AHMET YESEVİ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ LİSANS

TBML - 302

Mikroişlemciler - Ödev

HAZIRLAYAN

HASAN AYGIR - 142132079

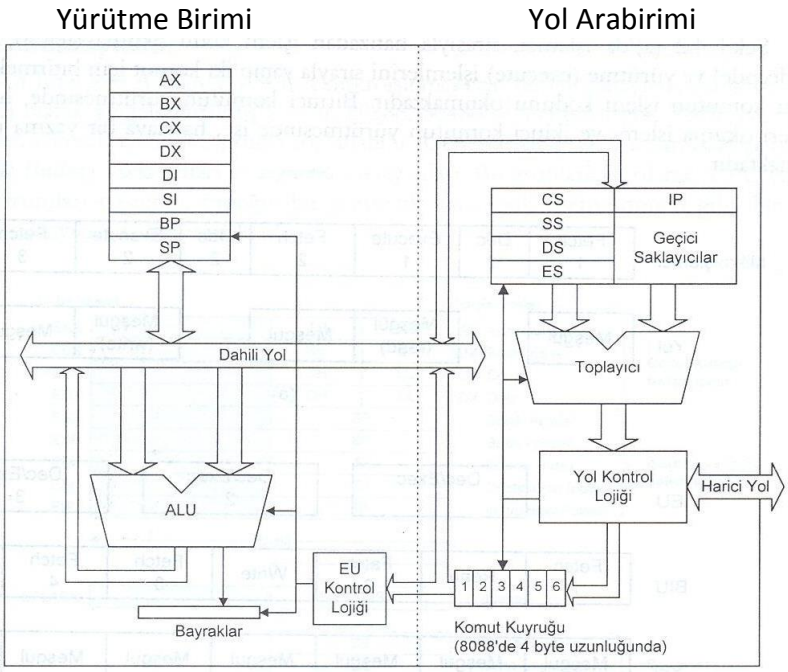
Öğretim Görevlisi

Doç. Dr. Oğuz FINDIK

8086 MİKROİŞLEMCİ

Bir mikroişlemci tüm işlemlerden sorumlu merkezi işlem birimidir (CPU). Bir mikroişlemcinin en temel görevleri; bellekten komut okuma, komutun kodunun çözülmesi ve komutun işlevini yerine getirmesidir. Bunların yanı sıra bulunduğu sistemdeki cihazların kontrol etmek, sisteme aritmetik ve lojik işlem yapma olanağı sağlamak, giriş/çıkış birimleri arasında veri transferinin sürekli sağlamak ve çevre birimlerinin hizmet istek sinyallerine cevap vermek de başlıca görevlerindendir.

Aşağıda x86 ailesinin 16-bit çekirdek mimarisinin basit bir gösterimi verilmiştir. Buna göre mikroişlemci iki temel ayrı çalışma birimine sahiptir; *Yürütme birimi* ve *Yol arabirimi*. Yürütme birimi komutları yorumlamakta ve yürütmektedir. Yol arabirimi ise yol işlemlerini (işlem kodu okuma, operand okuma ve griş/çıkış cihazlarıyla haberleşme gibi) yerine getirmektedir.



Yürütme Birimi: Komut çözme ve komut yürütme için bir kontrol birimine, ALU' ye, genel amaçlı saklayıcılara, işaretçi ve indis saklayıcılar ile bayrak saklayıcısına sahiptir. Kontrol birimi, Yol arabirimi tarafından komut kuyruğuna sırasıyla yerleştirilen makine dilindeki komutların yorumlanmasını, kodunun çözülmesi ve yürütülmesi için gereken ve işlemleri kontrol eder. Yürütme birimi hafızadan bir operanda ihtiyaç duyarsa veya yazmak isterse bu işi Yol Arabirimi' ya yönlendirir. Bu sırada Yol arabirimi için gereken fiziksel hafıza adresi hesaplama işlemlerini de sağlar.

Yol Arabirimi: Bütün dış yol işlemlerini kontrol eden bir kontrol birimine, yürütme biriminin yürüteceği komut byte'larını tutan komut kuyruğuna, fiziksel hafıza adresleri üretmek için bir toplayıcıya, 4 segment saklayıcısına (CS, DS, SS, ES), komut işaretçisine sahiptir. Anlaşılacağı gibi yol arabirimi hafıza ve Giriş/çıkış işlemleri dâhil bütün dış yol işlemlerinden sorumludur. Yol arabirimi komut kuyruğuna yerleştirmek üzere örneğin 8086 mikroişlemci için 6 byte komut kodunu önden okuyabilir. Komut kodunun bu şekilde önden okunması yol arabirimi ve yürütme biriminin paralel çalışmasına imkân verir. Önceden okunan komutlar icra edilirken yenilerinin bu esnada okunabilmesi işlemcinin performansını arttırır. Bu tarz çalışan mikroişlemci mimarisine *işhatlı mimari (pipelined architecture)* adı verilir.

