EEM 332 - Mikroişlemciler

Deney Numarası: 1
Deney Adı: Debug'a Giriş, x86 Yazmaçları ve Makina Komutları

Deneysel Çalışma

Aşağıdaki çalışmaları okuyup, verilen komutları uygulayınız.
DEBUG programını çalıştırmak için bir DOS penceresi açılır ve aşağıdaki komut girilir:

```
C:\> debug <enter>
```

DEBUG tire "-" istemi ile cevap verir. Tire istemi, DEBUG programının verilecek komutları etkileşimli olarak kabul etmeye hazır olduğunu belirtir. En önemli komutlardan birisi, tüm geçerli komutların bir özetini veren "**?**" komutudur. Tüm geçerli DEBUG komutlarının bir listesi ders kitabının 118. sayfasında verilmiştir.

Şimdi DEBUG'ı çalıştırınız ve bütün komutların listesini görüntüleyiniz;

```
C:\> debug      <enter>
-?              <enter>
```

Bu komutlar uygulandığında, ekranda geçerli DEBUG komutlarının bir listesini görülür. Tüm komutlar tek bir harften oluşmaktadır. Bazı komutlar için ek bağımsız değişkenlerin kullanımı gerekmektedir.

DEBUG Üzerinde Aritmetik İşlemler:

"h" komutu, DEBUG'da verilen iki bağımsız değişkenin toplamını ve farkını gösterir. DEBUG sadece 16lık (hexadecimal) sayı sisteminde işlem yapar!

DEBUG istem satırına aşağıdaki komutu giriniz:

```
-h 3 2 <enter>
```

Komut işletildikten sonra ekranda aşağıdaki satır gösterilecektir:

```
0005 0001
```

Burada, sırasıyla toplama ve çıkarma işlemlerinin sonucu gösterilmektedir, 3+2=5 ve 3-2=1.

Şimdi aşağıdaki komutu deneyiniz:

```
-h 2 3 <enter>
```

Bu durumda negatif sonuç, 2-3= -1, 2'nin tümleyeni şeklinde verilir. Negatif bir sayının tanımlanması için o sayının en anlamlı bitinin 1 olup olmamasına bakılır. Öte yandan toplama işlemi sonucu oluşacak pozitif bir sayının da en anlamlı biti 1 olabilir. Sonucu doğru olarak yorumlamak PROGRAMCI'ya kalmaktadır. Örnek olarak, aşağıdaki komutları çalıştırınız:

```
-h E1F6 1E09 <enter>
```

Toplamanın pozitif sonucu FFFF olarak görülmektedir! Şimdi aşağıdaki komutu çalıştırınız:

```
-h 5C3F0 4BC6 <enter>
```

Görülen hata mesajı DEBUG'ın WORD bazında çalışmasından kaynaklanmaktadır. x86 mimarisinde word, 16 bit uzunluğundadır.

Yazmaçlar ve Hafıza:

x86 işlemcileri (en azından 8086 versiyonu) 14 temel yazmaç içerir. Yazmaçların içeriklerini görmek için aşağıdaki komutu giriniz:

```
-r <enter>
```

İkinci satırın sonundaki 2-harfli kısaltmalara dikkat edin. Bu kısaltmalar FLAGS yazmacındaki özel amaçlı bitlerin değerlerini göstermektedir. Bir yazmacın içeriğini incelemek ve değiştirmek için "**r**" komutu ve değişken olarak o yazmacın adı kullanılabilir. Örnek olarak aşağıdaki komutları giriniz:

```
-r ax <enter>
ax    0000
:3A7 <enter>
```

Şimdi r komutunu kullanarak tüm yazmaçları tekrar inceleyiniz. AX yazmacının artık 03A7 sayısını sakladığını göreceksiniz.

r komutunu kullanarak BX yazmacına 92A değerini yükleyiniz. Daha sonra AX ve BX yazmaçlarının sırasıyla 3A7 ve 92A sayılarını sakladıklarını gözleyiniz.

Şimdi 3A7+92A işlemini DEBUG komutları kullanmak yerine *makina komutlarını* kullanarak x86 mikroişlemcisinin hafızasında yapılmasını sağlayacağız. Burada DEBUG sadece komutların işlemciye yüklenmesi için bir araç olarak kullanılacaktır. Hafızanın içeriğinin incelenmesi ve değiştirilmesi için "**e**" komutu kullanılır. Aşağıda verilen komutları uygulayınız:

```
-e 100               <enter>
3756:0100  E4.01     <enter>
-e 101               <enter>
3756:0101  85.D8     <enter>
```

