

**ATMEL
ATMEGA8
MİKRODENETLEYİCİ
ve
ASSEMBLER**

**Ögr.Gör. İbrahim KORANA
Konya, 2006**

1.	Genel Tanımlar.....	3
1.1	Microişlemci Nedir?.....	3
1.2	Microdenetleyici Nedir?.....	5
1.3	Microişlemci-Microdenetleyicinin Tarihsel Gelişimi.....	6
1.4	Neden Intel Tabanlı Microdenetleyici ?.....	7
2.	MCS-51 Microdenetleyici Ailesinin Mimari Özellikleri.....	8
2.1	MCS-51 Microdenetleyici Ailesinin Genel Mimari Yapısı.....	8
2.2	Pin Anlamları	12
2.3	MCS-51 Microdenetleyici Ailesinin RAM yapısı ve Register'leri.....	13
2.3.1	Genel Amaçlı Registerler ve Stack	13
2.3.2	Özel Amaçlı Registerler	15
3.	Atmel MCS-51 Microdenetleyici Komut Seti	24
3.1	Genel Assembler Bilgileri.....	24
3.2	Assembler Yönlendirme Komutları	24
3.3	Assembler Operatörleri	28
3.4	Hazır Fonksiyonlar	31
3.5	Komut Seti	33
3.5.1	Data Transfer Komutları	33
3.5.2	Aritmetiksel ve Logic Komutlar	35
3.5.3	Dallanma Komutları.....	37
3.5.4	Bit ve Bit Test Komutları.....	42
4.	Atmel MCS-51 Microdenetleyici I/O Portları	43
5.	Atmel MCS-51 Microdenetleyici Interrupt Sistemi.....	54
5.1	External Interrupt'lar (Dış Interrupt'lar).....	56
5.2	Internal Interrupt'lar (İç Interrupt'lar).....	58
6.	Atmel MCS-51 Microdenetleyici Özel Amaçlı Devreleri	58

1. Genel Tanımlar



1.1 Microişlemci Nedir?

Microişlemci, program yolu ile aritmetiksel ve mantıksal işlemler yapabilme gücü olan, kendisine bağlı iç ve dış çevre birimleri yönetebilen ve bu birimler arasındaki ilişkileri düzenleyebilen bir entegredir. Bir microişlemci, yapabileceği matematiksel ve mantıksal işlemlerin, çevrebirim yönetim fonksiyonlarının dizayn sırasında belirlendiği bir entegredir. Bu nedenle her microişlemci bir diğerinden farklıdır. (8088, Pentium III, Pentium II, Sparc, Alpha vb.)

Microişlemciler genel amaçlar için dizayn edilmiş yapılardır. Bu nedenle temel işlevlerini yerine getirebilmek için dizayn edilen komut setleri oldukça fazladır. Örneğin intel microişlemci ailesinin ilk işlemcilerinden olan 8088'in 92 adet komutu bulunmaktadır.

Microişlemcilerin komut setleri ve bu komut setlerinin nasıl işletildiği işlemcinin mimarisini belirler. Eğer komutların hafızada kapladığı alanlar farklı ise bu şekilde düzenlenmiş işlemciler CISC (Complex Instruction Set Computing) işlemciler adını alırlar. CISC mimari ile düzenlenmiş işlemcilerde, komutlar basitlik veya karmaşıklıklarına göre hafızada farklı uzunlukta yer kaplarlar. Bu durum işlemcinin içinde komutların decode edilmesi ile ilgili oldukça karmaşık bir birimin varlığını zorunlu kılar. Ancak, complex komutlar programlarda kullanılması gereken komut sayısını azaltırlar. 1970 yılında ortaya çıkan CISC mimari ilerleyen yıllarda hafıza birimlerinin büyümesi ve ucuzlaması, kısıtları nedeniyle RISC mimarinin ortaya atılmasına öncülük etmiştir. 1974 yılında IBM, bir işlemcinin daha az komut sayısı ve basit komutlardan oluşan bir komut seti ile çalışabileceğini önererek RISC (Reduced Instruction Set Computing) mimariyi çıkarttı. Bu mimaride komut uzunlukları sabittir ve her komut basit bir işlemi yerine getirir. Bir risc çipi bu iki karakteristik özelliği sayesinde komutların yorumlamasını kolaylıkla ve hızla yapabilir. RISC mimarinin en önemli dezavantajı CISC programlara göre programlarının hafızada daha fazla yer kaplamasıdır.

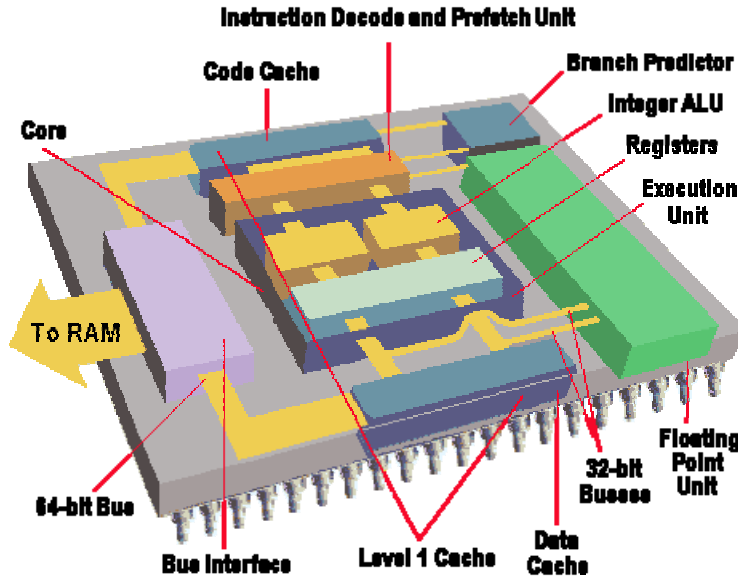
Gerek RISC mimaride gerekse CISC mimaride dizayn edilmiş olsun bir microişlemci herhangi bir komutu işleyebilmek için bir komut döngüsünü yerine getirmek zorundadır. Komut döngüsünün nasıl düzenlendiği genel olarak işlemcinin performansını belirler. Bir komut döngüsü 3 ana başlıktan oluşur.

1. Fetch
2. Decode
3. Execute

1. Fetch : İşlenme sırası gelen komutun ram'den okunarak microişlemcinin içine alınması
2. Decode : Komutun nasıl işleneceğinin belirlenerek çözümlenmesi

3. Execute : Komutun çalıştırılması ve elde edilen sonuçların ilgili yerlere aktarılması

İster CISC mimaride isterse RISC mimaride düzenlenmiş olsun bir microişlemci genel olarak aşağıdaki yapıdadır.



- **Execution Unit (Core):** Bu ünite komutları çalıştırır. Genel olarak matematiksel işlemlerin kontrolü ve mantıksal işlemleri gerçekleştirir. Execution Unit Aritmetik hesaplamalar için ALU (Arithmetic and Logic Unit) denen aritmetik ve mantık ünitelerini kullanılır, ALU için işlemcilerin yapıtaşdır denilebilir.
- **Branch Predictor:** Bu ünite, bir program çalışırken hangi komutun öncelikle çalışacağını belirleyerek Prefetch ve Decode ünitelerine hız kazandırır.
- **Floating Point Unit:** Bu ünite tamsayı olmayan floating point sayılarla yapılan matematiksel işlemlerin gerçekleştirilmesini sağlar.
- **L1 Cache:** Sistem belleğinden gelen veriler, bazı durumlarda CPU 'nun hızına yetişmeyebilir. Bu problemi çözmek için CPU içinde, küçük bir miktar cache (tampon) bellek bulunur. Önemli kodlar ve veriler bellekten cache belleğe getirilir ve burada tutulur. İhtiyaç duyulduğunda işlemci bunlara dış bellekte bulunan verilerden çok daha hızlı ulaşabilir. İşlenecek makine kodlarının ve verilerin saklandığı cache bellekler ayrı düzenlenirler.
- **BUS Interface:** Çevre birimlerden işlemciye veri – kod karışımını getirir, bunları ayırarak işlemcinin ilgili ünitelerine aktarılmasını ve işlem sonuçlarının dış ortama aktarılmasını sağlar. Bu arayüzün genişliği işlemcinin adresleyebileceği hafızayı belirler. Örneğin 32 bitlik hafıza genişliğine sahip bir işlemci 4 GB hafızayı adresleyebilir ve bu hafızadan aynı anda 32 bitlik veriyi transfer edebilir.

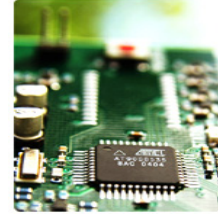
Microişlemcilerin genel amaçlı yapılar olmaları, komut setlerinin büyüklüğü, kullanıldığı sistemlerde ana eleman olmaları vb. nedenlerle çalışma hızları oldukça önemlidir. Bir işlemcideki bütün elemanlar saat vuruşlarıyla çalışır. Saat hızı bir işlemcinin saniyede ne kadar çevrim yapabileceğini belirler. 200 MHz saat hızı bir işlemci kendi içinde saniyede 200.000.000 çevrim yapabilir. Her çevrimde işlemcinin ne kadar işlem yapabileceği işlemcinin yapısına göre değişir. Genel olarak bir işlemcinin performansını;

İşlemcinin Mimarisi

Saat Hızı

L1/L2 Cache Miktarı

belirler. Ham işlemci performansını ifade etmek için **MIPS** (Million Instructions Per Second, saniyede işlenebilen komut sayısı) ve **MFLOPS** (Million Floating Point Operations Per Second, saniyede yapılabilen kayar nokta hesabı) birimleri kullanılır.



1.2 Microdenetleyici Nedir?

Bir mikroişlemcinin çevre birimleri (ram, rom, bus controller vb.) olmaksızın genel amaçlı bir programı çalıştırarak bir problemi çözmesi beklenemez. Mikroişlemcilerin anlamlı bir şekilde çalışabilmesi için minimum input ve output ünitelerinin bulunduğu bir sistemin kurulması gereklidir. Bir mikroişlemci ve minimum input-output ünitelerinin tek bir entegre içinde toplandığı yapılara microdenetleyici adı verilir.

Microdenetleyicilerde, çoğunlukla risc mimariye sahip bir mikroişlemci, programların ve verilerin saklanabileceği 2K ile 512Kb arasında değişebilen flash rom, 64 byte ile 16Kb arasında değişebilen statik ram, 4 ile 64 arasında değişebilen I/O, kontrol edilebilen timer/counter'lar, PWM, dışarıdan tetiklenebilen Interrupt kaynakları, Analog-digital çeviriciler, digital-analog çeviriciler bulunur. Bu özellikler mikroişlemci ile microdenetleyici arasındaki en önemli farkı oluşturur.

Microdenetleyiciler genel amaçlı uygulamalardan çok özel amaçlı uygulamalar için düzenlenmiş yapılardır. Kullanabilecekleri iç kaynakları çeşitli olmasına rağmen bu kaynakların sınırlı olması nedeniyle daha çok özel amaçlı cihazların kontrolünde kullanılmaktadırlar. Günlük hayatta kullandığımız birçok cihazın kontrolü microdenetleyiciler tarafından gerçekleştirilmektedir. Cep telefonu, televizyon, radyo, alarm sistemleri, taşıtlar, hesap makineleri, giriş-çıkış kontrol sistemleri, kameralar, bilgisayar vb.

Kısıtlı miktarda olmakla birlikte yeterince hafıza birimlerine ve input – output uçlarına sahip olmaları sayesinde tek başlarına (stand alone) çalışabildikleri gibi donanımı oluşturan diğer elektronik devrelerle irtibat kurabilir, uygulamanın gerektirdiği fonksiyonları gerçekleştirebilirler. Microdenetleyiciler çoğunlukla, yer aldıkları uygulama devresinin içine gömülmüş, sadece oraya adanmış olarak kullanılırlar. Bu özellikleri nedeniyle bilgisayarlardaki kullanıcı uygulama programlarını çalıştırma gibi esneklikleri olmamakla

birlikte kontrol ağırlıklı uygulamalarda alternatifsiz seçenek olarak karşımıza çıkarlar. Onları böyle cazip kılan, çok düşük boyutlu olmaları (az yer kaplamaları), düşük güç tüketimleri, düşük maliyetlerine karşın yüksek performansa sahip olmaları gibi özellikleridir.



1.3 Microişlemci-Microdenetleyicinin Tarihsel Gelişimi

Microişlemcilerin gelişimi 1970 li yıllarda başlar. 1971 yılında Intel firması hesap makinelerinde kullanılmak üzere ilk microişlemcilerden sayılabilecek 4004 işlemcisini üretti. Bu işlemci 4 bit veri yoluna sahipti ve 640 byte adres uzayına sahipti. Bu işlemcinin rağbet görmesi üzerine daha genel amaçlı ve daha büyük hafıza birimlerini adresleyebilecek 8080 microişlemcisi üretildi. 1974 yılında üretilen bu işlemci 8 bit veri yoluna ve 64Kb adres uzayına sahipti. 8080 8 bit microişlemcilerde endüstri standardı olan bir entegredir. Intel 1976 yılında 8080 işlemcinin gelişmiş bir versiyonu olan 8085 işlemcisini piyasaya sürerek işlemci gelişimini sürdürdü.

Intel 8080 işlemcisini duyurduktan kısa bir süre sonra Motorola 6800 microişlemcisini üretti. Bu işlemci 8080 işlemci ile karşılaştırılabilir bir güce sahipti. Motorola 6800 işlemcisinin devamı olarak bu işlemci ile uyumlu 6809 işlemcisini üretti. Bu işlemci 16 bit veri yoluna sahipti.

6809 işlemcisinden sonra Motorola 68000 işlemci serisinin üretimine başladı. Bu işlemci iç yapısında 32 bit dış yapısında ise 16 bit veri yoluna sahip bir işlemcidir ve 6800 serisi işlemcileri desteklememektedir. 68000 serisi işlemciler 68008, 68010, 68012, 68020, 68030, 68040 ve Power PC ile günümüze kadar gelmiştir.

Intel firması 1978 yılında ilk 16 bit veri yoluna sahip microişlemcisi olan 8086'yı duyurdu. 8086 gerek iç veri yollarında gerekse dış veri yollarında 16 bit uzunluk kullanan bir yapıya sahipti. Çevrebirimlerle uyum problemleri yaşanması üzerine intel 1979 yılında iç yapısında 16 bit, dış veri yolunda 8 bit yapıya sahip 8088 işlemcisini duyurdu. Bu işlemciler 8080-8085 microişlemci serisi ile benzeşmesine rağmen bu seri ile tam uyumlu değildir. 8086 ve 8088 1981 yılında ilk üretilen IBM-PC lerin gelişiminin temelini oluşturmuştur. Bu işlemciler geriye uyumluluk kaygıları ile gelişimini bugüne kadar sürdüren Intel x86 ailesinin başlangıcı olmuştur.

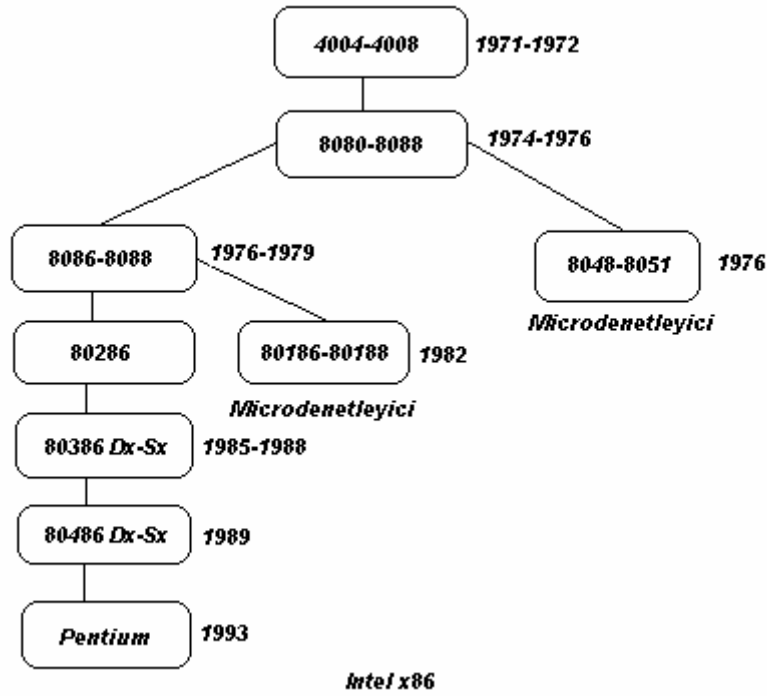
Intel firması 1976 yılında 8080-8085 microişlemcilerinin devamı olarak bazı microişlemci çevre birimlerini içinde barındıran ve özel amaçlı uygulamalarda kullanılmak üzere 8048 ve 8051 microdenetleyicilerin üretimini gerçekleştirdi. Bu microdenetleyiciler üzerinde kullanılan yapı günümüzde de kullanılan birçok 8 bit microdenetleyicinin temelini oluşturan standardı sağlamıştır. 8048 ve 8051 ile başlayan microdenetleyici serisi farklı özelliklerle günümüze kadar gelmiştir.

Intel firmasının PC lerin gelişimine temel oluşturan 8086-8088 serisinin 1Mb hafıza sınırlaması 80'li yıllarda birçok uygulama için ciddi problemler oluşturmaya başladı. Bu nedenle Intel 16Mb belleği adresleyebilen 80286 işlemcisini üretti. Bu işlemci 8088

işlemcisinin devamı olarak piyasaya sürüldü ve 8088 ile tam uyumlu olarak çalışabiliyordu. Intel firmasının bu işlemcisi IBM firmasının PC/AT makinelerinde kullanıldı.

1985 yılında Intel ilk 32 bit işlemcisi olan 80386 işlemcisini duyurdu. Bu işlemcinin 16 bit dış veri yoluna versiyonu 80386SX'in üretimini 1988 yılında gerçekleştirdi. 80386 işlemcinin devamı olarak piyasaya sürülen 80486, 80386'nın gelişmiş bir versiyonu görünümündedir. 80486 ile 80386 arasındaki en önemli fark 80486 içinde kayan noktalı sayılarla işlem yapabilecek bir FPU (Floatin-Point unit) bulunmasıdır.

80486 microişlemcinin ardından Intel, hızlarını, cache miktarlarını, işlem kapasitelerini artırarak x86 microişlemci serisini geliştirmeye devam etti. (Pentium I, Pentium pro, Pentium II, Pentium III, Pentium IV)



1.4 Neden Intel Tabanlı Microdenetleyici ?

Günümüzde microdenetleyicilerin birçok çeşidi bulunmaktadır. Intel, Microchip, Motorola, National, Atmel firmalarının ürettiği microdenetleyiciler bunlar arasında sayılabilir. Microdenetleyici gerektiren herhangi bir uygulama geliştirirken seçilecek microdenetleyicinin uygulamanın gereklerini karşılayıp karşılamadığı önemlidir. Bunun yanı sıra kullanılacak olan derleyici, programlayıcı, simülör, elektriksel özellikler, kaynak temini, seçenek çeşitliliği, kolay elde edilebilirlik, fiyat, Komut seti vb. özellikler de microdenetleyici seçiminde önemli etmenlerdir.