8086 BELLEK ORGANİZASYONU

8086 mikroişlemci 20 bit adres yolu ile 1048576 (1M) byte bellek hücresi adresleyebilmektedir. Lojik bellek genelde yazılım tarafından programcıya görülen hafızaya denir. 8086 işlemci 16-bit mimarisine sahip olmasına rağmen lojik bellek genişliği yine 8-bit' tir. Ancak saklayıcı genişliği (16-bit mimari) ve fiziksel bellek tasarımı sayesinde 16-bit yazma ve okuma yapılabilir. 8086 işlemci 00000H-FFFFFH arasında 1M byte adres uzunluğunda 8-bit

yani 1 byte lojik belleğe sahiptir. Mikroişlemci ile adreslenebilen 16-bit (2 byte) bellek kelimesi (word) herhangi bir byte adresinden başlar ve peş peşe iki byte işgal eder. Yani programcı için bellek her zaman 8-bit' tir fark sadece donanım tasarımındadır.

Fiziksel bellek ise daha çok donanım tasarımcısı tarafından görülen gerçek hafıza yapısını tanımlar ve lojik hafızadan farklı olabilir. Yanda görülen fiziksel bellek yapısının avantajı byte veya word verisinin doğrudan adreslenebilmesinde yatar. 8086 işlemci bu sayede ve veri adresinin çift olması sağlandığında 16-bit bir veriyi bir işlemde okuyup yazabilmektedir. 8-bit işlemci olan 8088 16-bit veri aktarımı için ancak 2 okuma veya yazmaya ihtiyaç duyar. Yani 8086 16-bit veriye iki kat daha hızlı erişir.

16-bit veri yolu (data bus) genişliğine sahip olan 8086 işlemci 20 bit adres yolu (adres bus) genişliğine sahiptir. Ancak bellek adresleme işlemleri için gereken bir 20 bit saklayıcısına sahip değildir. Bu durumda çözüm 16 bit saklayıcıların adres verilerinin aşağıdaki şekilde gösterildiği gibi kullanılmasıyla elde edilir. İşte bu yapı segment ve ofset kavramlarını doğurur. 8086 işlemcisinde bellek erişimi segment saklayıcılar ile sağlanır. Her segment saklayıcı 20-bit adresin 16-bit kısmını tutar. Bu 16-bit adresin düşük değerli bölümüne 0H (0000_2) eklenir. Sonrada 16-bit bir ofset(indis) bu adresle toplanarak 20-bit gerçek adres elde edilir. Bu işleme lojik adresin fiziksel adrese çevrilmesi denir.

Segment: Belleğin daha kolay adreslenebilmesi amacıyla bölündüğü düşünülen bellek kesimleridir. Diğer bir deyişle her bir bellek hücresini küçük bir kutucuk olarak düşünürsek segment kavramını da bu küçük kutucuklardan oluşan büyük kutucuklar olarak düşünebiliriz. Yani Segment adresin setlenmesi ile bu büyük kutucuklardan birisinin başlangıç adresi setlenmiş olur.

Ofset: Segment içerisinde belleğin her bir hücresi olarak düşünebilir. Segment adresin setlenmesinden sonra bu segment içindeki veya bu adresten sonra ulaşılmak istenen bellek hücrelerinin her biridir. Yani yukarıda bahsi geçen büyük kutucukların içindeki her bir küçük kutucuklardır.

Segment' li bellek yapısı sayesinde kod, veri ve gerekli diğer bellek alanları farklı hafıza bölgelerinde çalışabilmektedir. Örneğin DS saklayıcısına farklı bir adresi işaret eden adres yazılarak program ve data bellek alanları birbirinden kolayca ayrılabilir.

Register Yapısı: Bir mikroişlemcinin programlanması bu işlemcinin dâhili kaydedicilerinin yapısının tam olarak anlaşılmasını gerektirir. Bu bölümde bu yapıdan ve belleğin segment kaydediciler ve offset adresler kullanılarak nasıl adreslendiği açıklanacaktır.

INTEL ailesi bir 8086 mikroişlemcinin dahili kaydedicileri şekilde gösterilmektedir. Programcıya açık olan bu dahili kaydedicileri üç gurup altında toplayabiliriz: *Genel Amaçlı Kaydediciler***,** *Pointer (gösterici) ve Index (işaret) kaydediciler, Segment Kaydediciler***.** Bunlara ek olarak bir diğer gurup olarak da aritmetik işlem birimi veya aritmetik ve lojik birimi yaptığı işlemler hakkında çeşitli bilgilerin veya koşulların gösterildiği *Flag (bayrak) Kaydedici*'yi sayabiliriz.

32 bitlik kaydediciler EAX, 16 bitlik kaydediciler AX, 8 bitlik kaydediciler AH veya AL biçiminde gösterilmektedir.

Genel Amaçlı Kaydediciler: Genel amaçlı kaydediciler programcının isteğine bağlı olarak her hangi bir şekilde kullanılabilir. Bunların her biri şekilde gösterildiği gibi 32-bit (EAX, EBX, ECX, EDX), 16-bit (AX, BX, CX, DX) ve 8-bit (AH, AL, BH, BL, CH, CL, DH, DL) kaydediciler olarak kullanılabilir. Aynı zamanda bazı emirler, genel amaçlı kaydedicileri özel işlemler için de kullanırlar ve bu yüzden dolayı da her birine değişik bir isim verilmiştir.

