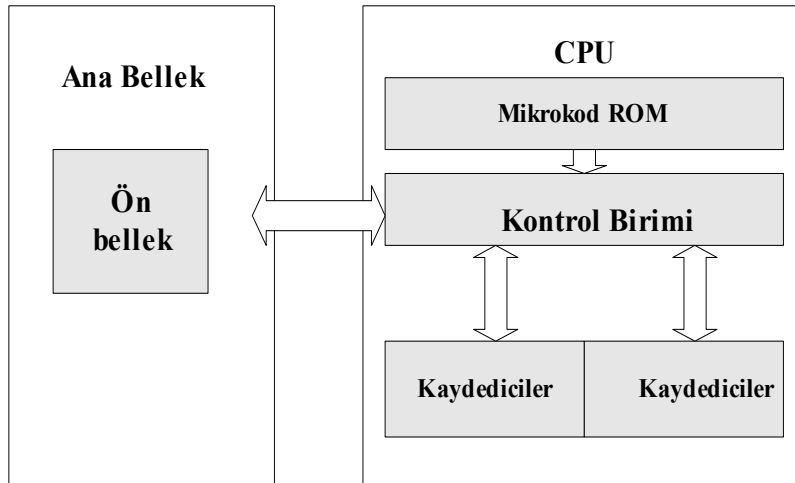
**Şekil 2.2.:** Komut Kümesi mimarisinin yazılım ve donanımla ilişkisi

### Programsal yaklaşım

Bilgisayar sistemlerinde bütün mesele sistemi meydana getiren tüm elemanların bir komutla nasıl devreye sokulacağıdır. Ufak tefek ayrıcalıkları olsa da birbirine benzer yapıdaki bilgisayarlar için farklı programlar yazmak oldukça maliyetli olduğundan, programcının yazdığı komutların her bilgisayar tarafından algılanarak yürütülmesi esas hedeftir. Ortaya atılan ilk çözüm mikrokod yaklaşımı daha sonraları iki standarttan biri olmuştur.

Donanımı devreye sokacak öz bilgilerin yani komut kümesinin yer aldığı bu yere (bölgeye) **mikrokod motoru** denilmektedir. (Şekil 2.3.).

Burası, CPU içinde CPU olarak da ifade edilebilir. Programcının yazdığı kodları işlemcinin daha çabuk anlayabileceği veya çalıştırabileceği küçük mikrokodlara dönüştüren bu mikrokod motoru, işlemci ROM bellek vasıtasıyla yerleştirilmiştir. Mikroprogram ve icra birimi tarafından meydana gelen mikrokod ROM'un görevi, özel komutların bir dizi kontrol sinyallerine çevirerek sistem elemanlarının denetlenmesini sağlar. Aynı zamanda, mikrokod CISC tipi işlemcilerdeki temel işlevi, alt düzey komut kümesiyle programcının çalıştığı üst düzey komutlar arasında soyutlama düzeyi oluşturmaktır.



**Şekil 2.3.:** Bir mikrokod ROM'un sistemdeki yeri



Mikroişlemci üreticileri, sistem tasarımında iki yönlü düşünmek zorundadırlar. Birincisi,mimariyi meydana getiren elemanların işlevleri,ikincisi bu elemanların nasıl devreye sokulacağıdır. Elemanları devreye sokmak için program yazmak gerekecektir. Bu işin bir yanı; diğer yanı ise donanımdır. Donanım ile tasarım mühendisleri ilgilenir. Fakat programcı öyle bir program yazmalı ki, sistem tarafından algılanarak doğru zamanda doğru eleman devreye sokulabilsin. Donanım mimarisini programcıya aktaracak en iyi yol ona kullanabileceği komut kümesini hazır vermektir. Bilgisayar sisteminin donanımsal tüm özelliklerini içeren sisteme **komut kümesi mimarisi** denildiğine göre, programcı bu kümeye bakarak veya bu kümeyi kullanabilen derleyicileri kullanarak hiçbir endişeye gerek duymaz. Programcının yazdığı bir komut işletildiğinde, mikrokod ROM bu komutu okur ve sonra o komuta karşılık gelen uygun mikrokodları yükler ve çalıştır.

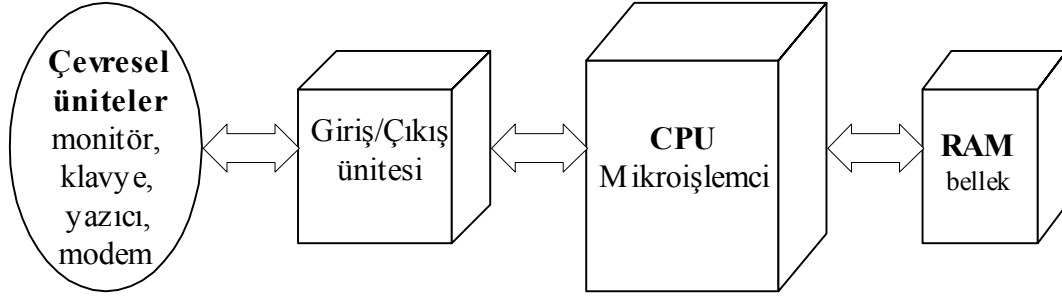
### **Donanımsal Yaklaşım**

Mikrokod kullanılarak ISA sisteminin yürütülmesinin başlıca sakıncası başlangıçta komutların doğrudan çalıştıran sisteme göre yavaş olmasıdır. Mikrokod, ISA tasarımcılarına programcının ara sıra kullandığı her çeşit komutların komut kümesine eklenmesini ister. **Daha çok komut demek daha fazla mikrokod, çekirdek büyüklüğü ve güç demektir.** ISA mimarisinin yaşanan aksaklıklarından dolayı daha sonraları ,komutların doğrudan donanım elemanları tarafından yorumlanarak sistemin denetlendiği diğer bir mimari yaklaşımda donanımsal çalışma modelidir..Komutların anlaşılır standart bir boyuta getirilerek çalışıldığı sisteme RISC modeli denilmektedir.Yani komutların donanımsal çalışma modeline sahip RISC tipi bilgisayarlarda, komut kümesindeki komutların sayısı azaltılmış ve her bir özel komutun boyutu düşürülmüştür.

## **2.2. MİKROİŞLEMCİLER VE MİKROBİLGİSAYARLAR**

**MİKROİŞLEMCİ**, saklı bir komut dizisini ardışıl olarak yerine getirerek veri kabul edebilen ve bunları işleyebilen sayısal bir elektronik eleman olarak tanımlanabilir. Günümüzde basit oyuncaklardan, en karmaşık kontrol ve haberleşme sistemlerine kadar hemen her şey mikroişlemcili sistemlerle kontrol edilmektedir.

**MİKROBİLGİSAYAR** veya sayısal bilgisayar üç temel kısım (CPU, Giriş/Çıkış Birimi ve Hafıza) ile bunlara ek olarak bazı destek devrelerinden oluşur. Bu devreler en basitten, en karmaşığa kadar çeşitlilik gösterir.



**Şekil 2.4 :** Bir mikroişlemci sisteminin temel bileşenleri

**Giriş / Çıkış (Input / Output) :** Sayısal, analog ve özel fonksiyonlardan oluşur ve mikroişlemcinin dış dünya ile haberleşmesini sağlar.

**CPU (Central Processing Unit – Merkezi İşlem Birimi) :** Sistemin en temel işlevi ve organizatörüdür. Bilgisayarın beyni olarak adlandırılır. Komutları yürütmek, hesapları yapmak ve verileri koordine etmek için 4, 8, 16, 32 ve 64 bitlik sözcük uzunluklarında çalışır.

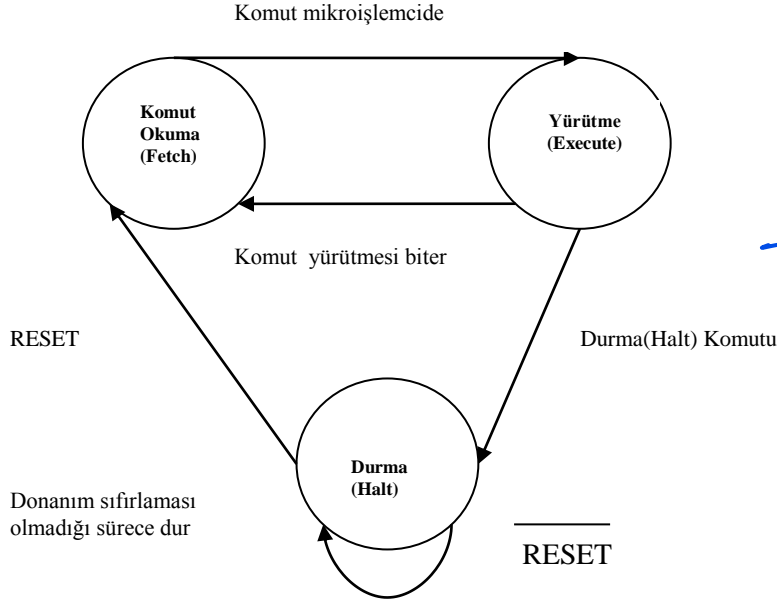
**Hafıza :** RAM, ROM, PROM, EPROM, EEPROM veya bunların herhangi bir birleşimi olabilir. Bu birim, program ve veri depolamak için kullanılır.

**Osilatör :** Mikroişlemcinin düzgün çalışabilmesi için gerekli olan elemanlardan biridir. Görevi; veri ve komutların CPU 'ya alınmasında, yürütülmesinde, kayıt edilmesinde, sonuçların hesaplanmasında ve çıktıların ilgili birimlere gönderilmesinde gerekli olan saat darbelerini üretmektir. Osilatör, farklı bileşenlerden oluşabileceği gibi hazır yapılmış bir modül de olabilir.

**Diğer devreler :** Mikroişlemci ile bağlantılı diğer devreler; sistemin kilitletmesini önlemeye katkıda bulunan *Watchdog Timer*, mantık aşamalarını bozmadan birden fazla yonganın bir birine bağlanmasını sağlayan adres ve veri yolları (BUS) için *tampon (buffer)*, aynı BUS 'a bağlanmış devrelerden birini seçmeyi sağlayan, adres ve I/O için kod çözücü elemanlar (*decoder*).

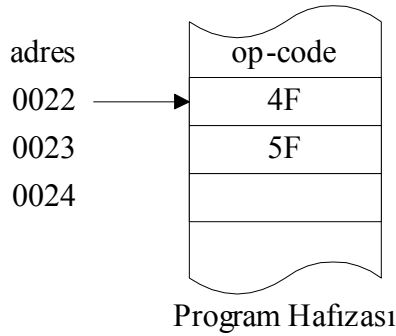
**Mikroişlemcinin Çalışması:** Bir mikroişlemcinin çalışmasında ,kontrol birimi tarafından yerine getirilen ve Şekil 2.5’de gösterilen temel iki işlem vardır. **Komut okuma (fetch)** ve **komut yürütme (execute)**. Komut okuma,mikroişlemcinin hafızadan bir **işlem kodu (operation code-opcode)** alıp **komut saklayıcısına (Instruction Register –IR)** getirme işlemine denir. Komut saklayıcısına gelen komut ile hangi işlemin yapılacağı komut kod çözücüsü tarafından belirlenir.Gereken sinyalleme kontrol birimi tarafından üretilir.Eğer komut ile belirlenen işlem için,bazı **işlem verisine (operand)** gerek var ise,bu veriler hafızadan okunur.

Son olarak komutun yürütülmesi gerçekleştirilir. Bir komutun yürütülmesi bittikten sonra, benzeri şekilde; tekrar komut okuma ve yürütme işlemleri, sonsuz bir çevrim içinde, bir **durma (halt)** komutu yürütülünceye kadar yapılır. Mikroişlemcinin çalışmasının durduran bu komut ile işlemci bir üçüncü duruma girer ve bu durumdan çıkabilmesi için bir donanım sıfırlaması (**reset**) gerekir.



**Şekil 2.5:** Bir mikroişlemcideki komut okuma ve yürütme çevrimleri

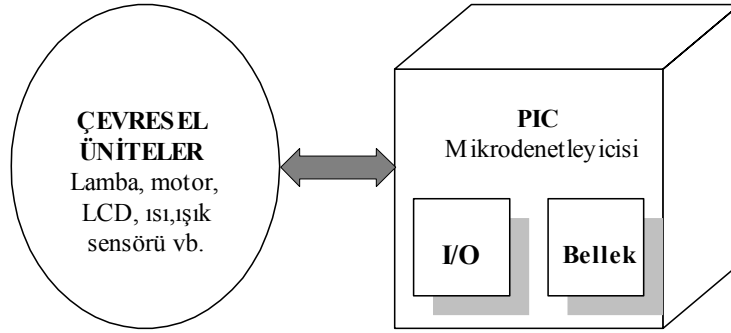
**Örnek:** Bir mikroişlemcinin program hafızasında bulunan programın çalıştırılması



**Şekil-2.6.** Program yüklü bulunan hafıza

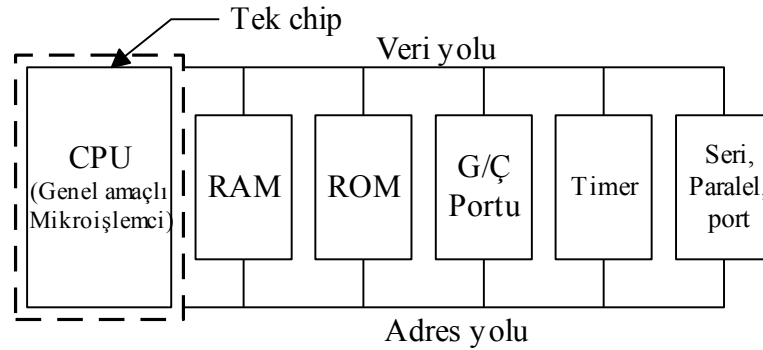
1. Program sayacı o anda çalışan komutun adresini üzerine alır. PC =[0022]
2. Komut kayıtcısı, o anda çalışan komutu üzerinde bulundurur. Ir = 4F
3. Kontrol ünitesi bu komuta göre uygun elektronik sinyalleri uygun yere göndererek komutu çalıştırır. 4F için A akümülatörünü sıfırlayıcı işaret gönderir.
4. Daha sonra PC bir sonraki adrese geçer ve aynı işlemler tekrarlanır.

**MİKROBİLGİSAYAR (MİKRODENETLEYİCİ, MICROCONTROLLER),** bir bilgisayar içerisinde bulunması gereken temel bileşenlerden Hafıza, I/O ünitesinin tek bir chip (yonga) üzerinde üretilmiş biçimine denir.

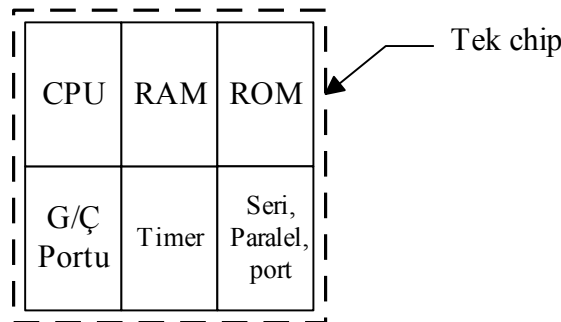


**Şekil 2.7 :** Bir mikrobilgisayar sisteminin temel bileşenleri

Mikroişlemci ile kontrol edilebilecek bir sistemi kurmak için en azından şu üniteler bulunmalıdır; CPU, RAM, Giriş/çıkış ünitesi ve bu üniteler arasında veri/adres alış verişi sağlamak için bilgi iletim yolları (DATA BUS) gerekmektedir. Bu üniteleri yerleştirmek için baskı devre organizasyonu da önemli bir aşamadır. Mikrodenetleyici ile kontrol edilebilecek sistemde ise yukarıda saydığımız ünitelerin yerine tek bir yonga (mikrodenetleyici) kullanmak yeterli olacaktır. Tek bir yonga kullanmak ile, maliyet düşecek, kullanım ve programlama kolaylığı sağlanacaktır. Bu avantajlardan dolayı son zamanlarda bilgisayar kontrolü gerektiren elektronik uygulamalarda gelişmiş mikroişlemci (*Embeded processor*) kullanma eğilimi gözlenmiştir.



a) Genel amaçlı mikroişlemci sistemi



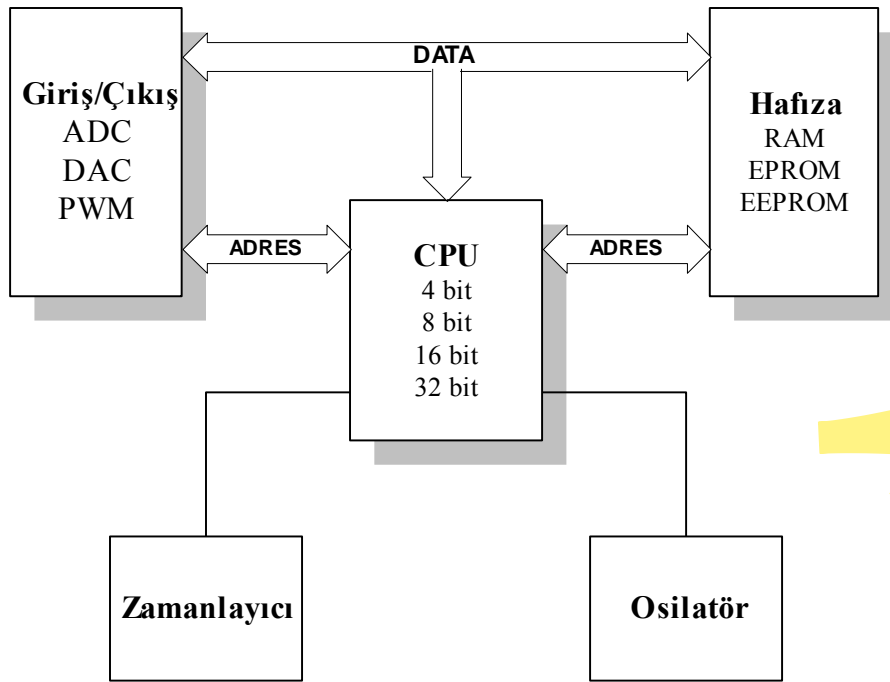
b) Mikrodenetleyici sistemi

**Şekil 2.8.** Mikroişlemcili sistem ile mikrodenetleyici sistemler

Günümüz mikrodnetleyicileri otomobillerde, kameralarda, cep telefonlarında, fax-modem cihazlarında, fotokopi, radyo, TV, bazı oyuncaklar gibi sayılamayacak kadar pek çok alanda kullanılmaktadır.

Mikrodnetleyiciler 1990'lı yıllardan sonra aşğıdaki ihtiyaçlara cevap verebilmek için gelişmeye başlamışlardır. Gelişim sebepleri;

- Karmaşık cihazlar da daha yüksek performansa ihtiyaç duyulması
- Daha geniş adres alanına sahip olması
- C gibi yüksek seviyedeki dillerde programlama desteğinin sağlanması
- Windows altında çalışan gelişmiş özelliklere sahip program geliştirme ortamlarının olması
- Daha az güç tüketimi ve gürültünün olması
- Büyük geliştirme yatırımları ve yazılım güvenliği açısından varolan çeşitli programların kullanılması
- Sistem fiyatlarının ucuz olması

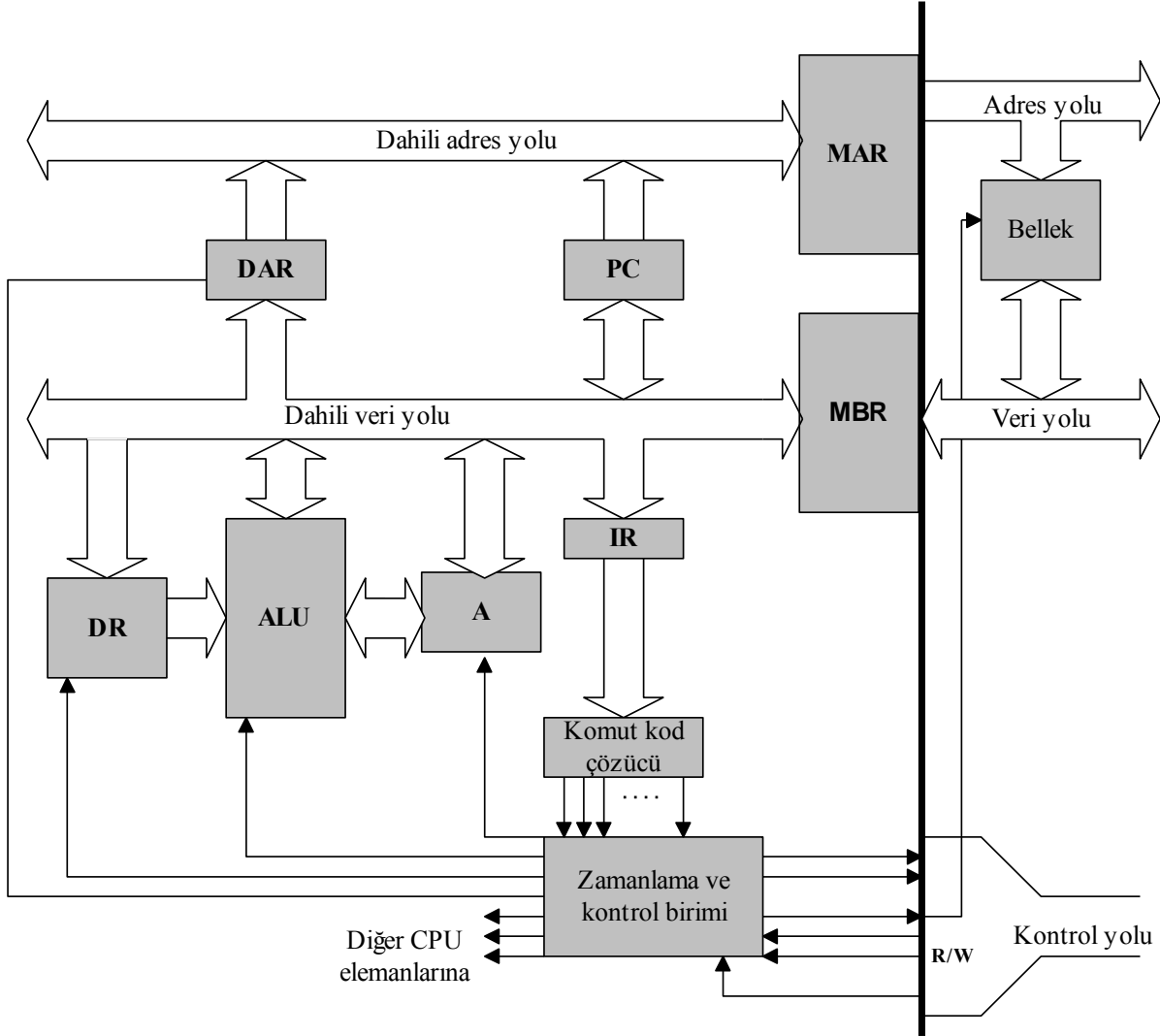


**Şekil 2.9: Mikrobilgisayar sistemi**

Teknolojik gelişmelerle birlikte mikroişlemcilerde zamanla gelişmeye başlamışlardır. Belirli bir sürede ele alınan bit sayısına bakılarak mikroişlemcinin güçlü olup olmadığı belirlenir. Bit uzunluklarına göre 8 bit, 16 bit, 32 bit ve 64 bitlik mikroişlemciler bulunur.

## 2.3. Mikroişlemci Yapıları

**2.3.1. 8-Bitlik Mikroişlemciler:** Basit bir işlemci kaydediciler, aritmetik-mantık birimi ve denetim birimi olmak üzere 3 ana bölümden meydana gelmiştir.

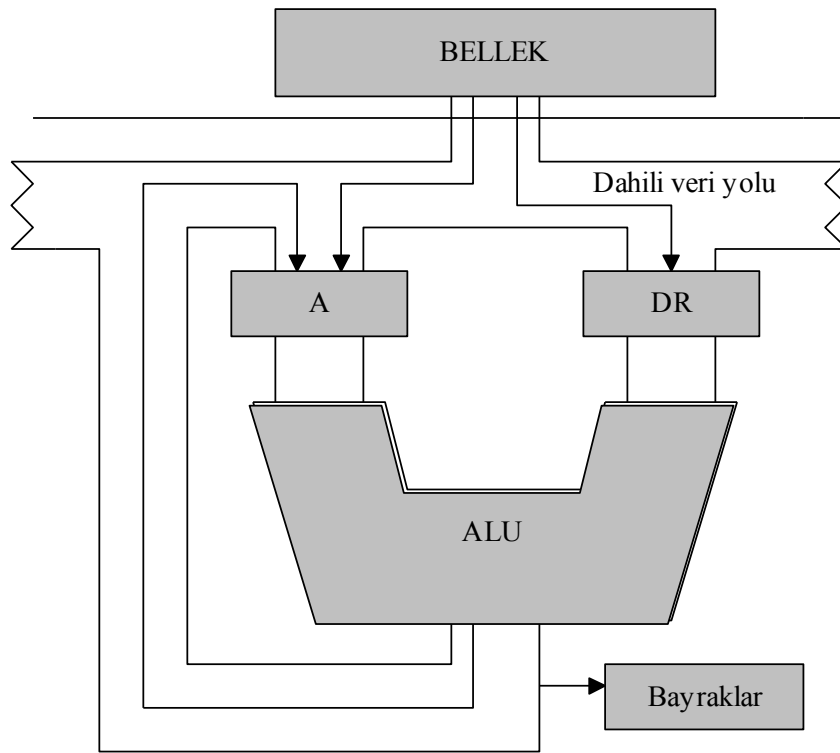


**Şekil 2.10 :** Basit bir 8-bitlik işlemcinin yapısını oluşturan ana birimler

**Kaydediciler:** Flip-floplardan oluşan birimlerdir. İşlemci içerisinde olduklarından belleklere göre daha hızlı çalışırlar. İşlemci çeşitlerine göre kaydedicilerin adı ve tipleri değişmektedir. Kaydediciler genel amaçlı ve özel amaçlı olmak üzere iki grupta incelenmektedir. Genel amaçlı kaydediciler grubuna A, B ve X gibi kaydediciler girer. A kaydedicisi Akümülatör teriminden dolayı bu adı almıştır. İndis kaydedicilerinin görevleri ise; hesaplamalar sırasındaki ara değerlerin üzerinde tutulması, döngülerde sayaç olarak kullanılmasıdır. Özel amaçlı kaydediciler ise; PC (Program Counter, Program Sayacı), SP (Stack Pointer- Yığın İşaretçisi) ve Flags (Bayraklar) verilebilir. Bunların dışında işlemcide programcıya görünmeyen kaydediciler vardır. Bu kaydedicileri alt düzey program yazan programcılar mutlaka bilmesi gerekir. Bunlar; IR (Instruction Register-Komut kaydedicisi), MAR (Memory Address Register- Bellek adres

kaydedicisi), MBR (Memory Buffer Register- Bellek veri kaydedicisi), DAR(Data Address Register- Veri adres kaydedicisi) ve DR (Data register- Veri kaydedicisi) olarak ele alınabilir.

**Aritmetik ve Mantık Birimi:ALU** mikroişlemcilerde aritmetiksel ve mantıksal işlemlerinin yapıldığı en önemli birimdir. Aritmetiksel işlemler denilince akla başta toplama, çıkarma, çarpma ve bölme gelir. Komutlarla birlikte bu işlemleri, mantık kapıları, bu kapıların oluşturduğu toplayıcılar, çıkarıcılar ve flipflopolar gerçekleştirir. Mantıksal işlemlere de AND, OR, EXOR ve NOT gibi işlemleri örnek verebiliriz.

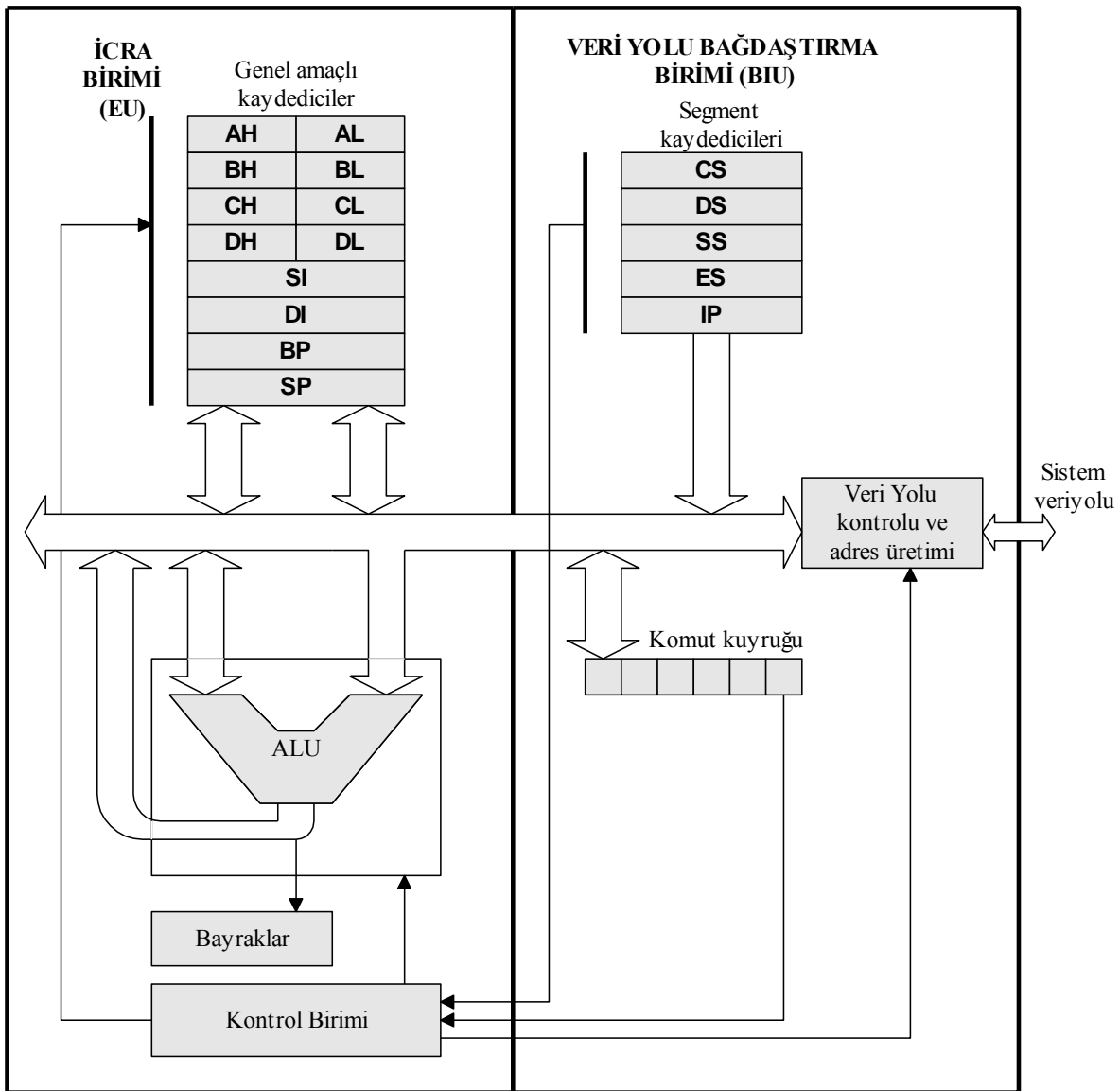


**Şekil 2.11 : Aritmetik ve mantık birimi**

**Zamanlama ve Denetim Birimi:** Bu kısım sistemin tüm işleyişinden ve işlemin zamanında yapılmasından sorumlu olan birimdir. Bu birim bellekte program bölümünde bulunan komut kodunun alınıp getirilmesi, kodunun çözülmesi, ALU tarafından işlenip, sonucun alınıp belleğe yüklenmesi için gerekli olan denetim sinyalleri üretir.

