Assembly Dersi Ödev 1 Raporu

Hazırlayan: Faruk Arslan
16011032

Soru 1- Sağa Döndürme
Kullanılan algoritma

```
//////////  resmin transpozunu alma //////////
for (i = 0; i < n; i++) {
        for (j = i; j < n; j++) {
            temp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = temp;
        }
    }
/////////////  resmin simetriğini alma ////////
for (i = 0; i < n; i++) {
        for (j = 0; j < (n / 2); j++) {
            temp = mat[i][j];
            mat[i][j] = mat[i][n - j-1];
            mat[i][n - j-1] = temp;
        }
    }
```

Öncelikle resmi çevirmede kullandığım algoritmayı buraya c kodu olarak eklemek istedim.

Resmi sağa çevirmek için önce resmin yani matrisimizin transpozesini elde ediyoruz.Bunun için yukarıdaki iç içe for döngüsünü kullanıyoruz.

Ardından oluşan bu yeni matrisin de simetriğini alarak resmimizi sağa çevirmiş oluyoruz. Bu işlem için de alttaki iç içe for döngülerini kullanıyoruz.

```
__asm {
    MOV EBX, resim

    XOR EAX, EAX
    XOR EDX, EDX
    XOR ESI, ESI    //ESI=i EDI=j
    XOR ECX, ECX
    XOR EDI, EDI
    MOV CX, n
```

Burada öncelikle ebx registerimizde resim dizisinin başlangıç adresini tutmak için ebx'e resim i atıyoruz.Ardından registerlerin içinde herhangi bir değer olması ihtimaline karşı xor ile registerlerin içini sıfırlıyoruz. Algoritmamdaki i degerine karşılık esı registerini ve j değişkenine karşılık edı değişkenini kullandım.
Ardından dışdaki döngünün dönüş sayısı n olacağı için cx'i n yapıyoruz. N değeri word olduğu için ecx yerine cx kullanıyoruz.

```
lopdis: PUSH CX
        MOV EDI, ESI

        MOV CX, n
        SUB CX, SI
```

Bu işlemler tamamlandıktan sonra dis döngümüz olan lopdis isimli döngüye başlıyoruz.Burada cx i push ederek stackte tutuyoruz çünkü içteki döngüde tekrar cx i kullanmamız gerekecek ve cx in değeri değişecek.İlk değeri korumak için stack e attık. Sonrasında iç döngüdeki şartlarımızda olduğu gibi edı registerine esı yı atıyoruz çünkü içteki döngümüzdeki değişkenin başlangıç değeri dış döngünün değeri olmalı ki karşılıklı olarak yer değiştirme yapılabilsin. Ardından içteki döngünün dönüş sayısını ayarlamak için cx'e n atıyoruz ve cx den sı çıkarıyoruz.Çünkü içteki for döngümüz i'den n'e kadar yani n-i kadar dönüyor.

```
lopic:  MOV EAX, ESI

        PUSH ESI
        XOR ESI, ESI
        MOV SI, n
        MUL ESI
        POP ESI

        ADD EAX, EDI
        SHL EAX, 1
```

Şimdi içteki döngümüz başlıyor. Adresleri oluşturmak için eax registerini kullandım çünkü aritmetik işlemler ax üzerinden daha kolay yapılmakta. Burada bizim ulaşmamız gereken adres mat[i][j] yani resimdizi[resim + n*i + j] nin adresi. İlk olarak eax'e esı'daki değeri atıyoruz.Ardından eax'i n ile çarpmamız lazım. Ancak 32 bit çarpma yapmamız gerektiği için başka bir register aracılığıyla yapmamız gerekli. Bunun için herhangi bir registeri push pop yaparak kullanabiliriz. Ben burada esı yı push yaparak değeri kaybetmemek için stackte tutuyorum.Ardından esı'yı sıfırlıyorum çünkü ben düşük anlamlı olan 16 bitine değer atacağım ve çarpmada kullanacağım. Sıfırlama yapmazsam soldaki 16 bit çarpma sonucunu değiştirebilir. SI ya n attıktan sonra mul esı ile eax*esı işlemini yapıyorum ve eax'in içinde n*i değerini elde etmiş oluyorum.Ardından pop esı ile gerçek esı değerimi stackten çekip esı ya atıyorum. Sonra eax'e edı yani j ekleyerek eax içinde [(n*i)+j] değerini elde etmiş oluyorum.