Atmel firmasının ürettiği Atmega serisi microcontrollerların yukarıda sayılan birçok özellikte öne çıktığı görülmektedir. Firma 8 bit Intel MCS-51 core'a sahip microdenetleyicilerin yanı sıra 16/32 bit veri yoluna sahip microdenetleyiciler, otomotiv temelli microdenetleyicileri de Intel MCS-51 core ile kullanıma sunmaktadır. Microdenetleyicilerin aynı core'u kullanıyor olmaları, komut setlerinin de birbirini desteklemesi anlamına gelmektedir. Bu nedenle herhangi bir microdenetleyici için yazılan program özelliklerin aynı olması durumunda hiçbir değişikliğe ihtiyaç duymadan farklı bir microdenetleyicide çalışabilmektedir. Atmel microdenetleyiciler elektriksel bozukluklardan etkilenmeyi minimize edebilmek için Watch dog timer, Brown-out dedektör vb. özellikler içermesi nedeniyle sanayi uygulamalarında oldukça yoğun kullanılmaktadır.

2. MCS-51 Microdenetleyici Ailesinin Mimari Özellikleri

2.1 MCS-51 Microdenetleyici Ailesinin Genel Mimari Yapısı

Sanayi Otomasyonu için günümüzde mevcut olan mikroişlemciler ve microdenetleyiciler arasında en uygunlardan birisi MCS-51 ailesinden olan microdenetleyicilerdir. Bu microdenetleyicilerin çekici tarafı, bu aileden olan her bir microdenetleyicinin çok küçük boy bir bilgisayar olması ile birlikte içermekte olduğu giriş-çıkış portlarının her birinin ve diğer önemli fonksiyon registerlerin (Special Function Registers SFR) büyük çoğunluğunun üzerinde bayt ve bit manipülasyonu yapmaya direkt imkan sağlamasıdır.

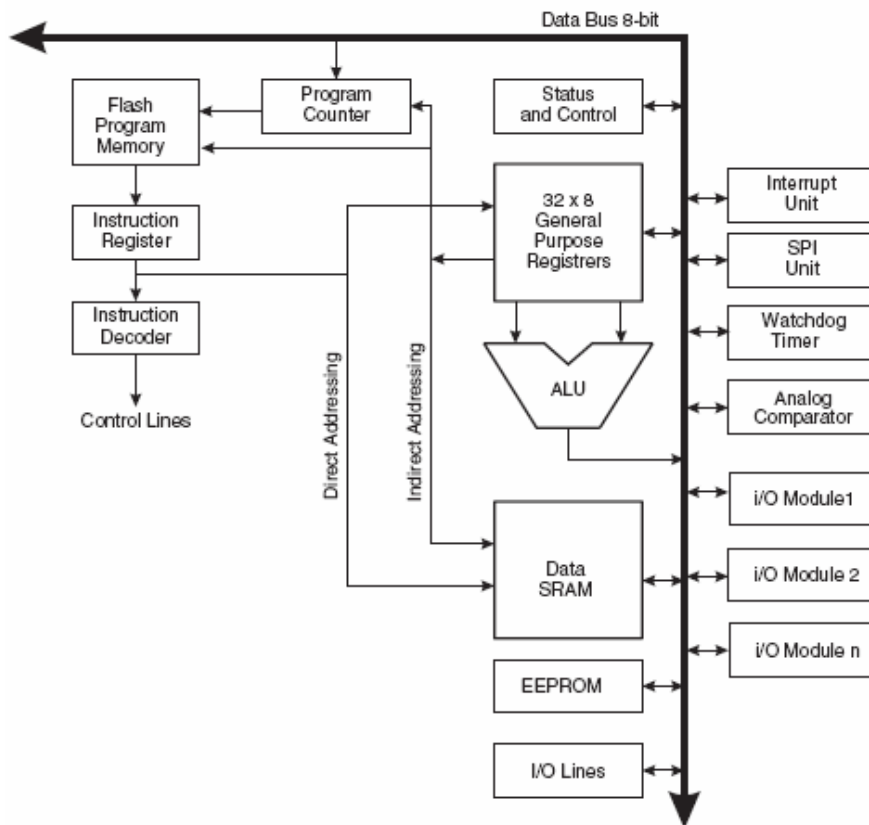
MCS-51 ailesinden olan microdenetleyici tabanlı kontrol kartına sanayi otomasyonuna yönelik küçük boy bilgisayar özelliği kazandırmaya imkan sağlayan özellikler sadece bir çip içerisinde yer alan aşağıdaki mimari özelliklerdir.

- 8/16/32 bit RISC merkezi işlem birimi (CPU),
- Dahili EEprom
- Dahili Flash Rom
- Harici veya dahili olarak seçilebilen osilatör,
- Bayt veya bit olarak düzenlenebilen, pull-up dirençleri aktif veya pasif yapılabilen Giriş / Çıkış portları,
- Static RAM bellek (İç Ram),
- Dış program/veri belleği (Bazı microdenetleyicilerde),
- 8/16 bit Zamanlayıcı/Sayaç/Pwm,
- İç ve dış kaynaklara hizmet edebilen, farklı vektörlere sahip kesme (Interrupt) sistemi,
- Seri Port,
- ISP Port,
- JTag Port,
- Security bitler aracılığı ile program ve verinin korunabilmesi,
- Watch dog timer, Brown-out dedektör,
- Analog-Digital çevirici (Adc)
- 20 MHZ kadar çalışma frekansı,
- Çalışma frekansına bağımlı (MIPS) çalışma performansı

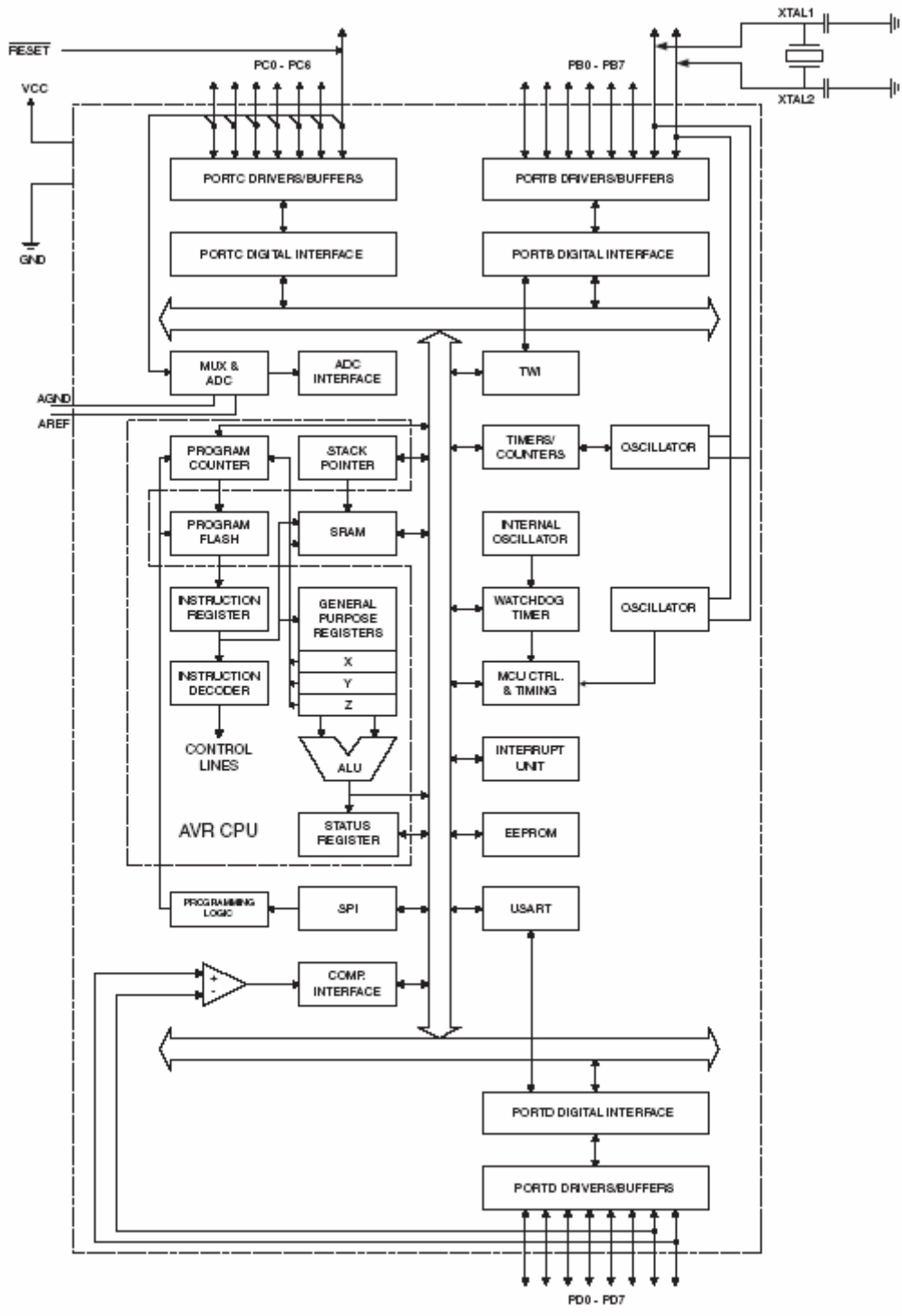
MCS-51 risc core'a sahip, Atmel firmasının atmega8 microdenetleyicisi yukarıda sözü edilen mimari yapının tümüne sahiptirler. Atmega8 microdenetleyicisinin özellikleri aşağıdaki şekilde özetlenebilir.

- Geliştirilmiş RISC mimari
 - 130 adet komut
 - 32 genel amaçlı register
 - Her clock'ta 1 komut çalıştırma
- 8Kb program tarafından da yazılabilen program belleği
- 512 Byte EEprom
- 1 Kb statik ram
- 2 adet 8 bit timer/counter
- 1 adet 16 bit timer/counter
- 3 adet PWM kanalı
- 6 kanal 10 bit ADC
- Byte temelli çift yönlü seri kanal
- Programlanabilir seri USART
- Master/Slave SPI seri kanal
- Programlanabilir Watchdog timer
- Chip içinde analog comparador
- Power on reset ve programlanabilen Brown-out dedektör
- Internal düzenlenebilir RC oscillator
- İç/Dış kaynaklı Interrupt sistemi
- 5 değişik uyuma modu
- 23 programlanabilir giriş-çıkış ucu
- 0-16 Mhz çalışma frekansı
- 4.5 – 5.5v çalışma voltajı

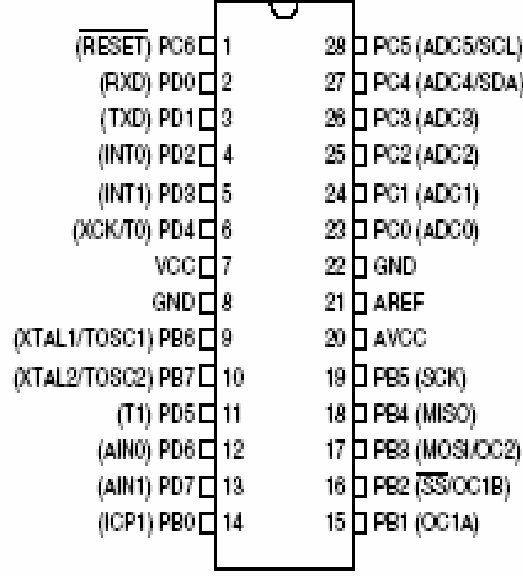
Şekil 2.1.1' de MCS-51 core mimarisinin blok diyagramını, şekil 2.1.2'de Atmega8 microdenetleyicinin blok diyagramı, şekil 2.1.3' de ise bu microdenetleyicinin bacak bağlantı şeması gösterilmiştir.



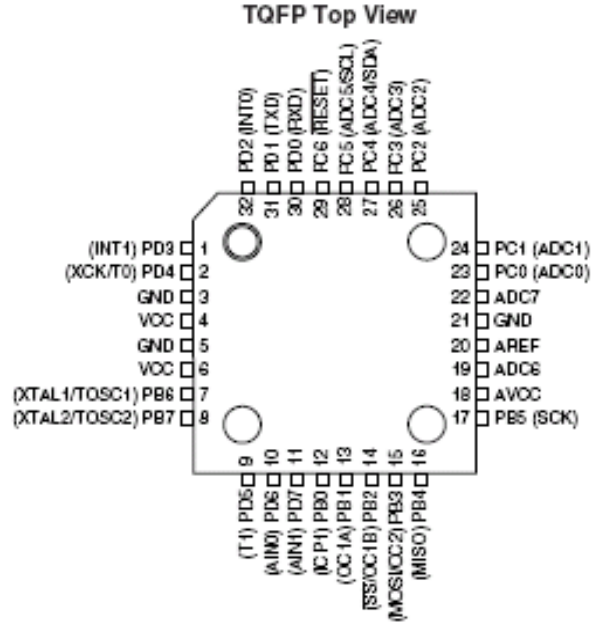
Şekil 2.1.1 MCS-51 core mimarisinin blok diyagramı



Şekil 2.1.2 Atmega8 Microdenetleyici Blok Diyagramı



Şekil 2.1.3 Atmega8 Microdenetleyici Bacak Bağlantı Şeması (PDIP)



Şekil 2.1.4 Atmega8 Microdenetleyici Bacak Bağlantı Şeması (TQFP)

2.2 Pin Anlamları

VCC

Sistem besleme voltajı. 4.5v-5.5v

GND

Sistem GND

Port B (PB0..PB7)

- Port B, programlanabilir pull-up dirençleri bulunan, 8 bit genişliğinde bir giriş-çıkış portudur. Porta byte olarak erişim yapılabileceği gibi bit olarak da erişim yapılabilir.
- Portun 6 ve 7. bitleri, Oscillator'ün internal RC seçilmesi durumunda port bitleri, aksi durumda oscillator giriş uçları

Xtal1/Xtal2/Tosc1/Tosc2

SPI

<p>Port C (PC0..PC6) PC7/Reset Avc, Seri giriş-çıkış</p>	<p>olarak kullanılır.</p> <ul style="list-style-type: none"> • Oscillator'un internal RC seçilmesi durumunda portun 6 ve 7. bitleri ASSR registerinin durumuna göre Timer/counter2'nin asenkron giriş uçları olarak kullanılabilir. Port ve portun fonksiyonlarının kullanımı ilerideki konularda detaylı olarak anlatılacaktır • Port C, programlanabilir pull-up dirençleri bulunan 7 bit genişliğinde bir giriş-çıkış portudur. Porta byte veya bit erişimleri yapılabilir. • Portun 7. biti RESDISBL yönlendirme biti ile anlam değiştirebilir. RESDISBL biti programlanmış ise pin giriş-çıkış pini, programlanmamış ise reset girişi olarak kullanılır. Port ve portun fonksiyonlarının kullanımı ilerideki konularda detaylı olarak anlatılacaktır.
<p>Port D (PD0..PD7) Usart, int, Timer/counter, Analog comparator</p>	<ul style="list-style-type: none"> • Port D, programlanabilir pull-up dirençleri bulunan 8 bit genişliğinde bir giriş-çıkış portudur. Port byte veya bit erimleri ile kullanılabilir. Port fonksiyonlarının kullanımı ilerideki konularda detaylı olarak anlatılacaktır.
<p>AVCC</p>	<p>Avcc analog digital çevirici için besleme voltaj girişidir. Bu voltaj girişi sağlanmadan ADC fonksiyonu kullanılamaz. ADC fonksiyonunun kullanımı için Avcc ucu alçak geçiren bir filtre ile besleme voltajına bağlanmalıdır.</p>
<p>AREF</p>	<p>Analog-Digital çevirici için referans voltaj girişidir.</p>

2.3 MCS-51 Microdenetleyici Ailesinin RAM yapısı ve Register'leri

2.3.1 Genel Amaçlı Registerler ve Stack

MCS-51 sisteminde register kavramı, diğer microdenetleyici sistemlerindeki genel amaçlı register kavramı ile aynıdır. Söz konusu sistemin mimari özelliklerinden biri bu registerlerle ilgilidir. MCS-51 sisteminde fiziksel olarak hiç bir register bulunmaz. Fakat MCS-51' in assembler dilini başka işlemcilerin alışılmış olan assembler diline yaklaştırmak amacıyla MCS-51'in assembler dilinde R0 – R31 gibi 32 adet register ismi kullanılmaktadır. Görünür ile gerçeklik arasında uyum her bir register ismi altında belli bir iç RAM hücresinin kullanılması sayesinde sağlanır. Bu uyumu sağlamak için gereken işlemler derleyici tarafından ve kullanıcıya şeffaf olarak gerçekleştirilir. Registerler iç ram'ın 00h -1Fh adresleri de yerleşir. Buna göre söz konusu registerlerin ismini veya adreslerini kullanmanın hiçbir farkı yoktur. Örneğin R5. registerinin içeriğinin akümülatöre gönderilmesi gerekiyorsa o zaman MOV A, R5 veya MOV A, \$05 yazılması yeterlidir.

Sonuç olarak sistemde 32 register vardır. Bu registerler iç ram'ın 00h - 1Fh adreslerine yerleşmektedir. Aslında registerler ram hücrelerin kendileridir. Bu hücreler komut görüntüsüne uygun olarak registerler diye adlandırılmaktadırlar. Böyle aldatmacalar

bilgisayar tekniğinde çok yaygın olarak kullanılmaktadır ve amacı kullanıcı alışkanlığı ile donanım gerçekliği arasındaki açıklığı mümkün olduğu kadar kapatmaktır.

7	0	Addr.	
R0		0x00	
R1		0x01	
R2		0x02	
...			
R13		0x0D	
R14		0x0E	
R15		0x0F	
R16		0x10	
R17		0x11	
...			
R26		0x1A	X-register Low Byte
R27		0x1B	X-register High Byte
R28		0x1C	Y-register Low Byte
R29		0x1D	Y-register High Byte
R30		0x1E	Z-register Low Byte
R31		0x1F	Z-register High Byte

Şekil 2.3.1.1 Registerlerin iç ram üzerindeki yerleşimi

Microdenetleyiciler iç ram olarak 8 bit ile adreslenemeyecek kadar büyük iç ram veya rom bulundurabilirler. Bu gibi durumlarda iç ram'a veya rom'a erişim için 16 bit registerler gerekli olurlar. Bu ihtiyacı karşılayabilmek için X,Y,Z registerleri düzenlenmiştir. Bu registerler R26 ile R31 registerleri üzerine yerleştirilmiş durumdadırlar ve genel olarak Indirect adresleme için kullanılırlar.



Şekil 2.3.1.2 X,Y,Z registerlerinin ram üzerindeki yerleşimi

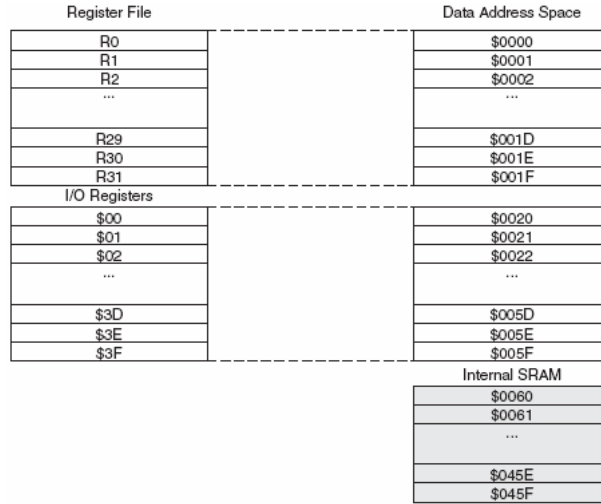
Stack, program içinde kullanılan geçici dataların saklanması amacıyla kullanılan bir alandır. Program içinde kullanılan local değişkenler, alt program çağrılarında ve Interrupt geçişlerinde geri dönüş adresleri ve parametreler stack alanı içinde tutulur. Stack LIFO (Last in First Out, Son giren ilk çıkar) metodu ile çalışır.