Pointer ve Index Kaydediciler: Bu tip kaydedicilerin genel amaçlı bir yapıya sahip olmalarının yanında daha çok sıklıkta bir çok emir için, mevcut segment içrerisinde olmak üzere işlenen datanın tutulduğu bellek lokasyonunu göstermek (pointing) veya işaret etmek (indexing) için kullanılır.

Segment Kaydediciler: Segment (kesim) kaydediciler mikroişlemci içindeki diğer kaydedici yardımı ile bellek adreslerinin üretilmesi ve belleğin adreslenmesi maksadıyla kullanılırlar.

FLAGS Kaydediciler: Bayrak (Flag) kaydediciler hem mikroişlemcinin durumunu hem de onun çalışmasının kontrolünü gösterirler veya tutarlar. Bu kaydedicideki bitler bir çok aritmetik veya lojik emrin icrasından sonra değişir, bazıları ise mikroişlemci özelliklerinin kontrol edilmesinde kullanılır.

8086 ADRESLEME MODLARI

8086 Kaydedici Adresleme: Adından anlaşılacağı gibi kaydediciden kaydediciye yapılan işlemlerde bu adresleme modları kullanılır. En hızlı adresleme modu'dur, çünkü işlem hafızada değil işlemcinin içinde gerçekleşir.

Genel amaçlı ve indeks kaydedicilerde kaydedici adresleme modları:

```
mov   ax, bx      ; BX teki değeri AX'e kopyalar
mov   dl, al      ; AL teki değeri DL'ye kopyalar
```

Kaydedici adreslemede en çok dikkat etmeniz gereken husus hedef ve kaynağın boyutlarıdır. Örneğin 16 bitlik bir kaydediciden 8 bitlik bir kaydediciye taşıma yapılamaz!

Bunlara ek olarak segment kaydedicilerinin kullanımında dikkat edilmesi gereken noktalar vardır.
- Segment kaydedicileri arasında bir transfer işlemi ancak genel amaçlı bir kaydedici vasıtasıyla yapılabilir.
- CS ve IP kaydedicilerinin değeri kaydedici adresleme ile değiştirilemez.

8086 Hafıza Adresleme:

Acil Adresleme (Immediate Addressing): Herhangi bir genel amaçlı veya indeks kaydedicisine doğrudan bir değer yükleye bilirsiniz. Yüklenecek olan veri kod segmentten alınacağından bu tür kullanımları şahsen ben pek tavsiye etmem. İyi bir program organizasyonu için, veriler hafızanın ayrı bir bölümünde (mesela data segmentte) değişkenler veya sabitler olarak belirtilmelidir.

```
mov   al, 17      ; AL'ye 11h yüklenir.
```

Direkt Adresleme (Displacement Only Addressing) : Acil adreslemenin doğru kullanılmış şeklidir. Bu adreslemede segment:ofset adresi kullanılarak hafızaya erişilir.

```
mov   al, ds:12   ; ds:000C adresinden 1 byte AL'ye kopyalanır.
mov   ds:12, al   ; AL'nin içeriği ds:000C adresine kopyalanır.
```

Kaydedici Dolaylı Adresleme (Register Indirect Addressing): Adının kaydedici olduğuna aldanmayın. Burada operand olarak kullanılan kaydedici köşeli parantez içine alınır ve bu andan itibaren bir offset adresi oluverir.

```
mov   al, [bx]    ; hafızadan AL'ye 1 byte taşınır. Alınacak verinin offset adresi BX'in değeridir.
```

İndeksli adresleme (Indexed Addressing): Kaydedici dolaylı adreslemenin operandına sabit bir değer eklenmiş halidir. Kullanım şekli;

```
mov   al, disp[bx]
mov   al, disp[si]
```

Taban İndeksli adresleme (Based Indexed Addressing): Bu adresleme modu da kaydedici dolaylı adreslemeye çok benzer. Kullanım şekli;

```
mov al, [bx][si]
mov al, [bp][si]
```

Taban İndeksli artı direkt adresleme (Based Indexed Plus Displacement Addressing):

Bu adresleme modu taban indeksli adreslemeye 8 yada 16 bitlik sabit bir değerin eklenmiş halidir.

```
mov al, disp[bx+di]
mov al, [bp][di][disp]
```

ASSEMBLY DİLİ

Assembly programlama dili, kullanılan bilgisayar sisteminin yapısına ve işletim sistemi gibi platformlara sıkı-sıkıya bağımlı bir dildir. Assembly programlama dili düşük seviyeli bir dil olup C, C++, Pascal, C# gibi yüksek seviyeli programlama dillerine göre anlaşılması biraz daha zordur. Assembly dili ile program yazarken kullanılan bilgisayarın donanımsal özelliklerinin bilinmesi gerekir. Yazılan program kullanılan mikroişlemcinin yapısına bağlıdır. Assembly dili ile program yazarken programcı doğrudan bilgisayarın işlemcisi ve hafızası ile uğraşır. Ana bellekteki ve işlemci kaydedicilerindeki değerleri doğrudan değiştirebilme imkânı vardır.