Sonra da eax'i 1 kere shift left yapıyorum yani 2 ile çarpıyorum çünkü bizim dizimizin elemanları word şeklinde.

```
PUSH ESI
MOV ESI, EAX
POP EDX
PUSH WORD PTR[EBX + ESI]

MOV ESI, EDX
MOV EDX, EAX
```

Eax’ de istediğim değeri elde ettikten sonra bu adrese erişip bunu stacke atmam kalıyor. Adres erişimi için kullanabileceğim esı, edı ve ebx registerleri var ve ben ebx de resmin adresini tuttuğum için esı yı kulanıyorum. Önce esı’ push yaparak değerini stacke atıyorum , ardından oluşturduğum adres değerini yani eax’deki değeri esı ya atıyorum. Burada pop edx yapıyorum çünkü aşağıda tmp değerimi push yapacağım için esı yı geri çekmek istediğimde tmp yi çekeceği için ben esı değerimi stackden edx’e çekiyorum. Ardından tmp değerimi yani oluşturduğum adresteki değeri push yaparak stacke atıyorum ve bunu en sonda kullanmak için geri çekeceğim.Ardından esı’ya kendi değerini geri vermek için edx’den değeri esı’ ya atıyorum. Bu işlemden sonra edx’e eax değerini yani n*i+j yi atıyorum çünkü bunu aşağıda kullanacağım.

```
MOV EAX, EDI

PUSH EDX
PUSH EDI
XOR EDI, EDI
MOV DI, n
MUL EDI
POP EDI
POP EDX
ADD EAX, ESI
SHL EAX, 1
```

Burada yeni değerim mat[j][i]'yi yani resimdizi[resim + n*j+i] yi oluşturmak için eax'e edı yani j atıyorum.Ardından yine yukarıda anlattığım yöntem ile çarpma işlemi yapıyorum fakat çarpma sonucundan edx'in etkilenme ihtimali olduğu için onu da push pop ile koruyorum. Burada çarpma ile n*j elde edip ardından esı ekleyerek eax'de n*j +i elde etmiş oluyorum.Sonra sola kaydırma işlemi ile eax*2 yapıyorum ve yeni değerimi elde ediyorum.

```
PUSH EDI
MOV EDI, EAX
MOV AX, WORD PTR[EBX + EDI]

PUSH ESI
MOV ESI, EDX
MOV WORD PTR[ESI + EBX], AX
POP ESI

MOV EDX, EDI
POP EDI
```

Ardından aynı yöntem ile edı'ya elde ettiğim değeri atıp Word ptr[ebx + edı] yaparak resmin istediğim elemanına ulaşıyorum ve buradaki değeri ax'de tutuyorum. Sonra önceki adresimi tuttuğum edx'deki değeri esı'ya atıp kullanıyorum ve ax'e attığım değeri buraya taşımış oluyorum. Ardından esı'yı pop yaparak gerçek değerine döndürmüş oluyorum. Sonra edx'e edı'yı yani ikinci olarak oluşturduğumuz adrei atıyorum çünkü birazdan tmp olarak stacke attığımız elemanı bu adrese atacağız.
Pop edı ile de edı2nın gerçek değerini stackten geri alıyorum.

```
POP AX
PUSH ESI
MOV ESI, EDX
MOV WORD PTR[ESI + EBX], AX
POP ESI

INC EDI
LOOP lopic
POP CX
INC ESI
LOOP lopdis
```

Tüm bu işlemlerden sonra sırada tmp yi yeni adresime atma işlemi kalıyor. Bunun için en başta push ile stacke gönderdiğim word ptr[ebx + esı] değerini yani tmp değerimi stackten ax'e çekiyorum.

Ardından bu değeri yine yukarıda yaptığım benzer işlem ile esı aracılığıyla ikinci olarak oluşturduğum adrese atıyorum. Burada edx'de tuttuğum ikinci adresi esı'ya atarak adrese erişiyorum.Sonra pop esı ile gerçek esı değerimi geri çekiyorum.