Registerler gibi stack'da ayrıca şema olarak mevcut değildir. Stack işlemleri SP (Stack Pointer) registerin yardımı ile iç ram'ın belli bir alanında simule edilir.

MCS-51 core içerisinde gerçekleştirilmiş olan mantığa göre microdenetleyicinin her bir resetlemesinden sonra SP = 0h durumu oluşur. Bu durum stack alanı ile register alanlarının çakışmalarına neden olur. Herhangi bir karışıklığın önlenmesi için

programların eğer stack kullanacaklarsa Stack Pointer'a stack olarak kullanacakları alanın başlangıç adresini yerleştirmeleri gereklidir. Stack teorik olarak 60h adresinden başlayarak FFh adresine kadar devam edebilir.

20h - 5Fh arasındaki iç ram alanı hayati önem taşımakta olan SFR (Special Function Register) için ayrılmıştır ve bu alanın başka amaçlar için kullanılması microdenetleyicinin işinin bozulmasına sebep olabilir. Bu registerler sonraki bölümde açıklanacaktır.



Şekil 2.3.1.3 Genel iç ram yerleşimi

2.3.2 Özel Amaçlı Registerler

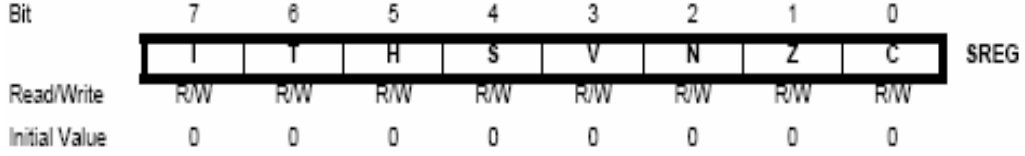
Bu registerler gurubunu sadece SFR' ler (Special Function Registers) olarak adlandıracağız. Bu registerlerden her biri çok özel olan bir veya birkaç fonksiyonun gerçekleştirilmesine hizmet ederler. Her bir programda SFR' lerin hepsinin kullanılması gerekmez. Yani herhangi bir programda yalnız o program için geçerli olan SFR alt gurubu kullanılır. Geride kalan SFR'ler hakkında düşünmeye gerek yoktur. Çünkü RESET sırasında bütün SFR' ler etkisiz duruma getirilir. SFR'leri açıklarken onları hizmet amaçlarına göre aşağıdaki guruplara bölünmesi uygun olacaktır.

- Microdenetleyicinin mikroişlemci kısmının temel görevlerini sürdürebilmesi için gereken SFR'ler.
- Microdenetleyicinin içinde bulunan Timer/Counter'ları denetleyen SFR' ler.
- Microdenetleyici içerisinde bulunan Usart'ı denetleyen SFR'ler,
- Microdenetleyici içerisinde bulunan interrupt devresi denetleyen SFR' ler.
- Giriş / Çıkış portları olarak kullanılan SFR' ler,
- Diğer SFR'ler

Özel Amaçlı Registerler başlığı altında sadece Microdenetleyicinin mikroişlemci kısmının temel görevlerini sürdürebilmesi için gereken SFR'lerden bahsedilecek, Timer/Counter, Usart, Interrupt ve diğer SFR'ler ayrı başlıklar halinde incelenecektir.

2.3.2.1 SREG (Status Register)

Bu register yapılan matematiksel işlemlerin ve microdenetleyicinin genel durumunu tutmak için kullanılır. Register bir bayt olarak veya 8 ayrı bit olarak adreslenebilir. Register iç ram'de 5Fh adresinde yer alır. SREG'in bit yapısı aşağıdaki gibidir.



- I Global Interrupt Enable**
- T Bit Copy Storage**
- H Half Carry Flag**
- S Sign Bit**
- V Two's complement overflow flag**
- N Negative Flag**
- Z Zero Flag**
- C Carry Flag**

SREG Registerinin bit değerleri, yapılan işlemlere göre microdenetleyici tarafından oluşturulur. Fakat programlar tarafından istenildiği şekilde değiştirilebilir.

(I) Global Interrupt Enable :

I biti, microdenetleyici içindeki iç ve dış interruptları genel olarak açmak ve kapatmak için kullanılır. Eğer bit set durumda ise microdenetleyici kurulmuş interruptların çalışmasına imkan verecektir. Aksi durumda interruptlar kurulmuş olsa bile interrupt aktif olmayacaktır.

Eğer I biti set edilmişse interrupt oluştuğunda microdenetleyici bit değerini kendiliğinden resetler ve reti komutu ile tekrar set yapar. Bu yöntem ile microdenetleyicinin peşpeşe oluşan interruptlarla stack taşmasının önüne geçilmiş olur.

I biti sei komutu ile set, cli komutu ile resetlenebilir.

(T) Bit Copy Storage

T biti, bit kopyalama işlemlerinde kaynak veya hedef olarak kullanılır. Bit BLD ve BST komutlarında geçici alan olarak kullanılır. BLD komutu T bitini belirlenen registerin istenen bitine kopyalar.

$Rd(b) \leftarrow T$

Örnek :

BLD r16,4

T bitinin değerini r16 registerinin 4. bitine kopyalar.

BST Komutu belirlenen registerin belirlenen bitini T registerine kopyalar.

$T \leftarrow Rd(b)$

Örnek :

BST r16,2

R16 registerinin 2. bitini T bitine kopyalar.

(H) Half Carry Flag (Auxiliary Carry)

Microdenetleyici matematiksel işlemlerin sonunda H flagını set veya reset yapar. Flag genel olarak matematiksel işlemlerde alıcı alanın alt nibble'ında bir taşma olursa set olur. Bu durum BCD matematiksel işlemlerin yapılabilmesini sağlar.

(S) Sign Bit

Microdenetleyicinin matematiksel işlemler sonunda etkilediği bir flagdır. Flag daima N ve V flaglarının toplamını ifade eder.

(V) Two's complement overflow flag

Flag iki tabanlı sayılarla yapılan matematiksel işlemlerde byte üzerindeki taşmayı kontrol etmek için kullanılır. Yapılan matematiksel işlemler sonunda microdenetleyici flagı set veya reset yapar.

(N) Negative Flag

İşaretili sayılarla işlem yaparken işlem sonucunun işareti hakkında bilgi verir. Flagın set olması işlem sonucunun negatif, aksi pozitif olduğunu belirtir.

(Z) Zero Flag

Bir matematiksel işlem sonunda sonuç 0 ise veya bir atama işleminde alıcı alana aktarılan değer 0 ise flag set olur. Bu özellik karşılaştırma işlemlerinde oldukça sık kullanılır.

(C) Carry Flag

Bu flag ALU vasıtası ile peşpeşe yapılan toplama ve çıkartma işlemlerinin gerçekleştirilmesi ve başka amaçlar için kullanılabilen bir flagdır. Toplama işlemlerinde flag elde (carry), çıkartma işlemlerinde borç (barrow) flagı olarak kullanılır. Çıkartma işleminde barrow'un dikkat dışı bırakılması gerektiğinde işlemden önce C=0 yapılmalıdır. Toplama işlemleri için ise iki değişik komut; ADD ve ADDC komutları kullanıldığından söz konusu ön hazırlığa ihtiyaç yoktur.

2.3.2.2 SP Registeri (Stack Pointer)

Bu register stack göstergesi olarak kullanılır. 5Dh ve 5Eh adreslerine yerleştirilmiştir. Stack teorik olarak tüm ram üzerinde olabileceği için 16 bit'lik bir register olarak düzenlenmiştir. SP olarak 16 bit erişim sağlanabileceği gibi SPH (5Eh) ve SPL (5Dh) olarak 8 bitlik erişim de sağlanabilir.

2.3.2.3 MCUCR Registeri (MCU Control Register)

MCUCR registeri genel olarak microdenetleyicinin güç kontrolü ve dış interruptların nasıl tetikleneceğini belirlemek için kullanılır. Registerin genel yapısı;

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SE SM0..SM2 bitleri ile belirlenen microdenetleyici güç uyuma modunun aktif edilmesini sağlayan bittir. Bitin set edilmesi durumunda SM0..SM2 ile belirlenen güç uyuma modunu aktif hale getirir.

SM0..SM2 Microdenetleyici güç uyuma modlarının seçimini sağlayan bitlerdir.

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby ⁽¹⁾

Not 1 : Bu mod sadece external oscillator kullanıldığında geçerlidir.

Idle Sleep Mode : SM0, SM1 ve SM2 bitleri 0 yapıldığında microdenetleyici idle güç uyuma moduna geçer. Bu modda CPU ve flash dışındaki tüm birimler çalışır durumdadır. Mod aktif edildikten sonra herhangi bir dış interrupt geldiğinde, herhangi bir iç timer overflow olduğunda, USART'tan bir karakter alındığında microdenetleyici Idle moddan çıkar.

ADC Noise Reduction : SM0 1, SM1 ve SM2 bitleri 0 yapıldığında ADC gürültü bastırma moduna geçilir. Bu modda I/O, CPU ve flash kapatılır. Bu birimler dışındaki birimler çalışır durumdadır. ADC Noise Reduction güç uyuma modu yüksek çözünürlüklü ADC çevrimlerinde oluşabilecek parazitleri engellemek içindir. Bu nedenle mod adc okuma çevrimine girildiğinde kendiliğinden devreye girer.

Power-Down : MCUCR registeri içinde SM0,SM1,SM2 bitleri 010 yapılırsa microdenetleyici Power-Down güç uyuma moduna geçer. Bu modda external interruptlar, Two wire serial Interface ve watchdog hariç her şey durdurulur. Power-Down modunda iken açık external interruptlardan birinin durum değiştirmesi moddan çıkmak için yeterlidir.

Power-Save : SM0, SM1 ve SM2 bitleri 011 yapıldığında microdenetleyici power-save moduna girer. Power-save modu power down modu ile çok benzer şekilde çalışır. Modun Power-down modundan farkı, timer2 asenkron modda çalışıyorsa timer2'nin durdurulmamasıdır.

Standby : Bu mod, microdenetleyici external crystal/resonator ile çalışacak şekilde düzenlenmişse geçerlidir. Standby, power-down modu ile aynı özelliklere sahiptir. Ancak standby modundan çıkış için 6 clock gereklidir.

Microdenetleyiciye ait tüm güç uyuma modları özet olarak aşağıdaki gibi gösterilebilir.

Sleep Mode	Active Clock Domains					Oscillators		Wake-up Sources					
	clk _{CPU}	clk _{FLASH}	clk _{IO}	clk _{ADC}	clk _{ASY}	Main Clock Source Enabled	Timer Osc. Enabled	INT1 INT0	TWI Address Match	Timer 2	SPM/EEPROM Ready	ADC	Other I/O
Idle			X	X	X	X	X ⁽²⁾	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X ⁽²⁾	X ⁽³⁾	X	X	X	X	
Power Down								X ⁽³⁾	X				
Power Save					X ⁽²⁾		X ⁽²⁾	X ⁽³⁾	X	X ⁽²⁾			
Standby ⁽¹⁾						X		X ⁽³⁾	X				

Active Clock Domains : Aktif clocklar

Wake-up Sources : Uyandırma Kaynakları

- (1) Clock kaynağı olarak external crystal/resonator seçilmişse
- (2) ASSR registerinde ASR biti set edilmişse
- (3) INT0 ve INT1 seviye tetiklemeli olarak seçilmişse

ISCxx ISCxx bitleri INT0 ve INT1 external interruptlarının nasıl tetikleneceğini gösteren bitlerdir. ISC00 ve ISC01 interrupt 0, ISC01 ve ISC11 interrupt 1 için tetikleme şekli belirler. Interrupt 0 için tetikleme tipleri aşağıdaki gibidir.

ISC01	ISC00	Açıklama
0	0	Interrupt 0 seviyesinde tetiklenir
0	1	Interrupt logic değişimlerde tetiklenir
1	0	Interrupt düşen kenarda tetiklenir
1	1	Interrupt çıkan kenarda tetiklenir

Interrupt 1 için tetikleme tipleri aşağıdaki gibidir.

ISC11	ISC10	Açıklama
0	0	Interrupt 0 seviyesinde tetiklenir
0	1	Interrupt logic değişimlerde tetiklenir
1	0	Interrupt düşen kenarda tetiklenir
1	1	Interrupt çıkan kenarda tetiklenir

2.3.2.4 MCUCSR Registeri (MCU Control and Status Register)

MCUCSR registeri microdenetleyicinin hangi kaynaktan resetlendiğini program içinde tesbit etmek için kullanılır. Registerin yapısı :

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	See Bit Description				

Bit 4..7 : Registerdeki 4-7 arasındaki bitler yedek bitlerdir ve değerleri daima 0'dır.

Bit 3 : WDRF Watchdog Reset Flag
Microdenetleyici Watchdog tarafından resetlenmişse flag set olur.

Bit 2 : BORF Brown-out Reset Flag
Microdenetleyici internal Brown-out devresi tarafından resetlenmişse flag set olur.

Bit 1 : EXTRF External Reset Flag
Microdenetleyici reset ucu ile resetlenmişse flag set olur.

Bit 0 : PORF Power-on Reset Flag
Microdenetleyiciye ilk kez enerji verildiğinde flag set olur.

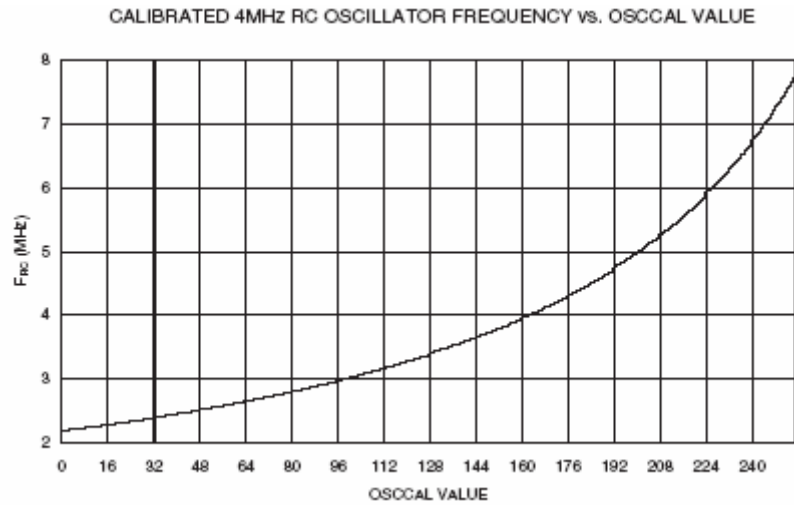
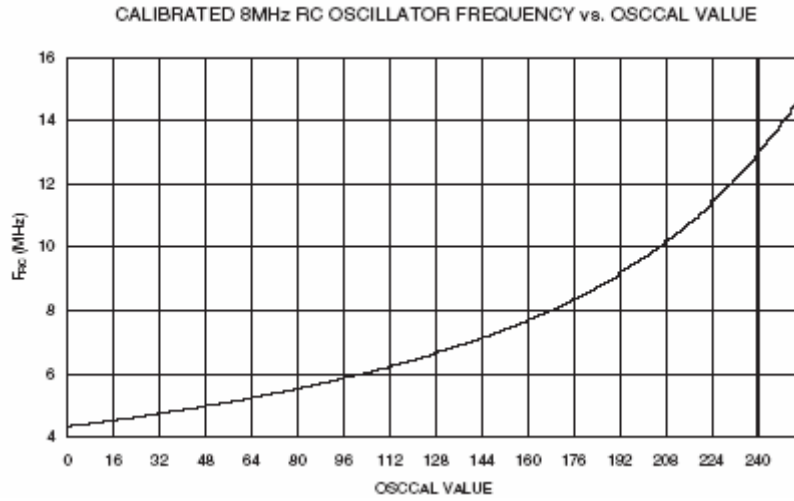
MCUCSR Registeri microdenetleyici resetlendikten sonra program yolu ile sıfırlanmalıdır.

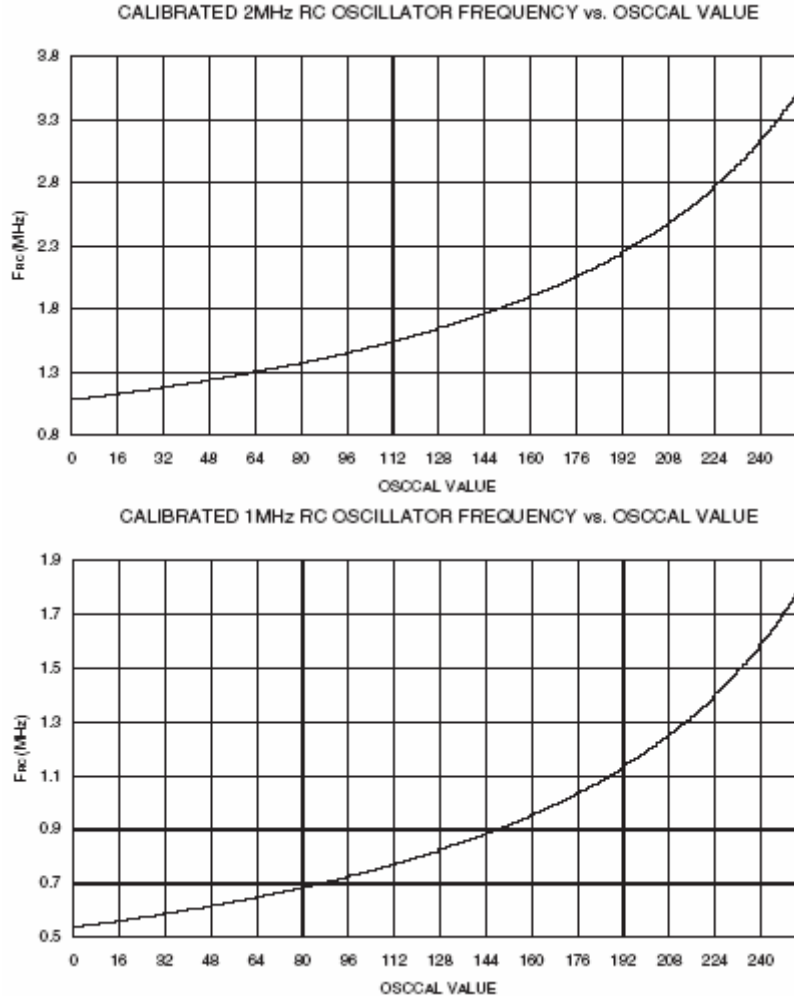
2.3.2.5 OSCCAL Registeri (Oscillator Calibration)

Register, microdenetleyici internal RC oscillator ile çalıştırıldığında, oscillator'ü ayarlamak için kullanılır. Internal RC ilk kez çalıştırıldığında oscillator 1 Mhz, OSCCAL registeri 0 olarak ayarlanmıştır. Oscillator frekansı, OSCCAL registerinin değeri değiştirilerek % cinsinden ayarlanabilir. Register yazılacak 0 dışındaki değerler oscillator frekansının artmasına neden olur. Bu yöntemle yapılan frekans değişimleri flash ve eeprom erişimlerini direkt etkiler. Bu nedenle flash ve eeprom erişimlerinde frekans %10 dan fazla artırılmamalıdır. OSCCAL frekans değişimleri aşağıdaki tabloda gösterilmiştir.