Mikroişlemci sadece ikili sayı sisteminde yazılan komut kodlarını, başka bir ifade ile makine dilinden anlar. Assembly dilinde yazılan programları makine diline çevirmek için Assembler adı verilen çevirici(derleyici) programlar kullanılır. Aşağıda verilen şekilde Assembly dili, Makine dili ve Assembler blok olarak görülmektedir. Bilgisayarımızda çalıştırılan tüm programlar önce bilgisayarımızın RAM belleğe yüklenir. Daha sonra RAM bellekten sırası ile mikroişlemci tarafından okunarak çalıştırılır. RAM'e yüklenen veri programın makine dili karşılığından başka bir şey değildir. Yani 0 ve 1 kümeleridir.

Makine dilinde program yazmak oldukça zordur. Buna karşılık makine dili ile birebir karşılığı olan ve komutları kısaltılmış kelimelerden oluşan Assembly dilinden yararlanılır. Assembly dilinde program yazmak makine dilinde program yazmaya göre daha hızlı ve daha kolay yapılabilir. Ayrıca yazılan programların bellekte kapladıkları yerde aynıdır. Başka bir ifade ile bellek kullanımları aynıdır. Yüksek seviyeli dillerle karşılaştırıldığında assembly dilinde yazılan programlar daha hızlıdır ve bellekte daha az yer kaplar. Buna karşılık program yazmak yüksek seviyeli dillerde daha kolaydır.

Assembly programlama dili günümüzde daha çok sistem programcıları tarafından diğer programlama dilleri içerisinde kullanılmaktadır.

Assembly dilinin avantajları

• Bigisayar donanımı üzerinde daha iyi bir denetim sağlar. İşlemcinizin gücünü en iyi şekilde ortaya koyabilecek tek programlama dilidir.

• Küçük boyutlu bellekte az yer kaplayan programlar yazılabilir. virüslerin yazımında kullanılırlar.

• Yazılan programlar daha hızlı çalışır. Çok hızlı çalıştıkları için işletim sistemlerinde kernel ve donanım sürücülerinin programlanmasında, hız gerektiren kritik uygulamalarda kullanılmaktadır.

• Herhangi bir programlama dili altında, o dilin kodları arasında kullanılabilir.

Assembly dilinin dezavantajları

• Assembly dilinde program yazmak için mikroişlemci içyapısı bilinmesi gerekir.

• Assembly dili mikroişlemci tipine göre değişir. Bir mikroişlemci için yazılan bir program başka bir mikroişlemcide çalışmayabilir. Program taşınabilir platformdan bağımsız değildir.

• Assembly dilinde program yazmak yüksek seviyeli dillere göre daha zor ve zaman alıcıdır.

Assembly dilinde program yazma

Assembly dilinde program yazmak için Windows altında yer alan note pad, word pad gibi herhangi bir text editör kullanılabilir. Text editör yardımı ile Assembly dilinde program yazılır. Yazılan program TASM veya MASM assembler çevirici programları yardımı ile .obj uzantılı olarak makine diline çevrilir. Bu halde elde edilen program işletim sisteminin anladığı bir formatta değildir. TLINK bağlayıcı programı kullanılarak .exe veya .com uzantılı hale dönüştürülür.

Bir Assembly dilinde yazılan programda temel olarak şu bölümler bulunur:Yorumlar, Label (Etiketler), Talimatlar, Komutlar

Yorumlar / Açıklamalar

Açıklamalar program satırlarının başına noktalı virgül konularak yapılır. Açıklama satırları assembler tarafından dikkate alınmaz. Program içinde daha detaylı bilgi vermek, kullanılan komutları izah etmek için kullanılır.

; MOV ES,AX bu komut dikkat alınmaz

; AL ye SAYI1 değerini at

Etiketler

Etiketler program içinde kullanılan özel kelimelerdir. Sonuna ":" konularak kelimenin etiket olduğu anlaşılır. Etiketlerden program akışını belirli bir noktaya yönlendirmek istediğimizde yararlanırız.

Son:

Basla: JMP ANA

Talimatlar

Veri tanımlama talimatları

Veri tanımlama talimatları DB, DW, DD,DF, DQ, DT ve DUP dur.

DB (Define Byte): 1 Byte'lık veri tanımlanır.

DW (Define Word):2 Byte'lık veri tanımlanır.

DD (Define double word):: 4 Byte'lık veri tanımlanır.

DF (Define Far Word): 6 Byte'lık veri tanımlanır.

DQ (Define Quad Word): 8 Byte'lık veri tanımlanır.

DT (Define Ten Byte): 10 Byte'lık veri tanımlanır.

DUP: Duplicate

SAYI 3 DUP(0); Bellekten SAYI değişkeni için 3 byte'lık yer ayır, içini 0 ile doldur.

SAYI DW 10 DUP(5) Bellekten SAYI değişkeni için 10x2 byte'lık yer ayır, içlerini 5 ile doldur.

String verileri tanımlama

YAZI DB 'HASAN'

YAZI DB 'H','A','S','A','N'

Dizi Tanımlama

DIZI DB 2, 4, 0, -5, 7

DIZI DB 12, 0FH, 01001001B

Sayıların sonunda B olması verinin ikilik sistemde olduğunu, H olması verinin hexadecimal olduğunu gösterir. Bir şey yazılmamışsa veri onluk sistemde yazılmış anlamına gelir.