**İletişim yolları:** Mikroişlemci mimarisine girmese de işlemciyle ayrılmaz bir parça oluşturan iletişim yolları kendi aralarında üçe ayrılır. Adres yolu; komut veya verinin bellekte bulunduğu adresten alınıp getirilmesi veya adres bilgisinin saklandığı yoldur. Veri yolu ise işlemciden belleğe veya Giriş/Çıkış birimlerine veri yollamada yada tersi işlemlerde kullanılır. Kontrol yolu ise sisteme bağlı birimlerin denetlenmesini sağlayan özel sinyallerin oluşturduğu bir yapıya sahiptir.

**2.3.2. 16-Bitlik Mikroişlemciler:** 16-bitlik mikroişlemciler basit olarak 8- bitlik mikroişlemcilerde olduğu gibi , Kaydediciler, ALU ve Zamanlama-Kontrol birimine sahiptir. Fakat mimari yapısı çoklu görev ortamına uygun hale getirildiğinden, işlemci içerisindeki bölümlerde fonksiyonel açıdan 2 mantıksal bölümden oluşurlar. Bu birimler Veri Yolu Bağdaştırma Birimi (BIU) ve İcra Birimi (EU) ‘dir. BIU birimi, EU birimini veriyle beslemekten sorumluyken, icra birimi komut kodlarının çalıştırılmasından sorumludur. BIU bölümüne segment kaydedicileriyle birlikte IP ve komut kuyrukları ve veri alıp getirme birimleri dahilken, EU bölümüne genel amaçlı kaydediciler, kontrol birimi, aritmetik ve mantıksal komutların işlendiği birim dahildir.

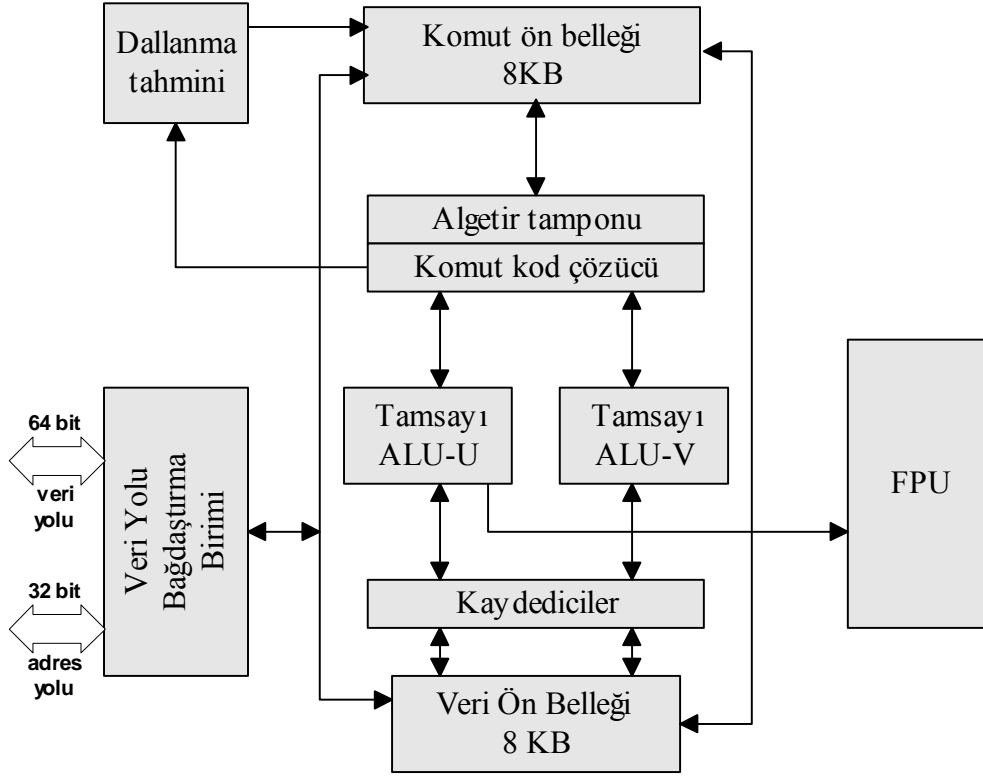


**Şekil 2.12:** 16- bitlik mikroişlemci mimarisi

**2.3.3. 32-Bitlik Mikroişlemciler:** 3. kuşak mikroişlemcilerdir. Diğerlerinden farklı olarak içerisinde FPU (Floating Point Unit- Kayan nokta birimi) denilen ve matematik işlemlerinden sorumlu olan bir birim eklenmiştir. Bu gelişmiş işlemci 64-bitlik geniş bir harici veri yoluna sahiptir. Geniş veri yolu, işlemcinin bir çevrimlik zamanda daha çok veri taşıması ve dolayısıyla yapacağı görevi daha



kısa zamanda yapması demektir. Bu, işlemcinin bir tıklanmasıyla, işlemci ile bellek arasında veya işlemci ile G/Ç birimleri arasında, 8-bitlik bir işlemciye göre 8 kat fazla bilgi taşınması demektir.



Şekil 2.13 : 32-bitlik mikroişlemci mimarisi

## 2.4. Mikrobilgisayarların Gelişimi

CPU, Bellek ve Giriş/Çıkış birimlerinin bir arada bulunması mikrodenetleyiciyi özellikle endüstriyel kontrol uygulamalarında güçlü bir dijital işlemci haline getirmiştir. Mikrodenetleyiciler özellikle otomobillerde motor kontrol, elektrik ve iç panel kontrol; kameralarda, ışık ve odaklama kontrol gibi amaçlarda kullanılmaktadır. Bilgisayarlar, telefon ve modem gibi çeşitli haberleşme cihazları, CD teknolojisi, fotokopi ve faks cihazları, radyo, TV, teyp, oyuncaklar, özel amaçlı elektronik kartlar ve sayılmayacak kadar çok alanda , mikrodenetleyiciler kullanılmaktadır. Bu kadar geniş bir uygulama alanı olan mikrodenetleyiciler aşağıda sıralanan çeşitli özelliklere sahiptirler.

- Programlanabilir sayısal paralel giriş / çıkış
- Programlanabilir analog giriş / çıkış
- Seri giriş / çıkış (senkron, asenkron ve cihaz denetimi gibi)
- Motor/servo kontrolü için darbe işaret çıkışı (PWM gibi)
- Harici giriş ile kesme
- Zamanlayıcı (Timer) ile kesme
- Harici bellek arabirimi
- Harici BUS arabirimi (PC ISA gibi)
- Dahili bellek tipi seçenekleri (ROM, PROM, EPROM ve EEPROM)

- Dahili RAM seçeneği
- Kesirli sayı (kayan nokta) hesaplaması
- D/A ve A/D çeviricileri

Bu özellikler mikrodnetleyicileri üreten firmalara ve mikrodnetleyicilerin tipine göre değişmektedir.

Mikrobilgisayar uygulamalarında dikkate alınması gereken en önemli özellikler *gerçek zaman* (real time) işlemi ve çok görevlilik (multi-tasking) özellikleridir. Gerçek zaman işlemi, mikrodnetleyicinin ihtiyaç anında çalışma ortamına, gereken kontrol sinyallerini göndermesi ve ortamı bekletmeyecek kadar hızlı olmasıdır. Çok görevlilik ise mikrodnetleyicinin birçok görevi aynı anda veya aynı anda gibi yapabilme kapasitesidir. Mikrodnetleyici özet olarak kullanıldığı sistemin birçok özelliğini aynı anda *gözleme* (monitoring), ihtiyaç anında *gerçek-zamanda cevap verme* (real-time response) ve sistemi *denetlemeden* (control) sorumludur.

Bir çok firma tarafından mikrodnetleyiciler üretilmektedir. Her firma üretmiş olduğu mikrodnetleyici yongaya farklı isimler ve özelliklerini birbirinden ayırmak içinde parça numarası vermektedir. Bu denetleyicilerin mimarileri arasında çok küçük farklar olmasına rağmen aşağı yukarı aynı işlemleri yapabilmektedir. Her firma ürettiği chip'e bir isim ve özelliklerini biri birinden ayırmak içinde parça numarası vermektedir. Günümüzde yaygın olarak 8051(intel firması) ve PIC adı verilen mikrodnetleyiciler kullanılmaktadır. Bunlardan başka Phillips, Dallas, Siemens, Oki, Temic, Haris, Texas gibi çeşitli firmalarda üretim yapmaktadır. Örneğin; bunlardan *Microchip* firması üretmiş olduklarına *PIC* adını verirken, parça numarası olarak da *12C508*, *16C84*, *16F877* gibi kodlamalar vermiştir, *Intel* ise ürettiği mikrodnetleyicilere *MCS-51* ailesi adını vermiştir, *Texas Ins.* ise işaret işlemeye yönelik olarak *Digital Signal Processing (DSP)* mikrodnetleyici yongası üretmektedir. *PIC* mikrodnetleyicileri elektronik cihazlarda çok yaygın olarak kullanılmaktadır. Çünkü her amaca uygun boyut ve özellikte mikrodnetleyici bulmak mümkündür. Çeşitli tiplerde mikrodnetleyiciler kullanılabilir fakat, uygulamada en küçük, en ucuz, en çok bulunan ve yapılan işin amacına yönelik olmasına dikkat edilmelidir. Bunun içinde mikrodnetleyicilerin genel özelliklerinin iyi bilinmesi gerekir.

Mikrodnetleyicili bir sistemin gerçekleştirile bilinmesi için, mikrodnetleyicinin iç yapısının bilinmesi kadar, sistemin yapacağı iş için mikrodnetleyicinin programlanması da büyük bir önem arz eder. Mikrodnetleyicili sistemler ile bir oda sıcaklığını, bir motorun hızını, led ışık gibi birimlerini kontrol edebiliriz. Bütün bu işlemleri nasıl yapacağını mikrodnetleyiciye tarif etmek, açıklamak gerekir. Bu işlemlerde mikrodnetleyicili sistemin program belleğine yerleştirilen programlar vasıtasıyla gerçekleştirilir. Mikrodnetleyiciler için programlar assembly veya C gibi bir programlama dilinde yazılabilir. Assembly dilinde yazılan bir program assembler adı verilen bir derleyici ile makine diline çevrildikten sonra mikrodnetleyiciye yüklenir. C dilinde yazılan programında bir çevirici ile makine diline çevrilmesi gerekmektedir. Makine dilindeki bir

programın uzantısı ‘.HEX’ dir. PIC mikrodnetleyicisi için program yazarken editör ismi verilen bir programa ihtiyaç vardır. En çok kullanılan editör programı ise MPLAB’tır. MPLAB’ da yazılan programlar proje dosyalarına dönüştürülerek, aynı editör içerisinde derlenebilmektedir.

Bütün bu özellikler dikkate alınarak en uygun mikrodnetleyici seçimi yapılmalıdır. Çünkü mikrodnetleyiciler ticari amaçlı birçok elektronik devrede yaygın olarak kullanılmaktadır.

## 2.5. Mikrobilgisayar Seçimi

Mikrodnetleyici seçimi kullanıcı için oldukça önemlidir, çünkü mikrodnetleyiciler ticari amaçlı bir elektronik devrede yaygın olarak kullanılmaktadır. Bu sistemlerin öncelikle maliyetinin düşük olması için mikrodnetleyicinin de ufak ve ucuz olması istenir. Diğer taraftan ürünün piyasada bol miktarda bulunması da önemlidir. Tüm bu hususlar dikkate alınarak, kullanıcılar öncelikle hangi firmanın ürününü kullanacağına karar verirler. Daha sonra da hangi seriden, hangi ürünün kullanacaklarına karar verirler. Burada mikrodnetleyicinin belleğinin yazılım için yeterli büyüklükte olması, kullanılması düşünülen ADC (Analog Dijital Dönüştürücü) kanalı, port sayısı, zamanlayıcı sayısı ve PWM (Pulse Width Modulation- Darbe Genişlik Modülasyonu) kanalı sayısı önemlidir. Ayrıca tasarımcı yapılacak iş için uygun hızda mikrodnetleyici kullanmalıdır. Tüm bu hususlar dikkate alınarak uygun mikrodnetleyiciye karar verilir. Ürün geliştirmek için pencereleli (EPROM) veya FLASH tipinde olan belleği silinip, yazılabilen mikrodnetleyici kullanılır. Çünkü ürün geliştirme aşamasında mikrodnetleyici defalarca silinip, yazılabilmektedir. Ayrıca belleği daha hızlı silinip, yazılabilen FLASH mikrodnetleyiciler öğrenmeye yeni başlayanlar için cazip olmaktadır.

Seçimi etkileyen bu noktaları kısaca açıklarsak;

- *Mikrodnetleyicinin İşlem Gücü:* Her uygulamada farklı bir işlem gücüne gereksinim duyulabilir. Bunun için yapılacak uygulamada kullanılacak mikrodnetleyicinin çalışabileceği en yüksek frekans aralığı seçilmelidir.
- *Belleğin Kapasitesi ve Tipi:* Geliştirilecek olan uygulamaya göre program belleği, veri belleği ve geçici bellek kapasitesi dikkate alınmalıdır. Kullanılacak olan belleğin tipide uygulama için önemli bir faktördür.
- *Giriş/Çıkış Uçları:* Mikrodnetleyicinin çevre birimler ile haberleşmesinin sağlayan uçlardır. Bu nedenle giriş/ çıkış uçlarının sayısı oldukça önemlidir. Yapılacak olan uygulamaya göre bu faktörde dikkate alınmalıdır.
- *Özel Donanımlar:* Yapılacak olan uygulamanın çeşidine göre mikrodnetleyiciye farklı çevre birimleri de eklenebilir. Mikrodnetleyici çevre birimleri ile iletişim kurarken kullanacağı seri, I<sup>2</sup>C, SPI, USB, CAN gibi veri iletişim protokollerini destekleyen veya ADC, analog karşılaştırıcı gibi analog verileri işleyebilecek donanımlara sahip olması dikkate alınmalıdır.

- *Kod Koruması*: Mikrodenetleyicinin sahip olduğu kod koruması özellikle ticari uygulamalarda program kodunun korunmasına olanak sağlamaktadır.

## **2.6. Bölüm Kaynakları**

1. O. Altınbaşak, 2001. “Mikrodenetleyiciler ve PIC Programlama”, Atlas Yayıncılık, İstanbul.
2. N. Gardner, 1998. “PIC Programlama El Kitabı”, Bileşim Yayıncılık, İstanbul.
3. O. Urhan, M.Kemal Güllü, 2004. “Her Yönüyle PIC16F628”, Birsen Yayınevi, İstanbul.
4. N. Topaloğlu, S. Görgünoğlu, 2003. “Mikroişlemciler ve Mikrodenetleyiciler”, Seçkin Yayıncılık, Ankara.
5. Y. Bodur, 2001. “Adım Adım PICmicro Programlama”, Infogate.
6. M. Kemal Güngör, 2003. “Endüstriyel Uygulamalar İçin Programlanabilir Kontrol Ünitesi” .

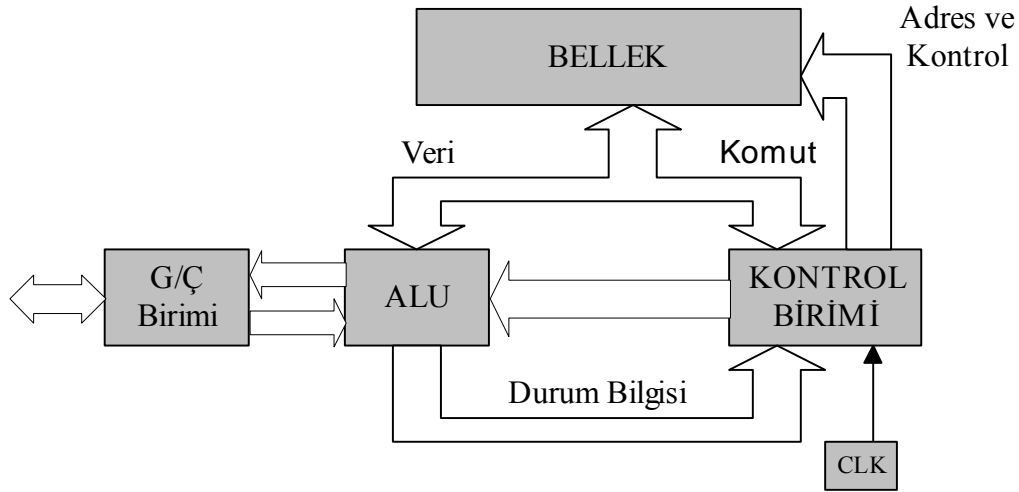
## BÖLÜM 3. MİMARİLER

### 3.1. Mikrobilgisayar Tasarım Yapıları

Bilgisayarın yüklenen tüm görevleri çok kısa zamanda yerine getirmesinde yatan ana unsur bilgisayarın tasarım mimarisidir. Bir mikroişlemci, mimari yetenekleri ve tasarım felsefesiyle şekillenir.

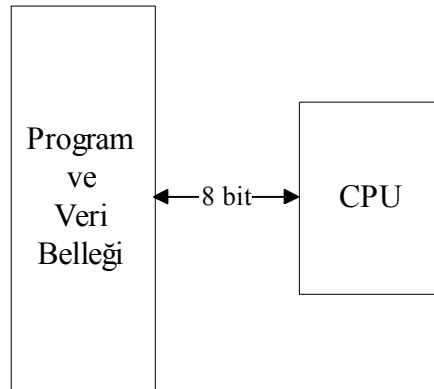
#### 3.1.1. Von Neuman (Princeton) Mimarisi

Bilgisayarlarda ilk kullanılan mimaridir. İlk bilgisayarlar Von Neuman yapısından yola çıkılarak geliştirilmiştir. Geliştirilen bu bilgisayar beş birimden oluşmaktaydı. Bu birimler; aritmetik ve mantıksal birim, kontrol birim, bellek, giriş-çıkış birimi ve bu birimler arasında iletişimi sağlayan yollardan oluşur.



Şekil 3.1. Von Neuman mimarili bilgisayar sistemi

Bu mimaride veri ve komutlar bellekten tek bir yoldan mikroişlemciye getirilerek işlenmektedir. Program ve veri aynı bellekte bulunduğundan, komut ve veri gerekli olduğunda aynı iletişim yolunu kullanmaktadır. Bu durumda, komut için bir algetir saykılı, sonra veri için diğer bir algetir saykılı gerekmektedir.



Şekil 3.2. Von Neuman mimarisi

Von Neuman mimarisine sahip bir bilgisayar aşağıdaki sıralı adımları gerçekleştirir.

1. Program sayıcısının gösterdiği adresten (bellekten) komutu algetir.
2. Program sayıcısının içeriğini bir artır.
3. Getirilen komutun kodunu kontrol birimini kullanarak çöz. Kontrol birimi, bilgisayarın geri kalan birimlerine sinyal göndererek bazı operasyonlar yapmasını sağlar.
4. 1. adıma geri dönlür.

### Örnek 3.1:

Mov acc, reg

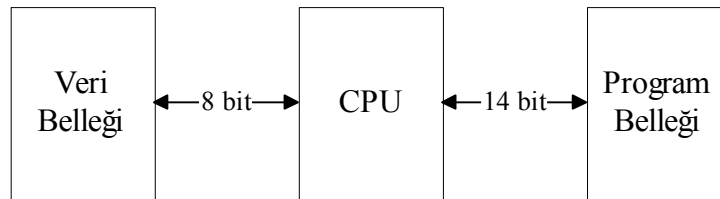
1. cp : Komut okur

2.,... cp : Veriyi okur ve *acc* ye atar.

Von Neuman mimarisinde, veri bellekten alınıp işledikten sonra tekrar belleğe gönderilmesinde çok zaman harcanır. Bundan başka, veri ve komutlar aynı bellek biriminde depolandığından, yanlışlıkla komut diye veri alanından kod getirilmesi sıkıntılara sebep olmaktadır. Bu mimari yaklaşıma sahip olan bilgisayarlar günümüzde, verilerin işlenmesinde, bilginin derlenmesinde ve sayısal problemlerde olduğu kadar endüstriyel denetimlerde başarılı bir şekilde kullanılmaktadır.

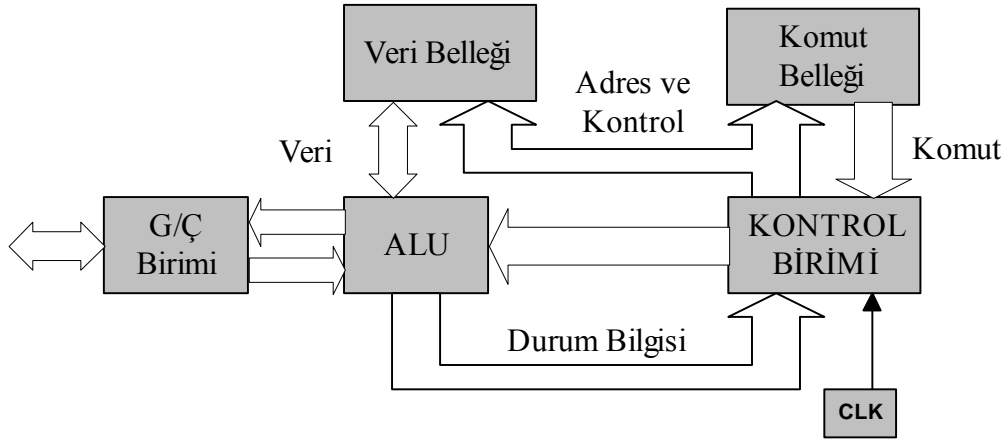
### 3.1.2. Harvard Mimarisi

Harvard mimarili bilgisayar sistemlerinin Von Neuman mimarisinden farkı veri ve komutların ayrı ayrı belleklerde tutulmasıdır. Buna göre, veri ve komut aktarımında iletişim yolları da bir birinden bağımsız yapıda bulunmaktadır.



Şekil 3.3. Harvard Mimarisi

Komutla birlikte veri aynı saykıl da farklı iletişim yolundan ilgili belleklerden alınıp işlemciye getirilebilir. Getirilen komut işlenip ilgili verisi veri belleğinden alınırken sıradaki komut, komut belleğinden alınıp getirilebilir. Bu önden alıp getirme işlemi, dallanma haricinde hızı iki katına çıkarabilmektedir.



**Şekil 3.4.** Harvard Mimarili bilgisayar sistemi

### Örnek 3.2:

Mov acc, reg

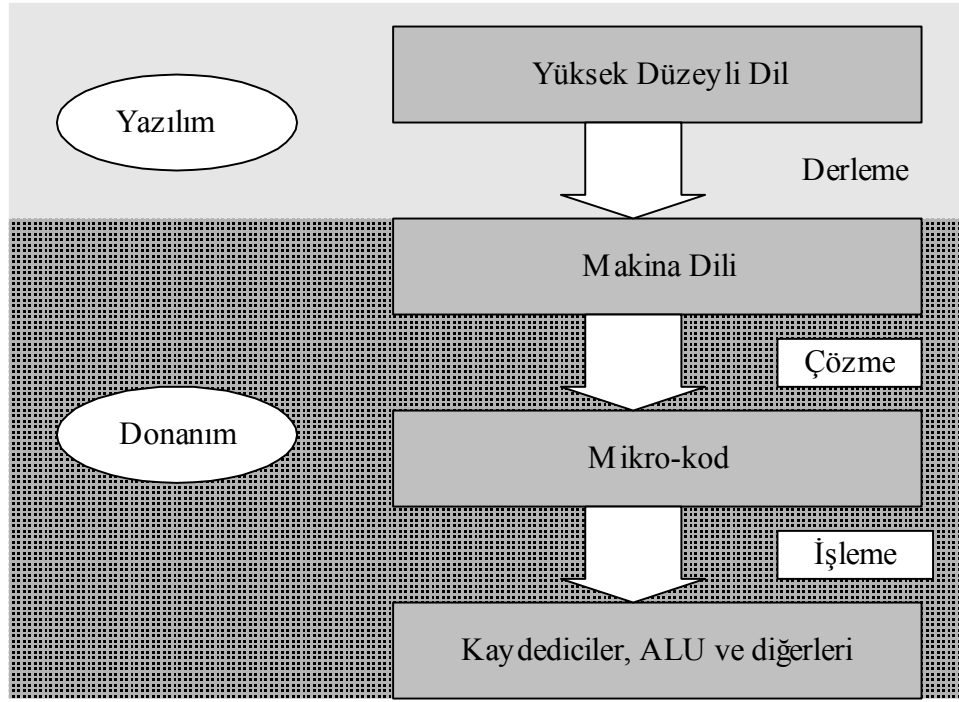
1. cp : Öncelikle “*move acc, reg*” komutunu okur.
2. cp : Sonra “*move acc, reg*” komutunu yürütür.

Bu mimari günümüzde daha çok sayısal sinyal işlemcilerinde (DSP) kullanılmaktadır. Bu mimaride program içerisinde döngüler ve zaman gecikmeleri daha kolay ayarlanır. Von Neuman yapısına göre daha hızlıdır. Özellikle PIC mikrodeneleyicilerinde bu yapı kullanılır.

## 3.2. Mikroişlemci Komut Tasarım Mimarileri

### 3.2.1. CISC (Complex Instruction Set Computer) Mimarisi

Bu mimari, programlanması kolay ve etkin bellek kullanımı sağlayan tasarım felsefesinin bir ürünüdür. İşlemci üzerinde performans düşüklüğü ve işlemcinin karmaşık bir hale gelmesine neden olsa da yazılımı basitleştirmektedir. Bu mimarinin en önemli iki özelliği, değişken uzunluktaki komutlar diğeri ise karmaşık komutlardır. Değişken ve karmaşık uzunluktaki komutlar bellek tasarrufu sağlar. Karmaşık komutlar birden fazla komutu tek bir hale getirirler. Karmaşık komutlar aynı zamanda karmaşık bir mimariyi de oluşturur. Mimarideki karışıklık işlemcinin performansını da doğrudan etkilemektedir. Bu sebepten dolayı çeşitli istenmeyen durumlar ortaya çıkabilir. **CISC komut seti mümkün olabilen her durum için bir komut içermektedir. CISC mimarisinde yeni geliştirilen bir mikroişlemci eski mikroişlemcilerin assembly dilini desteklemektedir.**



**Şekil 3.5.** CISC tabanlı bir işlemcinin çalışma biçimi

CISC mimarisi çok kademeli işleme modeline dayanmaktadır. İlk kademe, yüksek seviyeli dilin yazıldığı yerdir. Sonraki kademeyi ise makine dili oluşturur. Burada yüksek seviyeli dilin derlenmesi ile bir dizi komutlar makine diline çevrilir. Bir sonraki kademede makine diline çevrilen komutların kodları çözülerek , mikrokodlara çevrilir. En son olarak da işlenen kodlar gerekli olan görev yerlerine gönderilir.

#### **CISC Mimarisinin Avantajları**

- Mikroprogramlama assembly dilinin yürütülmesi kadar kolaydır ve sistemdeki kontrol biriminden daha ucuzdur.
- Yeni geliştirilen mikrobilgisayar bir öncekinin assembly dilini desteklemektedir.
- Verilen bir görevi yürütmek için daha az komut kullanılır. Böylece bellek daha etkili kullanılır.
- Mikroprogram komut kümeleri, yüksek seviyeli dillerin yapılarına benzer biçimde yazıldığından derleyici karmaşık olmak zorunda değildir.

#### **CISC Mimarisinin Dezavantajları**

- Gelişen her mikroişlemci ile birlikte komut kodu ve yonga donanımı daha karmaşık bir hale gelmiştir.
- Her komutun çevirim süresi aynı değildir. Farklı komutlar farklı çevrim sürelerinde çalıştıkları için makinenin performansını düşürecektir.
- Bir program içerisinde mevcut komutların hepsi kullanılamaz.

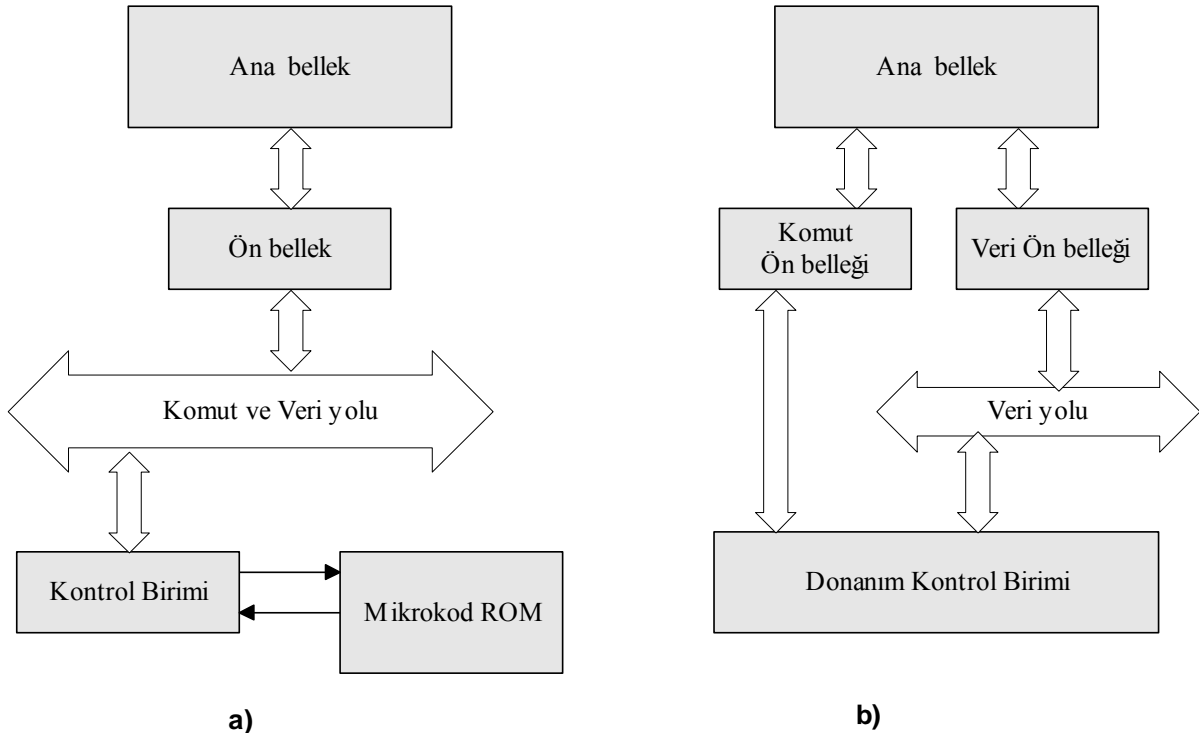


- Komutlar işenirken bayrak bitlerinin dikkat edilmesi gerekir. Buda ek zaman süresi demektir. Mikroişlemcinin çalışmasını etkilemektedir.

### 3.2.2. RISC ( Reduced Instruction Set Computer) Mimarisi

RISC mimarisi IBM, Apple ve Motorola gibi firmalarca sistematik bir şekilde geliştirilmiştir. RISC mimarisinin taraftarları, bilgisayar mimarisinin gittikçe daha karmaşık hale geldiğini ve hepsinin bir kenara bırakılıp en başta yeniden başlamak fikrindeydiler. 70’li yılların başında IBM firması ilk RISC mimarisini tanımlayan şirket oldu. Bu mimaride bellek hızı arttığından ve yüksek seviyeli diller assembly dilinin yerini aldığından, CISC’in başlıca üstünlükleri geçersiz olmaya başladı. RISC’in felsefesi üç temel prensibe dayanır.