OSCCAL Value	Min Frequency in Percentage of Nominal Frequency (%)	Max Frequency in Percentage of Nominal Frequency (%)
0x00	50	100
0x7F	75	150
0xFF	100	200

Farklı oscillator seçimlerinde OSCCAL register değerinin frekans üzerindeki etkisi :





2.3.2.6 SPMCR Registeri (Store Program Memory Control Register)

SPMCR registeri program hafızasının programlanması sırasında kullanılacak olan bitleri içerir. Microdenetleyici içinde çalışan bir program tarafından program hafızasının (Flash Rom) programlanması boot loader programlarının vazgeçilmez özelliğidir. Bu nedenle register içeriği boot loader başlığı altında detaylı olarak incelenecektir. Registerin genel yapısı aşağıdaki gibidir.

Bit	7	6	5	4	3	2	1	0	
	SPMIE	RWWSB	-	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCR
Read/Write	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

2.3.2.7 SFIOR Registeri (Special Function In/Out Register)

Register genel olarak I/O düzenlemelerinde kullanılır. Register bitleri ve bit anlamları aşağıdaki gibidir.

Bit	7	6	5	4	3	2	1	0	SFIOR
	-	-	-	-	ACME	PUD	PSR2	PSR10	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PSR10 : Timer/Counter0 ve Timer/Counter1 tarafından kullanılan prescaller'ın (bölücü) resetlenmesini sağlar. Resetleme bit set yapıldığında ve timer/counter yeniden start edildiğinde gerçekleşir. Bit okunduğunda her zaman 0 değerini verecektir.

PSR2 : Timer/Counter2 tarafından kullanılan prescaller'ın (bölücü) resetlenmesini sağlar. Resetleme bit set yapıldığında ve timer/counter yeniden start edildiğinde gerçekleşir. Bit okunduğunda her zaman 0 değerini verecektir.

PUD : I/O portlara ait açılmış tüm pull-up dirençlerin kapatılmasını sağlar.

ACME : Microdenetleyici içindeki ADC kapalı ve ACME biti set ise Analog comparator'un negatif girişi ADC multiplexer tarafından seçilir. ACME biti logic 0 ise Analog comparator'un negatif girişi AIN1 ucundadır. Bitin detaylı açıklaması Analog Comparator bölümünde detaylı olarak anlatılacaktır.

Özel Amaçlı registerlerin microdenetleyici fonksiyonları açısından anlatımına başlamadan önce Atmel Mcs-51 komut setinin, program yazımı ve verilen örneklerin anlaşılabilmesi açısından kısaca anlatımında yarar bulunmaktadır.

3. Atmel MCS-51 Microdenetleyici Komut Seti

3.1 Genel Assembler Bilgileri

Her bir assembler komutu bir, iki, üç veya dört kısımdan oluşabilir. Komut kısımları bir birinden virgöl (,) vasıtasıyla ayrılır. Her bir komutun birinci kısmı komutun görevini ifade eder. Mesela MOV - gönder, ADD - topla, ANL - lojik çarp, JMP - atla, XCH - değiştir vs. Komutun geriye kalan kısımlarını onun operandları (üzerinde işlem yapılacak olan bilgiler) oluşturur. Bu anlamda sıfır, bir, iki ve üç operandlı komutlar söz konusu olacaklardır. Komutta her hangi bir operand olarak veri (data), adres ve register ismi kullanılabilir. Komularda sabit veri olarak 2' li, 16' lı ve 10' lu sayı sisteminde ifade edilmiş sayılar kullanılabilir. Verilmiş olan sayıyı 2' li sistemde ifade edilmiş olduğunu bildirmek için sayının başına 0b eklenir. Aynı amaçla 16' lı sayının başına 0x eklenir. Eğer sayının sonuna veya başına hiç bir işaret eklenmemiş ise o zaman söz konusu sayının 10' lu sistemde verilmiş olduğu varsayılır. Örneğin :

```
ldi r16, 0b10001001
ldi r15, 0x0F
ldi r14, 112
```

Herhangi bir microdenetleyici komut seti ile programlama için kullanılan assembler derleyici arasında sıkı bir bağ bulunmaktadır. Assembler derleyiciler program yazımını kolaylaştırmak amacıyla komut setine ek olarak microdenetleyici komut setinden farklı komutlar sunabilmektedirler. Bu nedenle, bundan sonra anlatılacaklar Atmel firmasının AVR Studio 4 yazılımı esas alınarak anlatılacaktır. Yazılım bir assembler compiler'ın yanı sıra bir microdenetleyici simülatörü, bir flash programlayıcı ve debugger içermektedir. (Yazılım http://www.atmel.com/dyn/resources/prod_documents/aStudio4b460.exe linkinden ücretsiz olarak alınabilir.)

3.2 Assembler Yönlendirme Komutları

Avr Studio 4 yazılımı içinde bulunun compiler program yazılımını kolaylaştırmak için aşağıda listesi verilen komutları microdenetleyici komut seti dışında programcıya sunmaktadır.

BYTE : Komut, statik ram üzerinde değişkenler için yer ayırmak için kullanılır. Komut sadece Data segment üzerinde yer ayırmak amaçlı kullanılabilir. Örneğin :

```
.DSEG
Var1:      .BYTE 1      ;var1 için 1 byte yer ayırır
Var2:      .BYTE Abc    ;var2 için Abc kadar yer ayırır
Var3:      .BYTE 5      ;var3 için 5 byte yer ayırır
```

```
.CSEG
    Ldi r30,low(Var1) ;Var1 deęişken adresinin düşük kısmını
                        r30 registerine yükler. (Z registerinin
                        düşük kısmı)
    Ldi r31,high(Var1) ; Var1 deęişken adresinin yüksek kısmını
                        r31 registerine yükler. (Z registerinin
                        yüksek kısmı)
    Ld r1,Z           ; Var1 deęişkeni içindeki deęer r1
                        registerine yüklenir.
```

CSEG : Komut, code segmentin başlangıcını belirlemek için kullanılır. Örnek :

```
.DSEG           ; Data segment başlangıcı
Var1           .BYTE 5
.CSEG          ; Code segment başlangıcı
Const: .DW 2
Mov r0,r1
```

DB : Komut program hafızasında veya EEprom üzerinde 1 byte'lık yer ayırmak için kullanılır. Örnek:

```
.CSEG
    consts: .DB 0, 255, 0b01010101, -128, 0xaa
.ESEG
    const2: .DB 1,2,3
```

DEF : Registerlere sembolik isimler vermek için kullanılır. Örnek :

```
.DEF temp=R16
.DEF ior=R0
.CSEG
    ldi temp,0xf0 ; Temp registerine f0 deęerini yükler
    in ior,0x3f ; SREG registerinin içerięini ior registerine yükler
```

DEVICE : Komut, assembler derleyiciye hangi microdenetleyici için kod üreteceęi konusunda bilgi üretir. Komut ile kullanılabilen microdenetleyici isimleri aşıęıdaki tabloda verilmiştir.

Klasik	Tiny	Mega	Dięer
AT90S120	ATtiny11	ATmega8	AT94K
AT90S4433	ATtiny2313	ATmega48	AT86RF401
AT90S2313	ATtiny12	ATmega16	AT90CAN128
AT90S2323	ATtiny13	ATmega161	
AT90S2333	ATtiny22	ATmega162	
AT90S4414	ATtiny26	ATmega163	
AT90S4434	ATtiny25	ATmega169	

AT90S8515	ATtiny45	ATmega32	
AT90S8534	ATtiny85	ATmega323	
AT90S8535		ATmega103	
AT90S2343		ATmega104	
		ATmega8515	
		ATmega8535	
		ATmega64	
		ATmega128	
		ATmega165	
		ATmega2560	
		ATmega2561	

Örnek :

.DEVICE ATmega8

Komut derleyicinin program hafızası, data hafızası, eeprom uzunluklarının kontrol edebilmesini sağlar.

DSEG : Data segment başlangıcını tanımlamak için kullanılır. Örnek :

```
.DSEG          ; Data segment başlangıcı
Var1          .BYTE 5
.CSEG          ; Code segment başlangıcı
Const: .DW 2
Mov r0,r1
```

DW,DQ,DD : Komut program hafızasında veya EEprom üzerinde yer ayırmak için kullanılır. DW 2 byte, DD 4 byte, DQ 8 byte yer ayırır.

Örnek:

```
.CSEG
const: .DW 0, 255, 0b01010101, -128, 0xaa
cc1: .DD 111
cc2: .DQ 222
.ESEG
const2: .DW 1,2,3
```

ESEG : EEprom başlangıcını tanımlamak için kullanılır. Örnek:

```
.ESEG
const2: .DB 1,2,3
```

EXIT : Assembler dosyasının sonunu belirlemek için kullanılır.

EQU : Herhangi bir etikete değer vermek için kullanılır. EQU ile değer atanan herhangi bir etiket değeri program içinde değiştirilemez. Örnek

```
.EQU io_offset = 0x23
.EQU porta = io_offset + 2
.CSEG ; Code segment başlangıcı
    clr r2 ; r2 registeri temizlendi
    out porta,r2 ; r2 porta'ya yazıldı
```

INCLUDE : Assembler dosyasının içinden bir başka assembler dosyasını çağırmak için kullanılır. Örnek

```
; iodefns.asm:
.EQU sreg = 0x3f ; Status register
.EQU sphigh = 0x3e ; Stack pointer üst
.EQU splow = 0x3d ; Stack pointer alt
; incdemo.asm
.INCLUDE iodefns.asm ; I/O tanımlama dosyasını çağır
    in r0,sreg ; Status registeri oku
```

MACRO : Komut bir macronun başlangıcını gösterir. Macro deyiminden sonra macro'nun isminin yazılması gerekir.

```
.MACRO Abc
.ENDMACRO
```

```
.MACRO topla
.ENDMACRO
```

Macro deyimini, isimden sonra 0 ile 9 arasında adlandırılan parametre alabilir ve .endmacro deyimini ile sonlanmalıdır.

```
.MACRO topla
    Mov r0,@0
    Mov r1,@1
    Add r0,r1
.ENDMACRO
```

```
.CSEG
    Topla 5,152
```

ORG : ORG komutu bölüm sayacına değer atamak için kullanılır. Data hafızası, program hafızası veya eeprom içinde çalışma noktasını istenen herhangi bir yere kaydırır. Örnek

```
.DSEG ; Data segment başlangıcı
.ORG 0x120 ; Bölüm sayacına 120 hex değerini atar
    v1: .BYTE 1 ; 120 hex adresinde v1 için 1 byte yer ayırır
.CSEG
.ORG 0x10 ; Bölüm sayacına 10 hex değerini atar
    mov r0,r1 ; Program kodları 10 hex den başlayarak yerleşir.
```

3.3 Assembler Operatörleri

Assembler komut operandları içinde, operandlar üzerinde direkt işlem yapmak için kullanılan işaretlerdir. Bu işaretler :

Operatör	Açıklama
!	Logical Not
~	Bitwise Not
-	Unary Minus
*	Multiplication
/	Division
%	Modulo (AVRASM2)
+	Addition
-	Subtraction
<<	Shift left
>>	Shift right
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal
!=	Not equal
&	Bitwise And
^	Bitwise Xor
	Bitwise Or
&&	Logical And
	Logical Or
?	Conditional operator (AVRASM2)

Logical Not

Sembol : !

Açıklama : Operatörde kullanılan değer 0 ise 1, sıfır olmayan bir değer ise 0 döndürür.

Örnek : ldi r16,!0xf0 ;r16 registerine 0 değerini atar.

Bitwise Not

Sembol : ~

Açıklama : Operatörde kullanılan değerın tüm bit değerlerini tersine çevirir.

Örnek : ldi r16,~0xf0 ; r16 registerine 0x0f değerini atar.

Minus

Sembol : -
Açıklama : Operatör kendisinden sonra kullanılan değeri negatif yapar.
Örnek : ldi r16,-2 ; r16 registerine -2 (0xfe) değerini yükler.

Multiplication

Sembol : *
Açıklama : Bu operatör iki değerın çarpımını döndürür.
Örnek : ldi r30,label*2 ;r30 registerine label*2 değerini atar.

Division

Sembol : /
Açıklama : Bu operatör soldaki değerin sağdaki değere bölümünün tamsayı cinsinden sonucunu döndürür.
Örnek : ldi r30,label/2 ;r30 registerine label/2 sonucunu yükler

Modulo (AVRASM2)

Sembol : %
Açıklama : Operatör soldaki değerin sağdaki değere bölümünden kalanı tamsayı olarak döndürür.
Örnek : ldi r30,label%2 ;r30 registerine label%2 sonucunu yükler

Addition

Sembol : +
Açıklama : Operatör iki değerin toplamını döndürür.
Örnek : ldi r30,c1+c2 ;r30 registerine c1+c2 sonucunu yükler.

Subtraction

Sembol : -
Açıklama : Operatör soldaki ifade ile sağdaki ifadenin farkını döndürür.
Örnek : ldi r17,c1-c2 ;r17 registerine c1-c2 sonucunu yükler.

Shift left

Sembol : <<
Açıklama : Operatör kendisinden önce kullanılan değerin bitlerini sağında kullanılan sayı kadar sola kaydırarak sonucu döndürür.
Örnek : ldi r17,1<<4 ;r17 registerine 1 in 4 kez sola kaydırılmış halini yükler.

Shift right

Sembol : >>
Açıklama : Operatör kendisinden önce kullanılan değerin bitlerini sağında kullanılan sayı kadar sağa kaydırarak sonucu döndürür.
Örnek: ldi r17,c1>>c2 ;r17 registerine c1 değerini c2 kez sağa kaydırarak yükler

Less than

Sembol : <

Açıklama : Operatörün solundaki değer sağındaki değerden küçükse 1, aksi durumda 0 döndürür.

Örnek : ldi r18,bitmask*(c1<c2)+1 ;r18 registerine işlem sonucunu aktarır.

Less or equal

Sembol : <=

Açıklama : Operatörün solundaki değer sağındaki değerden küçük veya eşitse 1, aksi durumda 0 döndürür.

Örnek : ldi r18,bitmask*(c1<=c2)+1 ;r18 registerine işlem sonucunu aktarır.

Greater than

Sembol : >

Açıklama : Operatörün solundaki değer sağındaki değerden büyükse 1, aksi durumda 0 döndürür.

Örnek : ldi r18,bitmask*(c1>c2)+1 ;r18 registerine işlem sonucunu aktarır.

Greater or equal

Sembol : >=

Açıklama : Operatörün solundaki değer sağındaki değerden büyükse veya eşitse 1, aksi durumda 0 döndürür.

Örnek : ldi r18,bitmask*(c1>=c2)+1 ; r18 registerine işlem sonucunu aktarır.

Equal

Sembol : ==

Açıklama : Operatörün sonundaki değer ile sağındaki değer eşitse 1, aksi durumda 0 döndürür.

Örnek : ldi r18,bitmask*(c1==c2)+1 ; r18 registerine işlem sonucunu aktarır.

Not equal

Sembol : !=

Açıklama : Operatörün solundaki değer ile sağındaki değer eşit değilse 1, aksi durumda 0 döndürür.

Örnek : .SET flag=(c1!=c2) ;flag değişkenine 1 veya 0 değerini aktarır

Bitwise And

Sembol : &

Açıklama : Operatörün solundaki ve sağındaki değerlerin bitlerini karşılıklı olarak and logical işlemine tabi tutar.

Örnek : ldi r18,0x0A & 0x02 ;r18 registerine 2 değerini atar.

Not : And işleminde işleme tabi tutulan iki bit 1 ise sonuç 1 aksi halde sonuç 0 olur.

Bitwise Xor

Sembol : ^
Açıklama : Operatörün solundaki ve sağındaki değerlerin bitlerini karşılıklı olarak xor logical işlemine tabi tutar.
Örnek : ldi r18,0x0A ^ 0x02 ;r18 registerine 0x48 değerini atar.
Not : Xor işleminde işleme tabi tutulan iki bit 1 ise veya 0 ise sonuç 0 aksi durumda sonuç 1 olur.

Bitwise Or

Sembol : |
Açıklama : Operatörün solundaki ve sağındaki değerlerin bitlerini karşılıklı olarak or logical işlemine tabi tutar.
Örnek : ldi r18,0x0A | 0x02 ;r18 registerine 0x4A değerini atar.
Not : Or işleminde işleme tabi tutulan iki bitten herhangi birisi 1 ise sonuç 1 aksi durumda sonuç 0 olur.

Logical And

Sembol : &&
Açıklama : Operatörün her iki tarafındaki değerlerden bir tanesinin 0 olması durumunda 0, aksi durumda 1 döndürür.
Örnek : ldi r18,0x0A && 0x02 ;r18 registerine 1 değerini atar.

Logical Or

Sembol : ||
Açıklama : Operatörün her iki tarafındaki değerlerden her ikisinin de 0 olması durumunda 0, aksi durumda 1 döndürür.
Örnek : ldi r18,0x0A || 0x02 ;r18 registerine 1 değerini atar.

Conditional operator (AVRASM2)

Sembol : ? :
Düzen : Logic? exp1 : exp2
Açıklama : Logic olarak verilen ifade doğru ise exp1 değerini aksi halde exp2 değerini döndürür.
Örnek : ldi r18, 1 > 2? 10 : 20 ; r18 registerine 20 değerini atar.

3.4 Hazır Fonksiyonlar

Hazır fonksiyonlar avr studio 4 assembler içinde bulunan hazır fonksiyonlardır. Bu fonksiyonlar :

- LOW(exp) 16 bitlik exp ifadesinin düşük byte değerini döndürür
- HIGH(exp) 16 bitlik exp ifadesinin yüksek byte değerini döndürür
- BYTE2(exp) exp ifadesinin 2. byte değerini döndürür
- BYTE3(exp) exp ifadesinin 3. byte değerini döndürür

- BYTE4(exp) exp ifadesinin 4. byte deęerini döndürür
- LWRD(exp) exp ifadesinin 1. ve 2. byte deęerlerini döndürür.
- HWRD(exp) exp ifadesinin 1. ve 2. byte deęerlerini döndürür.
- PAGE(exp) exp ifadesinin 16-21. bit deęerlerini döndürür.
- EXP2(exp) exp ifadesinin karesini döndürür.
- LOG2(exp) exp ifadesinin logaritmasını tamsayı olarak döndürür.

Aşağıdaki fonksiyonlar sadece avrasm2 de geçerlidir.

- INT(exp) exp olarak verilen kayan noktalı sayıyı tamsayıya çevirir. Çevrim sırasında noktadan sonraki kısım göz ardı edilir.
- FRAC(exp) exp olarak verilen kayan noktalı sayının noktadan sonraki kısmını geri döndürür.
- ABS(exp) exp sayısının gerçek deęerinin döndürür. (-10=10 vb.)

3.5 Komut Seti

3.5.1 Data Transfer Komutları

Data transfer komutları genel olarak, herhangi bir hafıza adresine değer atamak veya herhangi bir hafıza adresindeki bir değeri başka bir hafıza adresine aktarmak için kullanılan komutlardır. Komutların genel özeti aşağıdaki gibidir.

Not : Komut operandı olarak kullanılan:

Rd	Hedef registeri
Rr	Kaynak registeri
b	0 ile 7 arasındaki sabit bir sayıyı
s	0 ile 7 arasındaki sabit bir sayıyı
K8	8 bitlik sabit bir sayıyı
K6	6 bitlik sabit bir sayıyı
k	Sabit hafıza(SRAM) adresini
q	64 bitlik sabit bir sayıyı
P	16 veya 32 bitlik sabit bir sayıyı
K	Sabit bir sayıyı

ifade etmektedir.