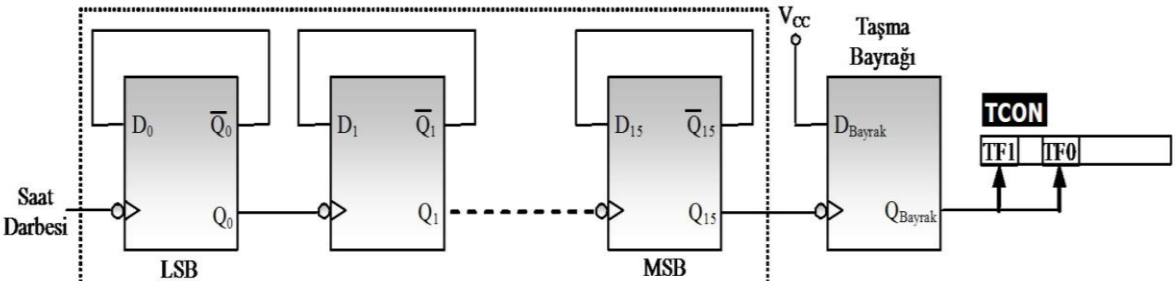
ZAMANLAYICILAR VE SAYICILAR

Mikrodenetleyicilerde Zamanlayıcı/Sayıcı (Z/S) biriminin işlevi :

- Dahili veya harici kaynaklı olarak zamanı ölçmek
- Dahili veya harici kaynaklı olarak olayları saymak

Standart 8086'da 4 farklı modda kullanılabilen 2 adet 16-bitlik Z/S vardır. (T0 ve T1)

16 adet negatif kenar tetiklemeli D tipi FF'un (Flip Flop) asenkron ve ardışık olarak bağlanmasından meydana gelmektedir



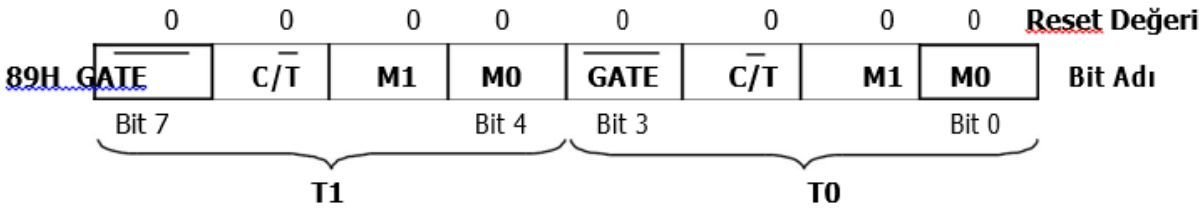
Zamanlayıcı/Sayıcı Saklayıcıları

İsim	Fonksiyonu	Adres	Bit-Adreslenebilir
TCON	Kontrol	88h	√
TMOD	MOD seçimi	89h	–
TL0	Zamanlayıcı-0 düşük-bayt	8Ah	–
TL1	Zamanlayıcı-1 düşük-bayt	8Bh	–
TH0	Zamanlayıcı-0 yüksek-bayt	8Ch	–
TH1	Zamanlayıcı-1 yüksek-bayt	8Dh	–
T2CON*	Zamanlayıcı-2 kontrol	C8h	√
T2MOD*	Zamanlayıcı-2 MOD seçimi	C9h	–
RCAP2L*	Zamanlayıcı-2 yakalama düşük-bayt	CAh	–
RCAP2H*	Zamanlayıcı-2 yakalama yüksek-bayt	CBh	–
TL2*	Zamanlayıcı-2 düşük-bayt	CCh	–
TH2*	Zamanlayıcı-2 yüksek-bayt	CDh	–

TMOD Saklayıcısı

T0 ve T1'in çalışma modlarını (Mod 0, 1, 2, 3)
T0 ve T1'in zamanlayıcı ya da sayıcı olarak çalışma durumunu belirler.

TMOD Zamanlayıcı MOD Seçim Saklayıcısı



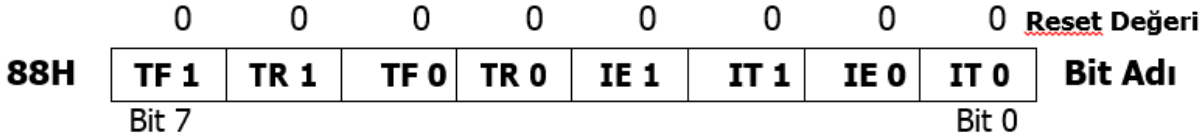
TCON Saklayıcısı

-Bit adreslenebilirdir

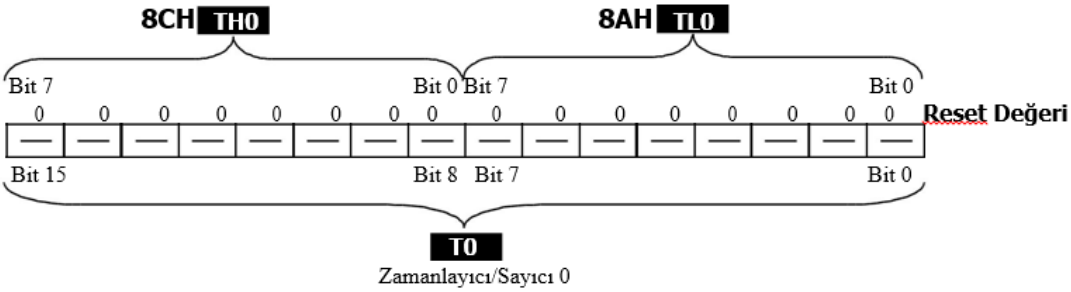
-Yüksek değerlikli dört biti, T0 ve T1'i başlatma, kontrol ve durdurma işlevlerini yerine getirir