- *Bütün komutlar tek bir çevrimde çalıştırılmalıdır:* Her bir komutun farklı çevrimde çalışması işlemci performansını etkileyen en önemli nedenlerden biridir. Komutların tek bir çevrimde performans eşitliğini sağlar.
- *Belleğe sadece “load” ve “store” komutlarıyla erişilmelidir.* Eğer bir komut direkt olarak belleği kendi amacı doğrultusunda yönlendirilirse onu çalıştırmak için birçok saykıl geçer. Komut alınıp getirilir ve bellek gözden geçirilir. RISC işlemcisiyle, belleğe yerleşmiş veri bir kaydediciye yüklenir, kaydedici gözden geçirilir ve son olarak kaydedicinin içeriği ana belleğe yazılır.
- *Bütün icra birimleri mikrokod kullanmadan donanımdan çalıştırılmalıdır.* Mikrokod kullanımı, dizi ve benzeri verileri yüklemek için çok sayıda çevrim demektir. Bu yüzden tek çevrimli icra birimlerinin yürütülmesinde kolay kullanılmaz.



Şekil 3.6. a) Mikrokod denetimli CISC mimarisi; b) Donanım denetimli RISC mimarisi

RISC mimarisi küçültülen komut kümesi ve azaltılan adresleme modları sayısı yanında aşağıdaki özelliklere sahiptir.

- Bir çevrimlik zamanda komut işleyebilme
- Aynı uzunluk ve sabit formatta komut kümesine sahip olma
- Ana belleğe sadece “load” ve “store” komutlarıyla erişim; operasyonların sadece kaydedici üzerinde yapılması
- Bütün icra birimlerinin mikrokod kullanmadan donanımsal çalışması
- Yüksek seviyeli dilleri destekleme
- Çok sayıda kaydediciye sahip olması

### RISC Mimarisinin Üstünlükleri

- **Hız:** **Azaltılmış komut kümesi**, kanal ve süperskalar tasarıma izin verildiğinden RISC mimarisi CISC işlemcilerin performansına göre 2 veya 4 katı yüksek performans gösterirler.
- **Basit donanım:** RISC işlemcinin komut kümesi çok basit olduğundan çok az yonga uzayı kullanılır. Ekstra fonksiyonlar, bellek kontrol birimleri veya kayan noktalı aritmetik birimleri de aynı yonga üzerine yerleştirilir.
- **Kısa Tasarım Zamanı:** RISC işlemciler CISC işlemcilere göre daha basit olduğundan daha çabuk tasarlanabilirler.

### RISC Mimarisinin Eksiklikleri:

CISC tasarım stratejisinden RISC tasarım stratejisine yapılan geçiş kendi problemlerinde beraberinde getirmiştir. Donanım mühendisleri kodları CISC işlemcisinden RISC işlemcisine aktarırken anahtar işlemleri göz önünde bulundurmamak zorundadırlar.

### CISC ve RISC Tabanlı İşlemcilerin Karşılaştırılması

CISC ve RISC tabanlı işlemcilerin karşılaştırılmasında iki önemli faktör farklılıklarını ortaya çıkarmada yeterlidir.

**Hız:** Genelde RISC çipleri kanal tekniği kullanarak eşit uzunlukta segmentlere bölünmüş komutları çalıştırmaktadır. Kanal tekniği komutları kademeli olarak işler ki bu RISC'in bilgi işlemini CISC'den daha hızlı yapmasını sağlar RISC işlemcisinde tüm komutlar 1 birim uzunlukta olup kanal tekniği ile işlenmektedir. Bu teknikte bazıları hariç komutlar, her bir basamağında aynı işlemin uygulandığı birimlerden geçerler. Kanal teknolojisini açıklamak için herhangi bir komutun işlenmesindeki adımlar ele alınırsa:

Komut kodu ve işlenecek veriler dahil bütün bilgilerin MIB'deki kaydedicilerde olduğu düşünülürse, birinci adımda yapılacak işin kaydedicide bulunan komut kodu çözülür, ikinci adımda üzerinde çalışılacak veri (işlenen) kaydediciden alınıp getirilir, üçüncü adımda veri, komuta göre

Aritmetik ve Mantık Biriminde işleme tabii tutulur ve dördüncü adımda da sonuç kaydediciye yazılacaktır. Böylece bir komutun işlemesi için her bir basamak bir saat çevrimi gerektirirse, dört çevrimle (adımda) gerçekleşmiş olmakta ve bir adım bitmeden diğeri başlayamamaktadır.

Kanal tekniği ile çalışan işlemcilerde birinci adımda komut kodu çözülür, ikinci adımda birinci komutun üzerinde çalışacağı veri (işlenen) kaydediciden alınırken, sıradaki ikinci işlenecek olan komutun kodu çözülür. Üçüncü adımda ilk komutun görevi ALU'da yerine getirilirken, ikinci komutun işleyeceği işlenen alınıp getirilir. Bu anda sıradaki üçüncü komutun kodu çözülür ve işlem böylece devam eder.

Kanal (Pipeline) tekniğinde çevrim zamanının düşmesi için komut kodlarının hızlı çözülmesi gereklidir. RISC mimarisinde tüm komutlar 1 birim uzunlukta oldukları için komut kodunu çözme işlemi kolaylaşır. Sistemde kullanılan kaydedicilerin simetrik bir yapıda olması, derleme işlemini kolaylaştırmaktadır. RISC işlemcilerde belleğe yalnız yükle ve depola komutlarıyla ulaşılır. Bazı eski CISC mimarisinde de olmasına rağmen RISC mimarisinin sabit uzunluktaki basit komutlarla çalışması pipeline sistemini daha iyi kullanmasına sebep olmaktadır. Bu yüzden hesaplama oranlarının birinci öncelik arz ettiği yerlerde iş-istasyonları ve dağıtıcılarda çok tercih edilmektedir.

**Transistör sayısı:** CISC mimarisinde kullanılan transistör sayısı RISC'e nazaran daha fazladır. Transistör sayısının bir yerde çok olması fazla yerleşim alanı ve ayrıca fazla ısı demektir. Bundan dolayı da fazla ısı üretimi soğutma olayını gündeme getirmektedir. CISC tabanlı Pentium işlemcilerde karışık ısı dağıtıcısı veya soğutma fanlar kullanılmaktadır.

RISC mimarisindeki önemli üstünlüklere karşı bazı mahzurları ortaya çıkmaktadır. RISC mimarisi, CISC'in güçlü komutlarından yoksundur ve aynı işlemi yapmak için daha fazla komut işlenmesini gerektirir. Bundan dolayı da RISC'in bant genişliği artar. Bu sistemde güçlü komutların yokluğu ikinci bir yardımcı işlemciyle ya da işlemci içinde oluşturulacak ayrı bir pipeline bölümüyle giderilebilir. Komut ön-belleğinin kullanılması yüksek komut alıp getirme işlemini azaltmaktadır. RISC mimarisi diğerine nazaran daha kompleks yazılımlara ihtiyaç duyar.

<b>RISC (Hard-wired Control Unit)</b>	<b>CISC (Microprogrammed Control Unit)</b>
Hızlı	Nispeten yavaş
Ucuz	Pahalı
Yeniden dizayn zor	Esnek
Daha az komut (instruction)	Daha fazla komut (instruction)
Daha fazla saklayıcı bellek (register)	Daha az saklayıcı bellek (register)

### 3.2.3. EPIC Mimarisi

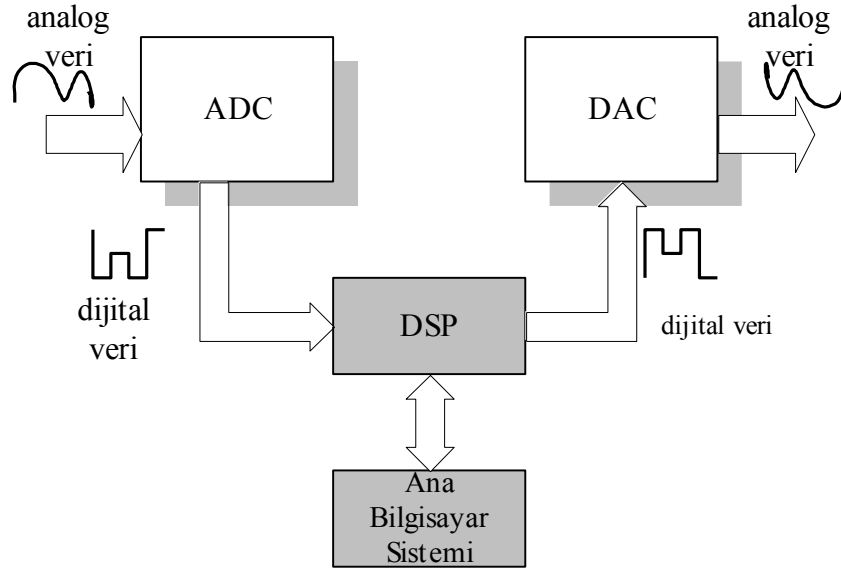
Bu mimari RISC ve CISC mimarisinin üstün yönlerinin bir arada bulunduğu bir mimari türüdür. EPIC mimarisi, işlemcinin hangi komutların paralel çalışabildiğini denetlemesi yerine, EPIC derleyicisinden açık olarak hangi komutların paralel çalışabildiğini bildirmesini ister. Çok uzun komut kelimesi (VLIW) kullanan bilgisayarlar, yazılımın paralelliğine ilişkin kesin bilgi sağlanan mimari örneklerdir. EPIC varolan VLIW mimarisinin dallanma sorunlarını çözmeye çalışarak daha ötesine gitmeyi hedeflemektedir. Derleyici programdaki paralelliği tanımlar ve hangi işlemlerin bir başkasından bağımsız olduğunu belirleyerek donanıma bildirir. EPIC mimarisinin ilk örneği, IA-64 mimarisine dayalı Itanium işlemci ailesidir.

#### EPIC Mimarisin Üstünlükleri

- Paralel çalıştırma ( çevrim başına birden çok komut çalıştırma)
- Tahmin kullanımı
- Spekülasyon kullanımı
- Derleme anında paralelizmi tanıyan derleyiciler
- Büyük bir kaydedici kümesi
- Dallanma tahmini ve bellek gecikmesi problemlerine karşı üstün başarı
- Gelişme ile birlikte eskiye karşı uyumluluk

### 3.2.4. DSP (Dijital Signal Processing -Dijital Sinyal işleme)

Dijital Signal Processing (Dijital Sinyal işleme) sözcüklerinin bir kısaltmasıdır. 1970'lerin sonlarında mikro-işlemcilerin ortaya çıkmasıyla, DSP kullanımı geniş bir uygulama alanı bulmuştur. Kullanım alanları, cep telefonlarından bilgisayarlara, video çalıcılardan modemlere kadar çok geniş bir alana yayılmaktadır. DSP yongaları, mikro-işlemciler gibi programlanabilir sistemler olup, saniyede milyonlarca işlem gerçekleştirebilir. DSP kartları, üzerlerindeki DSP'ler sayesinde aynı anda bir çok efekt uygulayabilir. Özellikle modemlerde bulunurlar. Çok yüksek hızlarda kayan nokta matematiksel işlemleri yapmak üzere geliştirilmiş bir donanımdır. Diğer birçok şeyin yanı sıra DSP donanımı ses ve görüntü sinyallerinin gerçek zamanlı sıkıştırma ve açma işlemleri için kullanıla bilinir.



**Şekil 3.7.** DSP sistem ve elemanları

### 3.3. Bölüm Kaynakları

1. O. Altınbaşak, 2001. "Mikrodenetleyiciler ve PIC Programlama", Atlas Yayıncılık, İstanbul.
2. N. Gardner, 1998. "PIC Programlama El Kitabı", Bileşim Yayıncılık, İstanbul.
3. O. Urhan, M.Kemal Güllü, 2004. "Her Yönüyle PIC16F628", Birsen Yayınevi, İstanbul.
4. N. Topaloğlu, S. Görgünoğlu, 2003. "Mikroişlemciler ve Mikrodenetleyiciler", Seçkin Yayıncılık, Ankara.
5. Y. Bodur, 2001. "Adım Adım PICmicro Programlama", Infogate.
6. Ö.Kalınlı, 2001. "Signal Processing With DSP".

## BÖLÜM 4. BAŞARIM ÖLÇÜTLERİ

Farklı tür bilgisayarların performansını değerlendirebilmek, bu makineler arasında en iyi seçim veya anahtar etmendir. Performans ölçümündeki karışıklık birçok temel etmenden doğar. Komut takımı ve bu komutları tamamlayan donanım önemli ana etmenlerdir. Aynı donanım ve komut takımına sahip iki bilgisayar bile bellek ve giriş/çıkış örgütlenmesi, ve de işletim sistemleri nedeniyle ya da sadece testlerde farklı iki derleyici kullanıldığından dolayı farklı başarımlar verebilir. Bu etmenlerin başarımı nasıl etkilediğini belirlemek, makinenin belirli yönlerinin tasarımının dayanağı olan ana güdüyü anlamak açısından çok önemlidir.

Yolcu uçaklarıyla ilgili bir örnek verelim. Aşağıda mevcut uçaklarla ilgili bilgiler verilmiştir. Buna göre; 5000 yolcu New York'tan Paris'e (1500 km) taşımamız gerektiğine göre hangi uçağı kullanmalıyız?

Uçak	P: Yolcu kapasitesi	R: Uçuş menzili (km)	S: Uçuş hızı (km/saat)	(P*S) Yolcu gönderme hızı
Boeing 737-100	101	1000	1000	101000
Boeing 777	375	7400	1000	375000
Boeing 747	470	6650	1000	460600
BAC/Sud Concorde	132	6400	2200	290400
Douglas DC-8-50	146	14000	880	128480

**Tablo 4.1.** Uçaklar ve özellikleri

Boeing 737-100 uçuş menzili New York'tan Paris'e uçmaya yetmeyeceğinden daha ilk başta listeden elenir. Ardından, geriye kalanlar arasında Concorde en hızlı olarak görülmektedir. Ancak uçağın sadece hızlı olması yeterli değildir. Boeing 747 daha yavaş olmasına karşın bir Concorde'un taşıyabileceğinden 3 kat daha fazla yolcu taşıyabiliyor. Uçakların performansları için daha iyi bir ölçüt uçakların yolcu taşıma hızı olabilir. Yolcu sayısının uçağın hızıyla çarpımından çıkan sayı yolcu taşıma hızıdır. Bu durumda 747'nin 5000 yolcu taşımada daha başarılı olduğunu görürüz, çünkü onun yolcu taşıma hızı Concorde'dan daha yüksektir. Diğer taraftan, eğer toplam yolcu sayısı 132'den az olursa Concorde elbetteki Boeing 747'den daha iyidir. Çünkü onları 747'den neredeyse iki kat hızlı taşıyacaktır. Yani performans büyük oranda yapılacak işe bağlıdır.

### 4.1. Başarım Tanımı

Bir bilgisayarın diğerinden daha iyi başarıma sahip olduğunu söylemekle, tipik uygulama programlarımız açısından o bilgisayarın birim sürede diğerinden daha çok iş bitirebildiğini kastederiz.

Zaman paylaşımlı çok-kullanıcılı çok- görevli bir bilgisayarda, bir programın başlangıcından bitişine kadar geçen toplam zamana toplam yürütme süresi denir. Genellikle giriş/çıkış ile programımızın işlemesi için geçen süre ayrı ayrı sayılır. Bunlar işimizin CPU süresi ve Giriş/ Çıkış işlem süresi olarak adlandırılır. Zaman paylaşımlı anabilgisayarlarda diğer kullanıcıların işleri arasında çalışan programın çalışma süresi CPU süresinden daha uzundur. Kullanıcıları genelde bu çalışma süresi ilgilendirirken, bilgisayar merkezinin yöneticisi bilgisayarın toplam iş bitirme hızıyla (throughput) ilgilenir.

Programların CPU sürelerini azaltmak için çeşitli yöntemler vardır. Bunlardan akla ilk gelen bilgisayarı aynı tip daha hızlı sürümüyle değiştirmektir. Bu yöntem başarıımı kısmen arttırır. Belli bir görevde , X bilgisayarının başarıımı temel olarak programın çalışma zamanıyla ters orantılıdır.

$$X'in \text{ Başarıımı} = \frac{1}{X'in \text{ Çalışma süresi}}$$

Bu da, X ve Y bilgisayarlarının başarıımı çalışma zamanıyla ters orantılıdır demektir.

$$Y'nin \text{ Çalışma Süresi} > X'in \text{ Çalışma Süresi}$$

ise

$$X'in \text{ Başarıımı} > Y'nin \text{ Başarıımı}$$

demektir. Nicel olarak,

$$\frac{X \text{ in Başarıımı}}{Y \text{ nin Başarıımı}} = \frac{Y \text{ nin Çalışma Süresi}}{X \text{ in Çalışma Süresi}} = n$$

ise X in Y den n kat hızlı olduğu söylenir.

## 4.2. Ölçme Koşulları ve Ölçme Birimleri

Çok görevli ve çok kullanıcılı bir bilgisayar ortamında yürütme süresi ve belli bir iş için harcanan işlem süresi farklı kavramlardır.

- Programın başlatılmasına bitişine kadarki zamana toplam yürütme süresi, yanıt zamanı, geçen süre yada duvar süresi denir.
- Program işleminde CPU tarafından harcanan zaman dilimlerinin toplamına CPU yürütme süresi yada basitçe CPU süresi denir.
- CPU süresi daha da ayrışarak program CPU süresi ve sistem CPU süresine bölünür. Sistem CPU süresi içinde giriş/çıkış, disk erişimi ve benzeri diğer çeşitli sistem görevleri yapılır. Program CPU zamanı ise yalnızca program kodunun yürütülmesi için geçen net süredir.

Zaman genellikle saniye(s) birimiyle ölçülür. Ancak saat dönüş süresi, yani bilgisayarın periyodu çoğunlukla nanosaniye (nano=1/1 000 000 000 ) kullanılarak ölçülür. Genelde bilgisayarların hızları verirken saat hızı (=1/saat-dönüşü) tercih edilir. Saat hızının birimi Hertz (Hz)

dir. Hertz saniyedeki dönüş sayısına eşittir. Daha hızlı saatler için Kilo-Hertz, Mega-Hertz yada Giga-Hertz terimleri kullanılır.

**Tablo 4.2.** Zaman Birimleri

Zaman Birimleri	Saniye	Mili-saniye	Mikro-saniye	Nano-saniye
Kısaltması	s	Ms	μs	ns
Saniye eşdeğeri	1	0.001	0.000 001	0.000 000 001

**Tablo 4.3.** Frekans birimleri

Frekans Birimleri	Hertz	Kilo- Hertz	Mega- Hertz	Giga- Hertz
Kısaltması	Hz	KHz	MHz	GHz
Saniyedeki dönüş	1	1000	1 000 000	1 000 000 000

Bilgisayarların başarımlarını karşılaştırırken, gerçekte kullanılacak uygulama programlarının iş-bitirme hızı son derece önemlidir. Bir programın CPU yürütme süresini belirleyen temel ifade;  
 $CPU\text{-}yürütme\text{-}süresi = CPU\text{-}saat\text{-}dönüş\text{-}sayısı \times Saat\text{-}dönüş\text{-}süresi$ ;  
 Biçimindedir.

CPU-saat-dönüş-sayısı ise;

$CPU\text{-}saat\text{-}dönüş\text{-}sayısı = komut\text{-}sayısı \times komut\text{-}başına\text{-}ortalama\text{-}dönüş\text{-}sayısı$

Komut başına ortalama dönüş sayısı genellikle CPI (cycle-per- instruction) diye adlandırılır.

**ÖRNEK 4.1:** A ve B aynı komut takımına sahip iki makine olsun. Herhangi bir program için A'nın saat dönüşü 10ns ve CPI 'si 2.0 ölçülmüş, aynı program için B'nin saat dönüşü 20ns ve CPI'si 1.2 ölçülmüştür. Bu program açısından hangi makine kaç kat hızlıdır.?

**Çözüm 4.1:** Programdaki komut sayısının I olduğunu varsayalım. Bu durumda;

$CPU\text{-}süresi\text{-}A = CPU\text{-}saat\text{-}dönüşü\text{-}sayısı\text{-}A \times saat\text{-}dönüş\text{-}süresi\text{-}A$

$$= I \times 2.0 \times 10\text{ ns} = 20\text{ I ns}$$

$CPU\text{-}süresi\text{-}B = I \times 1.2 \times 20\text{ ns} = 24\text{ I ns}$

$CPU\text{-}süresi\text{-}A < CPU\text{-}süresi\text{-}B$ , o halde A daha hızlıdır.

$$\frac{Başarımlar\ A}{Başarımlar\ B} = \frac{Çalışma\ Süresi\ B}{Çalışma\ Süresi\ A} = n$$

$$n = 24 \times I\text{ ns} / 20 \times I\text{ ns} = 1.2$$

A makinesi B den 1.2 kat daha hızlıdır.



### 4.3. Başarım Ölçütleri

MIPS ve MFLOPS, sistem başarımını karşılaştırmak için sık kullanılan başarım ölçütleridir. Bu iki başarım ölçütü birçok durumda yanıltıcı olabilir.

#### 4.3.1. MIPS Başarım Ölçümü

MIPS saniyede milyon komut için kısaltmadır. Bir programda,

$$\begin{aligned} MIPS &= \frac{\text{Komut Sayısı}}{\text{Yürütme Süresi} \times 10^6} = \frac{\text{Komut Sayısı}}{\text{CPU - saat - dönüş - sayısı} \times \text{saat - dönüş süresi} \times 10^6} \\ &= \frac{\text{Komut sayısı} \times \text{saat hızı}}{\text{Komut sayısı} \times \text{CPI} \times 10^6} \end{aligned}$$

burada  $\text{CPU saat dönüşü sayısı} = \text{komut sayısı} \times \text{CPI}$  olduğundan

$$\begin{aligned} MIPS &= \frac{\text{Saat hızı}}{\text{CPI} \times 10^6} \quad (\text{doğal MIPS}) \\ \text{Çalışma Süresi} &= \frac{\text{Komut sayısı} \times \text{CPI}}{\text{Saat hızı}} = \frac{\text{Komut sayısı}}{\text{Saat hızı} \times 10^6 / \text{CPI} \times 10^6} \\ \text{Çalışma Süresi} &= \frac{\text{Komut sayısı}}{MIPS \times 10^6} \end{aligned}$$

bu eşitliğe göre hızlı makinenin MIPS değeri yüksektir diyebiliriz.

#### MIPS Ölçümünü Kullanmanın Sakıncaları

- Aynı iş kullanılan komut sayıları farklı olacağından farklı komut takımlarına sahip bilgisayarları MIPS kullanarak karşılaştıramayız.
- Aynı bilgisayar da çalıştırılan farklı programlar farklı MIPS değerleri verir. Bir makinenin tek bir MIPS değeri olamaz.

Bazı durumlarda MIPS gerçek performansa ters yönde değişebilir.

#### 4.3.2.MFLOPS ile Başarım Ölçümü

MFLOPS saniyede milyon kayan noktalı işlem anlamına gelir. Her zaman “megaflops” diye okunur.

$$MFLOPS = \frac{\text{Bir programdaki kayan noktalı işlemler sayısı}}{\text{Yürütme süresi} \times 10^6}$$

MFLOPS programa bağlıdır. Komutlar yerine aritmetik işlemlerin üzerinde tanımlandığından, MFLOPS farklı makineleri karşılaştırmada daha iyi bir ölçüt olma eğilimindedir. Ancak, farklı makinelerin kayan noktalı işlem takımları birbirine benzemez ve gerçekte aynı iş için gereken kayan noktalı işlem sayısı her makinede farklı olabilir.

#### 4.3.3.BENCHMARK

MIPS ve MFLOPS yanıtıcı başarımlar ölçütleridir. Bir bilgisayarın başarımlarını ölçmek için, "benchmark" adı verilen bir grup karşılaştırma programını kullanarak değerlendirilir.

- Karşılaştırma programları kullanıcının gerçek iş yükünün vereceği başarımlar tahmin edecek iş yükünü oluşturur.
- En iyi karşılaştırma programları gerçek uygulamalardır, ancak bunu elde etmek zordur.

Seçilen karşılaştırma programları gerçek çalışma ortamını yansıtmalıdır. Örneğin; tipik bazı mühendislik yada bilimsel uygulama mühendis kullanıcıların iş yükünü yansıtabilir. Yazılım geliştirenlerin iş yükü ise, çoğunlukla derleyicidir, belge işleme sistemleri, vb. –den oluşur.

Benchmark sonuçları rapor edilirken, makinelerin karşılaştırma ölçümleri ile birlikte şu bilgilerde listelenmelidir.

- İşletim sisteminin sürümü
- Kullanılan derleyici
- Programa uygulanan girdiler
- Makine yapısı(bellek, giriş/çıkış hızı, vs)

Daha yüksek başarımlar elde edilen makine sisteminin belirlenmesinde;

<b>Donanım</b>	
Model no	Powerstation 550
CPU	41.67 MHz POWER 4164
FPU	Tümleşik
CPU sayısı	1
Önbellek Boyutu	64k veri, 8k komut
Bellek	64 Mb
Disk alt sistemi	2-400 SCSI
İletişim ağı arayüzü	Yok
<b>Yazılım</b>	
O/S tipi	AIX v3.1.5
Derleyici sürümü	AIX XL C/6000 ver 1.1.5 AIX XL Fortran ver 2.2
Diğer yazılım	Yok
Dosya sistemi tipi	AIX
Bellenim seviyesi	YOK
<b>Sistem</b>	
Uyum parametreleri	Yok
Art alan yükü	Yok
Sistem durumu	Çok kullanıcı (tek kullanıcı login)

**Tablo 4.3.** Daha yüksek başarımlar sonucu elde edilen makine sisteminin betimlenmesi

## **SONUÇ**

- Doğru başarımlı ölçüsü üç parametreyi: komut sayısı, CPI, ve saat hızı-nı şu şekilde içermelidir

$$Yürütme Süresi = \frac{Komut\ sayısı \times CPI}{Saat\ hızı}$$

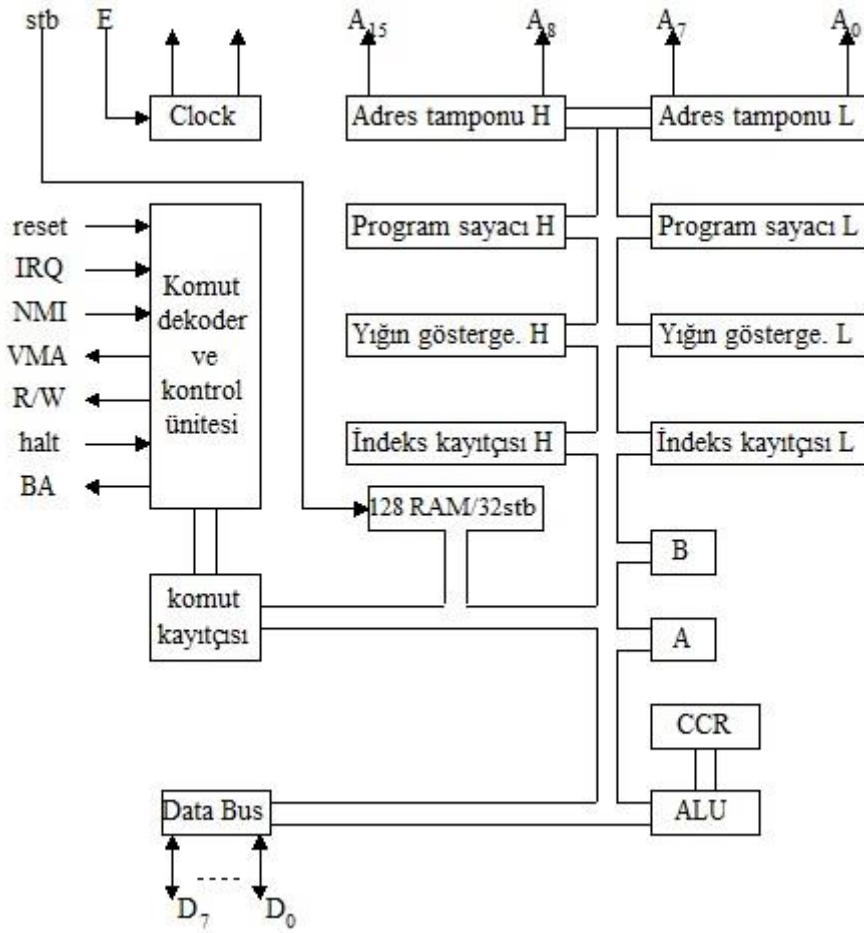
- Bir tasarım farklı yönlerinin bu anahtar parametrelerin her birini nasıl etkilediğini anlamamız gerekir: Örneğin,
  - Komut takımı tasarımı komut sayısını nasıl etkiler,
  - Ardışık düzen ve bellek sistemleri CPI değerini nasıl etkiler,
  - Saat hızı teknoloji ve organizasyona nasıl bağlıdır.
- Sadece başarıma bakmamız yetmez, maliyeti de düşünmemiz gerekir. Maliyet şunları kapsar:
  - Parça maliyeti
  - Makineyi yapacak iş gücü
  - Araştırma ve geliştirme giderleri
  - Satış, pazarlama, kar, vs.

## **4.4. Bölüm Kaynakları**

1. M. Bodur, “RISC Donanımına GİRİŞ”, Bileşim Yayınevi

## BÖLÜM 5. MİKROİŞLEMCİ PROGRAMLAMA

### 5.1. MC6802 Mikroişlemcisinin Yapısı Ve Kayıtları



Şekil-5.1. 6802 mikroişlemcisinin yapısı.

Mikroişlemci resetlendiğinde veya enerji verildiğinde adres çıkışı ilk anda;  $A_0 = 0$ ,  $A_1, \dots, A_{15} = 1$  olur. Yani FFFE dir. Bu adresteki bilgi otomatikmen program sayacına (PC) yüklenir ve program sayacındaki bu yeni adres değerinden itibaren mikroişlemci çalışmasına başlar. Yani;  $[FFFE] = 80$  ve  $[FFFF] = 00$  varsa bu değerler program sayacına yüklenir  $PC = 8000$ . Artık  $[8000]$  adresindeki programa göre mikroişlemci çalışır.

**Program sayacı (program counter, PC) :** Adres ucu kadar bite (16 bit) sahip kayıttır. O anda çalışacak olan komutun adresini üzerinde bulundurur. Komut çalıştırıldıktan sonra değerini bir artırır.

**Komut kayıtları (instruction register, Ir) :** O anda çalışan komutu üzerinde bulundurur. 8 bitlidir.

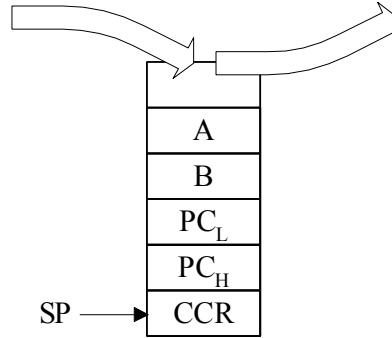
**Komut kod çözücü (instruction decoder) :** Komut kayıtlarından gelen bilgileri, kontrol sinyalleri oluşturacak şekilde kodlar.

**Akümülatör (accumulator, birikeç, A, B) :** A ve B olmak üzere iki tanedir. Data ucu kadar bite sahiptir (8 bit). ALU tarafından kullanılırlar. Genelde o andaki dataları veya işlem sonuçlarını üzerinde bulundurlar.

**İndis kayıtcısı (index register, X) :** 16 bitliktir. Kullanılacak gerçek hafıza yerini belirlemek için bu kayıtcı içindeki değer, komutla belirtilen adrese eklenir.