Komut	Operandları	Açıklama	Operasyon	Etkilenen Flaglar
Mov	Rd,Rn	Registeri Kopyala	$Rd = Rr$	Yok
Movw	Rd,Rr	16 bit register kopyala	$Rd+1:Rd = Rr+1:Rr$	Yok
Ldi	Rd,K8	Registere kopyala	$Rd = K8$	Yok
Lds	Rd,k	Hafıza adresinden kopyala	$Rd = (k)$	Yok
Ld	Rd,X	X registerinin gösterdiği yerden kopyala	$Rd = (X)$	Yok
Ld	Rd,X+	X registerinin gösterdiği yerden kopyaladıktan sonra X registerini 1 artır.	$Rd = (X), X=X+1$	Yok
Ld	Rd,-X	X registerinden 1 çıkarttıktan sonra X registerinin gösterdiği yerden kopyala	$X=X-1, Rd = (X)$	Yok
Ld	Rd,Y	Y registerinin gösterdiği yerden kopyala	$Rd = (Y)$	Yok
Ld	Rd,Y+	Y registerinin gösterdiği yerden kopyaladıktan sonra Y registerini 1 artır.	$Rd = (Y), Y=Y+1$	Yok
Ld	Rd,-Y	Y registerinden 1 çıkarttıktan sonra Y registerinin gösterdiği yerden kopyala	$Y=Y-1, Rd = (Y)$	Yok

Ldd	Rd,Y+q	(Y+q) ifadesinin gösterdiği yerden kopyala	$Rd = (Y+q)$	Yok
Ld	Rd,Z	Z registerinin gösterdiği yerden kopyala	$Rd = (Z)$	Yok
Ld	Rd,Z+	Z registerinin gösterdiği yerden kopyaladıktan sonra X registerini 1 artır.	$Rd = (Z), Z=Z+1$	Yok
Ld	Rd,-Z	Z registerinden 1 çıkarttıktan sonra Z registerinin gösterdiği yerden kopyala	$Z=Z-1, Rd = (Z)$	Yok
Ldd	Rd,Z+q	Z+q ifadesinin gösterdiği yerden kopyala	$Rd = (Z+q)$	Yok
Sts	k,Rr	Hafıza adresine kopyala	$(k) = Rr$	Yok
St	X,Rr	X registerinin gösterdiği hafıza adresine kopyala	$(X) = Rr$	Yok
St	X+,Rr	X registerinin gösterdiği hafıza adresine kopyaladıktan sonra X registerinin değerini 1 artır	$(X) = Rr, X=X+1$	Yok
St	-X,Rr	X registerinin değerinden 1 çıkarttıktan sonra registerin gösterdiği yere kopyala	$X=X-1, (X)=Rr$	Yok
St	Y,Rr	Y registerinin gösterdiği hafıza adresine kopyala	$(Y) = Rr$	Yok
St	Y+,Rr	Y registerinin gösterdiği hafıza adresine kopyaladıktan sonra Y registerinin değerini 1 artır	$(Y) = Rr, Y=Y+1$	Yok
St	-Y,Rr	Y registerinin değerinden 1 çıkarttıktan sonra registerin gösterdiği yere kopyala	$Y=Y-1, (Y) = Rr$	Yok
St	Y+q,Rr	Y+q ifadesinin gösterdiği yere kopyala	$(Y+q) = Rr$	Yok
St	Z,Rr	Z registerinin gösterdiği hafıza adresine kopyala	$(Z) = Rr$	Yok
St	Z+,Rr	Z registerinin gösterdiği hafıza adresine kopyaladıktan sonra Z registerinin değerini 1 artır	$(Z) = Rr, Z=Z+1$	Yok
St	-Z,Rr	Z registerinin değerinden 1 çıkarttıktan sonra registerin gösterdiği yere kopyala	$Z=Z-1, (Z) = Rr$	Yok
St	Z+q,Rr	Z+q ifadesinin gösterdiği yere kopyala	$(Z+q) = Rr$	Yok
Lpm	Yok	Program hafızasında Z registeri ile gösterilen yerdeki 1 byte'lık veriyi r0 registerine getir.	$R0 = (Z)$	Yok
Lpm	Rd,Z	Program hafızasında Z registeri ile gösterilen yerdeki 1 byte'lık veriyi rd registerine getir	$Rd = (Z)$	Yok
Lpm	Rd,Z+	Program hafızasında Z registeri ile	$Rd = (Z), Z=Z+1$	Yok

		gösterilen yerdeki 1 byte'lık veriyi rd registerine getirdikten sonra Z registerinin değerini 1 artır		
Elpm	Yok	Program hafızasında RAMPZ:Z registeri ile gösterilen yerdeki 1 byte'lık veriyi r0 registerine getir. RAMPZ registeri 64Kb daha büyük program hafızası bulunan sistemlerde kullanılan registerdir.	R0 = (RAMPZ:Z)	Yok
Elpm	Rd,Z	Program hafızasında RAMPZ:Z registeri ile gösterilen yerdeki 1 byte'lık veriyi rd registerine getir	Rd = (RAMPZ:Z)	Yok
Elpm	Rd,Z+	Program hafızasında RAMPZ:Z registeri ile gösterilen yerdeki 1 byte'lık veriyi rd registerine getirdikten sonra Z registerinin değerini 1 artır	Rd = (RAMPZ:Z), Z = Z+1	Yok
Spm	Yok	Program hafızasında Z registeri ile gösterilen yere r1:r0 registerlerindeki 1 word'lük datayı yazar.	(Z) = R1:R0	Yok
Espm	Yok	Program hafızasında RAMPZ:Z registeri ile gösterilen yere r1:r0 registerlerindeki 1 word'lük datayı yazar.	(RAMPZ:Z) = R1:R0	Yok
In	Rd,P	P adresindeki portun değerini Rd registerine getirir.	Rd = P	Yok
Out	P,Rr	Rr registerindeki değeri P adresindeki porta yazar.	P = Rr	Yok
Push	Rr	Rr registerindeki değeri stack'a yazar	STACK = Rr	Yok
Pop	Rd	Rd registerine stack'taki değeri getirir.	Rd = STACK	Yok

3.5.2 Aritmetiksel ve Logic Komutlar

Komut	Operandları	Açıklama	Operasyon	Etkilenen Flaglar
Add	Rd,Rr	Rd ve Rr registerlerindeki değerleri toplar. Sonuç Rd registerinde olacaktır.	$Rd = Rd + Rr$	Z,C,N,V,H,S
Adc	Rd,Rr	Rd, Rr ve Carry flag değerlerini toplar. Sonuç Rd registerinde olacaktır.	$Rd = Rd + Rr + C$	Z,C,N,V,H,S
Adiw	Rd, K	Rd+1 ve Rd registerlerindeki 1 word'lük sayıyı K sayısı ile toplar. Sonuç Rd+1:Rd registerlerinde olacaktır.	$Rd+1 : Rd, K$	Z,C,N,V,S
Sub	Rd,Rr	Rd registerindeki değerden Rr registerindeki değeri çıkarır. Sonuç Rd registerinde olacaktır.	$Rd = Rd - Rr$	Z,C,N,V,H,S

Subi	Rd,K8	Rd registerindeki değerden K8 sabit sayısını çıkarır. Sonuç Rd registerinde olacaktır.	$Rd = Rd - K8$	Z,C,N,V,H,S
Sbc	Rd,Rr	Rd registerinden Rr registerinin değerini ve barrow (carry) flag değerini çıkarır. Sonuç Rd registerinde olacaktır.	$Rd = Rd - Rr - C$	Z,C,N,V,H,S
Sbci	Rd,K8	Rd registerinin değerinden K8 sabit sayısının değerini ve barrow (carry) flag değerini çıkarır. Sonuç Rd registerinde olacaktır.	$Rd = Rd - K8 - C$	Z,C,N,V,H,S
And	Rd,Rr	Rd ve Rr registerlerinin içerikleri logical And işlemine tabi tutulur. Örnek: Ldi r18,0x13 Ldi r19,0x03 And r18,r19 R18 registerinde 3 değeri olur.	$Rd = Rd \cdot Rr$	Z,N,V,S
Andi	Rd,K8	Rd registerinin içeriği ile K8 sabiti logical and işlemine tabi tutulur. Sonuç Rd registerindedir.	$Rd = Rd \cdot K8$	Z,N,V,S
Or	Rd,Rr	Rd ve Rr registerlerinin içerikleri logical Or işlemine tabi tutulur. Sonuç Rd registerinde olacaktır.	$Rd = Rd \vee Rr$	Z,N,V,S
Ori	Rd,K8	Rd registerinin içeriği ile K8 sabiti logical Or işlemine tabi tutulur. Sonuç Rd registerinde olacaktır.	$Rd = Rd \vee K8$	Z,N,V,S
Eor	Rd,Rr	Rd ve Rr registerlerinin içerikleri logical Xor işlemine tabi tutulur. Sonuç Rd registerinde olacaktır.	$Rd = Rd \oplus Rr$	Z,N,V,S
Com	Rd	255 sabit sayısından Rd registerinin içeriğini çıkarır. Sonuç Rd registerinde olur.	$Rd = \$FF - Rd$	Z,C,N,V,S
Neg	Rd	0 Sabit sayısından Rd registerinin içeriğini çıkarır. Sonuç Rd registerinde olur.	$Rd = \$00 - Rd$	Z,C,N,V,H,S
Sbr	Rd,K8	K8 sayısına göre Rd registerinin bitlerini set eder. Örnek : Sbr r16,4 R16 registerinin 2. bitini set yapar Sbr r16,0x05 R16 registerinin 2. ve 0. bitlerini set yapar.	$Rd = Rd \vee K8$	Z,C,N,V,S
Cbr	Rd,K8	K8 sayısına göre Rd registerinin bitlerini resetler. Örnek : Sbr r16,4 R16 registerinin 2. bitini 0 yapar Sbr r16,0x05 R16 registerinin 2. ve 0. bitlerini 0 yapar.	$Rd = Rd \cdot (\$FF - K8)$	Z,C,N,V,S
Inc	Rd	Rd registerinin içeriği 1 artırılır.	$Rd = Rd + 1$	Z,N,V,S

Dec	Rd	Rd registerinin içeriği 1 azaltılır.	$Rd = Rd - 1$	Z,N,V,S
Tst	Rd	Rd registerinin içeriğinin 0 olup olmadığını kontrol eder. İşlem flagları etkiler ancak register içeriği değişmez.	$Rd = Rd \cdot Rd$	Z,C,N,V,S
Clr	Rd	Registerin içeriği sıfırlanır.	$Rd = 0$	Z,C,N,V,S
Ser	Rd	Register içeriğine 255 atanır.	$Rd = \$FF$	None
Sbiw	RdI,K6	Word uzunluklu Rd registerinin içeriğinden K6 sabit sayısını çıkarır. Sonuç Rd+1:Rd registerlerinde olur.	$Rdh:Rdl = Rdh:Rdl - K6$	Z,C,N,V,S
Mul	Rd,Rr	İşaretsiz tanımlanmış Rd ve Rr registerlerinin içerikleri çarpılır. Sonuç R1:R0 registerlerinde olacaktır.	$R1:R0 = Rd * Rr$	Z,C
Muls	Rd,Rr	İşaretili tanımlanmış Rd ve Rr registerlerinin içerikleri çarpılır. Sonuç R1:R0 registerlerinde olacaktır.	$R1:R0 = Rd * Rr$	Z,C
Mulsu	Rd,Rr	Biri işaretili diğeri işaretsiz tanımlanmış Rd ve Rr registerlerinin içerikleri çarpılır. Sonuç R1:R0 registerlerinde olacaktır.	$R1:R0 = Rd * Rr$	Z,C
Fmul	Rd,Rr	İşaretsiz tanımlanmış Rd ve Rr registerlerinin kesirli içerikleri çarpılır. Sonuç R1:R0 registerlerinde olacaktır.	$R1:R0 = (Rd * Rr) \ll 1$	Z,C
Fmuls	Rd,Rr	İşaretili tanımlanmış Rd ve Rr registerlerinin kesirli içerikleri çarpılır. Sonuç R1:R0 registerlerinde olacaktır.	$R1:R0 = (Rd * Rr) \ll 1$	Z,C
Fmulsu	Rd,Rr	Biri işaretili diğeri işaretsiz tanımlanmış Rd ve Rr registerlerinin kesirli içerikleri çarpılır. Sonuç R1:R0 registerlerinde olacaktır.	$R1:R0 = (Rd * Rr) \ll 1$	Z,C

3.5.3 Dallanma Komutları

Dallanma komutlarının özetlemesine geçmeden önce dallanma komutlarında sıklıkla kullanılan dört kavramın açıklanması gerekmektedir.

- Long address
- Absolute address
- Relative address
- Indirect address

3.5.3.1 Long Address

Bu tür adres yalnız atama ve çağırma komutlarında kullanılır. Eğer komutun adres operandı olarak 16 bitlik bir sayı kullanılırsa bu kod uzun adres olarak adlandırılır. Birçok microdenetleyici için adres hattı 16 bittir. Bu nedenler 16 bitlik bir adres long address olarak adlandırılır.

Long Address kullanımı tüm adres uzayında kolay ve güvenli atlamalar, çağırımlar yapmaya imkan sağlar. Fakat uzun adresleme bu türden olan başka adresleme yöntemlerine göre bellekte daha fazla yer tuttuğundan, bellek kapasitesinden tasarruf söz konusu olduğunda mümkün oldukça kaçınılacak bir yöntemdir.

3.5.3.2 Absolute Address

Bu tür adres de yalnız atlama ve çağırma komutlarında kullanılır. Eğer absolute adres kullanan bir komutun adres operandı olarak 11 bitlik bir ikili kod veya büyük rakamı 7' yi aşmayan 16' lı kod kullanırsa bu kod absolute adres olarak adlandırılır. Görüldüğü gibi bu tür adreslemede adres hattının 5 büyük bitinin durumu hakkında somut bilgi verilmiştir. Absolute adreslemenin şartlarına göre söz konusu bitlerin bir önceki adresleme sırasında oluşmuş olan durumu değişmez kalmalıdır. Yani A15 - A11 bitlerinin eski durumu ile A10 - A0 bitlerinin yeni durumu birleşerek 16 bitlik tam bir adres oluşturur. Absolute adreslemede adresin 5 büyük bitinin kullanılması adreslemenin söz konusu bitlerin belli bir durumunda gerçekleştirilmesi demektir.

3.5.3.3 Relative Address

Relative adresleme şartlı veya şartsız kısa atlamalar için kullanılır. Bu tür adreslemede adres kodunun uzunluğu 8 bittir. Long ve absolute adreslerden farklı olarak bu kod varılacak olan bellek hücrelerinin adresini değil bu hücrenin program sayacının (Program Counter) göstermekte olduğu hücrelerden uzaklığı belirtir. Atlamalar ileriye ve geriye doğru olabildiğinden dolayı uzaklıkları da artı ve eksi işareti olabilirler. Başka bir deyimle relative adres işaretli bir sayı olmak zorundadır. Relative adresleme yöntemi ile 128 baytlık geriye 127 baytlık ileriye doğru atlama yapılabilir. Relative adreslemede atamaların doğru gerçekleştirilmesi için söz konusu adreslerin değerleri değil onlar arasındaki fark önemlidir. Yani programın bellek içerisinde herhangi bir şekilde kaydırılması relative adreslere göre kurulmuş olan atamaları etkilenmez. Atlama ilişkileri tamamen relative adreslere göre kurulmuş olan bir program ram yerleşim değişimlerinden etkilenmez.

3.5.3.4 Indirect Address

Indirect adresleme okuma ve yazma işlerinde kullanılır. Bu adreslemede komutun adres operand kısmında okunacak veya yazılacak hafıza hücrenin adresinin bulunduğu register gösterilir. Adreslemede kullanılan registerler olarak R0 veya R1 genel registerlerin biri veya pointer registerleri (X,Y,Z) kullanılır. R0 ve R1 registerleri 1 baytlık olduğu için belleğin yalnız ilk 256 baytını adresleyebilir ve bunlardan dolayı çok özel durumlarda faydalı olabilir. Ancak pointer registerleri 16 bitlik olduğu için mevcut adres uzayının başından sonuna kadar adresleyebilir.

Atlama komutları özet olarak aşağıdaki gibidir.

Komut	Operantları	Açıklama	Operasyon	Etkilenen Flaglar
Rjmp	k	Relative atlama yapar. k geriye veya ileriye doğru yapılacak atlamanın uzaklığını gösterir.	$PC = PC + k + 1$	Yok
ljmp	Yok	Indirect atlama yapar. Komut program akışını Z registerinin gösterdiği yere aktarır.	$PC = Z$	Yok
Eijmp	Yok	EIND registeri ve Z registerlerinin gösterdiği yere Indirect atlama yapar. EIND ve Z registerleri ile 4Mb bir alan adreslenebilir.	$STACK = PC+1,$ $PC(15:0) = Z,$ $PC(21:16) = EIND$	Yok
Jmp	k	k sabiti ile belirlenen adrese direct atlama yapar. k ile 4Mb bir alan adreslenebilir.	$PC = k$	Yok
Rcall	k	Relative olarak altprogram çağırmak için kullanılır. k ileriye veya geriye doğru 2kb bir atlama yapabilmek için işaretli bir değerdir.	$STACK = PC+1,$ $PC = PC + k + 1$	Yok
lcall	Yok	Z registerinde adresi belirtilen adresten başlayan altprogramı çağırmak için kullanılır.	$STACK = PC+1,$ $PC = Z$	Yok
Eicall	Yok	EIND ve Z registerlerinin gösterdiği adresten başlayan altprogramı çağırmak için kullanılır.	$STACK = PC+1,$ $PC(15:0) = Z,$ $PC(21:16) = EIND$	Yok
Call	k	k ile adreslenen yerden başlayan altprogramı çağırmak için kullanılır. k 4Mb bir alanı gösterebilir.	$STACK = PC+2,$ $PC = k$	Yok
Ret	Yok	Altprogramlardan dönüş için kullanılır.	$PC = STACK$	Yok
Reti	Yok	Interruptlardan çıkış için kullanılır.	$PC = STACK$	I
Cpse	Rd,Rr	Rd ve Rr registerlerinin içeriğini karşılaştırır. Eğer değerler eşit ise alttan ikinci komuta, değilse alttan birinci komuta atlamak için kullanılır. Örneğin : inc r4 Cpse r4,r0 mov r0,r4 Nop R4 registerinin içeriği r0 registerinin içeriğine eşit değilse r4 içeriğini r0 içine taşır.	if (Rd ==Rr) PC = PC 2 or 3	Yok
Cp	Rd,Rr	Rd ve Rr registerlerinin içeriklerini karşılaştırır. Bu karşılaştırma işleminde registerler etkilenmez. İşlem sonunda registerlere bakılarak sayılar hakkında fikir yürütülebilir.	Rd -Rr	Z,C,N,V,H,S