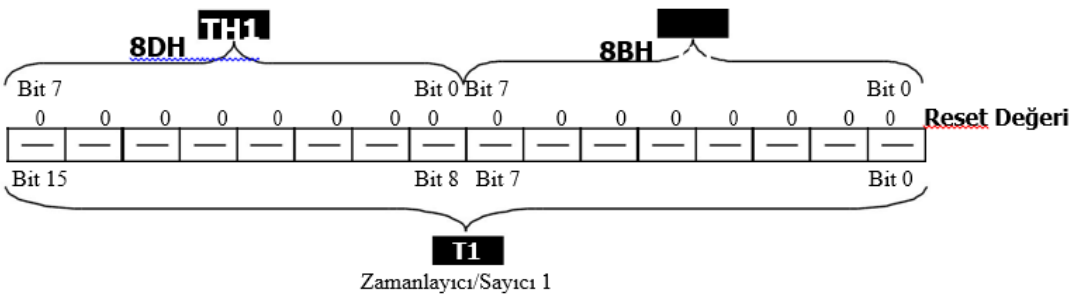
-Düşük değerlikli dört biti ise kesme işlemleri için kullanılır

TCON **Zamanlayıcı/Sayıcı Kontrol Saklayıcısı**



T0 ve T1 Zamanlayıcı/Sayıcıları





Zamanlayıcı/Sayıcı Çalışma Modları

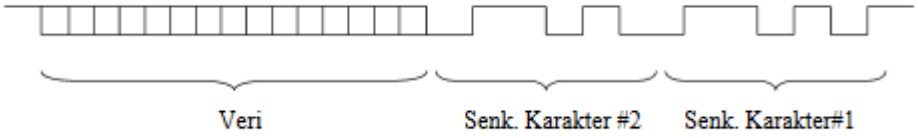
Z/S'ler 4 farklı çalışma moduna sahiptir.
Z/S'nin çalışma modu TMOD saklayıcısındaki M0 ve M1 bitleri ile belirlenir.

SERİ HABERLEŞMELER VE KESMELER

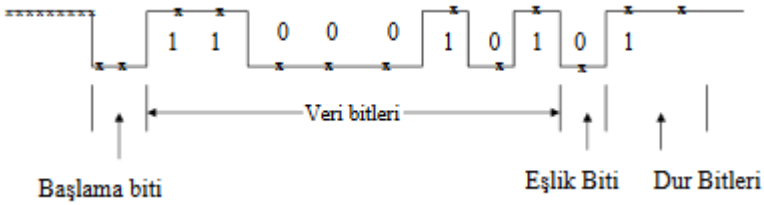
Seri Haberleşmeler

Seri haberleşmeler veri transferlerinde en fazla kullanılan protokolleri içeriyor. Çünkü veri transferlerindeki öncelikli amaç en az hat ile en fazla bilgiyi aktarabilmektir. Seri haberleşmeler kaba taslak iki ana başlık altında toplanır. İlki senkron seri haberleşmedir ki bu haberleşme tipinde veri hattının yanında bir de "clock sinyali" yani kare dalga taşıyan hat vardır. Verilerin her biti bu "clock sinyali"nin düşen ve ya yükselen kenarında okunur. İkincisi ise asenkron seri haberleşmedir. Bu haberleşme tipinde ise veriler bir "start" ve bir "stop" biti arasında paketler halinde taşınır.

Senkron Seri Haberleşme



Asenkron Seri Haberleşme



Genellikle küçük verilerin transferi için asenkron seri haberleşme kullanılır ki bu tür seri haberleşmelerin en başında RS232 protokolü gelir. RS232 protokolü, verileri genellikle 8 bitlik paketler halinde transfer eder.

Öncelikle RS232 veri trasnferi protokolünü destekleyen tüm denetleyicilerde **"SPBRG"** kaydedicisi bulunur. RAM'in "Special Function Register" bölümünde yer alan bu kaydedici, haberleşmenin "Bound Rate" oranını yani hız bilgisini tutar. Bu protokol ile ilgili diğer bir kaydedici **"TXREG"** dir. Bu kaydedici ise gönderilecek veriyi tutar.

Kesme (Interrupt)

Diğer işlemlerin yanında klavyeden kullanıcının giriş de yaptığı bir programı düşünelim. Mikroişlemcinin şimdiye kadar anlatılan dallanma ve altprogram yazılım teknikleriyle klavyeden yapılan karakter girişlerini okurken bu işlemin bitmesini beklemekten başka bir işlem yapması mümkün değildir. Oysa program normal çalışmasını sürdürürken sadece klavye ile ilgili bir olay olduğunda yapılan girişi okuması ve işine devam etmesi (Ekrana bir yazı yazılması veya yazıcıya yazırma..vs) daha mantıklıdır. Mikroişlemcilerdeki Kesme (interrupt) kavramı burada karşımıza çıkmaktadır.