**Yığın göstergesi (stack pointer, SP) :** 16 bitliktir. Hafızadaki herhangi bir hücre adresini üzerinde bulundurur. Herhangi bir dallanma alt programlara gitme ve kesme istekleri anında mikroişlemcinin o andaki bilgilerini dönüş anında kullanmak üzere saklamak gerekir. Bunun içinde geçici olarak yığın göstergesinin RAM üzerinde göstermiş olduğu adresten geriye doğru bir data yığını oluşturulur. SP ise bu data yığınının oluşturulacağı adres başlangıcını üzerinde tutar. Yığına son atılan bilgi ilk alınır. Yığının kapasitesine bağlı olarak içi içe dallanmalar yapılabilir. Eğer yığının kapasitesi yetersiz ise yığın taşması (stack overflow) problemi ortaya çıkar.

- Yığından bir okuma/yazma yapılacaksa, SP'nin işaret etmiş olduğu hafıza hücresinden okunur/yazılır.
- SP'nin değeri mikroişlemci tarafından otomatik olarak arttırılır ya da azaltılır.
- Yığın türleri:
  - LIFO (Last-In First-Out): Yığına son atılan bilgi ilk alınır.
  - FIFO (First-In, First-Out) : Yığına ilk atılan bilgi ilk alınır.
- Bir PUSH komutuyla veri, yığına atılırken, PULL komutu ile veri yığından alınır.



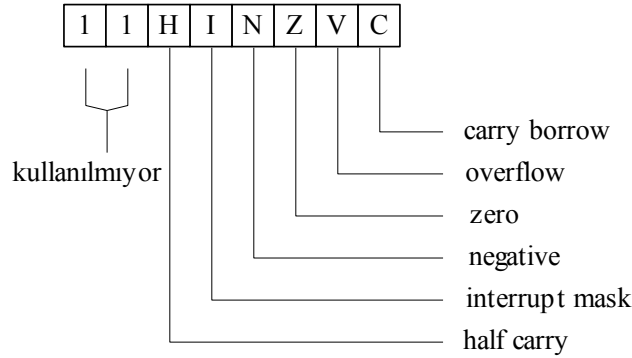
RAM üzerinde

**Şekil-5.2. Yığın**

- Yığın çok düzeyli kesmelerin kolayca gerçekleşmesini, sınırsız sayıda alt programın iç içe geçirilebilmesini ve birçok veri işleme türlerinin basitleştirilmesini sağlar.
- Mikroişlemcinin ana programdan alt programa gittiği zaman ana programa geri döneceği adresi sakladığı adres gözünün adresini içerir. Ana programdan alt programa gidildiği zaman PC' de o anda ana program komut satırının adresi vardır.

**Durum kod kayıtcısı (condition code register, CCR) :** ALU ile birlikte çalışır. Bayrak kaydedicisi, bütün mikroişlemcilerde olduğu gibi, tipine bağlı olarak 8-bit, 16-bit ve 32-bit olmak üzere, bir işlemin sonunda sonucun ne olduğunu kaydedici bitlerine yansıtan bir bellek hücresini oluşturur. Bu kaydediciye bayrak denmesinin sebebi, karar vermeye dayalı komutların yürütülmesinde sonuca göre daha sonra ne yapılacağını bit değişimiyle bu kaydedicinin 1-bitlik hücrelerine yansıtmasıdır. Kaydedicideki bitlerin mantıksal 1 olması bayrak kalktı, mantıksal 0 olması bayrak indi anlamındadır. Karşılaştırma ve aritmetik komutların çoğu bayraklara etki eder.

MC6802 Mikroişlemcisinin CCR kayıtcısı 8-bitlik olup, üzerinde şu bilgiler bulunur:



**Şekil-5.3.** Durum kod kayıtcısı bayrakları.

**Z-biti :** İşlem sonucu sıfır ise bu bit lojik-1 olur.

**N-biti :** İşlem sonucu negatif ise bu bit lojik-1 olur.

**H-yarım elde biti :** Yapılan toplam işlemi sonucunda elde biti oluşmuş ise bu bit lojik-1 olur.

**V-taşma biti :** Eğer bir elde biti varsa ve daha sonra yapılan işlem sonucunda tekrar elde biti oluşmuş ise bu bit lojik-1 olur.

**C- borç elde biti :** Bir çıkarma işleminde çıkan sayı çıkarılan sayıdan büyük ise borç alınır. Bu durumda bu bit lojik-1 olur.

**I-kesme biti :** Bu bit lojik-0 ise gelen IRQ kesme isteklerine izin verilir. Eğer bu bit lojik-1 ise gelen IRQ kesme isteklerine izin verilmez.

### 6802 Mikroişlemcisinin Kontrol Sinyalleri

- IRQ (interrupt request) : kesme isteği
- Reset : al baştan
- Halt : duraklatma kesmesi
- NMI (non-maskable interrupt) : maskelenmeyen kesme
- R/W : oku/yaz
- VMA (valid memory address) : geçerli adres ucu

- 3 durumlu kontrol
- BA (BUS available) : yol kullanılabilir.
- DATA BUS enable

**Halt :** Bu uç lojik-0 olduğunda 6802 son komutunu (en son yaptığı işlemi) tamamlar ve çalışmasını durdurur. Bu durumda adres yolu bir adresin komutunu gösterir. BA lojik-1 olur. VMA ucu ise lojik-0 olur. Eğer kesme (halt) işlemi yapılmayacaksa, halt ucu +5 volta bağlanmalıdır.

**R/W :** Lojik-0 ise hafızaya yaz. Lojik-1 ise hafızadan oku manasındadır. Bir çıkış ucu olup, hafıza ve giriş/çıkış ünitelerine yazmak ve okumak için kullanılır.

**VMA :** Adres hatları üzerindeki bilginin adres bilgisi olup olmadığını belirlemeye yarar. Buda bir çıkış ucu olup gerekli çevre birimlerle bağlantısı yapılmalıdır. Çevre birimler; hafıza, giriş/çıkış ünitesi ...v.s. olabilir.

**BA :** Bu çıkış data ve adres yollarının mikroişlemci dışındaki kullanıcılar için kullanmaya uygun olduğunu belirler. Örneğin halt kesmesi gelince o andaki adresteki bilgilerin kullanılabileceğini gösterir.

**Reset :** Bu uç lojik-0 yapıldığında program FFFE ve FFFF adresindeki yazılı olan adrese dallanır ve FFFE ile FFFF nin gösterdiği adres mikroişlemci programının başlangıç adresidir. Yani mikroişlemci her çalışmasında bu adrese göre çalışacaktır.

**NMI :** Bu uç lojik-0 olunca mikroişlemcinin o andaki bilgileri yığın göstergesi vasıtasıyla saklanır. NMI lojik-0 olunca kesme anında yapılması gereken işler için FFFC, FFFD adresinde belirtilen adresteki programa dallanır. O adresteki program bitince tekrar çalışmasına kaldığı yerden devam eder. Yani uca gelen kesme beklemeden devreye girer.

**IRQ :** CCR kayıtçısında belirtilen kesme (I) biti ile denetlenir. Eğer I biti sıfır ise gelen kesme isteğine cevap verilir. Bu bit lojik-1 ise kesme isteği geçersizdir. Bir kesme başlamışsa bir diğer kesmeye izin vermemek için bu bit lojik-1 yapılmalıdır. Mikroişlemcinin gerekli bilgileri yığın göstergesi yardımıyla saklanır ve sonra FFF8, FFF9 adreslerinde yazılı olan adresteki programa dallanır.

**SWI :** Diğer kesmelerin aksine bir yazılım kesmesidir. Bu kesme geldiğinde FFFA, FFFB adreslerinde belirtilen adresteki programı çalıştırır. Çoğu mikrobilgisayarda bu kesme geldiğinde monitörü durdurucu ve aynı zamanda mikrobilgisayarı duraklatma işlevini yapan bir program çalıştırılır.

FFF8	IRQ	H	H: yüksek değerklikli byte
FFF9	IRQ	L	
FFFA	SWI	H	L: düşük değerklikli byte
FFFB	SWI	L	
FFFC	NMI	H	
FFFD	NMI	L	
FFFE	RESTART	H	
FFFF	RESTART	L	

Şekil-5.4. Kontrol sinyallerinin adres yerleşimi.

## 5.2. MC6802 İle Gerçekleştirilmiş Mikrobilgisayar

### A) MİNİMUM 6802 MİKROBİLGİSAYAR DEVRESİ

“Minimum mikrobilgisayar = CPU + Hafıza + Giriş/Çıkış Ünitesi” den oluşur.

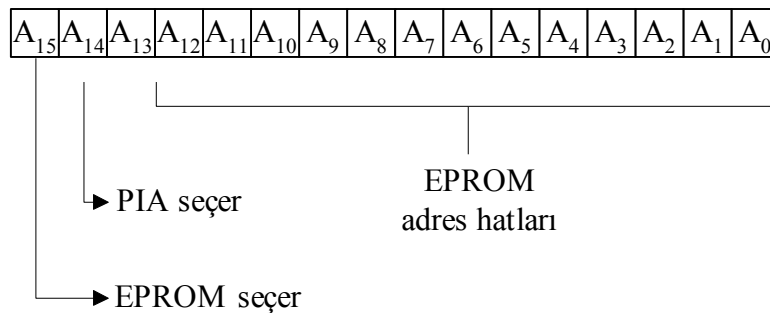
**Örnek:**

CPU → 6802-1Mhz,

Hafıza → 2764-8KB Eprom

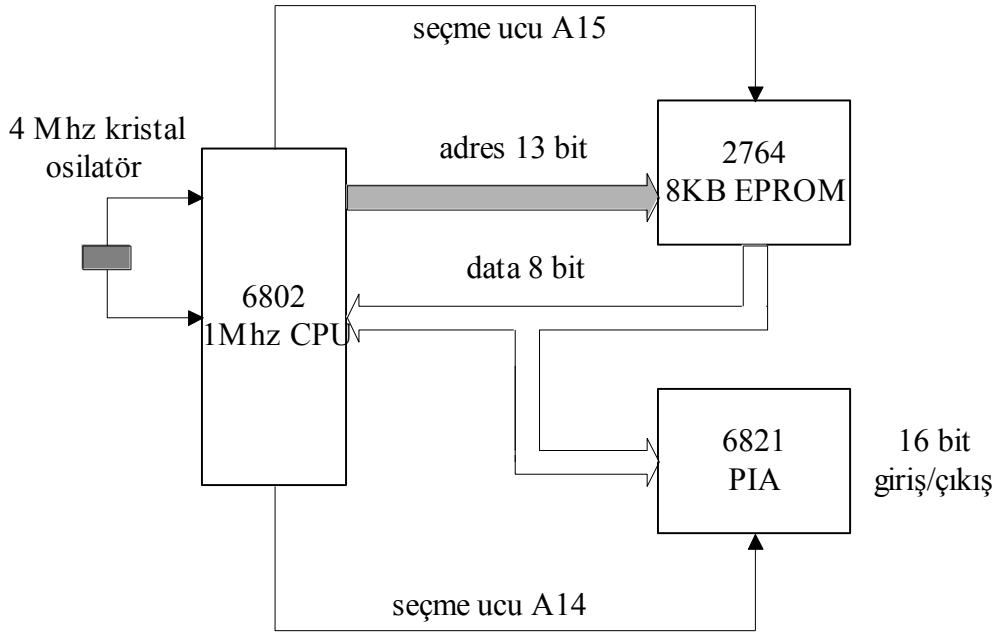
Giriş/Çıkış → 6821 PIA olsun.

Bunun blok şemasını çizmek istersek, CPU nun 16 bit adres hatlarını aşağıdaki gibi dağıtmamız mümkündür.



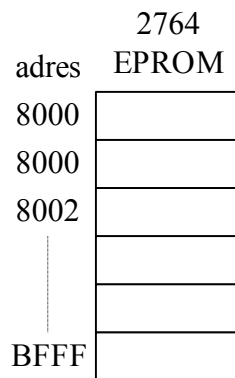
Şekil-5.5. CPU adres hatlarının dağılımı.





Şekil-5.6. Minimum mikrobilgisayar blok diyagramı.

- Giriş / çıkış ünitesi olarak PIA (paralel giriş/çıkış) kullanılmıştır. Kullanılan MC6821 PIA'nın 2 adet 8 bitlik giriş/çıkış portu vardır.
- MC6802 mikroişlemcisinin 16 adres ucu vardır. Bunun 13 adetini 2764 Epromu için, 2 tanesi de eprom ve PIA'yı seçme işlemi için kullanıldı.  
 $A_{15} = 1, A_{14} = 0$  ise EPROM seçilir.  
 $A_{15} = 0, A_{14} = 1$  ise PIA seçilir.
- Seçme uçlarında dikkat edilmesi gereken, herhangi bir anda sadece bir elemanın seçilmesidir.
- EPROM'un herhangi bir hafıza bölgesinden bilgi okuyacaksa,  $A_{15}$  adres ucunu aktif yapacak şekilde adres vermeliyiz.



Şekil-7.3. EPROM seçimi ve adresleri.

- PIA üzerinden giriş yada çıkış işlemi yapacaksa, A<sub>14</sub> ucu aktif yapacak şekilde bir adres vermeliyiz. Örneğin; 4000.
- MC6802 mikroişlemcisinin içerisinde 128 byte lik bir RAM vardır. Bu RAM 0000<sub>H</sub> adresine yerleştirilmiştir. Yani 128 bytelik RAM ın adres aralığı 0000-007F dir ve 7 adet adres ucu vardır. Bu RAM ın ilk 32 bytelik kısmı stand-by (stb) özelliği vardır. Bu özellik; mikroişlemcinin enerjisi kesilse bile dışardan bir pil ...v.s. ile stb li kısım beslenerek üzerindeki bilginin silinmesini engeller. Stb gerilimi 4.5 V + %10 dur.
- MC6802 mikroişlemcisinin içinde bir clock puls ünitesi vardır. Bu ünitenin çalışması için uçlarına bir kristal osilatör bağlamak gerekir. Bağlanacak olan bu osilatörün frekansı mikroişlemcinin frekansının 4 katı olmalıdır. MC6802-1 Mhz için 4Mhz lik, MC6802-2 Mhz için 8Mhz lik bir kristal osilatör bağlanmalıdır.
- MC6802 mikroişlemcisi 8 bit data bus, 16 bit adres ucu ile 74 assembly komuta sahip bir mikroişlemcidir.

### 5.3. Assembly (Birleştirici) Dil Kuralları

Bir assembly satırı şu şekildedir :

Etiket Alanı	Komut Alanı	Veri Alanı	Açıklayıcı Bilgi
--------------	-------------	------------	------------------

#### Etiket Alanı

Etiket kullanılmıyorsa, komut bir sütun içerden başlanmalıdır

İlk sütunda \* varsa bundan sonraki bilgilerin açıklama olduğunu gösterir.

Eğer etiket varsa şu kurallara uyulmalıdır :

Etiket 1 ile 6 karakter uzunluğa sahip olabilir.

İlk karakter sayı ve rakam olmamalıdır

İlk sütundan başlanılmalıdır.

Etiket program boyunca aynı isimden bir tane olmalıdır.

Bir etiket şu durumlarda kullanılır.

Herhangi bir şartlı dallanma komutu ile gidilecek yeri belirlemek.

Herhangi bir şartsız dallanma komutu ile gidilecek yeri belirlemek.

Herhangi bir alt programa gitmek için.

#### Komut Alanı

Bu alanda ilgili mikroişlemcinin assembly komutları bulunabilir.

Komut (assembly)	Operasyonel kod (op-code)	İkilik sistemde
CLR A	4F	0100 1111
CLR B	5F	0101 1111

MC6802 mikroişlemcisinin 74 komutu bulunmaktadır. Bu komutları dört guruba ayırabiliriz.

**a- 8 bitlik kayıtlar ile yapılan işlem komutları**

İki işlemli aritmetik : ABA, ADD,...

Tek işlemli aritmetik : CLR A, INC B, DEC A, ...

Karşılaştırma ve test etme : CMP, BIT A, ....

Kaydırma ve döndürme : ASL, ROR, ...

Mantıksal fonksiyonlar : EOR, AND, COM, ....

Yükleme ve depolama işlemleri : LDA, STA, LDX, STS,

Transfer : TAB, TBA, TXS, ...

**b- Atlama ve dallanma komutları**

Şartlı dallanma : BNE, BEQ,....

Şartsız dallanma : BRA, JMP, ....

Alt programlara gitme : BSR, JSR, ...

Kesme işlemlerinde belirtilen yerlere gitme : WAI, ...

**c- İndis kayıtları ve yığın göstericisi kontrolü yapan komutlar**

indis kayıtları ile ilgili işlemler : LDX, INX, STX, DEX, CPX, ...

Yığın göstericisi ile ilgili işlemler : STS, LDS, INS, ....

Transfer işlemleri : TSX, TXS, ....

**d- Durum kod kayıtları kontrolü yapan komutlar**

Bit kontrolü : CLI, SEI, CLC, ....

Byte kontrolü : TAP, TPA,

**Veri Alanı**

Adresleme modlarına göre bilginin girildiği alanlardır. Bu bölgedeki bilgilere göre komutlar 1, 2 veya 3 byte'lık olmaktadır. Eğer bir komut 2 veya 3 byte'dan oluşmakta ise 2. veya 2. ve 3. byte'lar bir işlem, bir adres veya adres elde etmekte kullanılacak bilgiyi içerir. Kısaca işlemler ve karakterler veya karakter dizileri bu bölgede bulunabilir. Bu bölgede bulunan bilgiler assembly derleyicisi tarafından şu şekilde anlaşılır :

Sayıların Temsili	Derleyici Tarafından Anlaşılması
sayı	Desimal –10
\$sayı sayıH	Hekzadesimal – 16
@sayı sayıO sayıQ	Oktal – 8
%sayı sayıB	Binary – 2

İşaretler	Anlamı
# (diyes)	Kendisinden sonra gelenin data olduğunu gösterir ve immediate adreslemede kullanılır.
+, -, *, /	Normal matematiksel işlemler için kullanılır (üst seviyeli programlama dillerinde : C, pascal, fortran, ...v.s.).

**Örnek :** 60<sub>H</sub> ile 61<sub>H</sub> adresindeki bilgileri toplayıp 62<sub>H</sub> adresine yazan bir assembly program yazınız.

Assembly Dil	Operasyonel kod	Makine dili
LDA A 60 <sub>H</sub>	96	1001 0110
	60	0110 0000
ADD A 61 <sub>H</sub>	9B	1001 1011
	61	0110 0001
STA A 62 <sub>H</sub>	97	1001 0111
	62	0110 0010

Programın makine kodundaki halini anlamak oldukça zordur. Hekzadesimal haldeki operasyonel kodlarında kullanıcı tarafından anlaşılması zordur. Bu nedenlerden dolayı daha anlaşılır olan assembly dili kullanılır. Daha sonra program EPROM a veya mikroişlemciye verilirken operasyonel kodlara çevrilerek verilir.

#### 5.4. MC6802 Mikroişlemcisinin Komut Kümesi ve Adresleme Modları

31 20

##### A) KOMUT KÜMESİ

6802 mikroişlemcisinin komut kümesi bir, iki veya üç bytelik komutlardan oluşur. Bir komutun uzunluğu komuta ve adresleme çeşidine bağlı olup, ilk (veya tek) byte komutu ve kullanılan adresleme çeşidini belirlemeye yeterlidir. 6802 mikroişlemcisinin 74 komutu bütün

geçerli adresleme çeşitleri için onaltılık tabanda operasyonel kodları EK-1,2 deki tablolar da verilmiştir. Kullanılabilecek 256 (00...FF) onaltılık sayıdan 197 si geçerli birer makine kodu olduğu, 56 sının ise kullanılmadığı verilen tablodan görülebilir.

## B) ADRESLEME MODLARI

LDA A 60<sub>H</sub> gibi bir assembly satırında; 60<sub>H</sub> bilgisini A akümülatörüne yüklenecek, yoksa 60<sub>H</sub> adresindeki bilgimi A akümülatörüne yüklenecek? İşte bunun gibi durumları ayırt etmek için adresleme modlarına gerek vardır.

6802 mikroişlemcisinde 7 adresleme modu kullanılabilmektedir.

- Immediate (hazır, hemen, derhal, anında, ivedi, öncel) adresleme
- Relative (bağlı, dolaylı, göreceli) adresleme
- Inherent (doğal, anlaşılır, içerilmiş) adresleme
- Indexed (indisli, sıralı) adresleme
- Akümülatör adresleme (Anlaşılır adreslemenin özel bir durumudur)
- Extended (genişletilmiş, mutlak) adresleme
- Direct (doğrudan) adresleme

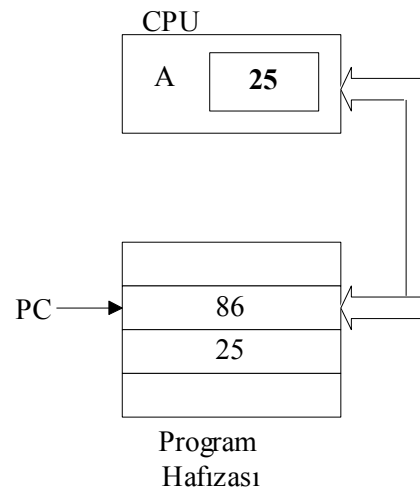
Bu adresleme modları mikroişlemciler için temelde aynı olmasına rağmen bazı değişiklikler gösterir.

### 1. Anında adresleme

Bu yöntemde işlenecek olan bilgi, komutun 2. byte dında yer alır. Anında adreslemeyi assembly dil yazılımında belirlemek için verinin önüne ‘#’ işareti konur.

#### Örnek :

Assembly	op-code	yaptığı iş
LDA A #\$25	86 25	A = 25 <sub>H</sub>
LDX #\$1000	CE 10 00	X = 1000 <sub>H</sub>
LDA A #45	86 45	A = 45

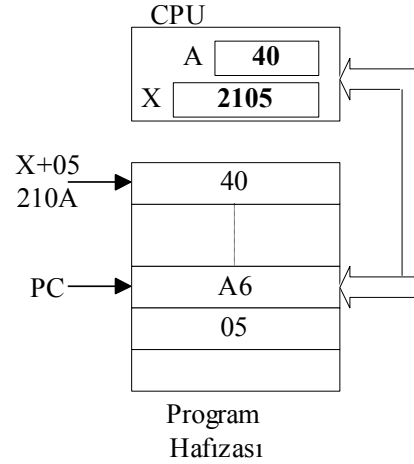


## 2. İndisli adresleme

İndis kayıtçısı kullanılarak adreslemenin yapılmasından dolayı, bu yönteme indisli adresleme yöntemi denilmiştir. Komuttan sonra gelen sayı indis kayıtçısındaki sayıya eklenerek, gerçek data adresi belirlenir. Belirlenen bu adresten data okunur veya yazılır. İndisli adresleme yöntemini belirtmek için komut ve datadan sonra 'x' yazılır.

### Örnek :

Assembly	op-code	yaptığı iş
LDA A \$05,X	A6 05	A = [X+05]

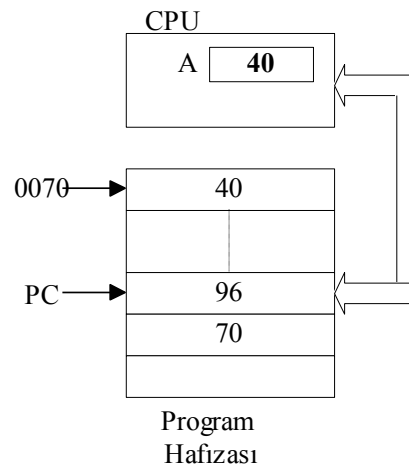


## 3. Doğrudan adresleme

Doğrudan adresleme yönteminde komutun operasyonel kodundan sonra işlenecek olan verinin bulunduğu adres yazılır. Bilindiği üzere 16 bitlik adresler 0000....FFFF arasındadır. Doğrudan adreslemede adres 8 bit kullanılarak, 8 bitlik adreslere ulaşılırken fazladan yer kaplamamak için 0000....00FF arasındaki adresler, 00....FF şeklinde kullanılmaktadır. Yani doğrudan adreslemede komuttan sonra gelen adres değeri bir bytedir.

### Örnek :

Assembly	op-code	yaptığı iş
LDA A \$70	96 70	A = [0070]
LDX \$70	DE 70	X = [0070,0071]

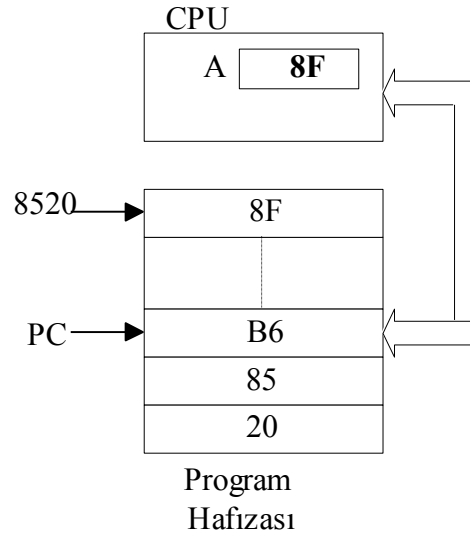


## 4. Genişletilmiş Adresleme

Bu adresleme, doğrudan adreslemenin genişletilmiş bir şekli olup \$0000....\$FFFF arasında tüm durumlara erişilmesini sağlar. Bu durumlar iki bytelik olduğu için genişletilmiş adresleme komutları 3 byteden oluşur.

### Örnek :

Assembly	op-code	yaptığı iş
LDA A \$45	96 45	A = [0045]
LDA A \$0045	B6 0045	A = [0045]
LDX \$8520	FE 8520	X = [8520,8521]
LDA A \$8520	B6 85220	A = [8520]



### 5. Anlaşılır adresleme

Bu adreslemede işlenecek olan veri (data, bilgi) komutun kendisi ile birlikte verilir. Böylece işlenecek olan bilgi herhangi bir bellek bölgesinde aranmaz. Bu şekilde 6802 mikroişlemcisinde 25 komut vardır.

### Örnek :

Assembly	op-code	yaptığı iş
ABA	1B	A = A + B
CLC	0C	C = 0

### 6. Akümülatör adresleme

Bu yöntem akümülatörün işlenen bilgiyi içerdiği, anlaşılır adreslemenin özel bir durumudur. 6802 mikroişlemcisinde 13 tane komut bu yöntemle adreslenebilmektedir. Bu adresleme doğrudan komuttan sonraki A ve B harfleri ile A akümülatörü veya B akümülatörü şeklinde tanımlanır.

### Örnek :

Assembly	op-code	yaptığı iş
CLR A	4F	A acc. sil.
COM A	43	A = A'

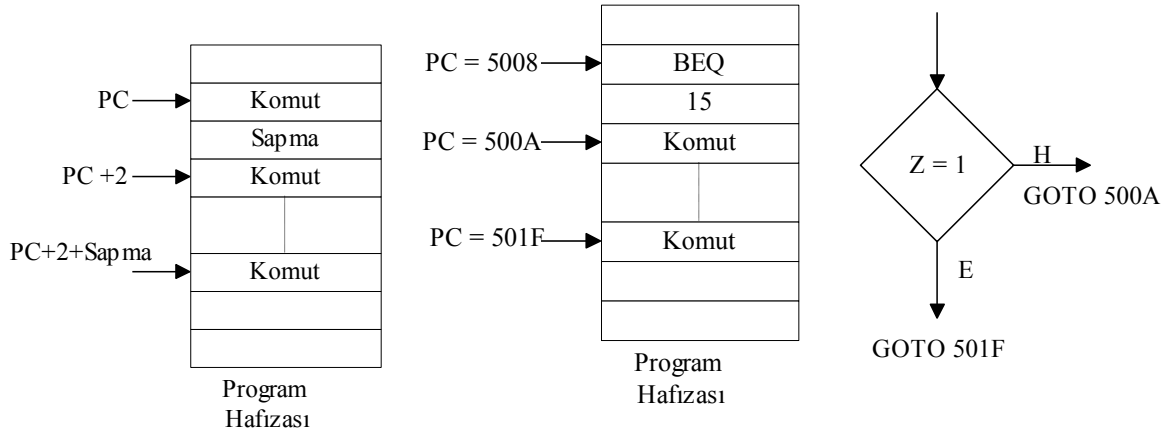
### 7. Relatif Adresleme

Sadece dallanma komutlarında kullanılan bu adresleme türünde, ulaşılması gereken adres program sayıcısının o andaki içeriğine bağlı olarak bulunur. Dallanma komutları iki byte den oluşur. Birinci byte işlem bytedir. İkinci byte ise öteleme (salınım, sapma, offset, dallanma) byte olup, program sayacına eklenir.

Bir dallanma sırasında ileriye gidilebileceği gibi, geriye de gidilmesi gerekebilir. Dolayısıyla öteleme sayısı işaretli bir sayıdır. Öteleme sayısı 8 bitlik bir sayı olup, bununla (00...FF) 256 durum ifade edilir. İşaret dikkate alınınca +127 ileriye (00...7F), -128 geriye (80...FF) gidilebilir. Fakat öteleme sayısı okunduğunda PC dallanma komutunun olduğu yerden 2 ilerisini göstereceği düşünülecek olursa, +129 ileri –126 geri dallanılabilir.

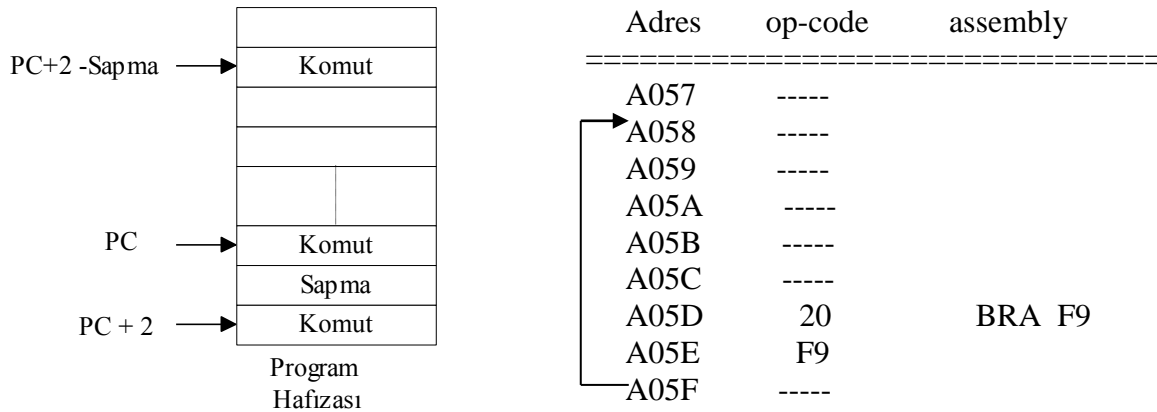
$$(PC + 2) - 128 \leq \text{ÖTELEME MİKTARI} \leq (PC + 2) + 127$$

#### - İleri doğru sapma



#### - Geriye doğru sapma

İleri doğru sapma yöntemiyle aynı prensiplere sahip bu yöntemde tek fark bağıl adresin negatif bir sayı olarak girilmesidir.



Geriye doğru sapma miktarını şu şekilde de bulunabilir :

Sapma komutundan itibaren FF den başlayarak geriye doğru sayılır, FF, FE, FD, FC,.....F9... sapılacak yere gelince durulur.



**Problem:** Aşağıdaki programda DNG1 ve DNG2 sapma miktarlarını 16 lık sayı sistemi tabanında veriniz.

```

DNG1    CLR A
        INC A
        NOP
        CMP A #$10
        BNE DNG1
        BRA DNG2
        DEC A
DNG2    NOP
        WAI

```

## 5.5. MC6802 Assembly Uygulamaları

### A) BİR ASSEMBLY PROGRAMIN YAZILIMDA İZLENECEK YOL

1. Probleme ait giriş/çıkış verileri ile istenen sonuç günlük dilde açık bir şekilde yazılmalı ve akış şeması çıkarılmalı
2. Kullanılacak olan bilgisayarın (mikroişlemcinin) kapasite ve özelliklerinin probleme cevap verip veremeyeceğinin belirlenmesi ve buna uygun mikroişlemci seçiminin yapılması.
3. Programın algoritmada belirtilen kurallara (adresleme modlarına) uygun olarak kodlanması.
4. Arzu edilen sonuçların elde edilip edilmediğinin kontrol edilmesi.