Cpc	Rd,Rr	Rd ve Rr registerlerinin içeriklerini carry flagını da dikkate alarak karşılaştırır. Bu karşılaştırma işleminde registerler etkilenmez. İşlem sonunda registerlere bakılarak sayılar hakkında fikir yürütülebilir.	$Rd - Rr - C$	Z,C,N,V,H,S
Cpi	Rd,K8	Rd registeri ve K8 sabitini karşılaştırır. Bu karşılaştırma işleminde registerler etkilenmez. İşlem sonunda registerlere bakılarak sayılar hakkında fikir yürütülebilir.	$Rd - K$	Z,C,N,V,H,S
Sbrc	Rr,b	Rr registerinin b sabiti ile belirlenen biti 0 ise alttan ikinci komuta değilse birinci komuta atlama yapmak için kullanılır.	$if(Rr(b)==0) PC = PC + 2 \text{ or } 3$	Yok
Sbrs	Rr,b	Rr registerinin b sabiti ile belirlenen biti 1 ise alttan ikinci komuta değilse birinci komuta atlama yapmak için kullanılır.	$if(Rr(b)==1) PC = PC + 2 \text{ or } 3$	Yok
Sbic	P,b	P ile belirlenen I/O registerinin b sabiti ile belirlenen biti 0 ise alttan ikinci komuta değilse birinci komuta atlama yapmak için kullanılır. Örnek: Wait: Sbic \$1c,1 ;EWE biti 0 ise Rjmp Wait ;Bekle Nop	$if(I/O(P,b)==0) PC = PC + 2 \text{ or } 3$	Yok
Sbis	P,b	P ile belirlenen I/O registerinin b sabiti ile belirlenen biti 1 ise alttan ikinci komuta değilse birinci komuta atlama yapmak için kullanılır.	$if(I/O(P,b)==1) PC = PC + 2 \text{ or } 3$	Yok
Brbc	s,k	Status registerin (SREG) s ile belirlenen biti 0 ise k ile belirlenen adrese atlamak için kullanılır	$if(SREG(s)==0) PC = PC + k + 1$	Yok
Brbs	s,k	Status registerin (SREG) s ile belirlenen biti 1 ise k ile belirlenen adrese atlamak için kullanılır	$if(SREG(s)==1) PC = PC + k + 1$	Yok
Breq	k	Eğer status registerdeki zero flagı set ise k ile belirlenen adrese atlamak için kullanılır.	$if(Z==1) PC = PC + k + 1$	Yok
Brne	k	Eğer status registerdeki zero flagı reset ise k ile belirlenen adrese atlamak için kullanılır.	$if(Z==0) PC = PC + k + 1$	Yok
Brcs	k	Eğer status registerdeki carry flagı set ise k ile belirlenen adrese atlamak için kullanılır.	$if(C==1) PC = PC + k + 1$	Yok
Brcc	k	Eğer status registerdeki carry flagı reset ise k ile belirlenen adrese atlamak için kullanılır.	$if(C==0) PC = PC + k + 1$	Yok
Brsh	k	Karşılaştırılan iki sayının eşit veya büyük olması durumunda k ile belirlenen yere atlama yapmak için kullanılır. Örnek	$if(C==0) PC = PC + k + 1$	Yok

		subi r19,4 Brsh besit Nop Besit: R19 4 e eşit veya büyükse Besit'e atlanır.		
Brlo	k	Karşılaştırılan iki sayıdan ilki ikincisine göre küçükse k ile belirlenen yere atlama yapmak için kullanılır. Örnek Eor r19,r19 Say: Inc r19 Cpi r19,16 Brlo Say Nop R19 registerinin 0 dan başlayarak 16 ya kadar saymasını sağlar	if(C==1) PC = PC + k + 1	Yok
Brmi	k	Negative flagı set ise k ile belirlenen yere atlama yapmak için kullanılır.	if(N==1) PC = PC + k + 1	Yok
Brpl	k	Negative flagı reset ise k ile belirlenen yere atlama yapmak için kullanılır.	if(N==0) PC = PC + k + 1	Yok
Brge	k	Karşılaştırılan iki sayıdan ilki ikincisine göre büyük veya eşitse k ile belirlenen yere atlama yapmak için kullanılır. (işaretli sayılarda)	if(S==0) PC = PC + k + 1	Yok
Brit	k	Karşılaştırılan iki sayıdan ilki ikincisine göre küçükse k ile belirlenen yere atlama yapmak için kullanılır.(işaretli sayılarda)	if(S==1) PC = PC + k + 1	Yok
Brsh	k	Half Carry flagı set ise k ile belirlenen yere atlama yapmak için kullanılır.	if(H==1) PC = PC + k + 1	Yok
Brhc	k	Half Carry flagı reset ise k ile belirlenen yere atlama yapmak için kullanılır.	if(H==0) PC = PC + k + 1	Yok
Brts	k	T flagı set ise k ile belirlenen yere atlama yapmak için kullanılır.	if(T==1) PC = PC + k + 1	Yok
Brtc	k	T flagı reset ise k ile belirlenen yere atlama yapmak için kullanılır.	if(T==0) PC = PC + k + 1	Yok
Brsv	k	overflow flagı set ise k ile belirlenen yere atlama yapmak için kullanılır.	if(V==1) PC = PC + k + 1	Yok
Brvc	k	overflow flagı reset ise k ile belirlenen yere atlama yapmak için kullanılır.	if(V==0) PC = PC + k + 1	Yok
Brie	k	Interrupt flagı set ise k ile belirlenen yere atlama yapmak için kullanılır.	if(I==1) PC = PC + k + 1	Yok
Brid	k	Interrupt flagı reset ise k ile belirlenen yere atlama yapmak için kullanılır.	if(I==0) PC = PC + k + 1	Yok

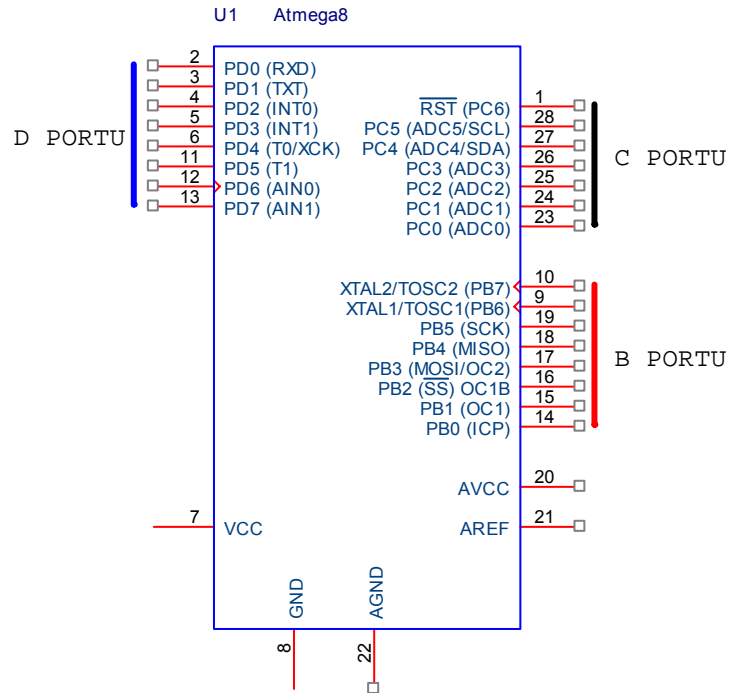
3.5.4 Bit ve Bit Test Komutları

Komut	Operandları	Açıklama	Operasyon	Etkilenen Flaglar
Lsl	Rd	Rd registerinin bitlerini sola doğru 1 kez kaydırır.	$Rd(n+1)=Rd(n)$, $Rd(0)=0$, $C=Rd(7)$	Z,C,N,V,H,S
Lsr	Rd	Rd registerinin bitlerini sağa doğru 1 kez kaydırır.	$Rd(n)=Rd(n+1)$, $Rd(7)=0$, $C=Rd(0)$	Z,C,N,V,S
Rol	Rd	Rd registerinin bitlerini 1 kez sola doğru kaydırır. En sağdaki yeni gelecek bit carry flagından alınır. En soldaki bit ise carry flag üzerine aktarılır.	$Rd(0)=C$, $Rd(n+1)=Rd(n)$, $C=Rd(7)$	Z,C,N,V,H,S
Ror	Rd	Rd registerinin bitlerini 1 kez sağa doğru kaydırır. En soldaki yeni gelecek bit carry flagından alınır. En sağdaki bit ise carry flag üzerine aktarılır.	$Rd(7)=C$, $Rd(n)=Rd(n+1)$, $C=Rd(0)$	Z,C,N,V,S
Asr	Rd	Rd registerinin bitlerini 1 kez sağa doğru kaydırır. En soldaki yeni gelecek bit registerin 7. biti tekrarlanarak sağlanırken en sağdaki bit carry flag üzerine kaydırılır.	$Rd(n)=Rd(n+1)$, $n=0,\dots,6$	Z,C,N,V,S
Swap	Rd	Rd registerinin nibble'larına yer değiştirir.	$Rd(3..0) =$ $Rd(7..4)$, $Rd(7..4)$ $= Rd(3..0)$	Yok
Bset	s	SREG üzerindeki s bitini set yapar.	$SREG(s) = 1$	SREG(s)
Bclr	s	SREG üzerindeki s bitini resetler.	$SREG(s) = 0$	SREG(s)
Sbi	P,b	P I/O registerinin b bitini set yapar.	$I/O(P,b) = 1$	Yok
Cbi	P,b	P I/O registerinin b bitini reset yapar.	$I/O(P,b) = 0$	Yok
Bst	Rr,b	Rr registerinin b. Bitini sreg registerinin T flagına kopyalar.	$T = Rr(b)$	T
Bld	Rd,b	T flagının içeriğini Rd registerinin b. Bitine kopyalar.	$Rd(b) = T$	Yok
Sec	Yok	Carry flagı set yapar.	$C = 1$	C
Clc	Yok	Carry flagı reset yapar	$C = 0$	C
Sen	Yok	Negative flagını set yapar	$N = 1$	N
Cln	Yok	Negative flagını reset yapar	$N = 0$	N
Sez	Yok	Zero flagını set yapar	$Z = 1$	Z
Clz	Yok	Zero flagını reset yapar	$Z = 0$	Z
Sei	Yok	Interrupt flagını set yapar.	$I = 1$	I

Cli	Yok	Interrupt flagını reset yapar.	I = 0	I
Ses	Yok	Signed flagını set yapar.	S = 1	S
Cln	Yok	Signed flagını reset yapar.	S = 0	S
Sev	Yok	Overflow flagını set yapar	V = 1	V
Clv	Yok	Overflow flagını reset yapar	V = 0	V
Set	Yok	T flagını set yapar	T = 1	T
Clt	Yok	T flagını reset yapar	T = 0	T
Seh	Yok	Half carry flagını set yapar	H = 1	H
Clh	Yok	Half carry flagını reset yapar	H = 0	H
Nop	Yok	Boş komut. Microdenetleyici bu komutta herhangi bir işlem yapmaz.	Yok	Yok
Sleep	Yok	Microdenetleyiciyi uyuma moduna geçirir.		Yok
Wdr	Yok	Watchdog counter'ı resetlemek için kullanılır		Yok

4. Atmel MCS-51 Microdenetleyici I/O Portları

Bir microdenetleyicinin dış dünya ile iletişim kurmada kullandığı en önemli sistemlerinden birisi digital I/O portlarıdır. Bu portlar genel olarak dış ortama bilgi aktarmak veya dış ortandan digital bilgi almak amacı ile kullanılır.



Atmel Atmega 8'in 23 adet I/O pini bulunmaktadır. Bu pinlerden 8 tanesi B Portu, 8 tanesi D portu, 7 tanesi ise C portu olarak adlandırılmaktadır. Portlara byte düzeyinde erişilebileceği gibi bit olarak erişim de mümkündür. Atmega 8'e ait hiçbir port çift yönlü (bi-directional) değildir. Bir diğer deyişle herhangi bir yönlendirmeye gerek kalmaksızın giriş ve çıkış yapılamaz. Bu nedenle herhangi bir port kullanılmadan önce kullanım amacına göre DDRx registeri ile yönlendirilmelidir. DDRx registerinin yapısı :

Bit	7	6	5	4	3	2	1	0	
	DDRB								DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	DDRC								DDRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	DDRD								DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRx registerinin her bir biti ilgili portun ilgili bitini yönlendirmek için kullanılır. Örneğin DDRB registerinin 0. biti B portunun 0. bitini yönlendirmek için kullanılır. DDRx registeri içine konulacak 0 bit değerleri ilgili bitin giriş, 1 değerleri ilgili bitin çıkış olarak yönlendirilmesini sağlar. Örneğin DDRB registerine atanacak 0xF0 değeri, B portunun ilk 4 bitini (0..3) giriş yapmak üzere, ikinci 4 bitini (4..7) çıkış yapmak üzere programlar.

Herhangi bir portu giriş veya çıkış olarak programlarken pull-up dirençlerinin durumunu ve portun başlangıç değerinin göz önüne alınması gerekir. Pull-up dirençleri SFIOR registeri, port başlangıç değerleri ise PORTx registeri ile kontrol edilir. PORTx registerinin yapısı aşağıdaki gibidir.

Bit	7	6	5	4	3	2	1	0	
	PORTB								PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

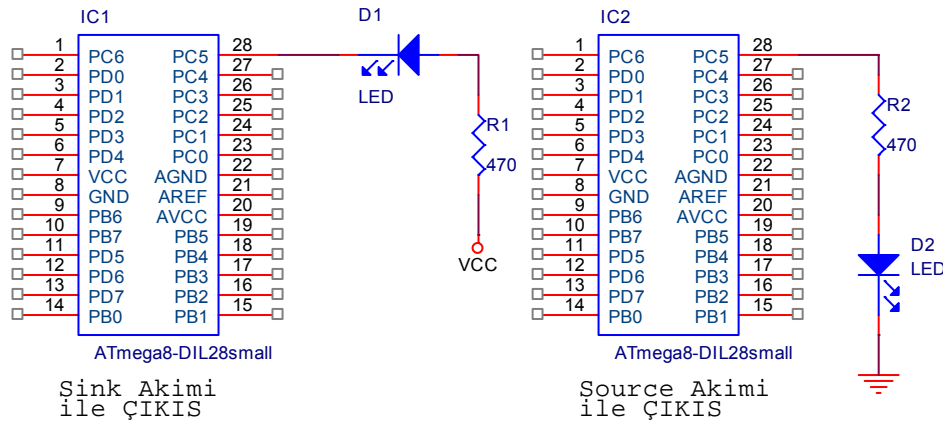
Bit	7	6	5	4	3	2	1	0	
	PORTC								PORTC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	PORTD								PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

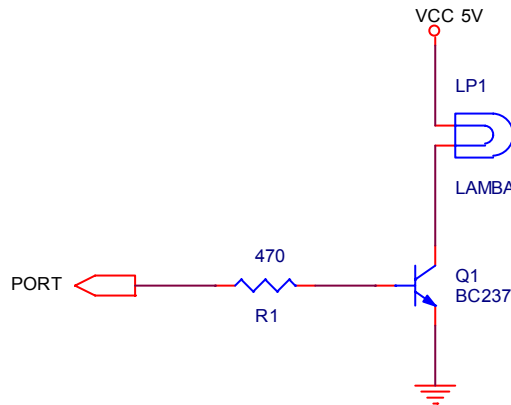
Herhangi bir portun herhangi bir pininin olası durumları aşağıda çıkartılmıştır.

DDRx	PORTx	SFIOR içindeki PUD	Giriş Çıkış	Pull-Up Direnci	Açıklama
0	0	X	Giriş	Yok	Yüksek empedans.
0	1	0	Giriş	Var	Giriş yapılabilir
0	1	1	Giriş	Yok	Yüksek empedans
1	0	X	Çıkış	Yok	Sink akımı ile çıkış
1	1	X	Çıkış	Yok	Source akımı ile çıkış

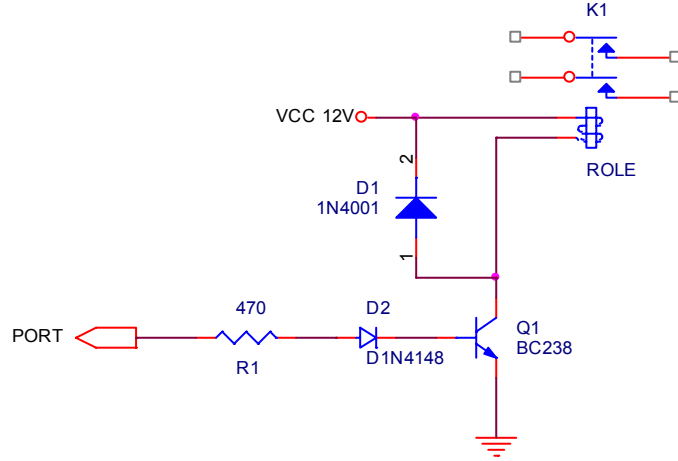
Yukarıdaki tablodan da anlaşılacağı gibi port pinlerinin yönlendirilebilmesi için DDRx registerinin yanı sıra PORTx registeri ve SFIOR registeri kullanılır. PORTx registeri portun çıkış değerini belirlerken SFIOR registeri pull-up dirençlerinin durumunu belirlemek için kullanılır.



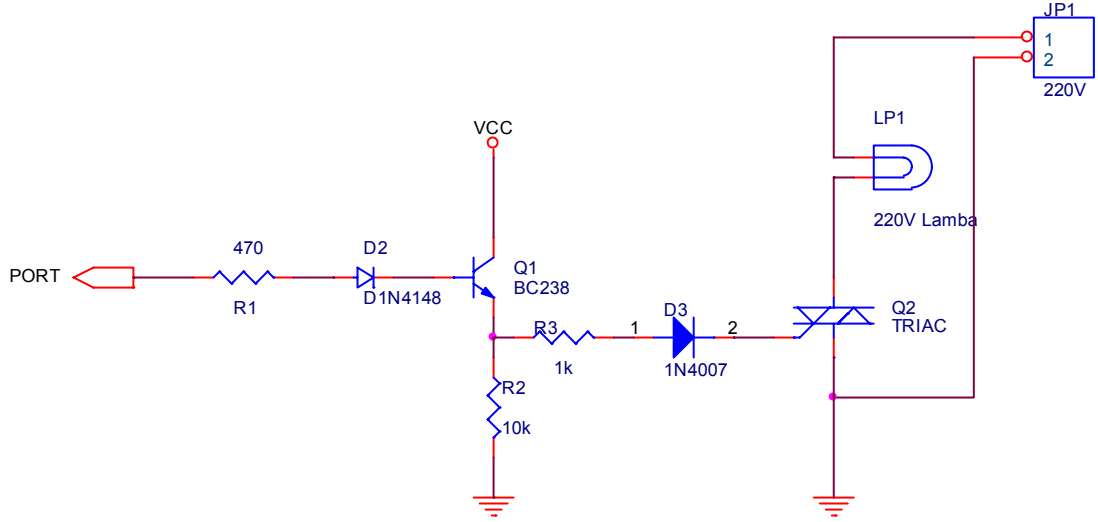
Çıkış olarak belirlenen bir port pininden 20 mA (miliamper) akım çekilebilir. 20 mA'den daha fazla akım gerektiren uygulamalarda pin'in zarar görmemesi için yükün transistör ile sürülmesi gereklidir. Port pinleri üzerinden değişik yapılarıdaki yükleri sürerken dikkat edilmesi gereken diğer bir husus da sistem beslemesinden daha yüksek voltajlarda çalışan yüklerden gelebilecek voltaj sızıntılarının önlenmesidir. Bu tür voltaj sızıntılarının önlenmesi tıkama diyotu yolu ile gerçekleştirilir.