Kesmeler tipik olarak bir donanımın mikroişlemcinin INTR (Interrupt Request) ucundan interrupt isteğini bildirmesiyle donanımla veya "INT XX" komutuyla program içinde yazılımla başlatılan veya tetiklenen, aslında bir CALL altprogram çağırma işlemidir.

Her iki durumda da herhangi bir program içindeyken kesme hizmet alt programı (Interrupt Service Routine - ISR) çağırılır.

Kesme Vektörleri

Kesme vektörü, her biri kesme vektör numarasıyla tanımlanan ve ulaşılan ve mikroişlemci gerçek modda çalışırken hafızanın ilk 1024 byte' lık alanında (003FFH-00000H) saklı olan her biri 4-byte' lık adreslerden oluşur. Bu adresler kesme vektör numarasıyla (Örneğin INT 10H) çağrılan kesme hizmet altprogramının bellek adresleridir. Her bir kesme vektörü bir kesme hizmet programının adresini oluşturan bir IP ve CS içerir. İlk 2 byte IP ve son 2 byte CS adresidir. Dolayısıyla 256 tane farklı kesme vektörü bulunur.Her kesme vektörü bu kesme numarası ile çağırılan bir kesme hizmet programının bellek adresini tutar. Intel ilk 32 kesme vektörünü (0-31) 8086-80486 ve gelecek ürünler için ayırmaktadır.Geri kalan kesme vektörleri (32-255) kullanıcı içindir.

Kesme Komutları

8086 üç farklı kesme komutuna sahiptir: **INT, INTO ve INT 3**.

Gerçek modda bu komutlardan her biri vektör tablosundan bir vektör okur ve sonra bu vektörle adreslenen kesme hizmet programını çağırır.

Korumalı modda bu komutlardan her biri kesme tanımlayıcı tablosundan bir kesme tanımlayıcısı okur. Bu okunan tanımlayıcı kesme hizmet programını belirtir.

Kesme çağırma uzak CALL komutuna benzer çünkü dönüş adresi (IP ve CS) yığına yerleştirilir.

INT

Programcının kullanabileceği 256 tane farklı yazılım kesme komutu (INT) bulunmaktadır. Her INT komutu 2-byte uzunluğunda olup değeri 0 ile 255 (00H- FFH) arasında değişen bir nümerik operand' a sahiptir. İlk byte işlem kodu ikinci byte vektör tipi numarasıdır. Sadece yazılım kesmesi INT 3 farklılık gösterir ve bu komut l-byte uzunluğundadır.

Bir yazılım kesmesi yürütüldüğünde:

1) Bayrak saklayıcısı yığına atılır.
2) T ve I bayrakları temizlenir.
3) CS yığına atılır ve yeni CS için vektörden yeni bir değer okunur. IP yığına atılır ve yeni IP için vektörden yeni bir değer okunur.
4) Yeni CS:IP ile adreslenen yere dallanılır.

INT komutu yürütüldüğünde harici donanım kesme giriş ucu INTR' i (Interrup Request) kontrol eden kesme bayrağı (I) temizlenir. I = O olduğunda mikroişlemci INTR ucunu pasifler (disabled) yani bu uçtan gelecek kesmelere cevap vermez. I bayrağı tekrar 1 olduğunda bu kesme giriş ucu aktif duruma gelir.

INT3

Özel bir yazılım kesmesi olan INT 3 programlarda durma noktası (break point) oluşturma-da kullanılır ve uzunluğu l-byte' tır. Genellikle program hatalarını bulurken program akı-şını kırmada veya kesmede kullanılır.

INTO

Taşma durumunda oluşan bu kesme taşma bayrağını (O) test eden duruma bağlı bir yazı-lım kesmesidir. Eğer O = O ise INTO komutu bir işleme neden olmaz, fakat O = 1 ise ve INTO komutu yürütülmüş ise vektör tip numarası 4 olan bir kesme oluşur.

INTO komutu işaretli ikili sayıları toplayan veya çıkaran programlarda kullanılır. Bu işlemlerde her zaman bir taşma durumu söz konusudur. JO veya INTO komutu bir taşma durumunu bulmak için kullanılabilir.

Kaynaklar:

http://web.karabuk.edu.tr/emelkocak/indir/MTM305/Adresleme%20Modlar%C4%B1.pdf

http://members.comu.edu.tr/boraugurlu/courses/bm307/content/week5/hafta5_a.pdf

http://www.yildiz.edu.tr/~uzun/MST_PDF/MST_04_80x86_Adresleme.pdf

http://ww3.ticaret.edu.tr/eyavuz/files/2017/02/8086-CPU-Tan%C4%B1t%C4%B1m.pdf

http://www.hmyoelk.sakarya.edu.tr/bilgifazlasi/elektronik/mikroislemciler.htm

http://www.emo.org.tr/ekler/d9b816ba072629f_ek.pdf

http://eng.harran.edu.tr/~nbesli/RBT/helptr.htm

http://members.comu.edu.tr/boraugurlu/courses/bm307/content/week10/hafta10.pdf

http://content.lms.sabis.sakarya.edu.tr/Uploads/48893/36956/07_hafta.pdf

http://content.lms.sabis.sakarya.edu.tr/Uploads/65664/51038/mikrosislab_h10_8051zamanlayicisayici.pdf