### B) ÖRNEK PROGRAMLAR

1.

<u>Komut</u>	<u>Adresleme Modu</u>	<u>Operasyonel Kodu</u>	<u>Yaptığı İş</u>
ADD A #\$33	Anında	8B 33	$A = A + 33$
ADD A \$33	Doğrudan	9B 33	$A = A + [33]$
ADD A \$0133	Genişletilmiş	BB 01 33	$A = A + [0133]$
ADD A \$06,X	İndisli	AB 06	$A = A + [X + 06]$
ABA	Anlaşılır	1B	$A = A + B$
LDA B #\$FF	Anında	C6 FF	$B = FF$
LDA B \$55	Doğrudan	D6 55	$B = [55]$
STA A \$1235	Genişletilmiş	B7 12 35	$[1235] = A$
INC A	Akümülatör (Anlaşılır)	4C	$A = A + 1$
LDX #\$0011	Anında	CE 00 11	$X = 0011$
LDX \$50	Doğrudan	DE 50	$X = [0050, 0051]$
JMP \$0180	Genişletilmiş	7E 01 80	$[0180]$ adresine git
JMP \$02,X	İndisli	6E 02	$[X + 02]$ adresine git

DEX	Anlaşılır	09	$X = X - 1$
CMP A #\$50	Anında	81 50	A ile 50 i karşılaştırır
CMP \$50	Doğrudan	91 50	A ile [0050] i karşılaş.
CMP A \$0250	Genişletilmiş	B1 02 50	A ile [0250] i karşılaş.
ADC A #\$05	Anında	89 05	$A = A + C + 05$
AND A #\$1C	Anında	84 1C	$A = A \times 1C$
ASL A	Akümülatör (Anlaşılır)	48	A sola kaydırılır
ASL \$0D80	Genişletilmiş	78 0D 80	[0D80] sola kaydırılır
ROR \$004E	Genişletilmiş	76 00 4E	[004E] sağa döndürülür
NOP	Anlaşılır	01	2 cp zaman geçiktirir
BCC xx	Relatif	24	$C = 0$ ise xx kadar sapar

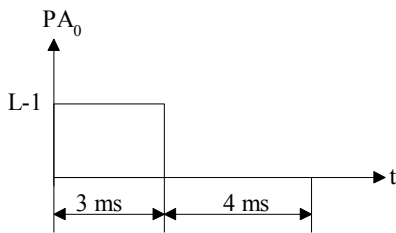
2. 55 adresinde CA sayısı ve B akümülatöründe 13 sayısı vardır.

ADD B #\$55	Anında	CB 55	$B = B + 55 = 68$
ADD B \$55	Doğrudan	DB 55	$B = B + [0055] = 13 + CA = DD$

3. Aşağıdaki programın çalışması sonucunda A akümülatöründe hangi sayı vardır.

LDA A #\$22	86 22	$A = 22$
STA A \$01C3	B7 01C3	$[01C3] = A = 22$
LDX #\$0123	CE 01 23	$X = 0123$
ADD A \$A0,X	AB A0	$A = A + [X + A0] = 22 + [0123 + A0] = 22 + [01C3] = 22 + 22 = 44$

4. 8000<sub>H</sub> adresine yerleştirilmiş bir PIA nın PA<sub>0</sub> ucundan aşağıdaki şekilde bir kare dalga alınmak isteniyor. Gerekli assembly programı yazınız.



```

zz LDA A #$01
STA A $8000
LDX #$0000
xx INX
NOP
CPX #$00E7
BNE xx
CLR A
STA A $8000
LDX #$0000
yy INX
NOP
CPX #$0134
BNE yy
BRA zz

```

**Ödev :** Yandaki programın operasyonel kodlarını çıkarınız



xx etiketindeki 3 ms'lik döngünün değeri, bu döngü içerisinde bulunan komutlarının çalışma sürelerinden çıkarılır :

INX → 4 + NOP → 2 + CPX → 3 + BNE → 4 = 13 cp; CPU → 1 Mhz ise 13cp = 13 µs

xx döngü miktarı = 3ms/13 µs = (231)<sub>10</sub>=(00E7)<sub>16</sub>

Benzer şekilde 4ms lik yy döngüsünün değeri :

yy döngüsü miktarı = 4 ms / 13 µs = (308)<sub>10</sub> = (0134)<sub>16</sub>

**5. Kopyalama programı :** 1000<sub>H</sub>...2000<sub>H</sub> arasındaki bilgileri 3000<sub>H</sub>...4000<sub>H</sub> arasına kopyalayan bir assembly program yazınız.

	LDX #\$1000	CE 10 00	X = 1000
	STX \$80	DF 80	[80] = X = 1000
	LDX #\$3000	CE 30 00	X = 3000
	STX \$82	DF 82	[82] = X = 3000
xx	LDX \$80	DE 80	X = [80]
	LDA A \$00,X	A6 00	A = [X+00]
	INX	08	X = X + 1
	STX \$80	DF 80	[80] = X
	LDX \$82	DE 82	X = [82]
	STA A \$00,X	A7 00	[00 + X] = A
	INX	08	X = X + 1
	STX \$82	DF 82	[82] = X
	CPX #\$4001	8C 40 01	X – 4001 durumu CCR ye setler.
	BNE xx	26 ED	Z = 0 ise xx e dallanır
	SWI	3F	Dur

**6. Aşağıdaki programın çalışması sonucu durum kod kayıtçısındaki H ve V bitlerinin durumu nedir?**

LDA A #\$E0	86 E0	A = E0	CCR = 1 1 H I N Z V C
LDA B #\$09	C6 09	B = 09	CCR = 1 1 1 0 1 0 0 1
ABA	1B	A = A+B = E9	
TAP	06	CCR = E9	H = 1
SWI	3F		V = 0

**7. Bilindiği üzere SWI kesmesi IRQ, NMI kesmelerinin aksine bir yazılım kesmesidir ve birçok mikrobilgisayarda mikroişlemciyi durdurucu bir işlev yapmaktadır. Bir SWI yazılım kesmesi programı yazınız.**

xx NOP	01	Sonsuz döngü
BRA xx	20 FD	

8. Aşağıdaki programın çalışması sonucu indis kayıtçısında hangi sayı vardır?

LDS #007F	8E 00 7F	SP = 007F
LDA A #\$23	86 23	A = 23
LDA B #\$34	C6 34	B = 34
PSH A	36	[007F] = 23
PSH B	37	[007E] = 34
LDX \$7E	DE 7E	X = [007E, 007F] = 3423
SWI	3F	

9. İki tane iki byte lık sayıyı toplamak için assembly program yazınız. Sayılardan biri 80<sub>H</sub> ve 81<sub>H</sub> adreslerine, diğeri ise 60<sub>H</sub> ve 61<sub>H</sub> adreslerinde saklıdır. Sonucu indis kayıtçısına yükleyiniz.

LDA A \$81	96 81	A = [81]
LDA B \$61	D6 61	B = [61]
ABA	1B	A = A + B
STA A \$91	97 91	[91] = A
LDA A \$60	96 60	A = [60]
ADC A \$80	99 80	A = A + C + [80]
STA A \$90	97 90	[90] = A
LDX \$90	DE 90	X = [0090,0091]
SWI	3F	

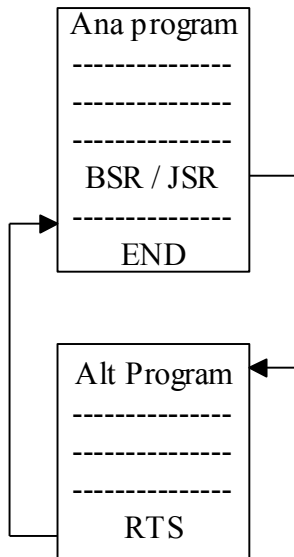
10. Aşağıdaki programın her komutunun çalışmasından sonra A birikecinde bulunan değeri yazınız.

LDA A #\$23	A = 23
AND A #\$F0	A = 20
COM A	A = DF
NEG A	A = 21
DEC A	A = 20
INC A	A = 21
ORA A #\$34	A = 35
ASL A	A = 6A
LSR A	A = 35
SUB A #\$43	A = F2
SWI	

**Ödev :** Aşağıda operasyonel kodları verilen programın assembly dil karşılığını yazarak ne iş yaptığını bulunuz?

CE  
00  
00  
DF  
50  
4F  
5F  
9B  
51  
D9  
50  
08  
DF  
50  
8C  
01  
2C  
26  
F4  
97  
53  
D7  
52  
3F

### C) ALT PROGRAM

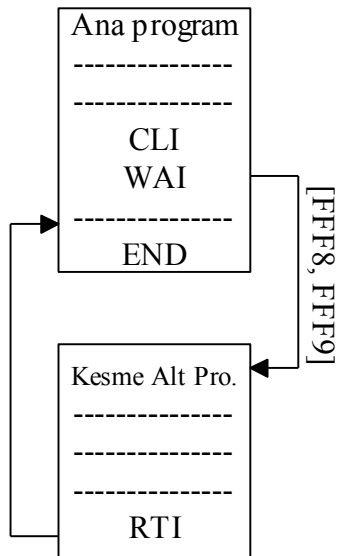


1. Alt programa giderken geri dönülecek adres değeri yığına atılır.
2. Alt program kullanılacaksa yığın göstergesinin (SP) başlangıç adresini ana programın başında mutlaka vermeliyiz ve verilen bu adresin RAM üzerinde olmasına dikkat edilmelidir.

**Örnek :** 0300<sub>H</sub> ....0600<sub>H</sub> adresleri arasına 02 yazan ve bir alt programda ise bunları toplayarak sonucu 0800<sub>H</sub> adresine kayıt eden bir assembly programı operasyonel kodlarıyla birlikte veriniz.

	LDS #\$007F	8E 00 7F
	LDX #\$0300	CE 03 00
	LDA A #\$02	86 02
DONGU1	STA A \$00,X	A7 00
	INX	08
	CPX #\$0601	8C 06 01
	BNE DONGU1	26 F8
	BSR DONGU2	8D 01
	SWI	3F
DONGU2	CLR A	4F
	CLR B	5F
	LDX #\$0300	CE 03 00
DONGU3	ADD A \$00,X	AB 00
	ADC B #\$00	C9 00
	INX	08
	CPX #\$0601	8C 06 01
	BNE DONGU3	26 F6
	STA A \$0801	B7 08 01
	STA B \$0800	F7 08 00
	RTS	39

#### D) KESME ALT PROGRAMI



1. Kesme alt programına giderken geri dönülecek adres değeri yığına atılır.
2. Kesme alt programı kullanılacaksa yığın göstergesinin (SP) başlangıç adresini ana programın başında mutlaka vermeliyiz ve verilen bu adresin RAM üzerinde olmasına dikkat edilmelidir.
3. IRQ kesme alt programı kullanılacak ise FFF8, FFF9 → adreslerine IRQ kesme alt programının başlangıç adresi önceden yazılmalıdır.
4. NMI kesme alt programı kullanılacak ise FFFC, FFFD → adreslerine NMI kesme alt programının başlangıç adresi önceden yazılmalıdır.

5. IRQ kesmesi kullanılacak ise, CCR deki I biti 0 yapılmalıdır. Bir IRQ kesmesi geldikten sonra, ikinci bir IRQ kesmesine izin vermemek için I biti 1 yapılmalıdır.

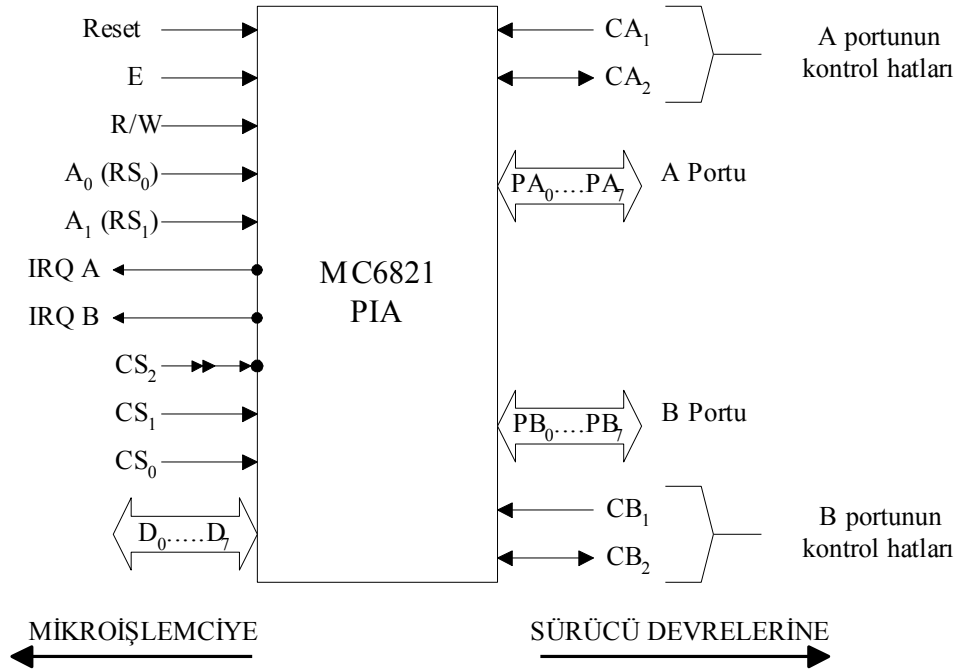
**Örnek :** Bir önceki alt program örneğini IRQ kesme alt programı ile yapınız.

	LDS #\$007F	8E 00 7F
	LDX #\$0300	CE 03 00
	LDA A #\$02	86 02
DONGU1	STA A \$00,X	A7 00
	INX	08
	CPX #\$0601	8C 06 01
	BNE DONGU1	26 F8
	CLI	0E
	WAI	3E
	SWI	3F
[FFF8, FFF9]	CLR A	4F
	CLR B	5F
	LDX #\$0300	CE 03 00
DONGU2	ADD A \$00,X	AB 00
	ADC B #\$00	C9 00
	INX	08
	CPX #\$0601	8C 06 01
	BNE DONGU2	26 F6
	STA A \$0801	B7 08 01
	STA B \$0800	F7 08 00
	RTI	3B

## 5.6. Mikrobilgisayarlarda Giriş/Çıkış İşlemi

### A) PIA (Peripheral Interface Adapter)

Paralel giriş / çıkış portudur. 68 serisi PIA: MC6821. PIA'nın işlev tarzı (G/Ç portlarının ne kadarı giriş, ne kadar bit çıkış olduğu) assembly programının başlangıcında programlanmalıdır. Yani PIA'nın her bir veri yolu giriş veya çıkış olarak programlanabilmektedir.



MC6821 de 3 tane 8 bitlik kayıtçı tipi vardır ;

- 1-) Veri kayıtçısı (Data Register → DRA, DRB) : Bu kayıtçı giriş yada çıkışa ait veriyi saklar.
- 2-) Veri yönü kayıtçısı (Data Direction Register → DDRA, DDRB) : Bu kayıtçıdaki her bir bit, buna karşılık gelen veri kayıtçısındaki bitin giriş yada çıkış olduğunu belirler. DDRA, DDRB de 0 → giriş, 1 → çıkış olarak tanımlıdır.
- 3-) Kontrol kayıtçısı (Control Register → CRA, CRB) : El sıkışma (handshake) için gerekli durum sinyallerini ve mantık bağlantılarını seçen bitleri üzerinde taşımaktadır

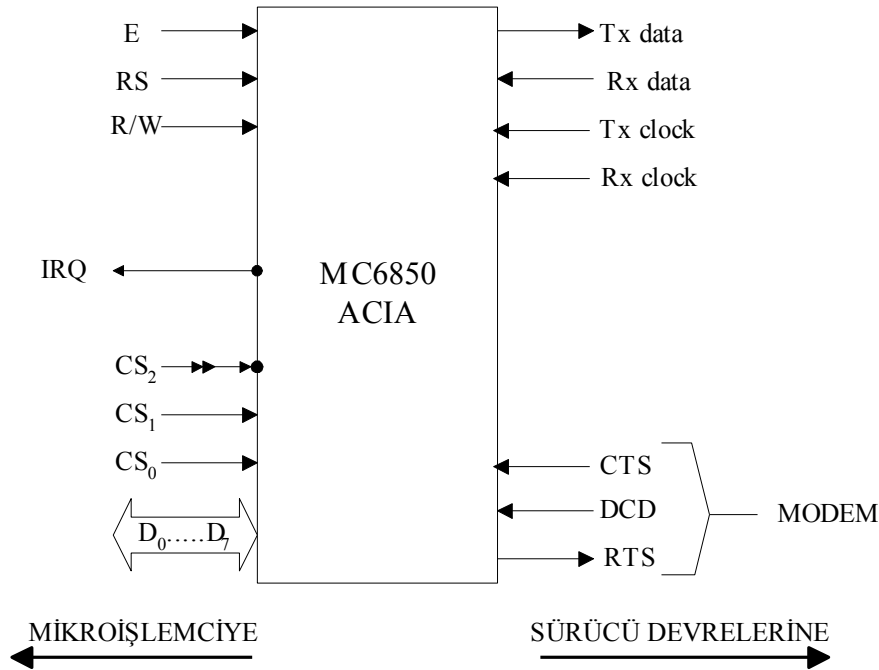
PIA CS sinyalleri ile seçilmektedir. Seçilme yapıldıktan sonra altı kayıtçıdan (DRA-DRB, DDRA-DDRB, CRA-CRB) her birine ulaşılabilir. Altı kayıtçı olmasına rağmen, bunları seçen iki adres yolu (RS<sub>0</sub>-A<sub>0</sub>, RS<sub>1</sub>-A<sub>1</sub>) vardır.

### B) ACIA (Asynchronous Communication Interface Adapter)

Asenkron olarak seri bilgi iletimi gerçekleştirir. 68 serisi ACIA MC6850 entegresidir. ACIA, bir mikroişlemcinin veri hattından aldığı paralel bilgiyi yazıcı, modem, gibi çevre birimlere seri biçimde olarak aktarabilen veya bu birimlerden aldığı seri biçimdeki bilgiyi mikroişlemcinin veri hatlarına paralel olarak verebilen eleman olup, UART (Universal Asynchronous Receiver



Transmitter – Evrensel Asenkron Alıcı Verici) olarak ta bilinir. Veri formatlama ve asenkron seri araçların bağlanabilmesi için kontrol sinyallerine sahiptir. Genel özelliklere ise aşağıdaki şekilde gösterilmiştir.



ACIA veri anayoluna ait paralel veriyi uygun format ve hata kontrolü ile seri olarak gönderebilmekte ve alabilmektedir. Dört tane kayıtçıya sahiptir. İki kayıtçı sadece okunabilir : Receiver (alıcı) ve Status (durum). Diğer ikisi ise sadece yazılabilir : Transmitter (gönderici) ve Kontrol kayıtçıları. Bu kayıtçılara RS ve R/W' sinyallerinin kombinasyonu ile ulaşılabilmektedir.

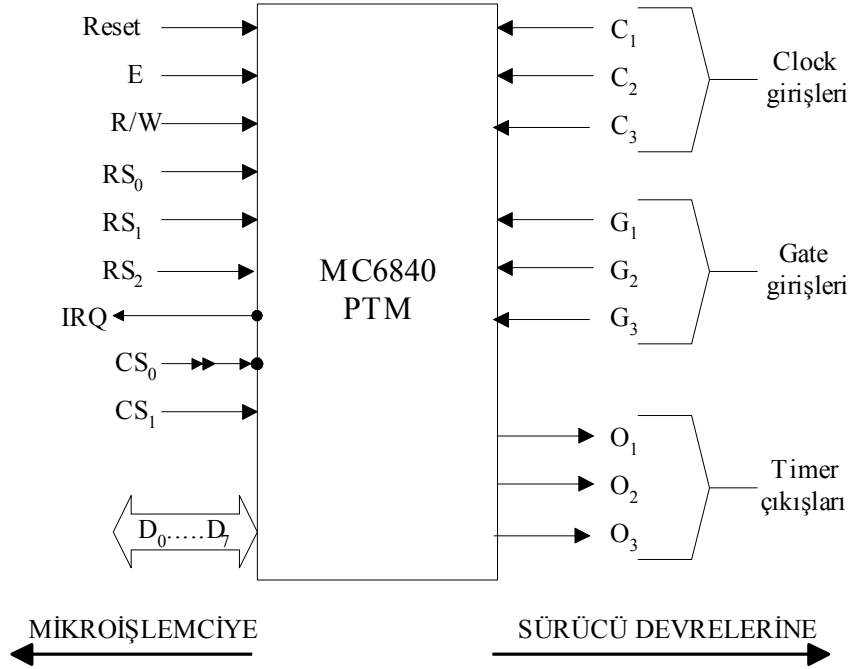
### C) SSDA (Synchronous Serial Data Adapter – Senkron Seri Veri Adaptörü )

Senkron olarak seri bilgi iletimi için kullanılır. 68 serisi olarak MC6852 mevcuttur. Bu birim birçok bakımdan ACIA ya benzer.

### D) PTM (Programmable Timer Module – Programlanabilen Zamanlayıcı Modülü)

Mikroişlemcilerin endüstriyel uygulamalarında çok sık karşılaşılan bir gereksinim belirli olayları başlatma, bitirme ve uygun aralıklarla tekrarlamak için süre ölçmektir. Bir başka gereksinim de belirli olayları takip edip saymak ve belirli sayılarda belirli bazı işlemleri yapmaktır. Her iki işlem içinde mikroişlemci kullanılabilir. Örneğin süre ölçmek için mikroişlemci, başlama işareti ile birlikte bir döngüye sokulur ve durma işaretinde döngüden çıkılır. Döngü kaç defa yürütülmüşse, bir döngüyü yürütmek için gerekli süre o döngüdeki komutların yürütme sürelerinden bulunulabileceğine göre, toplam süre de bulunabilir. Aynı şekilde olay saymak için mikroişlemci bir bekleme durumuna sokulabilir, her bir olayda üretilen bir kesme, belirli bir

saklayıcının içeriğinin bir artırılmasını sağlayabilir. Fakat bu yöntemlerin sakıncası, mikroişlemcinin bu işlemleri yapabilmesi için zamanının büyük bir kısmını ayırması ve hatta bazı durumlarda başka hiçbir iş yapmamasıdır. Dolayısıyla süre ölçme ve olay sayma için genellikle ayrı bir donanım kullanılır. 68 serisinde bu amaçla kullanılabilecek MC6840 zamanlayıcı (timer) entegresi mevcuttur. Aşağıdaki şekilde MC6840 verilmiştir.



MC6840 ta 3 tane 16 bit ikili sistemde sayaç ve bunları kontrol eden üç kayıtcısı ile bir tane durum (status) kayıtcısı vardır. Kayıtcı seçici yolları RS<sub>0</sub>, RS<sub>1</sub>, RS<sub>2</sub> ve R/W yolları ile kayıtcılara, sayaçlara ve tutuculara (latch) ulaşabilmektedir. Zamanlayıcının (timer) işleyiş şekli kontrol kayıtcısına yazılan veri ile yönlendirilmektedir. 6840 değişik uygulama alanlarında kullanılacak şekilde tasarımılandırılmıştır.

## BÖLÜM 6. MİKROBİLGİSAYAR PROGRAMLAMA

Genel olarak mikrobilgisayarlar giriş portlarından aldığı veriyi kendisine verilen komutlar dahilinde işleyerek çıkış portlarına ileten, bir çok defa farklı uygulamalar için programlanabilen bir mikroişlemcidir. Mikrobilgisayarlar çalışmak için kendilerine yüklenen program haricinde bilgisayarlar gibi bir işletim sistemine çoğu zaman ihtiyaç duymazlar. Günümüzde çok çeşitli mikrobilgisayar marka ve modelleri mevcuttur, fakat bu bölümde Arduino mikrobilgisayarların özellikleri, programlaması ve uygulamaları ele alınacaktır.

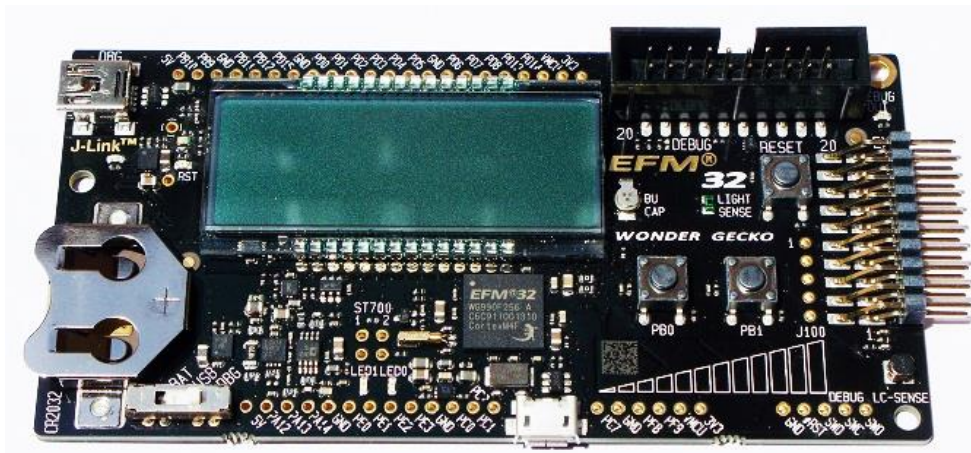
### 6.1. Mikrobilgisayar Kartları

Mikrobilgisayar (mikrodenetleyici) kartları, gömülü sistemin kalbidir. İster akıllı bir robot yapın isterseniz de basit bir sensör ile kontrol yapın, mikrobilgisayarlar, verilerin işlenmesi ve anlamlandırılması için gereklidir. Mikrobilgisayar geliştirme kartları, fiziksel tasarımları ve bağlantı zorlukları açısından kullanıcıya kolaylık sağlıyor. Gömülü sistem tasarımcısına, kullanmak istediği cihazı bu kartlara bağlamak ve programlamak görevleri kalıyor. Aşağıda günümüzde kullanılan birkaç mikrobilgisayar kartları listelenmiştir:

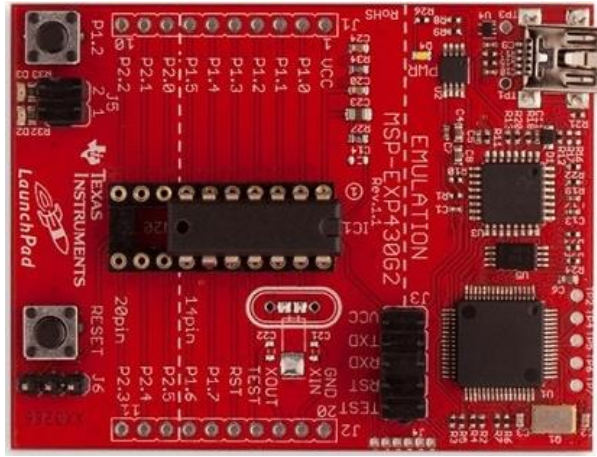
- a) **Parallela Micro ServerBoard:** Parallela Micro ServerBoard, güçlü bir mikrobilgisayar kartıdır. Çift çekirdekli ARM, FPGA ve Adapteva'nın 16 çekirdek Epifani işlemcisi, hızlı ve kolay işlemlere izin veriyor. Kart hem LINUX hem de Android OS sistemlerde çalıştırılabilir.



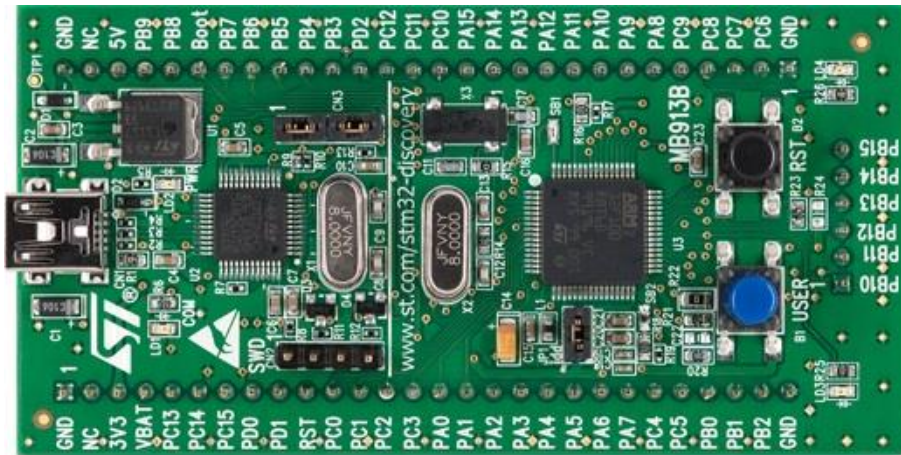
- b) **EFM32 Dev Gecko Starter Kit:** EFM 32GG Starter Kit, 32 bit mikroişlemci içeren bir geliştirme kartıdır. Kart, ortam ışığı sensörü, LCD, 20 adet I/O pinleri içermektedir.



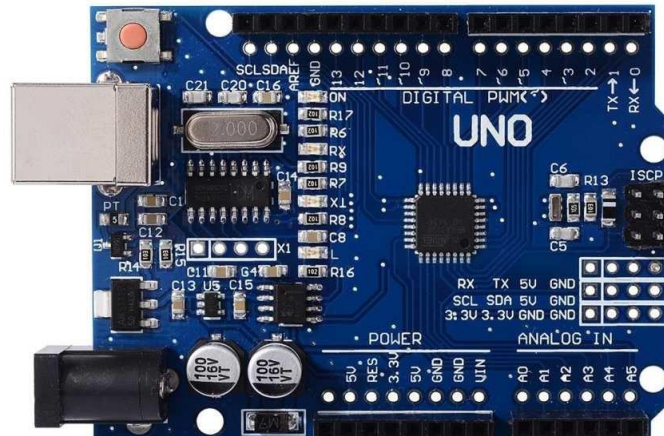
- c) **MSP 430 Launch Pad:** MSP 430 Launch Pad, düşük güç tüketimi ve düşük maliyetli bir kart olma özelliğine sahiptir. Kartın üreticisi, Texas Instruments'tir.



- d) **STM32 Discovery:** STM 32 Discovery, 24 MHz 32 bit ARM cortex ve 8 KB RAM içermektedir.



- e) **Arduino:** Arduino temelde açık kaynaklı bir elektronik geliştirme platformudur. Açık kaynak olduğu için çok sayıda farklı yetenek ve farklı seçeneklerde Arduino kart bulunmaktadır. Arduino kartları temel olarak Atmel mikrocipleri, özellikle Atmega328 veya Atmega168, üzerine kuruludur.