5 V Yüksek Akımlı Yükün Sürülmesi



12 V Yüksek Voltaj, Yüksek Akımlı Yükün Sürülmesi



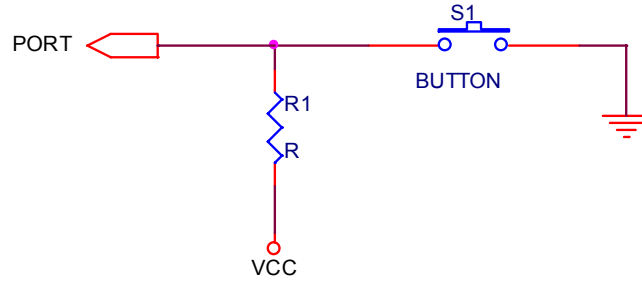
Triak Kullanarak AC Lamba Sürülmesi

Portun giriş olarak belirlenebilmesi durumunda dışarıdan bilgi okuma PINx registeri üzerinden gerçekleştirilir. PINx registerinin yapısı aşağıdaki gibidir.

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

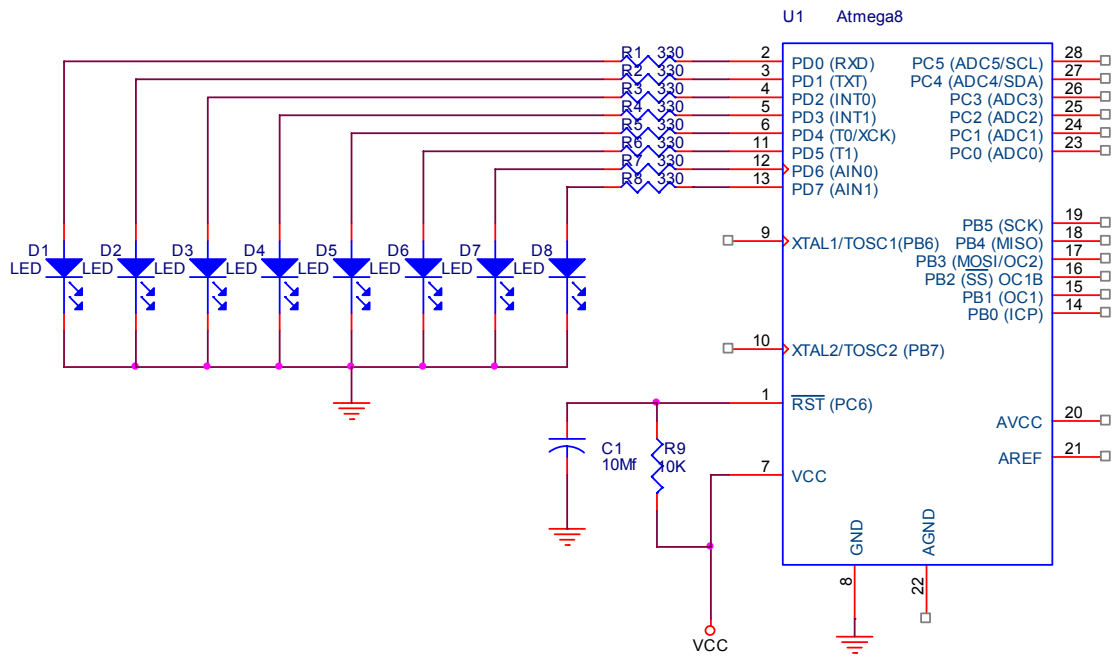
Bit	7	6	5	4	3	2	1	0	
	-	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Bit	7	6	5	4	3	2	1	0	
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	



Buton ile Giriş Portu

Digital Output Örnek :

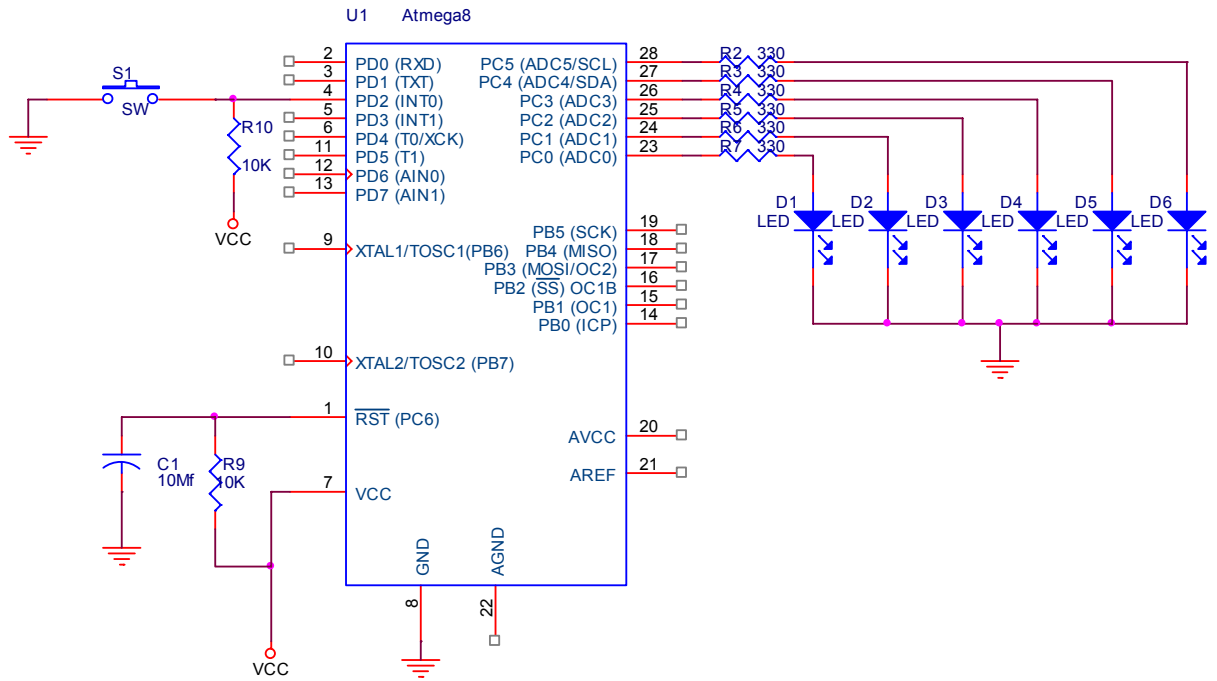


```
#include "m8def.inc"
.CSEG
.ORG 0x000
;Osilator ayarlanıyor
ldi r16,0xFF
out OSCCAL,r16
;stack pointer ayarlanıyor
ldi r16,high(RAMEND)
out sph,r16
ldi r16,low(RAMEND)
out spl,r16
;port C çıkış için ayarlanıyor
ldi r16,0xFF
out DDRD,r16
ldi r16,0x01
out PORTD,r16
```

DNN:

```
rol r16
out PORTD,r16
rjmp DNN
```

Digital Input-Output Örneği:



```
#include "m8def.inc"
```

```
.CSEG
```

```
.ORG 0x000
```

```
;Osilator ayarlanıyor
```

```
ldi r16,0xFF
```

```
out OSCCAL,r16
```

```
;stack pointer ayarlanıyor
```

```
ldi r16,high(RAMEND)
```

```
out sph,r16
```

```
ldi r16,low(RAMEND)
```

```
out spl,r16
```

```
;port C çıkış için ayarlanıyor
```

```
ldi r16,0xFF
```

```
out DDRC,r16
```

```
ldi r16,0x01
```

```
out PORTC,r16
```

```
;port D giriş için ayarlanıyor
```

```
ldi r16,0x00
```

```
out DDRD,r16
```

```
ldi r16,0xFF
```

```
out PORTD,r16
```

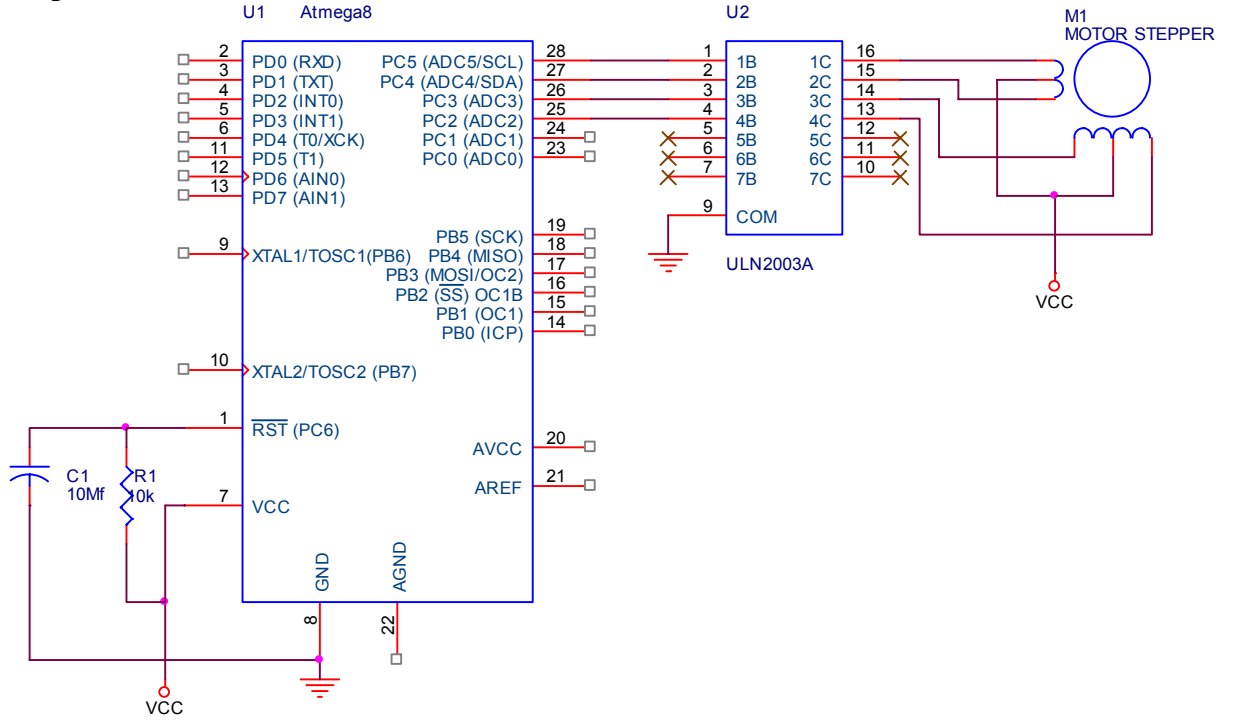
```

    clt                                ;T Flagı reset
    ldi r16,0x01
DNN:
    sbic PORTD,2                       ;2. bit reset ise (Tuşa basıldı)
    rjmp PRG0
    rcall TBIT

PRG0:
    brts SAGA                           ;T flag set ise
    rjmp SOLA
SOLA:
    rol r16
    out PORTC,r16
    rjmp DNN
SAGA:
    ror r16
    out PORTC,r16
    rjmp DNN
TBIT:
    brts RESETLE
    set                                  ;T flagını set et
    rjmp CIKIS
RESETLE:
    clt                                  ;T flagını resetle
CIKIS:
    sbis PORTD,2                       ;2. bit set ise (Tuş Bırakıldı)
    rjmp CIKIS
    ret

```

Step Motor Sürme



Step Motor Sürme biçimleri:

1. Tam Adım Sürme (Full Step)

T0	T1	T2	T3
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

2. Yarım Adım Hassas Sürme (Half Step)

T0	T1	T2	T3
1	0	0	0
1	1	0	0
0	1	0	0
0	1	1	0
0	0	1	0
0	0	1	1
0	0	0	1
1	0	0	1

3. Tam Adım Güçlü Sürme (Full Step Full Torque)

T0	T1	T2	T3
1	0	0	1
1	1	0	0
0	1	1	0
0	0	1	1

```
;  
; Program C Portuna bağlanmış  
; step motoru tam adım olarak sürer  
;
```

```
#include "m8def.inc"
```

```
.CSEG
```

```
;Osilator ayarlanıyor
```

```
ldi r16,0xFF
```

```
out OSCCAL,r16
```

```
;stack pointer ayarlanıyor
```

```
ldi r16,high(RAMEND)
```

```
out sph,r16
```

```
ldi r16,low(RAMEND)
```

```
out spl,r16
```

```
;port C çıkış için ayarlanıyor
```

```
ldi r16,0xFF
```

```
out DDRC,r16
```

```
;C portunun tüm bitleri set ediliyor
```

```
ldi r16,0xFF
```

```
out PORTC,r16
```

```
DNN:
```

```
;1. Adım
```

```
ldi r16,0x01
```

```
out PORTC,r16
```

```
rcall BEKLE
```

```
;2. Adım
```

```
rol r16
```

```
out PORTC,r16
```

```
rcall BEKLE
```

```
;3. Adım
```

```
rol r16
```

```
out PORTC,r16
```

```
rcall BEKLE
```

```
;4. Adım
```

```
rol r16
```

```
out PORTC,r16
```

```
rcall BEKLE
```

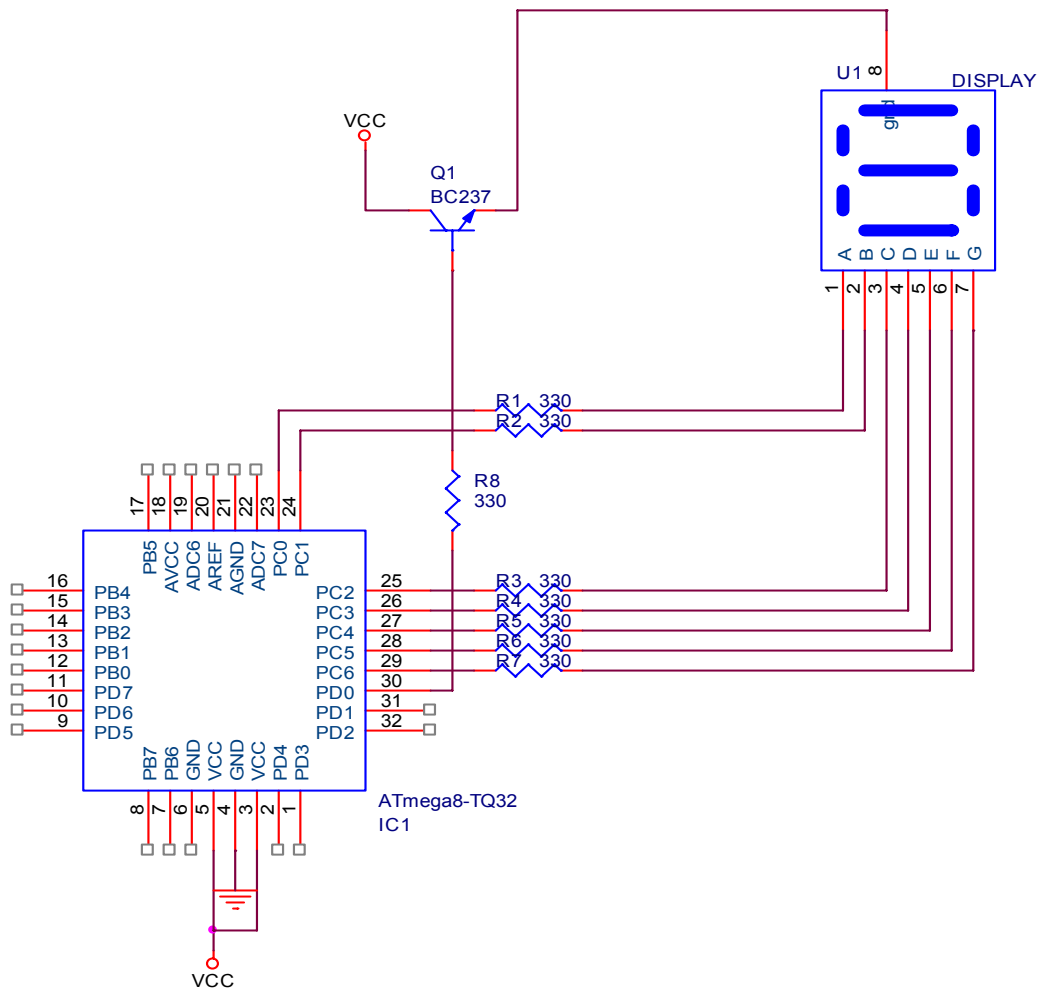
```
rjmp DNN
```

```

BEKLE:
    ldi r17,0xFF
BEK0:
    ldi r18,0xFF
BEK1:
    dec r18
    cpi r18,0x00
    brne BEK1
    dec r17
    cpi r17,0x00
    brne BEK0
    ret

```

7 Segment Display Sürme



```

#include "m8def.inc"

.CSEG
.ORG 0x000

;Osilator ayarlanıyor
    ldi r16,0xFF
    out OSCCAL,r16
;stack pointer ayarlanıyor
    ldi r16,high(RAMEND)
    out sph,r16
    ldi r16,low(RAMEND)
    out spl,r16
;port C çıkış için ayarlanıyor
    ldi r16,0xFF
    out DDRC,r16
ldi r16,0x00
    out PORTC,r16
;port D Çıkış için ayarlanıyor
    ldi r16,0xFF
    out DDRD,r16
    ldi r16,0x00
    out PORTD,r16
;
;
;
PRG:
;0 Yazdırılıyor
    ldi r16,0b00111111
    out PORTC,r16
    sbi PORTD,0
    call BEKLE
;1 Yazdırılıyor
    ldi r16,0b00000110
    out PORTC,r16
    sbi PORTD,0
    call BEKLE
    rjmp PRG

BEKLE:
    ldi r17,0xFF
BEK1:
    dec r17
    cpi r17,0x00
    breq CIK
    rjmp BEK1
CIK:
    ret

```

5. Atmel MCS-51 Microdenetleyici Interrupt Sistemi

Interrupt (Kesme) denildiğinde aktif olan bir A programını durdurarak çalıştırılması acil olan B programının aktifleşmesi ve B programı bittiğinde A programının kaldığı yerden devam ettirilmesini sağlayan bir süreç anlaşılır.

Genellikle interrupt programı, B programına girmeden önce A programına düzgün olarak dönebilmek için PC (program counter – program sayacı)’ın içeriği stack’a depolanır, interrupt programı bittiğinde ise stack’tan alınarak PC’ye aktarılır. Yani herhangi bir interrupt alt programı, durdurmuş olduğu ana programa düzgün dönüşü sağlamak mecburiyetindedir. Bu görevi her bir interrupt alt programının son komutu olan “reti” gerçekleştirir. Bazı sistemlerde PC’nin içeriği ile birlikte SREG (Status Register)’in de içeriği interrupt alt programına girmeden önce stack’a depolanır ve sonradan ilk yerine aktarılır. MCS-51 ailesi için bu geçerli değildir. PC’nin içeriğinin dışında kalan ne varsa hepsinin kaderi kullanıcıya bırakılır ve bu sebepten ana programın düzgün devam ettirebilmesi için nelerin içeriğinin korunacağına kullanıcı karar verir. Bunun için alt program başında push komutları, sonunda ise pop komutları kullanılır. Interrupt alt programından ana programa dönüş, interrupt alt programının en sonuncu komutu olan reti komutu tarafından gerçekleştirilir.

Interrupt alt programları Interrupt kaynaklarından gelen sinyallere göre aktifleştirilir. Genellikle her bir interrupt sinyali bir interrupt alt programının aktifleşmesine sebep olur. Interrupt kaynakları, microdenetleyiciye göre farklılık gösterebilir.