Yukarıdaki komutları girerken büyük bir ihtimalle 3756, E4 ve 85 değerlerinden farklı değerler göreceksiniz. Bu sayılar DEBUG çalıştırılmadan önce hafızanın durumuyla ilgilidir ve bu aşamada bunun bir önemi yoktur. 01 ve D8 değerlerini 0100 ve 0101 numaralı göreli konumlara (offset) girmeniz gerekmektedir. e 100 komutunu girerek, DEBUG'a mevcut kod parçasında (code segment) 0100 numaralı göreli konumdaki bilgi sekizlisini (byte of data) yükleme emrini vermiş oluruz. Yukarıdaki örnekte 3756:0100 adresindeki mevcut sekizli (byte) E4, kullanıcı tarafından girilen 01 sekizlisi ile değiştirilmiştir. Aynı örnekte kod parçası (code segment)'nın adresi 3756'dir. Bu değer kendi bilgisayarlarınızda farklı olabilir, ancak bunun şu an için bir önemi yoktur.

Kod parçasının adresini 16'lık tabanda 10 ile çarpıp, sonuca göreli konum değeri (offset) eklendiğinde fiziksel adres (physical address) elde edilir. X86 sistemleri hafıza adreslerini genellikle parçalı olarak (segmented form) belirtirler.

x86 Kullanarak Toplama:

Yazmaç içeriklerini inceleyiniz:

```
-r <enter>
```

Ekrandaki görüntü aşağıdakilere benzeyecektir:

```
AX=03A7 BX=092A CX=0000 DX=0000 SP=#### BP=#### SI=#### DI=####
DS=#### ES=#### SS=#### CS=#### IP=0100 NV UP DI PL NZ NA PO NC
#### : 0100 01D8 ADD AX, BX
```

Burada # 16'lık bir basamağı temsil eder. Görüntünün son satırı `01D8` değerinin hafızada saklanmakta olduğunu belirtir. Bu, aslında `ADD AX, BX` komutunun makina dili karşılığıdır ve bu komutun çalıştırılması ile belirtilen yazmaçların içerikleri toplanarak sonuç `AX` yazmacına yazılır. `x86` yapısı sekizli-adreslenebilir (byte-addressible) bir yapıda olduğu için bilgi aşağıdaki şekilde düzenlenir:

```
3756 : 0100     01
3756 : 0101     D8
```

`x86`'da komut hafızada bulunduğu halde, komutun çalıştırılabilmesi için komutun nerede bulunduğunun belirtilmesi gerekir. `x86` sistemlerde kullanılan yöntemde iki özel yazmaç kullanılır. Bunlar `CS` ve `IP`'dir. Bu yazmaçlar bir sonra işlenecek komutun parçalı adresini barındırırlar. Böylece işlenecek olan komutun hafızada bulunduğu yer, kod parçası değeri (`CS` yazmacında tutulur) ve göreli konum değeri (`IP` yazmacında tutulur) yardımıyla etkili bir şekilde gösterilir. Gereken durumlar da **`r`** komutu kullanılarak `IP` yazmacına doğru göreli konum değeri yüklenir (örnekte `0100`).

`DEBUG`'da bir izleme (trace) komutu "**`t`**" bulunur ve bu komut sayesinde her defasında bir tane olmak üzere makina dili komutları çalıştırılabilinir. Mevcut hafıza ve yazmaç içerikleriyle `DEBUG`'a izleme komutunu uygulayalım:

```
-t <enter>
```

`AX` yazmacının içeriği artık `03A7` ve `092A` sayılarının toplamı olan `0CD1`'dir. `AX` yazmacının içeriğinin işlem sonucu elde edilen toplam ile değiştiği görülür. Aynı zamanda `IP` yazmacının içeriği de `0102` olarak değişir. `x86` sistemleri `IP` yazmacının içeriğini, işlenecek bir sonraki komutun adresine göre kendiliğinden değiştirir.

Şimdi sistemlerinizi kullanarak `092A` (`BX` yazmacında) değerini, `0CD1` (`AX` yazmacında) değerine ekleyiniz (toplayınız). `DEBUG`'ın `r` komutunu kullanarak IP yazmacını uygun değere ayarlamayı unutmayın!

Çıkarma işlemini gerçekleştirmek için gerekli makina dili komutunu kod parçasına (code segment) eklemek gerekmektedir. `29D8` olarak makina dilinde kodlanmış komutu, göreli konum değerini `0100` olarak ayarlayıp, **`e`** komutunu kullanarak mevcut kod parçasına ekleyiniz. **`r`** komutunu kullarak `SUB AX, BX` komutunun hafızaya yerleştirildiğini gözlemleyiniz ve daha sonra izleme (trace) komutunu kullanarak bu işlemin gerçekleştirilmesini sağlayınız.