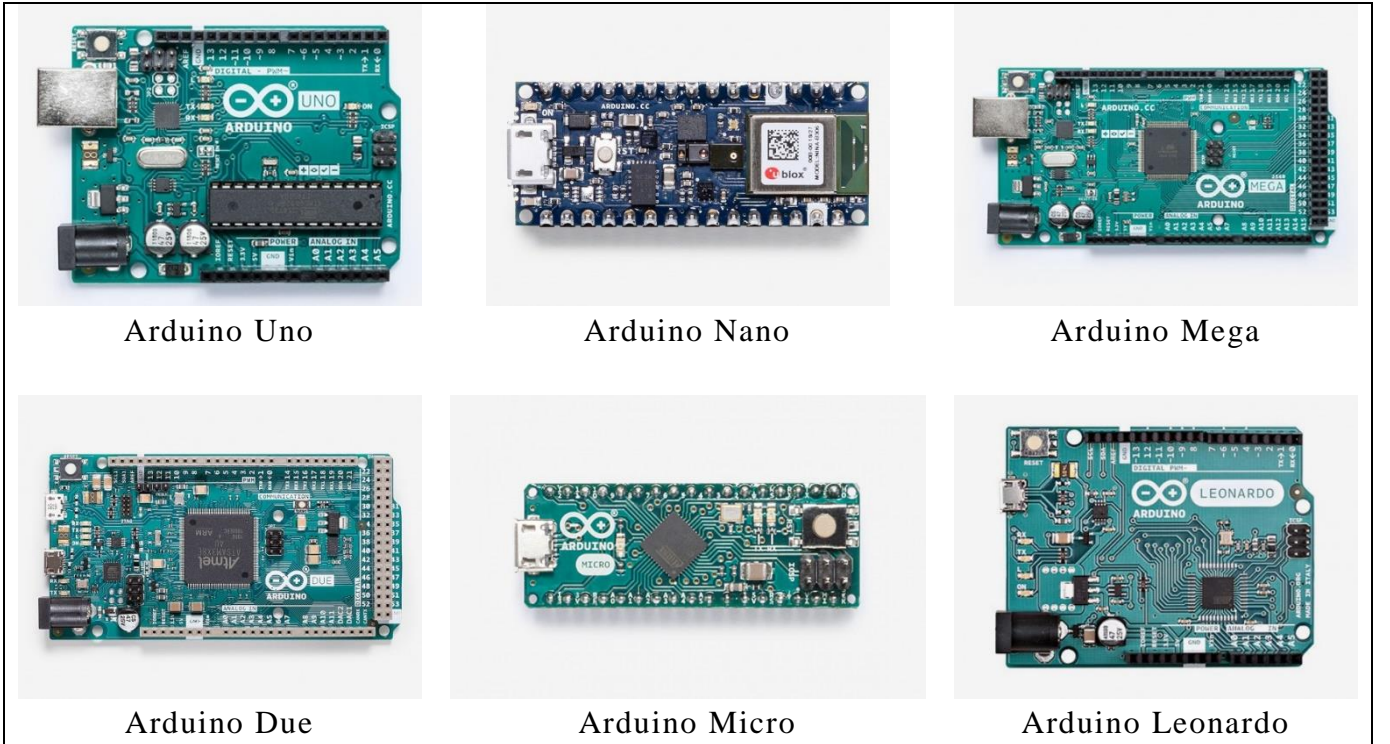


- f) **Raspberry Pi:** Raspberry Pi markası tarafından sunulan geliştirme kartıdır. Kart, bir denetleyiciden öte bilgisayar özelliklerine sahiptir. Raspberry Pi 2 kartı, 1 GB SDRAM, 900 MHz 4 çekirdekli ARM cortex CPU, 4 USB port, full HDMI portu ve 40 I/O pini içermektedir.



## 6.2. Arduino Mikrobilgisayar Kartları

Arduino, açık kaynak kodlu yazılım ve donanıma sahip bir mikrobilgisayar kartıdır. Açık kelimesi ile gerçek anlamda açık tasarımı ifade edilmektedir. Baskılı devresi, şematik tasarımı, pc üzerinde çalışan derleyicisi, kütüphaneleri ve tüm detayları ile internet ortamında paylaşılmaktadır. Arduino aynı zamanda mikrobilgisayar cihazın adı olarak da kullanılmaktadır. Baş tasarımcılarının (Massimo Banzi ve David Cuartielles) İtalyan olmaları nedeniyle cihazın adı da doğal olarak İtalyancadan seçilmiş ve arduino kelimesi “Sıkı arkadaş” anlamına gelmektedir. Arduino mikrobilgisayar kartlarının resmi web sitesi: <http://arduino.cc/>



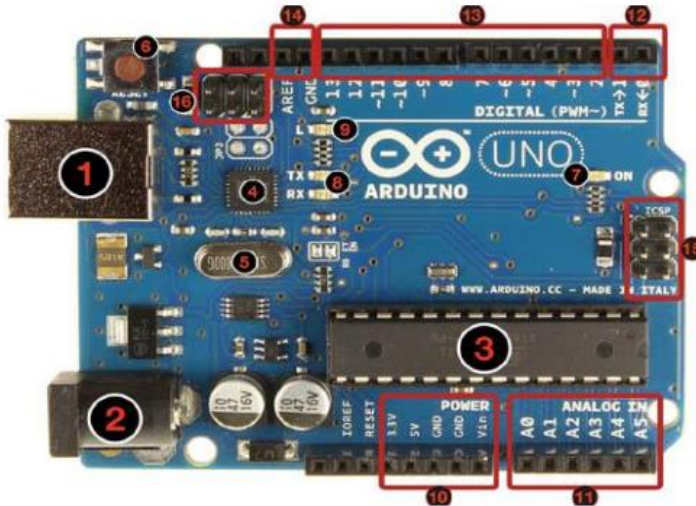
Şekil – Çeşitli Arduino Mikrobilgisayar Kartları

## A. Arduino Tercih Edilme Nedenleri

- Alt seviye mikroişlemci donanım bilgisi gerektirmemesi
- Zengin bir kütüphane desteği olması sebebiyle ileri teknolojileri destekler.
- Arduino programlamada C/C++/ Java tabanlı bir dil kullanmaktadır.
- Hem donanımı hem de yazılımı açık kaynaklıdır.
- Ucuz olması
- USB, Bluetooth gibi evrensel iletişim tekniklerini kullanabilmektedir.
- Motor, LCD Ekran, Röle, Sensör gibi birimler karta kolay entegre edilebilmektedir.
- Labview, Matlab ve Simulink gibi farklı platformlarda kullanılabilmektedir.
- İşlemci ile veri giriş çıkışları kolaylıkla sağlanabilir. Bu ise, serigrafi ile kart yapma ihtiyacını ortadan kaldırmaktadır.
- Kontrolleri GSM ve internet yolu ile kolayca yapılabilir.

## B. Arduino Donanımsal Yapısı

- Donanımsal özellikler mikrobilgisayar kartının türüne göre farklı özellikler göstermektedir.
- Arduino Uno Mikrobilgisayar Kartının Özellikleri:
  - 14 adet dijital giriş / çıkışı vardır
  - Bu çıkışlardan 6 tanesi PWM çıkışı olarak kullanılır.
  - Bunun yanı sıra, 6 adet analog giriş vardır.
  - Atmega328 işlemcisi kullanır.
  - 13 nolu pine bağlı debug LED bulunur. Böylece ilave bir donanıma ihtiyaç olmadan, geliştirilen yazılımlar kart üzerinde test edilebilir.
- Arduino Uno Mikrobilgisayar Kart Modelinin Bağlantıları:



Yukarıdaki şekilde görülen ATmega328 mikrodnetleyicisi kullanan Arduino Uno Kartı çeşididir. 14 dijital giriş/çıkış pini bulunur, bunlardan 6'sı PWM çıkışı olarak kullanılabilir ve dijital dalga genişliği ayarlanarak analog değerler elde edilebilir. 6 analog giriş pinine sahiptir. 16 MHz kristal osilatörü, USB bağlantısı, 2.1mm güç girişi, ICSP başlığı ve reset butonu vardır. Mikroişlemciyi destekleyecek herşeye sahiptir. Çalıştırmak için DC 7~12V güç kaynağına bağlamak yeterlidir

- 1 : USB jakı
- 2 : Power jakı (7-12 V DC)
- 3 : Mikrodnetleyici ATmega328
- 4 : Haberleşme çipi
- 5 : 16 MHz kristal
- 6 : Reset butonu
- 7 : Power ledi
- 8 : TX / NX ledleri
- 9 : Led
- 10 : Power pinleri
- 11 : Analog girişleri
- 12 : TX / RX pinleri
- 13 : Dijital giriş / çıkış pinleri (yanında ~ işareti olan pinler PWM çıkışı olarak kullanılabilir.)
- 14 : Ground ve AREF pinleri
- 15 : ATmega328 için ICSP
- 16 : USB arayüzü için ICSP

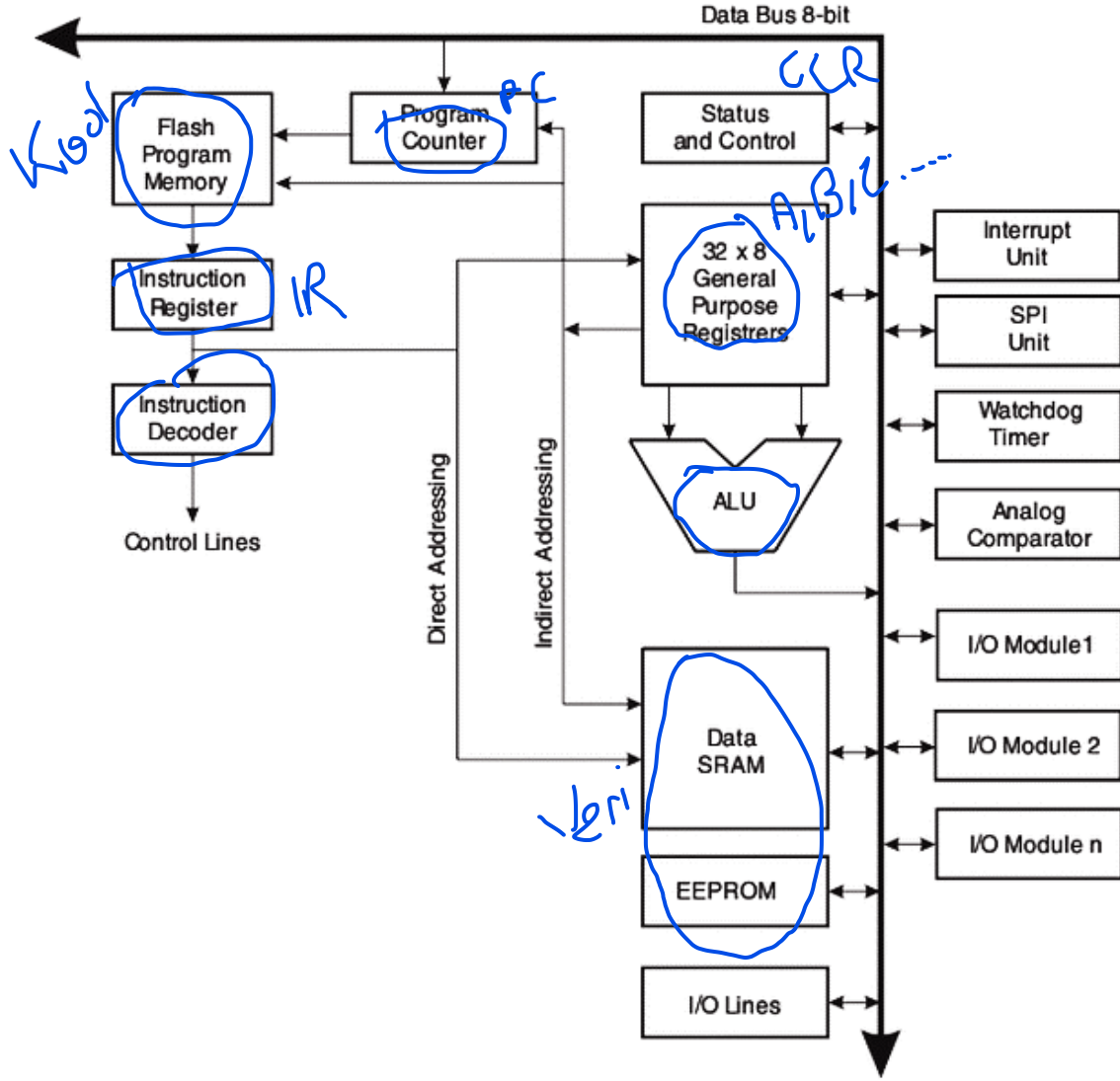
### İşlemci Özellikleri:

Flash: 32KB (0.5KB bootloader için kullanılır)  
SRAM: 2KB  
EEPROM: 1KB  
Saat frekansı: 16MHz

- Mikrobilgisayar kartında kullanılan Atmega mikrodenetleyici (gelişmiş mikroişlemci) iç mimarisi ve özellikleri:
  - 8 bitlik AVR mikroişlemci mimarisi: ALU ile bağlantılı 32 adet genel amaçlı 8-bitlik kayıtçı içerir.
  - RISC komut kümesi mimarisi: Komutların karmaşıklığı azaltılmıştır.
  - Harvard donanım mimarisi: Program ve veri belleği ayrıdır.

Microcontroller	ATmega168/328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	16 KB (ATmega168) or 32 KB (ATmega328) of which 2 KB used by bootloader
SRAM	1 KB (ATmega168) or 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) or 1 KB (ATmega328)
Clock Speed	16 MHz
Programlanabilme Kapasitesi	10000 Flash, 100.000 EEPROM

- AVR Mikrodenetleyicisinin Mimarisi:
  - Atmel firmasının üretmiş olduğu 8 bitlik RISC mimarisine sahip mikrobilgisayar (mikrodenetleyici) dir.
  - Ucuz ve hızlı çalışan bir mikrodenetleyici olup, gelişmiş özellikleri bulunmaktadır.
  - 2 KB ile 128 KB arasında değişen kapasitelerde yazılıp-silinebilen belleğe sahiptir.
  - Atmel, şirket ismidir, herhangi bir mikrodenetleyici ailesine verilen (PIC, AVR, Piccolo gibi) isim değildir.
  - Atmel firmasının üretmiş olduğu mikrodenetleyiciler bir kaç guruba ayrılır:
    - a) AT90Sxxxx: Klasik AVR'ler, yeni tasarımlar için önerilmemektedir.
    - b) ATtiny: En az seviyede giriş/çıkışlı, küçük bellekli, ucuz ve az güç harcayan gruptur.
    - c) ATmega: Aynı çekirdek üzerinde daha fazla bellek ve daha fazla giriş çıkışa sahiptir.
    - d) Diğer özel denetleciler: Örneğin LCD denetlecileri, akü şarj denetleyicileri, vb.



Şekil – Atmel AVR Mikroişlemcisinin Mimarisi

### C. Arduino ile Yapılabilecekler ve Uygulamaları

- Kolay bir şekilde çevresiyle etkileşime girebilen sistemler tasarlanabilir,
- Arduino kütüphaneleri ile mikrobilgisayar kolaylıkla programlanabilir,
- Analog ve dijital girişleri sayesinde analog ve dijital veriler işlenebilir,
- Sensörlerden gelen veriler kullanılabilir,
- Dış dünyaya çıktılar (ses, ışık, hareket vs...) üretilebilir.
- Medikal Uygulamalar
- Robotik Uygulamalar
- Mekatronik Uygulamalar
- Mobil Uygulamalar
- Giyilebilir Uygulamalar
- Kablosuz Haberleşme Uygulamaları
- Algoritmik Uygulamalar
- RFID Uygulaması
- Akıllı Ev/Şehir Uygulamaları





Giyilebilir Uygulamalar  
(Bisikletçinin arkasındaki  
dönüş sinyalleri).



Medikal Uygulamalar  
(Kalp sinyali izleme)



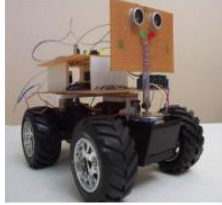
RFID Uygulamalar  
(Kimlik Giriş Sistemleri)



Askeri Uygulamalar  
(Mini drone'lar)



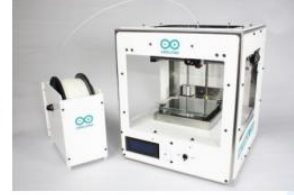
Tarımsal Uygulamalar



Robotik Uygulamalar



Ev Güvenliği ve Otomasyonu  
Uygulamaları



3 Boyutlu Yazıcı  
Uygulamaları



Mobil Uygulamalar

## D. Arduino'nun Kullanımı

Arduino ya da herhangi bir mikrobilgisayarlı sistemin kullanmak için gerekli çalışma şartlarının sağlanması gerekir. Arduino grubu mikrobilgisayarlar için gerekenler:

- Arduino Uno, Nano, Micro, Mega, Leonardo, Due, gibi bir Arduino kartı.
- Karta uygun USB kablo
- Arduino IDE programı
- Arduino ile uyumlu çalışan bilgisayar ya da cep telefonu



## E. Arduino ile Program Geliştirme

Program geliştirme adımları:

- 1) Algoritmanın oluşturulması
- 2) Yazılımın kodlanması
- 3) Devre bağlantılarının yapılması
- 4) Test
- 5) Test başarısız ise adım-1'e git
- 6) Test başarılı ise Dur

### 6.3. Arduino Mikrobilgisayar Kartlarında Program Yazma

#### A. Arduino IDE Yazılımı

• Arduino kartlarını programlamak için ilk önce Arduino web sitesinden (<https://www.arduino.cc/en/main/software>) işletim sistemimize uygun Arduino IDE sürümünü indirmek olacaktır. Arduino için Entegre Geliştirme Ortamı (IDE- Integrated Development Environment), C/C++/Java dilleri ile yazılmış bir platformlar (Linux, macOS, Windows) arası uygulamadır. Arduino uyumlu kartlara program yazmak ve yüklemek için kullanılır.

- Arduino IDE'si, özel kod yapılandırması kuralları kullanarak C ve C ++ dillerini destekler.
- Arduino IDE, birçok yaygın giriş ve çıkış prosedürünü bir yazılım kütüphanesi ile sağlar.
- Kullanıcı tarafından yazılan kod, iki temel fonksiyon (setup, loop) gerektirmektedir.
- Arduino IDE temelde, çalıştırılabilir kodu hexadecimal format ile metin dosyasına işler. Ardından kullandığımız Arduino IDE'si bu metin dosyasını bağlı olan arduino kartına yükleyici program ile aktarmayı gerçekleştirir.
- Ekim 2019'da Arduino organizasyonu, hata ayıklama ve diğer gelişmiş özellikler ile yeni bir Arduino Pro IDE'ye erken erişim sağlamaya başladı.

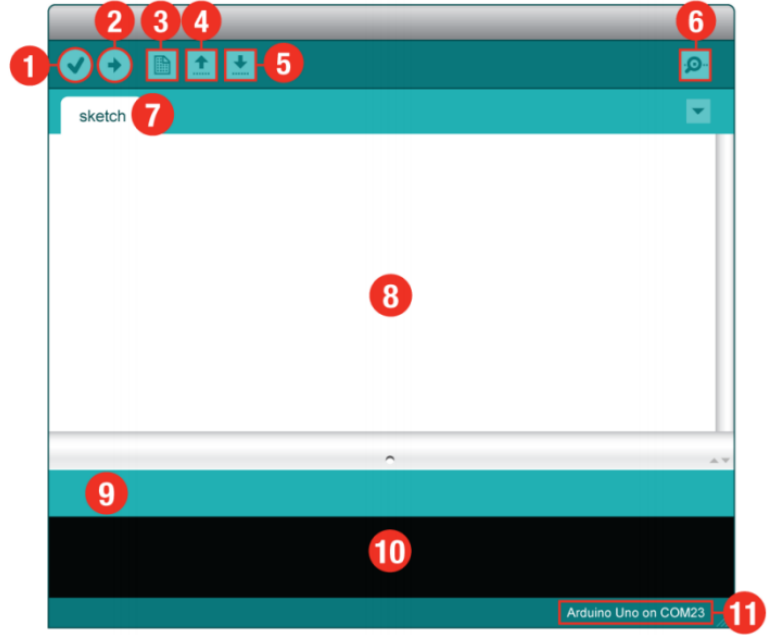
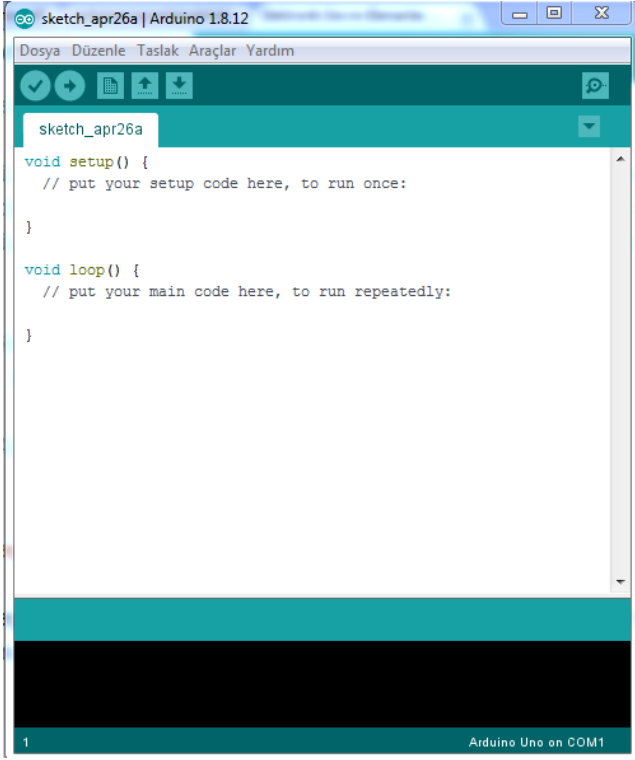
#### • Arduino IDE Programını Çalıştırma ve Arduino Kartına Bağlama:

Arduino IDE kurulumunu yaptıktan sonra, USB kablo ile Arduino kartımızı bilgisayar bağladığımızda Windows işletim sistemi driver isteyebilir. Driver otomatik kurulmaz ise aşağıdaki adımlar uygulanabilir:

1. Otomatik kurulum işlemi başarısız oluncaya kadar bekleyiniz.
2. Aygıt Yöneticisinden Portlar kısmında (COM, LPT) arduino kartının bağlı olduğu port sarı ünlem işareti görülecektir. Burada sürücü yazılımı güncellenir.
3. Windows driver kurulumu tamamlanır.
4. Arduino kartının ve bilgisayarın doğru çalışıp çalışmadığını anlamak için ilk pratik uygulama olarak, Blink LED uygulaması yüklenerek çalıştırılır.

#### • Arduino IDE Yazılımının Tanıtımı:

- Arduino IDE yazılım geliştirme ortamı, kodları yazmak için bir text editörü, bir mesaj alanı, bir konsol ekranı, araç çubuğunda fonksiyonlar için butonlar ve menülerden oluşur. Bu yazılım Arduino kartını bilgisayara bağlayıp, onunla iletişim sağlamak için kullanılır.
- Arduino için yazılan her programa sketch (taslak) denilmektedir. Sketch'ler bu text editöründe yazılır ve .ino uzantısı ile saklanır. Mesaj alanı, hataları ve uyarıları gösterir. Konsol ekranı ise hataları, uyarıların ayrıntılarını ve varsa düzeltilmesi gerekenleri gösterir. Ekranın sağ alt köşesinde ise bağlı olunan Arduino kartı (uno) ve port numarası (com) gösterilir. Araç çubuğundaki butonlar, kodları doğrulama ve yükleme, yeni sketch oluşturma, var olanı açma, kaydetme işlemleriyle beraber seri port monitörünü açar.



Şekil – Arduino IDE ana ekranı

- Arduino IDE ana ekranındaki her bir bileşenin görevi:
  - 1) **Derleme** : Yazdığımız programı derler hataları bulur.
  - 2) **Yükleme** : Yazdığımız kodu derler, Arduino içine atar.
  - 3) **Yeni** : Yeni çalışma sayfası açar.
  - 4) **Açma** : Kayıtlı bir programı açar.
  - 5) **Kaydetme** : Yazdığımız programı kaydeder.
  - 6) **Seri Monitör** : Arduino ile seri iletişim yaparak ekran açar.
  - 7) **Sketch** : Yazdığımız programın dosya ismi.
  - 8) **Boş alan** : Yazacağımız program alanı.
  - 9) **Gösterge** : Yaptığı işlemin ilerleme durumunu gösterir.
  - 10) **Rapor** : Derleme sonucu yapılan hataların veya programımızın yükleme sonrası mikrodenetleyicide kapladığı alanı gösterir.
  - 11) **Gösterge** : Bilgisayarımıza USB ile bağladığımız Arduino'nun bağlandığı portu ve hangi Arduino modeli ile çalışıyorsak onu gösterir.

## B. Arduino IDE ile Program Yazma ve Çalıştırma

- Arduino programları iki temel fonksiyon ile birlikte üç bölümden çalışır:

**Tanımlamalar** // değişken, sabit, pin isimlerini, vb. tanımlama

**void setup() {**

*//...led pinleri, motorlar, sensorler, kesmeler vb. kurulumlar*

**}**

**void loop() {**

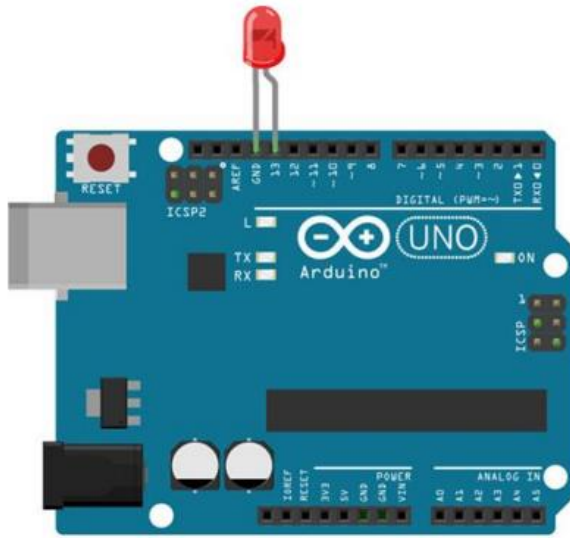
*// ...sensorlerden bilgi al, motorları kontrol et veya ledi yak/söndür vb. ana işlemler*

**}**

- **Tanımlamalar** en başta yapılırsa global tanımlamalar, setup veya loop veya kullanıcı tanımlı alt fonksiyonlar içinde yapılırsa lokal tanımlamalar denir.
- **Setup** fonksiyonu, programın çalışmaya başladığını ilk kısımdır. Bu kısım sadece bir kere okunur ve program esnasında yeniden okunmaz. Bu alanda, pinlerin çalışma modları, seri iletişim başlatılması, kesme tanımlamaları gibi önemli ve bir kere yapılması yeterli olacak ayarlamalar/kurulumlar yapılır.
- **Loop** fonksiyonu, setup fonksiyonu okunduktan sonra başlar. Buraya yapılması istenen işlemler yazılır. Loop fonksiyonu, sonsuz döngü şeklinde çalışır. Yani buradaki görevler tamamlandığında, program tekrar loop fonksiyonunun başına dönerek işlemleri yeniden yapar. Bu döngü, Arduino çalıştığı sürece devam eder.

• **Örnek-1:** İlk Arduino Programı – led yakıp söndürme (blink) uygulaması.

- Arduino IDE programında, hazır bulunan örnek uygulamalardan “Dosya/Örnekler/01.Basics/Blink” dosyasını açalım. Derleme yaptıktan sonra, Arduino’ya yükleyelim.
- Sistemin bağlantı yapısı aşağıdaki gibi olacaktır. Aşağıdaki şekil-a da 13 nolu pine bağlanan LED, gerçek kartta şekil-b de kırmızı ile işaretlenen yerde bulunmaktadır.



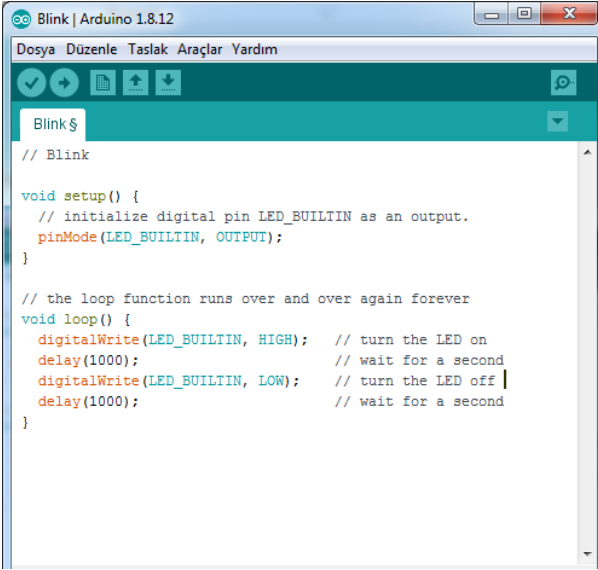
(a)



(b)

Şekil – Blink devre şeması: a) temsili, b) gerçek gösterim

- 13 nolu dijital giriş/çıkış pinine bağlı olan LED’in, birer saniye aralıklarla yanıp sönmelerini sağlayan program aşağıda verilmiştir.



(a)

`int ledPin=13; // 13.dijital G/Ç pini, ledPin olarak tanımlandı`

`void setup(){ //kurulumlar – sadece bir defa çalışır  
pinMode(ledPin, OUTPUT); // ledPin çıkış yapıldı.  
}`

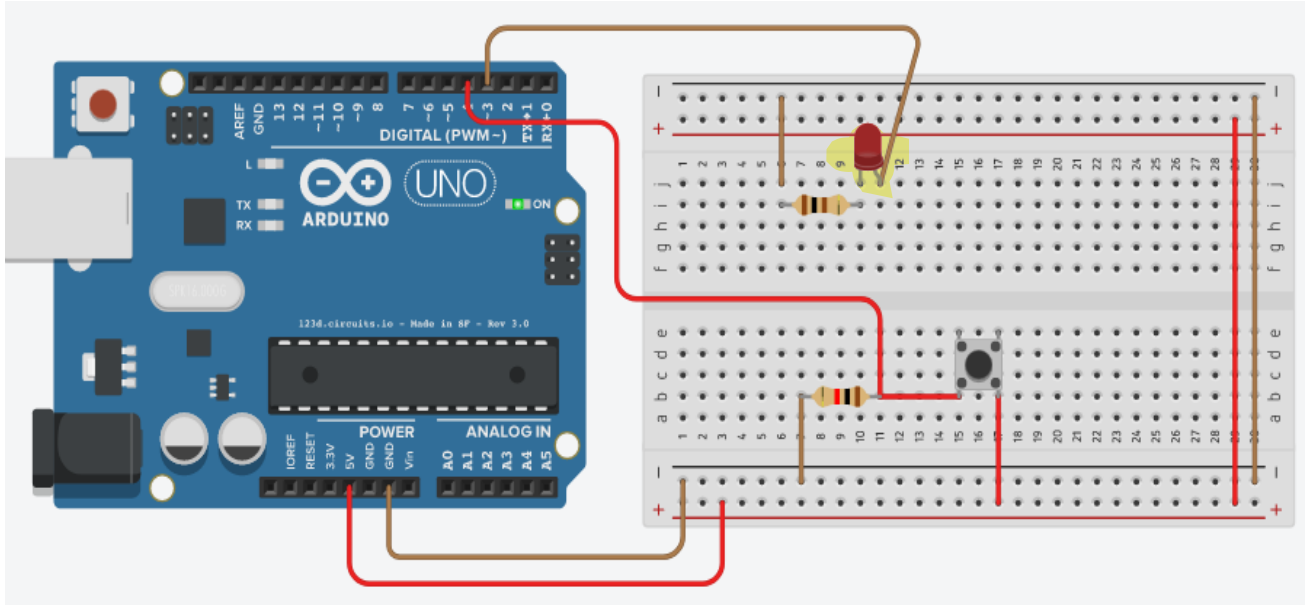
`void loop() { // ana işlemler – sürekli tekrar ederek çalışır  
digitalWrite(ledPin, HIGH); // led yandı  
delay(1000); // 1 saniye bekle  
digitalWrite(ledPin, LOW); // led söndü  
delay(1000); // 1 saniye bekle  
}`

(b)

Şekil – Blink programı: a) Arduino IDE de bulunan, b) Açık bir şekilde kodlanan


• **Örnek-2:** Buton ile led yakma uygulaması.

- Sistemin bağlantılarını gösteren devre şeması aşağıda verilmiştir. Led 3 nolu pine, buton ise 4 nolu pine bağlanmıştır.