Atmel Atmega8 için 19 adet interrupt kaynağı bulunmaktadır. Bunlardan INTO (External Interrupt Request – Dış interrupt isteği) ve INT1 olarak adlandırılan ikisi dış kaynaktır. Geride kalan onyediyi kaynak mikrodenetleyicinin içinde bulunan devrelerden üretildiğinden iç kaynaklar olarak adlandırılmaktadır. Gerek iç kaynaklardan gerekse dış kaynaklardan tetiklenmiş olsun herhangi bir interrupt tetiklendiğinde, microdenetleyici işletmekte olduğu programı bırakarak interrupt’ın tetiklenme kaynağına göre belirlenmiş adrese program akışını aktarır. Interrupt kaynaklarına göre belirlenmiş bu adreslere Interrupt Vektör Table (Interrupt Vektör Tablosu) adı verilir. Atmel Atmega8 için Interrupt vektör tablosu aşağıda verilmiştir.

Vektör NO	Program Adresi	Kaynak	Açıklama
1	0x000	Reset	External reset, Power on reset, Brown-out reset, Watchdog reset
2	0x001	Int0	External Interrupt Request 0
3	0x002	Int1	External Interrupt Request 1
4	0x003	Timer2 Comp	Timer/counter2 compare match
5	0x004	Timer2 Ovf	Timer/counter2 overflow
6	0x005	Timer1 Capt	Timer/counter1 capture event
7	0x006	Timer1 CompA	Timer/counter1 compare match A
8	0x007	Timer1 CompB	Timer/counter1 compare match B
9	0x008	Timer1 Ovf	Timer/counter1 overflow

10	0x009	Timer0 Ovf	Timer/counter0 overflow
11	0x00A	Spi Stc	Serial Transfer complete
12	0x00B	Usart Rxc	Usart, Rx complete
13	0x00C	Usart Udre	Usart, Data Register Empty
14	0x00D	Usart Txc	Usart, Tx complete
15	0x00E	Adc	ADC Conversion Complete
16	0x00F	EE_Rdy	EEPROM Ready
17	0x010	Ana_Comp	Analog Comparator
18	0x011	Twi	Two wire serial Interface
19	0x012	Spm_Rdy	Store program memory ready

Interrupt vektör tablosu incelendiğinde her bir interrupt için ayrılan alanın 1 byte olduğu görülecektir. Bu nedenle vektör adresleri, interrupt alt programına atlama yapabilmek için geçici bir alan olarak kullanılmalıdır. Örnek:

```
.CSEG
.ORG 0x00
    Rjmp AnaPrg
    Rjmp Int0Prg
    Rjmp Int1Prg
.ORG 0x09
    Rjmp Tim0Ovf
.ORG 0x15
AnaPrg:
    Rjmp AnaPrg

Int0Prg:
    iret
Int1Prg:
    iret
Tim0Ovf:
    iret
```

Yukarıdaki programda, reset, int0, int1 ve timer0 overflow interruptları için hafıza düzenlemesi yapılmıştır. Reset, int0 ve int1 vektörleri peşpeşe ve hafızanın 0. bytından itibaren yerleştirilmesi gerektiğinden ORG 0x00 komutu ile buraya yerleştirilmiş, timer0 overflow interrupt vektörü ise 9h adresinde olduğundan ORG 0x09 komutu ile 9h adresine yerleştirilmiştir. Ana program ORG 0x15 komutu ile 15h adresinden başlatılmıştır.

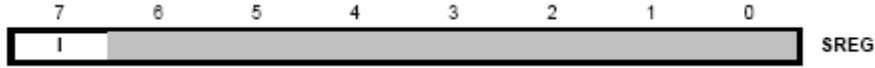
Kullanılmayan interrupt vektörleri yukarıdaki örnekte olduğu gibi göz ardı edilebilir.

Birçok microdenetleyici bootloader adı verilen microdenetleyici çalıştıktan sonra kontrollü olarak flash ram'e program yüklemeye ve çalıştırmaya izin veren program yapılarına uygun üretilmişlerdir. Bootloader programlar erişim kontrollü flash bellek alanlarına yazılmaları gerekir. Bu gereklik reset interrupt vektörünün veya diğer interrupt vektörlerinin adreslerinin değişmesi anlamına gelir. Bootloader programları ve bu programların vektör adreslerini nasıl etkiledikleri ile detaylı bilgi ilerleyen bölümlerde anlatılacaktır.

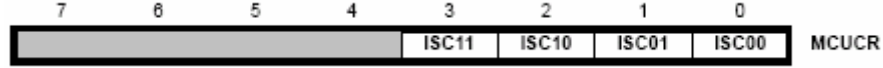
5.1 External Interrupt'lar (Dış Interrupt'lar)

MCS-51 core'a sahip microdenetleyicilerin farklı sayıda external interrupt kaynakları bulunabilmektedir. Atmel Atmega8 microdenetleyicinin INT0 ve INT1 olarak adlandırılan iki adet dış interrupt kaynağı bulunmaktadır. Bu interrupt kaynakları özel amaçlı registerler yardımı ile açılıp kapatılabildiği gibi düşen kenarda tetiklenmek üzere, çıkan kenarda tetiklenmek üzere veya düşük seviyede tetiklenmek üzere programlanabilir.

External interrupt kaynaklarının çalışabilmesi için aşağıda gösterilen registerlerin ilgili bitlerinin programlanması gerekir.



Interruptların aktif olabilmesi için SREG registerindeki I bitinin aktif olması gerekir.



MCUCR registerindeki ilgili bitler ile interrupt'ın tetikleme biçiminin belirlenmesi gerekir. Register ile ilgili tablo aşağıdadır.

Interrupt 0 için tetikleme tipleri aşağıdaki gibidir.

ISC01	ISC00	Açıklama
0	0	Interrupt 0 seviyesinde tetiklenir
0	1	Interrupt logic değişimlerde tetiklenir
1	0	Interrupt düşen kenarda tetiklenir
1	1	Interrupt çıkan kenarda tetiklenir

Interrupt 1 için tetikleme tipleri aşağıdaki gibidir.

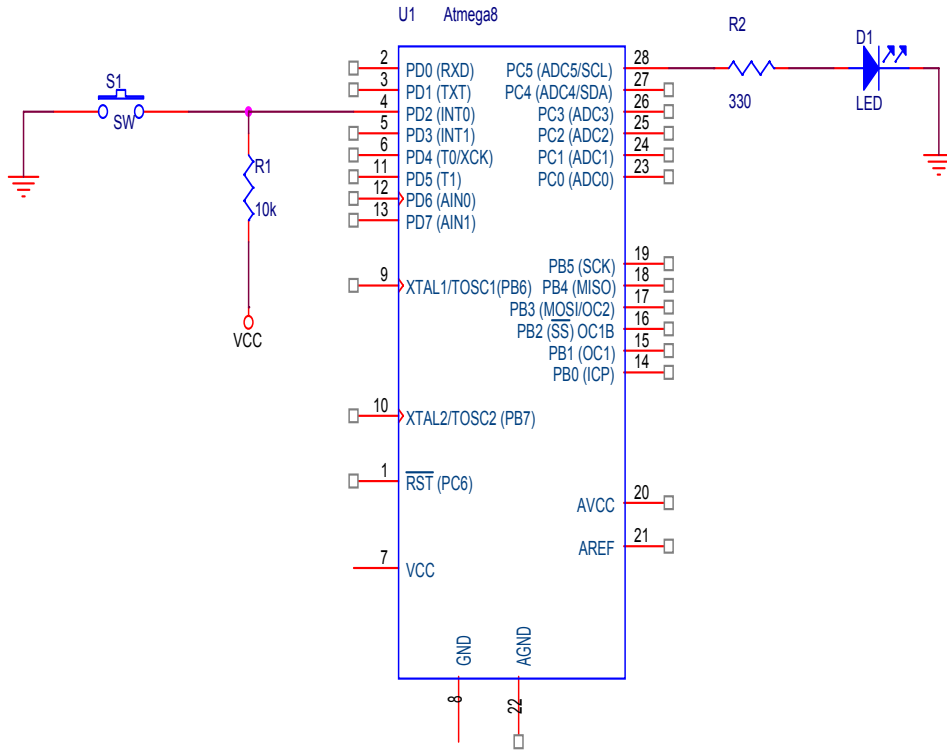
ISC11	ISC10	Açıklama
0	0	Interrupt 0 seviyesinde tetiklenir
0	1	Interrupt logic değişimlerde tetiklenir
1	0	Interrupt düşen kenarda tetiklenir
1	1	Interrupt çıkan kenarda tetiklenir



GICR registeri ile çalışması istenen interrupt'a izin verilmelidir. Register üzerinde ilgili biti set yapılan interrupt eğer SREG içindeki I biti set durumda ise MCUCR registerinde belirlenen şekilde tetiklenmek üzere aktif edilecektir.



GIFR registeri, external interrupt kaynaklarının durumunu göstermek üzere düzenlenmiş bir registerdir. INT0 veya INT1 kaynaklarından bir interrupt isteği geldiğinde, interrupt isteğini göstermek üzere ilgili bit set yapılır ve diğer registerlerin durumları uygun ise ilgili alt programa atlanır. Interrupt alt programı çalışırken ilgili bit resetlenir. Örnek:



```
#include <m8def.inc>

.CSEG
.ORG 0x00
    rjmp BASLA
    rjmp INT0VEC
BASLA:
    ;Osilator ayarlanıyor
    ldi r16,0xFF
    out OSCCAL,r16
    ;stack pointer ayarlanıyor
    ldi r16,high(RAMEND)
    out sph,r16
    ldi r16,low(RAMEND)
    out spl,r16
    ;port C çıkış için ayarlanıyor
    ldi r16,0xFF
    out DDRC,r16
    ldi r16,0x00
    out PORTC, r16
```

```

;interrupt 0 ayarlanıyor
ldi r16,0b00000010
out MCUCR,r16
ldi r16,0b01000000
out GICR,r16
ldi r16,0b01000000
out GIFR,r16
sei
;int0 düşen kenarda tetiklenecek

BEKLE:
rjmp BEKLE
;int0 açılıyor

INT0VEC:
sbic PORTC,5
rjmp RESETLE
rjmp SETET
rjmp CIK
;int0 istekleri açılıyor
;Global Interruptlar açılıyor

RESETLE:
cbi PORTC,5
rjmp CIK
;Port C nin 5.biti 0 ise

SETET:
sbi PORTC,5

CIK:
Reti

```

5.2 *Internal Interrupt'lar (İç Interrupt'lar)*

İç Interruptlar, microdenetleyici içindeki özel amaçla düzenlenmiş devreler tarafından tetiklenen interruptlardır. Timer/Counter taşma interruptları, Spi haberleşme arayüzü transfer tamamlandı interruptı, Usart gönderme ve alma tamamlandı interruptı, ADC çevrim tamamlandı interruptı, EEPROM hazır Interruptı vb. interruptlar iç interruptlara örnek olarak gösterilebilir. Bu interruptların hazırlanma, tetiklenme ve diğer özellikleri Özel Amaçlı Devreler bölümünde detaylı olarak anlatılacaktır.

6. **Atmel MCS-51 Microdenetleyici Özel Amaçlı Devreleri**

6.1 *Sayıcı Devreler (Timer/Counter)*

Microdenetleyiciler, sayma işlemlerini gerçekleştirebilmek için içlerinde özel devreler bulundurlar. Bu devreler sayesinde işlemci başka bir iş gerçekleştirirken aynı zamanda sayma işlemlerini de yapabilir. Bu devrelerin sayma giriş uçları içeriden veya dışarıdan tetiklenmek üzere programlanabilir. Devrenin sayma giriş ucu dışarıdan tetiklenmek üzere ayarlanmış ise devre counter, içeriden sabit bir sinyal ile tetiklenmek üzere ayarlanmış ise timer adını alır.

Sayıcı devreler sayma işlemlerini gerçekleştirebilmek için SFR'ler içinde yer alan registerleri kullanırlar. Bu registerler 8 veya 16 bit olabilir. Sayma işlemi sırasında bu register dolarsa isteğe bağlı olarak bir interrupt oluşturularak register sıfırlanır ve sayma işlemi yeniden başlatılır. Bu nedenle Timer/counter belirtilirken bu registerin uzunluğu da belirtilir. 8 bit Timer/Counter, 16 bit Timer/Counter vb.

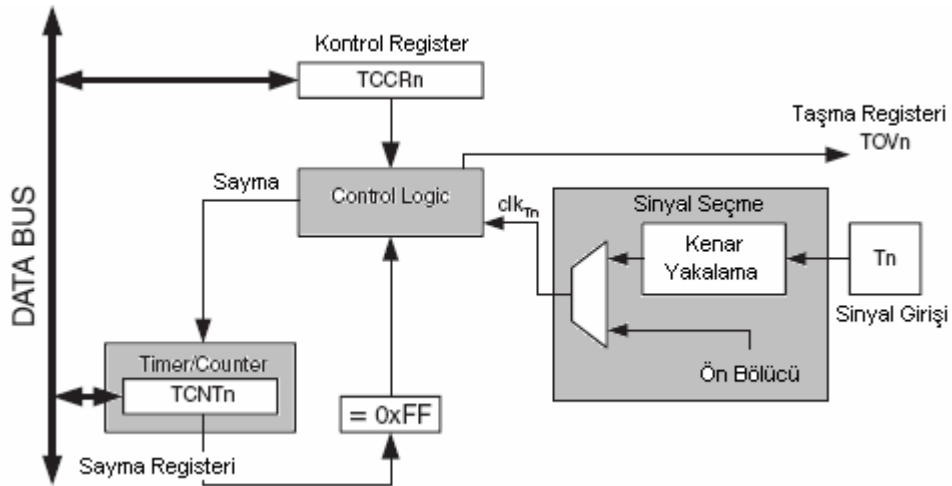
Atmel Atmega8 içinde 2 adet 8 bit timer/counter, 1 adet 16 bit timer/counter bulunur. Microdenetleyici içinde yer alan timer/counter'ların farklı fonksiyonlarının olması nedeniyle her bir timer/counter ayrı ayrı anlatılacaktır.

6.1.1 Timer/Counter0

Timer/Counter0 genel olarak tek kanallı bir sayma devresidir. Bu devre ile;

- Dış sinyalleri sayma,
- İç sinyalleri sayma,
- Frekans üretme işlemleri gerçekleştirilebilir.

Devrenin blok diyagramı aşağıdaki gibidir.

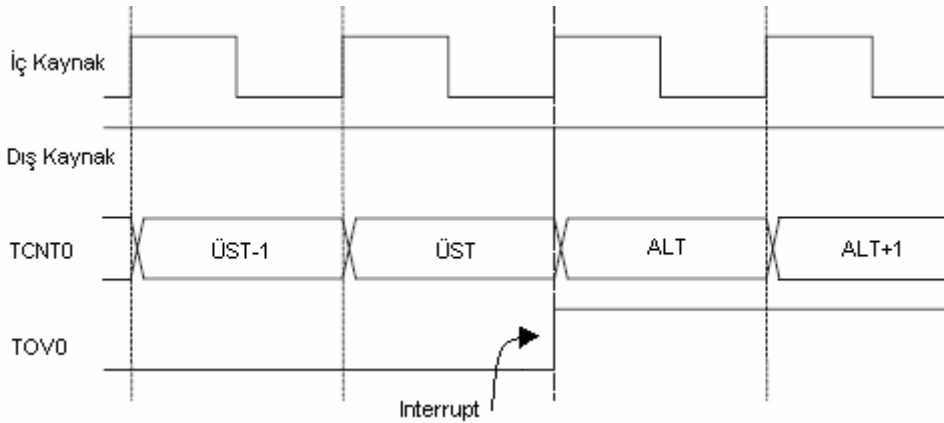


Sayıcı devrenin kontrolü TCCRn registeri tarafından gerçekleştirilir. (timer/counter0 için TCCR0) Bu register ile sayma kaynağının yönlendirilmesi ve dış sinyal girişinin yükselen veya alçalan kenarda sayılacağına karar vermek mümkündür. Registerin genel yapısı:

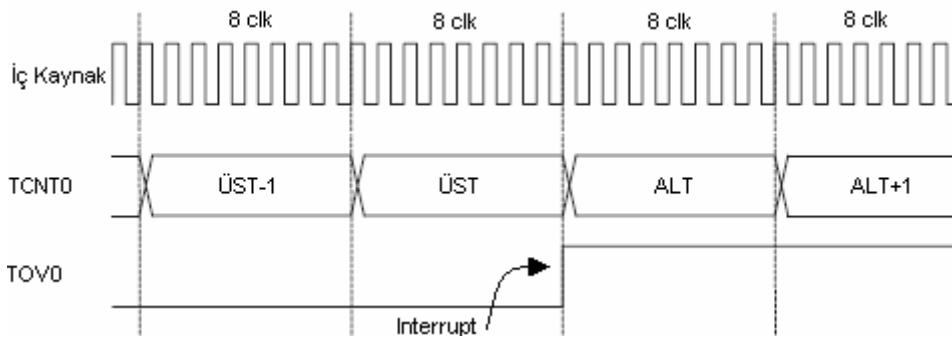
Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	-	CS02	CS01	CS00	TCCR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

CS02	CS01	CS00	Açıklama
0	0	0	Kaynak Yok (Timer/Counter'ı durdur)
0	0	1	İç kaynaktan tetikle (Bölme Yok)
0	1	0	İç kaynaktan tetikle (8'e böl)
0	1	1	İç kaynaktan tetikle (64'e böl)
1	0	0	İç kaynaktan tetikle (256'ya böl)
1	0	1	İç kaynaktan tetikle (1024'e böl)
1	1	0	Dış kaynaktan düşen kenarda tetikle
1	1	1	Dış kaynaktan yükselen kenarda tetikle

Sayıcı devresi içindeki sinyal seçme ünitesi, TCCR0 registerinin değerine göre çalışır ve dışarıdan gelen veya içeriden gelen sinyalleri düzenleyerek kontrol ünitesine aktarır. Kontrol ünitesi TCNT0 (sayma registeri), TIMSK (Timer/Counter Interrupt Mask registeri) ve TIFR (Timer/Counter Interrupt Flag registeri) registerlerini dikkate alarak TCNT0 registerinin değerini 1 artırır. Eğer TCNT0 registerinin değeri 0xFF (maksimum) değerine ulaşmışsa registerin değerini 0'a eşitler ve TIMSK registerinde ilgili bit set edilmişse TIFR registerindeki ilgili bit değerini set ederek Timer Overflow interruptunun aktif hale getirilmesini sağlar. Aşağıdaki diyagramlar sayma işleminin nasıl gerçekleştiğini göstermektedir.



İç Kaynaktan Bölme Olmaksızın Sayma



İç Kaynaktan 8 Bölme Oranı İle Sayma

Sayma işlemi TCNT0 registerinin üzerinde gerçekleştirilir. Register değeri her bir clock için 1 artırılır. TCNT0 registerinin yapısı aşağıdaki gibidir.

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCNT0 değeri, maximum seviyeye ulaştığında TIMSK registerinin ilgili bitine bakılarak Overflow interruptı oluşturulabilir. TIMSK registerinin yapısı aşağıdaki gibidir.

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Register üzerindeki TOIE0 biti set edilirse TCNT0 registeri 0xFF değerine ulaştığında SREG üzerindeki I biti set edilmişse Overflow interrupt'ı oluşturulur.

6.1.2 Timer/Counter1

6.1.3 Timer/Counter2