Çevirici Dili Programlarındaki (Assembler Language Programs) DOS İşlevlerinin Kullanımı:

INT komutu, x86 sistemlerinin bilgisayarın önyüklemesi (boot) sırasında DOS aracılığıyla sabit diskten hafızaya kopyalanan özel programlara ulaşılmasını sağlar. Şimdilik INT komutu bir "işlev çağrıcı" olarak düşünülebilir.

DEBUG kullanarak 0200 değerini AX yazmacına ve 0041 değerini DX yazmacına yükleyelim. INT 21 komutunun makina dili karşılığı CD21'dır (INT komutu ve yazmaçlara atanan değerlerin detaylarına internet sayfasında verilen bağlantıdan ulaşılabilir). Hafızada, 0102 göreli konum değerini kullanarak mevcut kod parçasına CD ve bir sonraki göreli konum değerinin gösterdiği parçaya 21 değerlerini giriniz. IP yazmacını 0102 olarak ayarlayınız ve hafıza ve yazmaç içeriklerinin doğru olduğunu kontrol ediniz. Daha sonra komutu, aşağıdaki işlemi uygulayarak çalıştırınız:

```
-g 104 <enter>
```

g komutu DEBUG'ın "go until" komutudur. Bu satırı uygulayarak x86'ya mevcut IP değerinden başlayarak, belirtilen IP değerine kadar olan bütün komutların çalıştırılmasını sağlarız. Burada kod parçasında değişiklik olmaz (CS:IP yapısında sadece IP değişir). Eğer izleme (**t**) komutunu kullanmış olsaydık, INT 21 işlevinin içinde bulunan bütün komutların işlenebilmesi için her komut için aynı komutu defalarca kullanmamız gerekirdi ve bazı durumlarda bu oldukça uzun kodların satır satır çalıştırılması anlamına gelmektedir. **g** komutunun kullanılması sayesinde CS:0104 değerine ulaşılana kadar gereken bütün komutların çalıştırılması sağlanır.

Yukarıdaki işlemin sonunda ekranda A karakterinin yazdığını görürüz. Burada kullandığımız INT işlevi, tüm INT işlevleri içinde 21 numaralı olanıdır. Bu işlevin kullanılmasıyla diğer DOS işlevlerine erişim sağlanabilir. AH yazmacındaki 02 (AH, AX'in en anlamlı sekizlisidir) değeri DOS'a ekrana bir karakter yazacağını gösterir. Ekrana basılacak karakteri ise DL (DL, DX'in en az anlamlı sekizlisidir) yazmacına yazdığımız 41 değeri belirtir. 41, A karakterinin ASCII kodundaki karşılığıdır.

Programların Sonlandırılması:

Çevirici veya makina dili programları çalışmalarını bitirdiğinde, x86 sisteminin işletim sistemi tarafından kontrol edilebilir hale gelmesi gerekmektedir. Bunu gerçekleştirmenin bir yolu INT 20 işlevini çağırmaktır. Yüksek seviye programlama dillerinde (C/C++ gibi) yazılan programlarda derleyici kendiliğinden bu satırı eklemektedir. Kullandığımız alt seviye çevirici dilinde (assembler language) ise programın sonuna INT 20 satırını eklemek programcıya ve onun ihtiyaçlarına bağlıdır.

Şimdi 2-satırlı şu programı yazalım:

```
INT 20
INT 21
```

Bunun için aşağıdaki işlemleri gerçekleştirmemiz gerekmektedir:

1. DEBUG'ın **r** komutunu kullanarak AX ve DX yazmaçlarına sırasıyla 0200 ve 0043 değerlerini kaydedelim
2. aşağıda verilenleri sırasıyla hafızaya yükleyelim

```
CD
21
CD
20
```

3. `IP` ve `CS` yazmaçlarına `INT 21` komutunun yerini gösteren hafıza adreslerini yükleyelim
4. geçerli bir bitiş göreli konum değeri belirterek `DEBUG`'ın **`g`** komutunu kullanıp bu komutları çalıştıralım

Tebrikler! İlk makina dili programınızı yazıp çalıştırdınız. Programın çalışmasını laboratuvar sorumlusuna gösteriniz.

SORU 1: 2-satırlı programı çalıştırdığınızda ekranda ne gördünüz? Bunun nedenini açıklayınız.