Şekil – Buton ile led yakma sisteminin devre şeması

- Yukarıdaki şekilde verilen sistemi çalıştıran program aşağıda verilmiştir.



```

int butonPin = 4; // Butonu bařladıřımız Pini 4 olarak ayarladık.
int ledPin = 3;   // Ledi bařladıřımız Pini 3 olarak ayarladık.
int butondurum = 0; // Butonun durumunu atadıřımız bir deęiřken tanımladık.
void setup()
{
    pinMode(ledPin, OUTPUT); // Led ıkıř olarak ayarlandı.
    pinMode(butonPin, INPUT); // Buton giriř olarak ayarlandı.
}
void loop()
{
    butondurum = digitalRead(butonPin); // Buton pininden dijital okuma iřlemi

    if (butondurum == HIGH) // Butona basılmıř ise durumu
    {
        digitalWrite(ledPin, HIGH); // Ledi yak .
    }
    else // Butona basılmamıř ise durumu
    {
        digitalWrite(ledPin, LOW); // Ledi sndr.
    }
}

```

řekil – Buton kontroll led programı

#### 6.4. C ile Arduino Programlama

Gnmzde programlama dillerinin en bilineni C dilidir. Programlama aısından, en temelden en st dzeye kadar kullanılmaktadır. İster bilgisayarlar iin ister gml (mikrobilgisayar) sistemler iin geliřtirilmek istenen programlarda ihtiyaa cevap verebilen gl bir programlama dilidir.

Bir program, her hangi bir programlama dilinin kodlama tekniklerini kullanarak iřlemcinin yetenekleri ile donanım elemanlarını kontrol edebilen ve makinaları akıllı hale getiren komutların tmdr. Bir program geliřtirme sreci genel olarak, tasarım, kodlama, test, hata ayıklama ve gncelleme adımlarından oluřur. Yazılan program, basit bir gml sistem mikrodenetleyicisi iin bile olsa, aynı srelerden geirilmesi bir yazılım mhendislięi disiplindir.

rneęin, bir sıcaklık kontrol sistemi iin mikrobilgisayar programlama yapalım. Program tasarımı řu řekilde gerekleřir:

- a) Sensr ile sıcaklık bilgisini l,
- b) llen deęer 40 derece ve zerinde ise sesli uyarı vererek, kırmızı ledi yak,
- c) Eęer deęilse, yeřil ledi yak,
- d) Adım-(a) srecine tekrar dn.

#### A. C Programlama Dilinin Temel Yapıları

- **Kořullu Yapılar (if-else-elseif):** Yazılım dillerinin en temel komutlarından birisidir. Elinizde bir durum var ve bu durum doęru ise a iřini yapmasını eęer yanlıř ise b iřini yapmasını istiyorsanız, bu yapı kullanılır.



```

if (a == 3){
    //Buraya a=3 olduğu durumda çalışması istenilen kodlar yazılır
}
else{
    a'nın 3 olmadığı durumda çalışacak kodlar yazılır.
}

```

Koşul alanında kullanılabilecek karşılaştırma ifadeler;

==	Eşitse	!=	Eşit değilse
<	Küçüktür	>	Büyüktür
<=	Küçük eşitse	>=	Büyükse
Kosul1 && Kosul2	VE bağlacı	Kosul1    Kosul2	VEYA bağlacı

Birden fazla koşulumuz var ise *elseif* ile yeni koşullar eklenebilir:

```

if (x == 1){
    // x = 1 durumunda burası çalışır
}
elseif (x == 2){
    // x = 2 durumunda burası çalışır
}
elseif (x == 3){
    // x = 3 durumunda burası çalışır
}

```

- **for döngüsü:** Yazdığımız kodların belli bir sayıya kadar tekrar etmesini isteyebiliriz. Bunun için döngüler kullanılır.

```

for (int i =0; i < 5; i ++){
    // burası 5 kez tekrar edecek,
    // her defasında i değeri bir arttırılır,
    // i=5 oluncaya kadar döngü devam eder
}

```

- **while döngüsü:** *for* gibi *while* kodları da döngü oluşturmak için kullanılır.

```

b = 10;
while (b > 5){
    b = b - 1 ; // her döngüde b'nin değerini bir azaltılır, b > 5 oldukça döngü devam eder
}

```

## B. Arduino da C Dili Programlama Yapıları ve Fonksiyonları

- **void setup( ):** Bu fonksiyon, Arduino program yapısının temel fonksiyonlarından ilki olup, program yüklendiğinde, karta enerji verildiğinde veya reset atıldıktan sonra 1 defa mahsus çalışır. Bu fonksiyon içinde, pin giriş/çıkış tanımlamaları, PWM, ADC, DAC, iletişim hızı, kesme gibi kurulum ayarları yapılır.

```
void setup () {
    Serial.begin (9600); // Seri iletişim hızı 9600 baud olarak ayarlandı
    pinMode (5, INPUT); // 5 nolu pin giriş yönlü tanımlandı
}
```

- **void loop( ):** *setup( )* fonksiyonu çalıştıktan sonra çalışan fonksiyondur. Bu fonksiyon, Arduino program yapısının temel fonksiyonlarından ikincisi olup, içindeki kodlar sonsuz döngü içerisinde çalışarak, sürekli kendini tekrarlar.

```
void loop(){
    digitalWrite(3, HIGH); // 3 nolu dijital pini lojik-1 yapar
    delay(1000); // 1000 ms (veya 1 sn) gecikme sağlar
    digitalWrite(3, LOW); // 3 nolu dijital pini lojik-0 yapar
    delay(2000); // 2000 ms (veya 2 sn) gecikme sağlar
}
```

- **Niteleyiciler:** Değişkenleri, fonksiyonları ve sabitleri tanımlamak için kullanılır.
  - **Global (Scope) Değişkenler:** Bu değişkenler, public olarak bilinir. Program kodu içerisinde her yerden ulaşılabilir olan değişkenlerdir. *setup()*, *loop()* veya kendi yazdığımız fonksiyonlar dışında tanımlanan bu değişkenlere *global (genel)* değişken denir. Fonksiyonlar içerisinde tanımlanan değişkenler ise ömürleri sadece o fonksiyonla sınırlı olan *lokal (yerel)* değişkenlerdir.

```
int ledPin; // ledPin, her yerden ulaşılabilir global değişkendir.
void setup(){
    ....
}
void loop(){
    int a; // a değişkeni sadece loop() fonksiyonunda tanımlı, lokal değişkendir.
    float b; // b değişkeni sadece loop() fonksiyonunda tanımlı, lokal değişkendir.
    for(int c=0; c<100; c++){
        ..... // c değişkeni sadece for döngüsü içerisinde tanımlı, lokal değişkendir.
    }
}
```

- **static:** static tanımlanan değişken bellekte tutulur ve daha sonra kullanılmak istendiğinde yeniden oluşturulmaz bellekten çağırılır. static ile oluşturulan değişkenler, lokal değişkenlere benzer. Fakat, lokal değişkenler fonksiyon her çağırıldığında yeniden oluşturulur ve fonksiyon bitince ömürleride biter, static ile oluşturulan değişken ise fonksiyon bitse bile değişken bellekte kalmaya devam eder, fonksiyon tekrar çağırılrsa yeniden oluşturulmaz, üzerindeki en son değer ile birlikte tekrar kullanılmaya izin verir.

```
int topla(int a){
    static int b=0;
    b = b + a;
    return b;
}
```



Burada, *topla()* fonksiyonu verilen *a* parametresini, ilk değeri 0 olan *b* static değişkene ekliyor ve fonksiyon her çağrıldığında kendisine verilen *a* parametresinin değerini tekrar topluyor. Aynı işlem lokal değişken ile yapılmış olsaydı, her defasında *a* parametresinin değeri neyse *topla()* fonksiyonu o değeri geri döndürecekti. Çünkü *b=0* alınacaktı.

- *volatile*: volatile ile tanımlanan değişkenler mikrobilgisayarın ram bölgesine kaydedilir. Bu değişkenler okuma veya yazma amaçlı kullanıldığında doğrudan ram bellekten okunur veya yazılır. Diğer değişkenler gibi, bellekten kayıtçılara aktarılabilir. volatile ile tanımlanan değişkenlerin değerinin değiştirilmesi, ancak kesme (interrupt) işlemi ile yapılabilir.

```
int pin = 13;
volatile int durum = LOW;
void setup(){
    pinMode(pin, OUTPUT);
    attachInterrupt(0, blink, CHANGE); // 0:kesme pini, blink:kesme fonksiyonu, CHANGE:kesme
durumu
}
void loop(){
    digitalWrite(pin, durum);
}
void blink(){
    durum = !durum;    // durum değişkeninin değeri, kesme ile değiştiriliyor
}
```

Burada, kesme oluştuğunda 13 nolu pine bağlı olan LED'in yanma durumunun değiştirilmesi istenmektedir.

- *void*: void ile bir fonksiyon tanımlanırsa, bu fonksiyon çalıştıktan sonra her hangi bir değer geriye döndürmeyeceğini ifade eder. Android program yapısındaki, *setup()* ve *loop()* fonksiyonlarında bu niteleyici kullanılmaktadır. Ayrıca kendi tanımladığımız fonksiyonlarda da kullanabiliriz.
- *const*: const ile oluşturulan değişkenlerin değerleri sabitlenir ve bu değerleri daha sonradan değiştirilemez.

```
const float pi= 3.14;    // Pi sayısı
```

- *#define*: #define ön işlemci komutu olup, bir isim yerine başka bir ismin kullanımını sağlar.

```
#define ledpin 13; // Programda ledpin gördüğü yere 13 rakamını yerleştirir.
#define PI 3.14;  // Programda PI kullanılan yerlerde, PI yerine 3.14 değerini alır.
```

string yada sayısal sabit oluşturmak için *#define* veya *const* kullanılabilir. Fakat *const* ile tanımlanmış bir sabit, dizi sınır boyutunu belirtirken kullanılmaz.

- **#include:** *#include* programımız dışındaki kütüphanelere erişmek ve programımıza ilave etmek için kullanılır.

```
#include "ABC.h"    // Tanımlama-1
#include <ABC.h>    // Tanımlama-2
```

Programımızda, ABC kütüphanesini kullanmak ve onun içerisindeki komutlara, fonksiyonlara ulaşmak istediğimizde, programımızın başına ABC kütüphanesini, yukarıdaki her iki tanımlama türünden biri ile eklememiz gerekir.

Arduino programımız içerisine, kullandığımız donanımlara yönelik hazır kütüphaneleride bu şekilde ekleyebiliriz.

```
#include <EEPROM.h>    // EEPROM kullanabilmek için ilgili kütüphane eklendi
```

- **Kontrol Yapıları:** Her hangi bir koşulun sağlanma durumuna göre belirtilen kodlar çalıştırılır.
  - *if kontrol yapısı:* *if*, “eğer” manasına gelir. Şart sağlanırsa *if* ile verilen komutları çalıştırır, sağlanmazsa *if* den sonraki komutları çalıştırır.

```
if (digitalRead(buton) == HIGH){    // buton değeri 1 ise
    digitalWrite(led, HIGH);          // ledi yak
}
```

- *if/else kontrol yapısı:* Koşullara bağlı olarak yürütüm süreçlerini kontrol etmek için kullanılır.

#### Örnek-1:

```
if (digitalRead(buton) == HIGH){    // buton değeri 1 ise
    digitalWrite(led, HIGH);          // ledi yak
}
else{                                // değil ise (buton değeri 1 değilse)
    digitalWrite(led, LOW);           // ledi söndür
}
```

#### Örnek-2:

```
if (digitalRead(butonA) == HIGH){    // butonA değeri 1 ise
    digitalWrite(ledA, HIGH);          // ledA'yi yak
}
elseif (digitalRead(butonA) == HIGH){ // butonB değeri 1 ise
    digitalWrite(ledB, HIGH);          // ledB'yi yak
}
else{                                // yukarıdakilerden hiçbiri değil ise
    digitalWrite(ledC, LOW);           // ledC'yi söndür
}
```

- *switch/case yapısı:* *switch/case* bir ifadenin sabit değerlerinden hangisiyle eşleşirse, o değere karşılık gelen durumdaki kodlar yürütülür. Bir *switch/case* yapısından çıkışı sağlamak ya da sonlandırmak için *break* yada *return* komutu kullanılır.

**Örnek:** Işık sensöründen okunan değere göre seri monitöre bilgi mesajının *switch/case* kullanarak yazılması.

```
void setup(){
    Serial.begin(9600); //Seri iletişimin hızı belirleniyor
}

void loop(){
    // A0 nolu analog portdan bilgi okunup, dijitalle çevrilir
    int okunanDeger = analogRead(A0);
    // 0-1023 aralığında dijital bilgi, 0-3 aralığına dönüştürülür.
    int deger= map(okunanDeger, 1, 1024, 0, 3);
    switch (deger){
        case 0: //sensör ışık görmüyorken, dijital bilgi 1-256 aralığında ise
            Serial.println("karanlık"); //Seri porta yazılır ve seri monitörden izlenir
            break; // switch yapısından çıkış
        case 1: //sensör az ışık alırken, dijital bilgi 257-512 aralığında ise
            Serial.println("az aydınlık");
            break;
        case 2: //sensör biraz ışık alırken, dijital bilgi 513-768 aralığında ise
            Serial.println("hafif aydınlık");
            break;
        case 3: //sensör tam ışık alırken, dijital bilgi 769-1024 aralığında ise
            Serial.println("tam aydınlık");
            break;
    }
    delay(10); // her değer okuması arasında kararlı çalışması için 10 milisaniye bekleme
}
```

- *while yapısı:* *while* döngüsündeki, ifade doğru olduğu sürece {...} parantezi içindeki kodlar çalışır.

```
while (ifade){
    // ifade doğru ise buradaki kodlar çalışır
}
```

**Örnek:** Aşağıdaki program LED'i 10 kez yakıp söndürmektedir.

```
int ifade=0;
int ledPin=0;
void setup(){
    pinMode(ledPin, OUTPUT); // 0 nolu pini çıkış yaparak, LED bağlanıyor.
}

void loop(){
    while (ifade<10)
        digitalWrite(ledPin, HIGH);    // ledi yak
        delay(1000);                    //1 saniye bekle
        digitalWrite(ledPin, LOW);      // ledi söndür
        ifade=ifade+1;
    }
}
```

- *do/while yapısı:* Döngüyü bir defa çalıştırdıktan sonra ifadeyi kontrol eder, ifade doğru olduğu sürece {...} parantezi içindeki kodlar çalışır.

```
do{
    // ifade doğru ise buradaki kodlar çalışır
} while (ifade);
```

**Örnek:** Aşağıdaki programda sıcaklık sensöründen gelen bilgi 100 den küçükse LED yanar.

```
int ledPin=3;
void setup(){
    pinMode(ledPin, OUTPUT); // 3 nolu pini çıkış yaparak, LED bağlanıyor.
}

void loop(){
    do{
        int deger = analogRead(A0); // A0 nolu analog portdan sıcaklık bilgisi okur
        digitalWrite(ledPin, HIGH); // ledi yak
        delay(1000)                  // sensörden okuma kararlılığı için 1 sn bekleme
    }while (deger<100)
    digitalWrite(ledPin, LOW);      // ledi söndür
}
```

- *break:* Döngü yapılarından çıkılmasını sağlar. *do-while, for, while, switch-case*, vb. yapılarda kullanılır.
- *continue:* Döngü yapılarında bir döngü sürecinin işlem yapılmadan geçilmesini sağlar. *do-while, for, while, switch-case* vb. yapılarda kullanılır.
- *goto:* Program akışını istenilen etikete yönlendirir.

### Örnek:

```
for (x=0; x<255; x++){  
  xx: digitalWrite(ledPin, HIGH);  
  deger=analogRead(Sensor);  
  if (deger>200) break; // seonsörden okunan deger>200 ise for döngüsünden çıkılır  
  if (x>50 && x<100) continue; // x'in değeri 50-100 aralığında ise işlem yapma  
  while(deger<100){  
    digitalWrite(ledPin, LOW);  
    goto xx;  
  }  
}
```

- *return*: Fonksiyonun geri döndüreceği değeri belirtir. Fonksiyonun geri dönüş değeri yoksa ve fonksiyonda belli bir yere kadar olan kodları çalıştırmak istiyorsak, return kullanılır.

### Örnek-1: Fonksiyondan geri dönüş değeri döndürmek.

```
int kontrolSensor(){  
  if (analogRead(A0) > 100)  
    return 1;  
  else  
    return 0;  
}
```

### Örnek-2: Fonksiyondan geri dönüş yoksa, belli bir yere kadar olan kodların çalıştırılması.

```
void loop ( ){  
  ... // çalışan kodlar  
  return; // sadece loop-return arası kodlar çalışır, return altındaki kodlar çalışmaz.  
  ... // çalışmayan kodlar  
}
```

- **Operatörler:** Operatörler, önceden tanımlanmış matematiksel ve mantıksal işlemleri yerine getiren ve belli işlevleri olan işaretlerdir.
  - Aritmetiksel Operatörler: Değer atama, değerler arasında toplama, çıkarma, bölme ve çarpma işlemleri gerçekleştiren işaretlerdir.

### Örnek:

```
int a,b,c,d,e;  
a = 5;           // atama  
b = a + 3;       // toplama  
c = a * b;       // çarpma  
d = c / 10;      // bölme  
e = b - d;       // çıkarma
```

- Mod alma (%) Operatörü: Bir sayının başka bir sayıya bölümünden kalan sayı, mod olarak ifade edilir.

### Örnek:

```
int a,b;
a = 5 % 2;      // a değişkeni 1 değerini alır
b = 49 % 5;     // b değişkeni 4 değerini alır
```

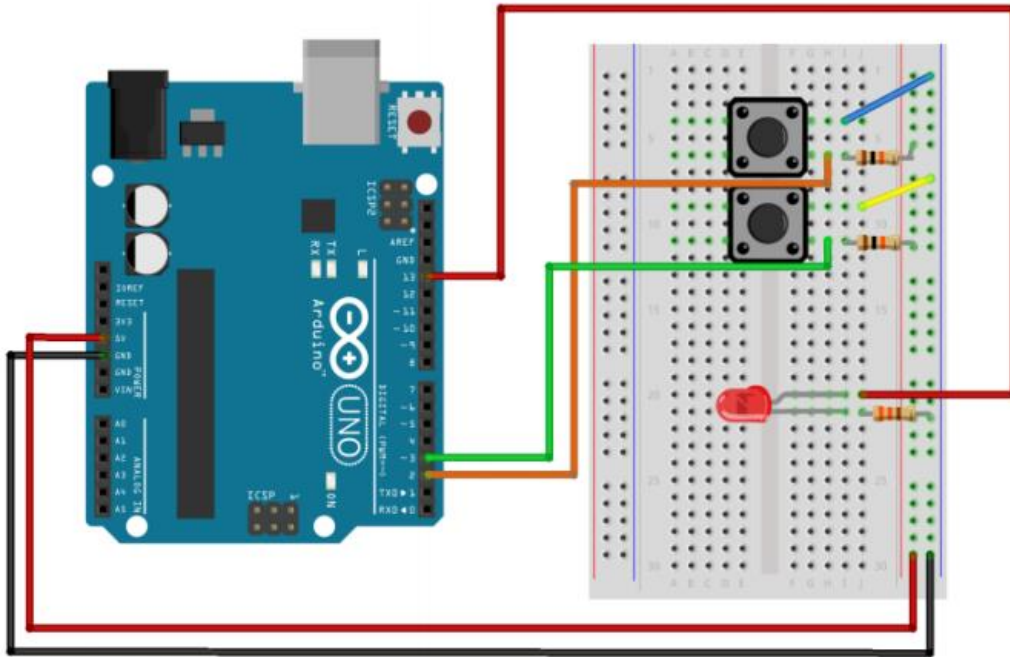
- Karşılaştırma Operatörleri: Şartlı yapılar (*if*, *while*, *vb.*) içerisinde karşılaştırma yapmak için kullanılır.

==	: eşit,	a == b	(a, b ye eşit)
!=	: eşit değil,	a != b	(a, b ye eşit değil )
<	: küçük,	a < b	(a, b den küçük)
>	: büyük,	a > b	(a, b den büyük)
<=	: küçük eşittir,	a <= b	(a, b den küçük eşit)
>=	: büyük eşittir.	a >= b	(a, b den büyük eşit)

- Mantıksal Operatörler: Şart gerektiren her yerde (*if*, *while*, *vb.*) kullanılan sembollerdir.

&& : VE  
 // : VEYA  
 ! : DEĞİL

**Örnek:** İki buton ve bir led'den oluşan aşağıdaki devrede, mantıksal operatörlerin çeşitli uygulamaları.



Şekil – Butonlar ile led kontrolü

```

int butonA = 2; // butonu 2. pine tanımladık
int butonB = 3; // butonu 3. pine tanımladık
int ledPin = 13; // ledi 13. pine tanımladık

void setup() {
    pinMode(butonA, INPUT); // 2. pin giriş oldu
    pinMode(butonb, INPUT); // 3. pin giriş oldu
    pinMode(ledPin, OUTPUT); // 13. pin çıkış oldu
}

void loop(){
    int A = digitalRead(butonA); // A butonunu oku
    int B = digitalRead(butonB); // B butonunu oku
    // her iki butona birlikte basılmışsa ledi yak – VE işlemi
    if (A==HIGH && B==HIGH) digitalWrite(ledPin, HIGH);
    // her iki butona veya sadece bir butona basılmışsa ledi yak – VEYA işlemi
    if (A==HIGH || B==HIGH) digitalWrite(ledPin, HIGH); // ledi yak
    // butonA'ya basılmıyorsa ledi söndür – DEĞİL işlemi
    if (!A) digitalWrite(ledPin, LOW); // ledi söndür
}

```

- Bitisel Operatörler: Mantıksal işlemlerin bit seviyesinde yapıldığı sembollerdir.

& : VE (AND)  
 / : VEYA (OR)  
 ~ : DEĞİL (NOT)  
 ^ : ÖZEL VEYA (XOR)

### Örnek:

```

int a = 12; // binary: 1100
int b = 10; // binary: 1010
int c = a & b; // binary: 1000 veya desimal: 8
int d = a | b; // binary: 1110 veya desimal: 14
int e = ~b; // binary: 0101 veya desimal: 5
int f = a ^ b; // binary: 0110 veya desimal: 6

```

- Birleşik Operatörler: İki operatörün birlikte kullanımı söz konusudur.  
Bu operatörler: ++, --, +=, -=, \*=, /=, %=, &=, |=, >>, <<.

### Örnek:

```

int a = 2; // binary: 00000010
int b = ++a; // önce artırma yapılarak a=3 olur, sonra atama yapılarak b=3 olur
int c = a--; // önce atama yapılarak c=3 olur, sonra azaltma yapılarak a=2 olur
int x = 1;
x += 4; // x=5 olur

```

```

x -= 3;           // x=2 olur
x *= 10;          // x =20 olur
x /= 2;           // x=10 olur.
x %= 5;           // x=0 olur
int m = 12;        // binary: 1100
int n = 10;        // binary: 1010
int m = &n;        // m = m & n; binary: 1000 veya desimal olarak m=8
int m = |n;        // m = m | n; binary: 1010 veya desimal olarak m=10
int y = 5;         // binary: 00000101
int z = y << 3;    // binary: 00101000, hegzadesimal olarak z=28, desimal olarak z=40
int v = z >> 2;    // binary: 00001010, hegzadesimal olarak v=A, desimal olarak v=10

```

- **Dijital Giriş/Çıkışlar:** Arduino karttan dijital (0 veya 1) giriş/çıkış yapmak için kullanılan fonksiyon ve yapılar verilecektir. Arduino Uno modelinde 14 adet dijital giriş/çıkış pini bulunmaktadır. Bu pinlerin numaraları; 0-13 aralığıdadır.

- *pinMode:* Belirtilen pinlerin giriş veya çıkış olmasını ayarlar.

#### Kullanım şekli:

pinMode(pin, mod)

pin : Hangi pin ayarlanacaksa, o pinin numarası.

mod : Ayarlanacak durum türleri; INPUT, OUTPUT, INPUT\_PULLUP

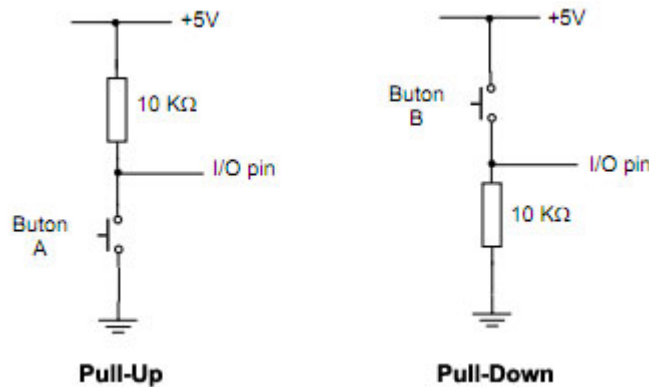
#### Örnek:

```
pinMode(3, INPUT);           // 3 nolu pin GİRİŞ olarak ayarlandı
```

```
pinMode(5, OUTPUT);         // 5 nolu pin ÇIKIŞ olarak ayarlandı
```

```
// 2 nolu pin, PULLUP dirençleri aktif yapılarak GİRİŞ olarak ayarlandı
```

```
pinMode(2, INPUT_PULLUP);
```



Buton girişi uygulamalarında, akım sınırlandırıcı direğin butonun üzerinde veya altında olduğunu gösterir.

- *digitalWrite:* Çıkış olarak ayarlanmış her hangi bir pinin değerinin HIGH (lojik-1) veya LOW (lojik-0) yapar



**Kullanımı:**

`digitalWrite(pin, deger)`

`pin` : İşlem yapılaca pinin numarası.

`deger` : Ayarlanacak durum türleri; HIGH, LOW.

**Örnek:**

`digitalWrite(3, HIGH);` // 3 nolu pinden lojik-1 verilir

`digitalWrite(7, LOW);` // 7 nolu pinden lojik-0 verilir.

- *digitalRead*: Giriş olarak ayarlanmış her hangi bir pin değeri okuması yapar. Okunan değeri HIGH (lojik-1) veya LOW (lojik-0) şeklindedir.

**Kullanımı:**

`digitalRead(pin)`

`pin` : İşlem yapılaca pinin numarası.

**Örnek:**

`digitalRead(3);` // 3 nolu pinden dijital bilgi (0 veya 1) okur

**Uygulama:** Buton ile lambanın yanma durumunun kontrol edilmesi.

`int ledPin = 7;` // lambanın bağlantısı dijital pin

`int butPin = 13;` // butonun bağlantısı dijital pin

`int durum = 0;`

`void setup(){`

`pinMode(ledPin, OUTPUT);`

`pinMode(butPin, INPUT);`

`}`

`void loop(){`

`durum = digitalRead(butPin);`

`digitalWrite(ledPin, durum);` // butona basılınca lamba yanar

`}`

- **Analog Giriş/Çıkışlar:** Arduino Uno'da 6 adet analog giriş/çıkış bulunmakta olup, her bir analog girişin dijitale dönüşüm çözünürlüğü ise 10 bittir. Bu çözünürlük,  $2^{10} \rightarrow 0 \dots 1023$  aralığındaki değerlere karşılık gelir..

- *analogRead*: Belirtilen analog pinden analog bilgi okur. Arduino Uno modelinde analog giriş pin numaraları, 0-5 aralığındadır

**Kullanımı:**

`analogRead(pin);`

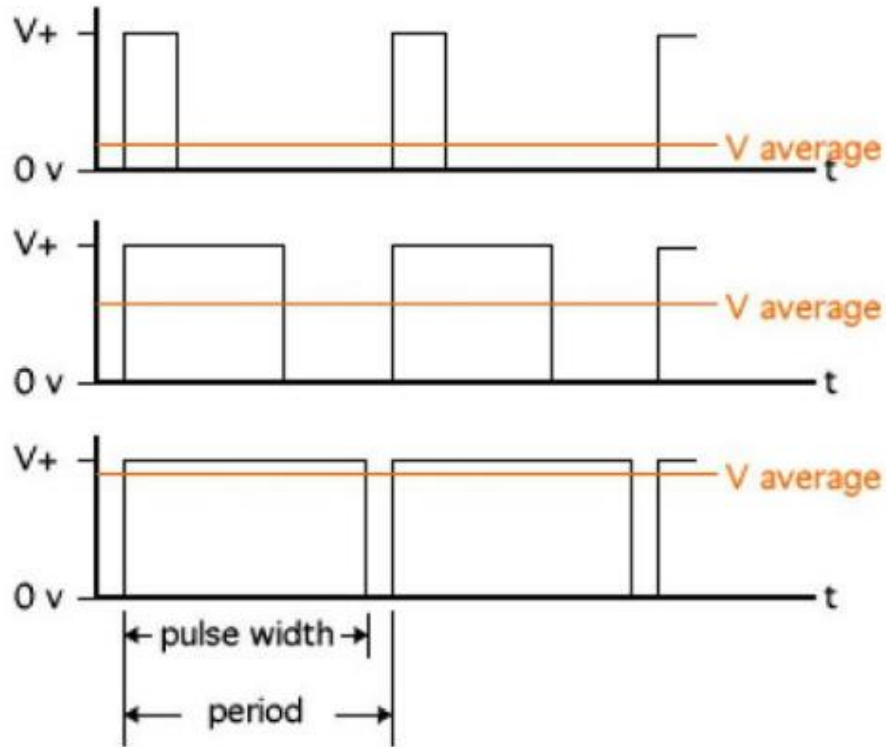
`pin` : İşlem yapılacak pinin numarası

### Örnek:

```
analogRead(3);           // 3 nolu analog pinden bilgi okunur.
```

- *analogWrite*: Belirtilen analog pine analog bilgiyi yazar. Analog bilgi olarak, 0-5 V değerleri arası bir gerilimdir. Bu işlem, PWM tekniği olarakta bilinir. Arduino Uno modelinde, PWM çıkışları dijital giriş/çıkış pinleri üzerinden yapılır. Bu pin numaraları, 3,5,6,9,10,11 nolu pinlerdir.

PWM tekniği, bir kare dalga'nın peryodu sabit kalmak kaydı ile lojik-1 'de kalma süresi değiştirilerek işaretin ortalama değerini ayarlama tekniğidir.