ADAM ASMACA OYUNU

Kodlar:

```
.data

kelime:          .word 'h' 'a' 's' 'a' ' n'
bosluk:          .word '-' '-' '-' '-' '-'
n:          .word 8
say:       .word 6
pos:       .word 0
sonuc:       .word 1
sonucyok:     .word 0

harf1:    .asciiz "Doğru harf seçtiniz. Devam edin!"
harf2:    .asciiz "Yanlış harf. Tekrar deneyin!!!"
sonuc1:       .asciiz "Tebrikler... Cevabınız doğru."
sonuc2:       .asciiz "Oyun Bitti!"
sonuc3:       .asciiz "Doğru cevap: "
sonuc4:       .asciiz "Sizin tahmininiz: "
soru1:       .asciiz "Ad: "
soru2:       .asciiz "Harf tahmin edin?: "
hak:       .asciiz "hak kaldı: "

.text

.global main
main:

goruntu:

   addi $v0, $zero, 11
   add $a0, $zero, 10
   syscall

   addi $v0, $zero, 4
   la $a0, soru1
```

```
    syscall

    la $t0, n
    lw $s0, 0($t0)
    la $s1, bosluk
    add $s2, $0, $0
    add $s3, $0, $0

dongu1:
    slt $t0, $s3, $s0
    beq $t0, $zero, son1

    sll $t0, $s3, 2
    add $t1, $s1, $t0
    lw $s4, 0($t1)

    addi $v0, $zero, 11
    add $a0, $zero, $s4
    syscall

    addi $s3, $s3, 1
    j dongu1

son1:
    addi $v0, $zero, 11
    add $a0, $zero, 10
    syscall

    addi $v0, $zero, 4
    la $a0, soru2
    syscall

    addi $v0, $zero, 12
    syscall

    la $t0, pos
    lw $s3, 0($t0)
```

```
la $s1, kelime
add $s2, $0, $0

sll $t0, $s3, 2
add $t1, $s1, $t0

beq $v0, $t4, adim1

la $s0, sonuc
la $s2, sonucyok
lw $s3, 0($s2)

sw $s3, 0($s0)

la $s0, say
lw $s1, 0($s0)
sub $s1, $s1, 1
la $s2, say
sw $s1, 0($s2)

addi $s1, $s1, 48

addi $v0, $zero, 11
add $a0, $zero, 10
syscall

addi $v0, $zero, 4
la $a0, harf2
syscall

addi $v0, $zero, 11
add $a0, $zero, 10
syscall

addi $v0, $zero, 4
la $a0, hak
syscall
```

```
    addi $v0, $zero, 11
    add $a0, $zero, $s1
    syscall

    sub $s1, $s1, 48


    addi $v0, $zero, 11
    add $a0, $zero, 10
    syscall

adim1:


    sll $t0, $s3, 2
    add $t1, $s1, $t0

    sw $v0, 0($t1)

    addi $v0, $zero, 11
    add $a0, $zero, 10
    syscall

    addi $v0, $zero, 4
    la $a0, harf1
    syscall

adim2:

    addi $v0, $zero, 11
    add $a0, $zero, 10
    syscall

    la $s0, pos
    lw $s1, 0($s0)
    addi $s1, $s1, 1
```

```
    la $s2, pos
    sw $s1, 0($s2)

    beq $s1, 8, sonuc

    j goruntu

sonuc:

    addi $v0, $zero, 11
    add $a0, $zero, 10
    syscall

    addi $v0, $zero, 4
    la $a0, sonuc3
    syscall

    la $t0, n
        la $s0, pos
    lw $s1, 0($s0)
    add $s2, $0, $0
    add $s3, $0, $0

dongu2:
    slt $t0, $s3, $s0
    beq $t0, $zero, son2

    # load b[i] into register
    sll $t0, $s3, 2
    add $t1, $s1, $t0
    lw $s4, 0($t1)


    addi $v0, $zero, 11
    add $a0, $zero, $s4
    syscall
    addi $s3, $s3, 1
```

```
    j dongu2

son2:
    la $s0, sonuc
    lw $s1, 0($s0)

    beq $s1, 0, yanlis

    addi $v0, $zero, 11
    add $a0, $zero, 10
    syscall

    addi $v0, $zero, 4
    la $a0, sonuc1
    syscall

    j end



dongu3:
    slt $t0, $s3, $s0
    beq $t0, $zero, son3

    sll $t0, $s3, 2
    add $t1, $s1, $t0
    lw $s4, 0($t1)

    addi $v0, $zero, 11
    add $a0, $zero, $s4
    syscall

    addi $s3, $s3, 1
    j dongu3

son3:
    addi $v0, $zero, 11
```

```
    add $a0, $zero, 10
    syscall

    addi $v0, $zero, 4
    la $a0, sonuc4
    syscall

    la $t0, n
    lw $s0, 0($t0)
    la $s1, bosluk
    add $s2, $0, $0
    add $s3, $0, $0

yanlis:

    addi $v0, $zero, 11
    add $a0, $zero, 10
    syscall

    addi $v0, $zero, 4
    la $a0, sonuc2
    syscall

end:
```

Ekran Görüntüleri:

Çok uğraşmama rağmen bir türlü çalıştıramadım. O yüzden maalesef ekran görüntüsü yok. ☹