Çevirme/Geri Çevirme (Assemble/Disassemble):

Pek çok program iki satırdan daha uzundur. Yazdığınız kodların insan tarafından anlaşılabilen hali olan anımsatıcıları (mnemonics) görmek için `IP` yazmacının değerini 2-satırlı programınız başlangıç konumunu gösterecek şekilde ayarlayınız ve `DEBUG`'ın geri çevirme (unassemble) komutunu aşağıdaki şekilde giriniz:

```
-u <enter>
```

Programınızın anımsatıcılar şeklinde yazıldığını ve her anımsatıcının yanında hafızada saklanan 16'lık (hexadecimal) bir makina kodunun olduğunu göreceksiniz. Şu ana kadar yaptıklarımız makina dili kodlarını içermekteydi. Anımsatıcıları kullanarak bir program yazmak, yazım ve anlama açısından daha kolaydır. Bu nedenle `DEBUG` bir çevirme (assemble) komutu "**`a`**" içerir. Örnek olarak 2-satırlı programı ele alalım. Mevcut kod parçasında `0100` numaralı göreli konum değerinden başlayarak aynı programı anımsatıcılar olarak yazalım:

```
-a 100     <enter>
int 21     <enter>
int 20     <enter>
           <enter>
```

Yukarıdaki komutlar sayesinde, aynı programı 16'lık makina kodları yerine anımsatıcılar (mnemonics) olarak ifade edebilirsiniz.

`DEBUG` Senaryoları (Scripts):

Sadece `DEBUG` kullanarak çevirici dili programları yazılabilir. Bu durumda bir takım kısıtlamalar söz konusudur. `DEBUG`'daki bu kısıtlamalarla karşılaşılacağı zaman ayrıntılı bilgi verilecektir. `DEBUG`'da iki farklı şekilde çevirici dili programlaması yapılabilir. İlk yöntemde `DEBUG` penceresinde yazılacak kod çalıştırılabilir. İkinci yöntemde ise çevirici dili kodlarının bulunduğu bir dosya yaratılarak `DEBUG` tarafından çalıştırılması sağlanabilir. İkinci yöntemi gösteren aşağıdaki çalışmayı yapınız:

1. Bilgisayarınızdaki herhangi bir kelime işlemci programı veya yazım (Ör: Notepad) programını kullanarak `TEMP.DBG` adlı bir dosya yaratınız. Bu dosyanın içeriğinin nasıl olması gerektiği aşağıda verilmiştir. İstediğiniz herhangi bir kelime işlemci veya yazım programını kullanabilirsiniz, ancak dosyayı kaydederken <u>mutlaka</u> `TEXT` veya `ASCII` dosyası olarak kaydetmeniz gerekmektedir!

```
a 100
mov ah, 02
mov dl, 2a
int 21
int 20

n temp.com
```

```
r bx
0
r cx
08
w
q
```

Yukarıdaki komutları yazarken arada bırakılan boş satırı sizin de uygulamanız gerekmektedir. Bu satır DEBUG'a çevirici kodlarının bitip, DEBUG komutlarının kullanılmaya başlandığını belirtir.

2. DOS komut istemini çalıştırarak aşağıdaki komutu yazınız:

```
C:\> DEBUG < TEMP.DBG   <enter>
```

Bu işlemden sonra sabit diskinizde TEMP.COM adlı bir çalıştırılabilir dosya oluşturulacaktır.

SORU 2: TEMP.COM adlı dosyayı çalıştırın. Bu dosya ne iş yapıyor?

SORU 3: Bu belgedeki bilgileri ve DEBUG'ın yardım menüsünü (**?** komutu) kullanarak TEMP.DBG dosyasındaki satırların hangi işlevleri gerçekleştirdiğini yazınız.

EEM 332 – Mikroişlemciler
Deney 1 Çalışma Formu

1- Debug'ı çalıştırın ve yazmaç (register) komutunu kullanarak her yazmacın içindeki ilk değerleri not edin.

AX	BX	CX	DX	SP	BP	SI	DI
DS	**ES**	**SS**	**CS**	**IP**			
????:0100							

2- Aşağıda verilen komutları çevirme (assemble) ve izleme (trace) komutlarını kullanarak çalıştırın ve yazmaçların durumlarını ve göreli konum değerlerini her adım için not edin. (başlama adresi: CS:0100)

Göreli Konum Değerleri		AX		BX		CX		DX	
	KOMUTLAR	AH	AL	BH	BL	CH	CL	DH	DL
CS:	MOV AL,57								
CS:	MOV DH,86								
CS:	MOV DL,72								
CS:	MOV CX,DX								
CS:	MOV BH,AL								
CS:	MOV BL,AL								
CS:	MOV BL,9F								
CS:	MOV AH,20								
CS:	ADD AX,DX								
CS:	ADD CX,BX								
CS:	ADD AX,1F35								

3- İkinci soruda yazdığınız anımsatıcıların (mnemonics) makina dili karşılıklarını döküm (dump) komutunu "d" kullanarak bulunuz ve aşağıdaki tabloda uygun konum değerlerine göre yazınız.

100	101	102	103	104	105	106	107	108	109	10A	10B	10C	10D	10E	10F
110	**111**	**112**	**113**	**114**	**115**	**116**									