Şekil – PWM tekniği

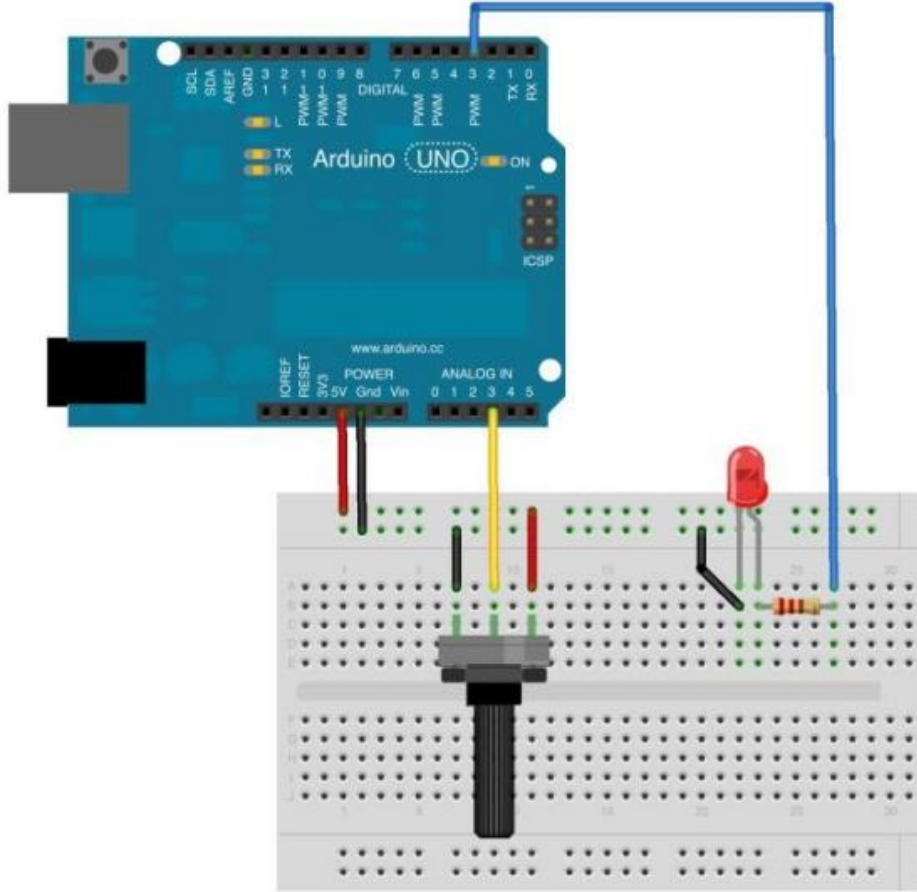
### Kullanımı:

```
analogWrite(pin);  
pin      : İşlem yapılacak pinin numarası
```

### Örnek:

```
analogWrite(3);           // 3 nolu pinden PWM işareti şeklinde analog bilgi çıkış verilir.
```

**Uygulama-1:** Bir potansiyometre ile lambanın ışık şiddetinin ayarlanması.



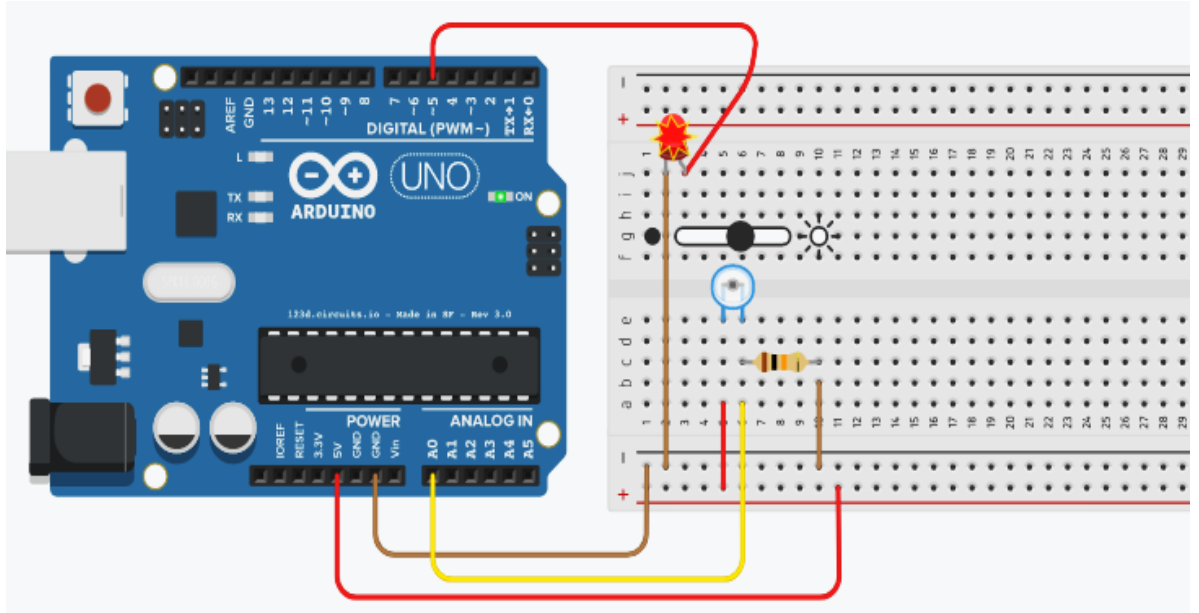
```
int ledPin = 3;           // lambanın bağlantı PWM pini
int potPin = 3;           // potansiyometrenin bağlantı analog giriş pini
int deger = 0;

void setup(){
    pinMode(ledPin, OUTPUT);
}

void loop(){
    deger = analogRead(potPin); // potansiyometreden analog değer okuma
    analogWrite(ledPin, deger); // degerin PWM çıkışı şeklinde lambaya verilmesi
}
```

**Uygulama-2:** Gün ışığının şiddetine göre lambanın yanması.

Gün ışığının bilgisini okuman için kullanılacak eleman LDR dir. LDR (Light Dependet Resistance – Foto Direnç) üzerine düşen ışığın şiddetine göre direnci değişen bir sensördür. LDR üzerine ışık gelmediği durumlarda direnç çok yüksek olur ve üzerinden akım geçmez. Otomatik gece ve sokak lambaları, kamera flaşların da vb. alanlarda kullanılır.



```

int ldrPin = A0;      // LDR sensörünün bağlandığı analog giriş pini
int ledPin = 5;       // Lambanın bağlı olduğu dijital pin numarası
int isikDegeri;

void setup(){
    pinMode(ledPin, OUTPUT);
}

void loop(){
    isikDegeri=analogRead(ldrPin);
    if(isikDegeri>500){
        digitalWrite(ledPin, HIGH); // lamba yanar
    }
    else{
        digitalWrite(ledPin, LOW); // lamba söner
    }
}

```

### • Gelişmiş Giriş/Çıkışlar

- *tone*: Belirtilen dijital giriş/çıkış pininden, istenilen frekansta kare dalga üretir. Üretilen kare dalganın, HIGH (lojik-1) ve LOW (lojik-0) bileşenleri eşit sürelerle sahiptir.

#### Kullanımı:

```
tone(pin, frekans, süre);
```

pin : Kare dalga çıkışı alınacak pin numarası

frekans: Kare dalganın frekansı

süre : Kare dalganın ne kadar süre devam edeceği

- *noTone*: Belirtilen dijital giriş/çıkış pininden üretilen kare dalgayı sonlandırır.

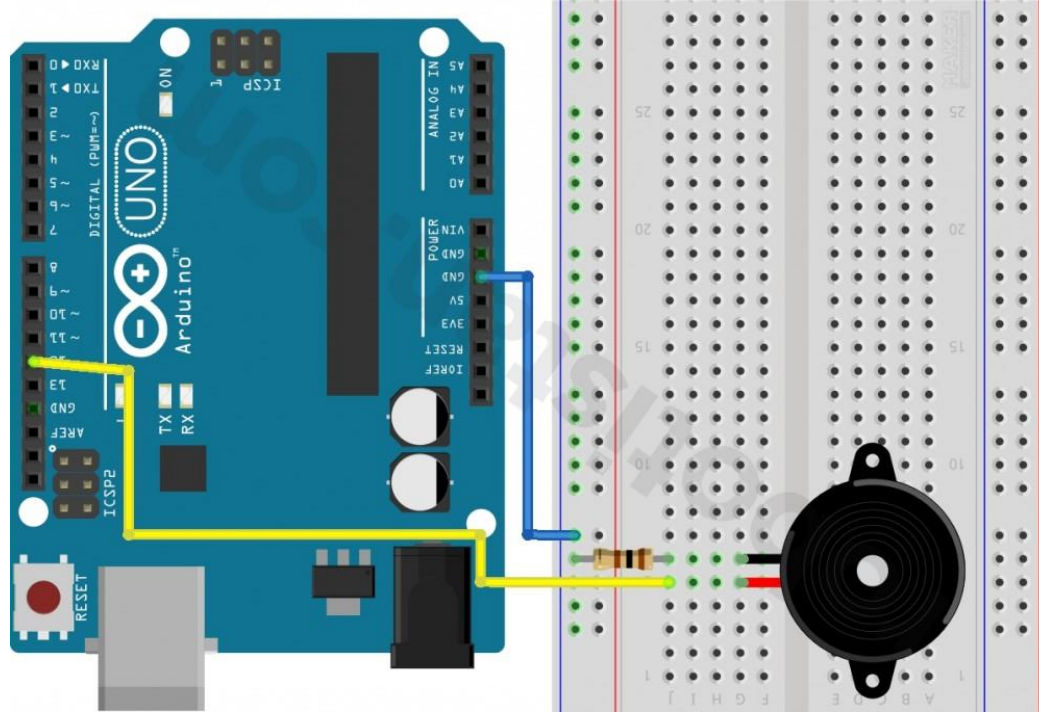
#### Kullanımı:

```
noTone(pin);
```

pin : Kare dalga çıkışı alınan pin numarası

**Uygulama:** Müzik notalarını çalan bir uygulama.

Her bir notanın, bir frekansı vardır. Bu frekanslar dikkate alarak tone ve noTone fonksiyonlarının kullanılması ile müzik notalarının oluşturulması.



Şekil – Müzik notalarının hoparlörden alınma devresi

```
int hoparlörPin = 12;
int notaSayisi = 7;
int C = 262;           // “do” notası
int D = 294;           // “re” notası
int E = 330;           // “mi” notası
int F = 349;           // “fa” notası
int G = 392;           // “sol” notası
int A = 440;           // “la” notası
int B = 494;           // “si” notası
int notalar[] = {C, D, E, F, G, A, B};
```

```
void setup(){
  pinMode(hoparlörPin, OUTPUT);
}
```

```
void loop(){
  for (int i = 0; i < notaSayisi; i++){
    tone(hoparlörPin, notalar[i]); //notanın hoparlöre kare dalga olarak gönderilmesi
    delay(500);                    // notanın hoparlörden 500 ms boyunca çalması
    noTone(hoparlörPin);           // notanın durdurulması
    delay(20);
  }
  noTone(hoparlörPin);
}
```

- *pulseIn*: Belirtilen pinin ne kadar süre HIGH veya LOW durumunda kaldığının süresini mikrosaniye olarak ölçer.

#### Kullanımı:

```
pulseIn(pin, değer);
pulseIn(pin, değer, zaman aşımı);
    pin      : Sinyalin alınacağı pin numarası
    değer    : HIGH veya LOW olabilir
    zaman aşımı : Pinden beklenen sinyalin gelmesi için bekleme süresi.
```

#### Örnek:

```
sure = pulseIn(5, HIGH, 10000000); //zaman aşımı 10 sn
```

5 nolu pindeki işareti 10 saniye boyunca izler, ne zamanki HIGH değeri LOW olursa geçen süreyi geri döndürür. 10 sn boyunca LOW değeri almaz ise *sure=0* olarak ölçülür.

- **Geçikmeler:** Bazı uygulamalarda zaman geçikmesine veya geçen sürenin ölçülmesine ihtiyaç duyulur. Bu geçikmeyi sağlayan ve süre ölçümü yapan fonksiyonlar:
  - **millis**: Arduino kartında aktif olan program çalışmaya başlamasından bu fonksiyonun bulunduğu yere kadar geçen süreyi milisaniye olarak ölçer.
  - **micros**: Arduino kartında aktif olan program çalışmaya başlamasından bu fonksiyonun bulunduğu yere kadar geçen süreyi mikrosaniye olarak ölçer.
  - **delay**: Milisaniye cinsinden geçikme oluşturur.
  - **delayMicroseconds**: Mikrosaniye cinsinden geçikme oluşturur.

#### Uygulama:

```
unsigned long zaman;
unsigned long sure;

void setup(){
    Serial.begin(9600); //seri iletişim hızı 9600 baud olarak ayarlandı
}

void loop(){
    Serial.print("Geçen Süre (milisaniye): "); // Seri monitöre yazar
    zaman = millis(); //programın çalışmaya başlamasından buraya kadar geçen süre
    Serial.println(zaman); // Seri monitöre yazar
    delay(500); // 500 milisaniye geçikme sağlar
    Serial.print("Geçen Süre (mikrosaniye): "); // Seri monitöre yazar
    sure = micros(); //programın çalışmaya başlamasından buraya kadar geçen süre
    Serial.println(sure); // Seri monitöre yazar
    delayMicroseconds(1000); // 1000 mikrosaniye geçikme sağlar
}
```

- **Fonksiyon Tanımlama:** Fonksiyonlar, bir program içerisinde aynı işlem defalarca yapılıyorsa, her seferinde aynı kodları yazmak yerine o işlem için ayrı bir fonksiyon yazılıp o fonksiyonu gereken yerde çağırıp kullanmayı sağlar. Fonksiyonların, parametre alan ya da almayan, geriye değer döndüren ya da döndürmeyen şeklinde çeşitleri bulunmaktadır.

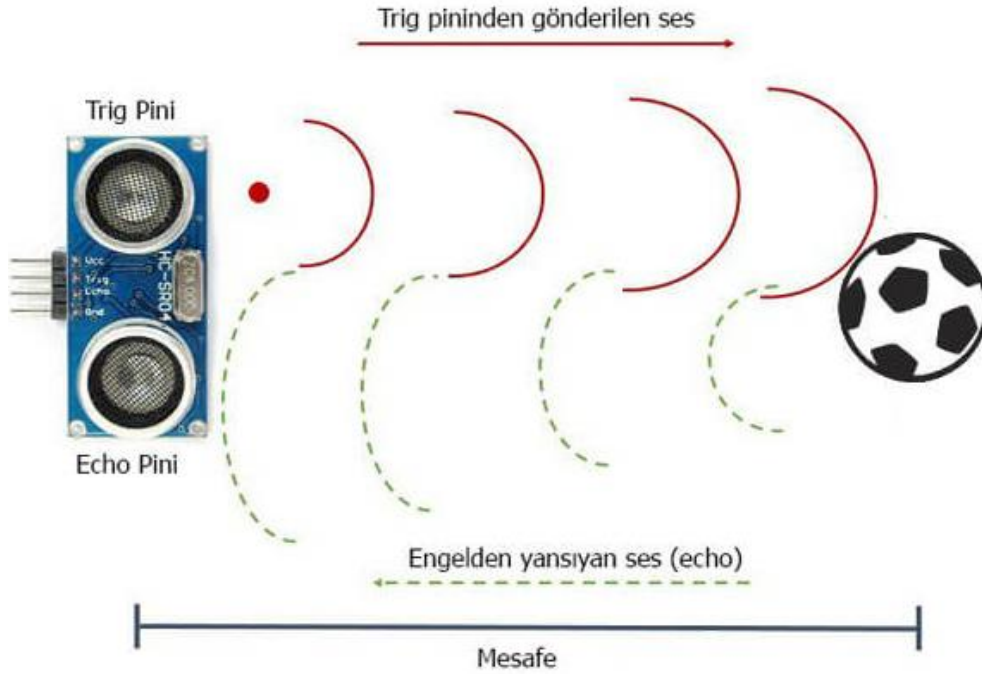
#### Kullanımı:

Fonksiyon tanımlama : `void FonksiyonAdi(){ }`

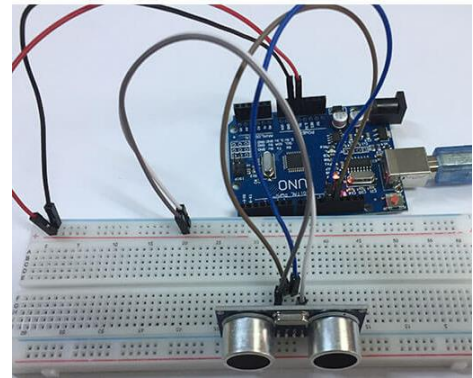
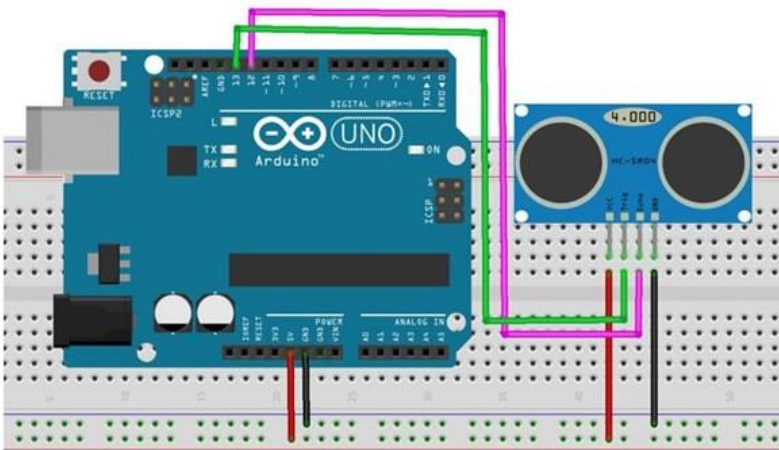
Fonksiyonu çağırma : `FonksiyonAdi();`

#### Uygulama: Arduino da Ultrasonik Sensör ile Mesafe Ölçümü.

Ultrasonik sensör, çıkış (trig) pininden ultrasonik ses dalgaları gönderirken, giriş (echo) pininden bu ses siyalinin herhangi bir engelden dönmesini bekler. Yansıyan ses sinyalini algıladığında geçen süre hesaplanarak aradaki mesafeyi ölçülür.



Şekil – Ultrasonic sensörün çalışma prensibi.



Şekil – Mesafe ölçüm sisteminin bağlantıları



```

int outputPin=13;           // trig, ses dalgası gönderme pini
int inputPin=12;           // echo, ses dalgası alma pini

void setup(){
  pinMode(outputPin, OUTPUT);
  pinMode(inputPin, INPUT);
  Serial.begin(9600);
}

unsigned long olcme(){
  int zaman;
  int mesafe;
  digitalWrite(outputPin, LOW);           // lojik-0 dalgasını 5 mikrosaniye gönderme
  delayMicroseconds(5);
  digitalWrite(outputPin, HIGH);          // lojik-1 dalgasını 10 mikrosaniye gönderme
  delayMicroseconds(10);
  digitalWrite(outputPin, LOW);
  zaman = pulseIn(inputPin, HIGH);        // dalga dönüş süresini ölçme
  mesafe = (zaman /29.1)/2;               // zamanın cm'ye dönüşümü
  return mesafe;
}

void loop(){
  int y=0;
  y = olcme();
  Serial.print("Uzaklik: ");              // Seri monitöre ölçülen değerin yazılması
  Serial.print(y);
  Serial.println(" cm");
  delay(1000);
}

```

Bu mesafe ölçüm uygulaması ile engelden kaçan robot, park sensörü, dijital metre vb. yapılabilir.

- **Seri Haberleşme:** Arduino da seri haberleşme, seri port için ayrılan Tx (1 nolu) ve Rx (0 nolu) dijital pinleri üzerinden yapılır. Bunun için kullanılan fonksiyonlar:

#### Kullanımı:

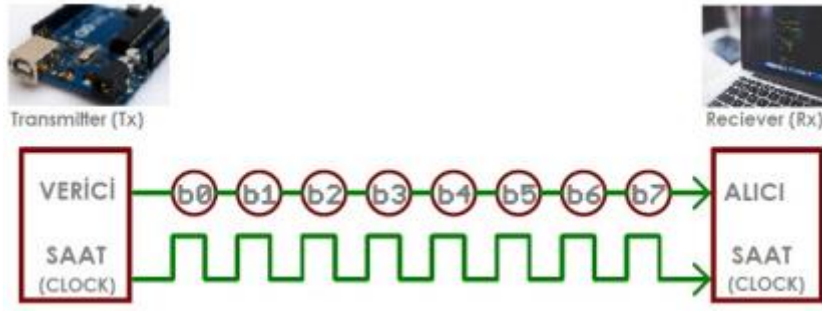
```

Serial.begin(hız);           // seri haberleşmenin başlatılması için setup() kısmına yazılır.
                             hız: alınan ve gönderilen bilgilerin baud olarak hızıdır. Örneğin; 9600.
Serial.print("...");         // seri ASCII (string ve char) bilgilerin gönderilmesini sağlar
Serial.println("...");
Serial.read();               // seri porttan gelen verileri okur - veri alma -
Serial.write();              // seri porta veri yazar - veri gönderme -
Serial.available();          // seri bağlantının kullanılabilir olup olmadığını kontrol eder

```

**Uygulama:** Bilgisayardan verilen komutlar ile 8 adet LED'in yakılması veya söndürülmesi. **74HC595** entegresi ile sürülen 8 adet LED'i yakıp söndürme işlemini, bilgisayardan verilecek komutlar ile yakılıp söndürülebilecek. Bunun için bilgisayar ile arduino kart arasında seri haberleşme kullanılmıştır. Bu haberleşmeyi takip etmek için Arduino IDE programının "Seri Monitör – Seri Port Ekranı" açılarak veri akışı izlenebilir.

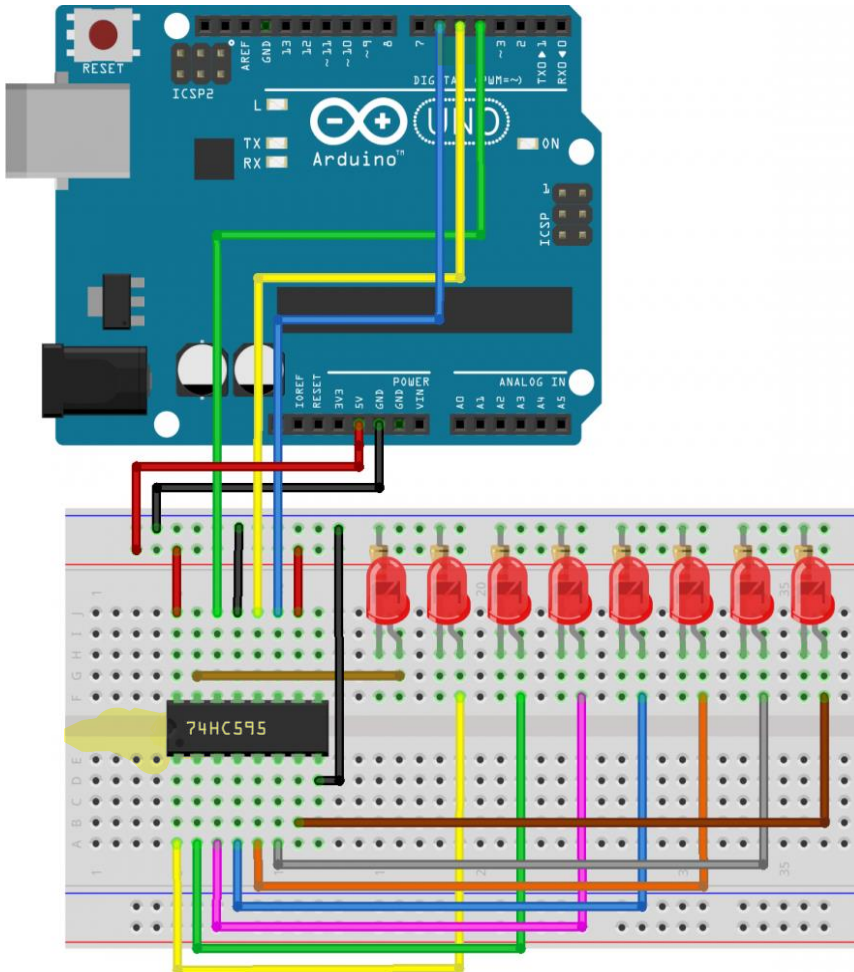




Şekil – Arduino ile Bilgisayarın Haberleşmesi.



Şekil – Seri Monitörün Açılması.



Şekil – Devre Bağlantı Şeması.

Verilen devre bağlantısını istenen şekilde çalıştıran yazılım kodları ise:

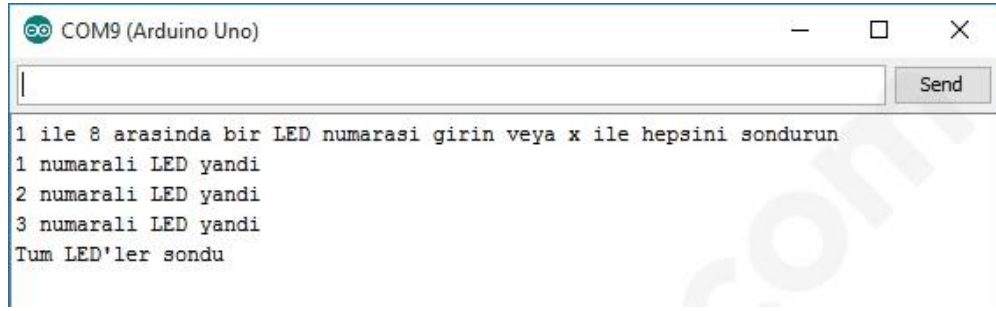
```
int latchPin = 5;
int clockPin = 6;
int dataPin = 4;
byte leds = 0; // leds isimli 8 bitlik bir değişken tanımlandı, her bir biti bir ledi temsil ediyor

void setup(){
    pinMode(latchPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    registraYaz();
    Serial.begin(9600);          // Seri iletişim hızı 9600 baud
    Serial.print("1 ile 8 arasında bir LED numarası girin veya ");
    Serial.println("x ile hepsini sondurun");
}

void loop(){
    if (Serial.available()){
        char ch = Serial.read();
        if (ch >= '1' && ch <= '8'){
            int led = ch - '1';
            bitSet(leds, led);
            registraYaz();
            Serial.print(led + 1);
            Serial.println(" numaralı LED yandı");
        }
        if (ch == 'x'){
            leds = 0;
            registraYaz();
            Serial.println("Tüm LED'ler sondu");
        }
    }
}

void registraYaz(){ // Bu fonksiyon, shift register'in çalışması için gerekli işlemleri yapar.
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, LSBFIRST, leds); // 74HC95 için yazılmış özel bir fonksiyon.
    digitalWrite(latchPin, HIGH);
}
```

Kodun ilk kısmında, çıkış pinleri ve leds isimli 8 bitlik bir değişken tanımlandı (byte tipindeki değişkenler 8 bit büyüklüğündedir). Bu baytın her bir biti, shift register'in çıkışına bağlı olan LED'leri temsil etmektedir. *registraYaz* fonksiyonu, shift register'in çalışması için gerekli işlemleri yapar. *loop* fonksiyonu, *registraYaz* fonksiyonunu çağırarak leds değişkeninde yapılan değişiklikleri LED'lere aktarır. *loop* fonksiyonunda iki adet for döngüsü kullanıldı: İlk for döngüsü, leds değişkenindeki 8 bittten her birini sırayla 1 yaparak 00000001, 00000011, 00000111... şeklinde bir desen elde edilmesini sağlar. Her bir bit 1 olduktan sonra (11111111), ikinci for döngüsü başlayarak bu sefer bitleri 0'layarak 01111111, 00111111, 00011111... desenini oluşturacaktır.



Şekil – Seri Port Ekranı Çıktısı.

- **Kesmeler (Interrupt):** Kesmeler donanım (dış-external) kesmesi ve zaman (timer) kesmesi olmak üzere ikiye ayrılır. Donanımsal kesme, beklenen her hangi bir sensör, buton, vb. fiziksel bir sinyalin gelmesi durumunda, programın normal çalışmasını durdurup, kesmeye ilişkin alt programın devreye alınmasıdır. Arduino Uno modelinde, iki adet donanımsal kesme bulunmaktadır. Bunlar 2 ve 3 nolu dijital pinlerdir.

#### Kullanımı:

Donanımsal Kesme:

`attachInterrupt(pin, fonksiyon, mod);`

pin : kesme sinyali için kullanılacak pin numarası.

fonksiyon : kesme geldiğinde çalıştırılacak alt program fonksiyonunun adı

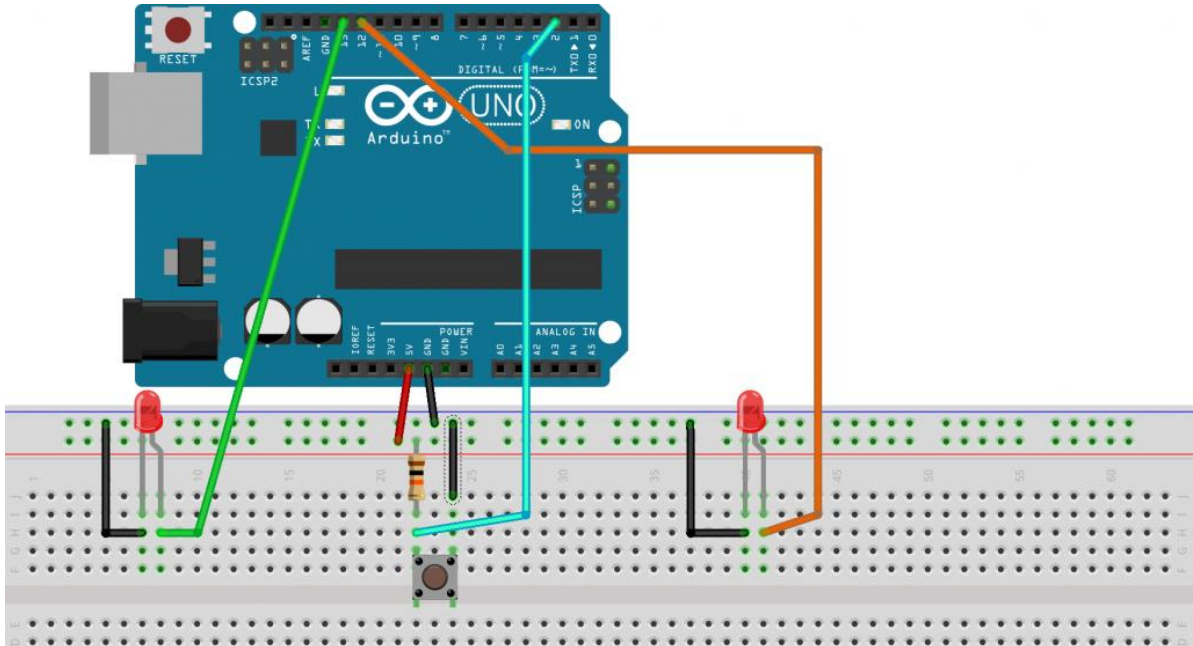
mod : kesme hangi durumda kabul edilecek {LOW, CHANGE, HIGH}

Zaman Kesmesi:

`ISR();` // belirlenen süre sonunda, tanımlanan görevleri yapar.

`noInterrupts();` //kesmeleri devre dışı bırakır.

**Uygulama:** İki adet led ve bir adet butondan oluşan bir devrede, ledlerden biri 1 saniye aralıklar ile yanıp sönüyor, diğeri ise butona basılıp bırakılınca yanıyor, söner, sönmükse yanar. Butona basılma olayını kesme ile algılayıp verilen işlemlerin gerçekleştirilmesi.



Şekil – Devre Bağlantı Şeması.

```

int led1 = 13;           // led tanımları
int led2 = 12;           // led tanımları

void setup() {
    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);
    attachInterrupt(2, kesmeFonk, CHANGE);
}

void kesmeFonk(){         // kesme fonksiyonu
    digitalWrite(led2, !digitalRead(led2));
}

void loop(){              //led1'i 1 sn aralıklar ile yakıp söndürür
    digitalWrite(led1, !digitalRead(led1));
    delay(1000);
}

```

- **Kütüphaneler:** Arduino ile birlikte kullanılabilecek cihazlar, Arduino IDE de eklenen kütüphaneler ile tanıtılır. Ayrıca, programcı kendi özel amaçlarına yönelik kütüphane yazabilir. Arduino IDE'ye kütüphane eklemek *#include* ile olur veya Arduino IDE'nin "Taslak" menüsü bölümünde "Include Library" seçeneğinde program ile beraber gelen belli başlı kütüphanelerden istenilenlerde eklenebilir.

#### Örnek:

```
#include <EEPROM.h>      // EEPROM kullanımı (okuma/yazma) için gerekli
```

### 6.5. Bölüm Kaynakları

1. Erdal Delebe – Projeler ile Arduino, Kodlab Yayıncılık, 2014.
2. Bülent Çobanoğlu, Derinlemesine Arduino, Abaküs yayınları, 2019.
3. Uğur Demir, Arduino Programlama Kitabı, İnternet, 2016.
4. Aslı Ergün, Arduino ile Programlama – Ders Notları, Dokuz Eylül Üniversitesi, İnternet, 2020.
5. Çeşitli internet kaynakları, Mayıs 2020.

SON