Bunlardan sonra iç döngü değişkenim olan j'yi yani edı'yı 1 arttırıyorum çünkü adres erişimlerinde shift left ile (adres * 2) yaptığımız için 2 kere arttırmama gerek kalmıyor.Sonra loop ile iç döngümün tekrar dönmesi için döngünün başına gidiyorum.İç döngü bitene kadar bu tekrar ediyor.
Ardından en başta push yaptığım cx değerimi çekiyorum ki dış döngü sayım doğru sayılsın.Sonra dış döngü değişkenim olan esı,2yı da aynı sebepten dolayı bir artırıyorum ve dış döngümü tekrar dönüyorum.

```
----------ilk iç içe döngü bitti , ikinci iç içe döngü başlıyor

MOV EBX, resim

XOR EAX, EAX
XOR EDX, EDX
XOR ESI, ESI   //ESI=i EDI=j
XOR ECX, ECX
XOR EDI, EDI
MOV CX, n
```

Yukarıda ilk iç içe döngümüz ile resmin transpozunu almıştık.Şimdi ikinci iç içe for döngülerimiz ile oluşan resmin simetriğini, yani aynalanmış halini oluşturacağız.Yine ilk döngülerde yaptığımız gibi ebx'de resmin adresini tutacağım.Ardından registerleri sıfırlayıp dış döngümün döngü sayısı olan n'i cx'e atarak dış döngüme başlıyorum.

```
lopdis2:PUSH CX
        XOR EDI, EDI

        MOV CX, n
        SHR CX, 1
```

Burada döngüye başlarken önce cx değerimi iç döngüde de kullanacağım için stacke yedekliyorum.Ardından edı değerini yani iç döngümün değişkeni olan j'yi sıfırlıyorum çünkü bu sefer iç döngü i'den başlamayacak, j=0 şeklinde başlayacak.

Ardından cx'e n atıp shiftright ile 2 ye bölüyorum çünkü içteki döngüm j=0 dan n/2 ye kadar dönecek.Burada n/2 olmasının sebebi resmi çevirmek için ortasaına kadar gitmemiz gerekmesi.Eğer n olsaydı yer değiştirdiğimiz elemanlar tekrar yer değişikliği yapardı ve bir sonuç elde edemezdik.

```
lopic2: MOV EAX, ESI

        PUSH ESI
        XOR ESI, ESI
        MOV SI, n
        MUL ESI
        POP ESI

        ADD EAX, EDI
        SHL EAX, 1
```

Burada iç döngümüz başlıyor. Yine adres değerlerini elde etmek için eax kullandım. Burada ilk ulaşmamız gereken adres mat[i][j] yani resimdizi[n*i + j] adresi. Bu yüzden başta eax'e esı yani i değerimi atıyorum. Ardından push pop kullanarak registerimin değerini koruyarak eax'i n ile çarpıp eax'de n*i elde ediyorum. Sonra bu değere edı yani j ekleyip 2 iki çarpmak için shiftleft yapıyorum.

```
PUSH EDI
MOV EDI, EAX
POP EDX
PUSH WORD PTR[EDI + EBX]
MOV EDI, EDX

MOV EDX, EAX
```

İstediğim değeri elde ettikten sonra raporun başında da açıkladım yöntem ile tmp değerimi stacke atıyorum çünkü en sonda oradan çekip kullanacağım. Bunun için de yine bir registerimi push edip kullanıyorum ve gerçek değerini edx ile stackten alıp esı'ya geri veriyorum. Ardından oluşturduğum bu ilk adresi yer değiştirmede kullanacağım için edx'de tutuyorum.

```
MOV EAX,ESI

PUSH EDX
PUSH ESI
XOR ESI, ESI
MOV SI, n
MUL ESI
POP ESI
POP EDX

PUSH ESI
XOR ESI,ESI
MOV SI,n
ADD EAX,ESI
POP ESI

SUB EAX,EDI
DEC EAX


SHL EAX, 1
```

Şimdi sıra ikinci adresimi oluşturmaya geldi. Burada elde etmek istediğim değer mat[i][n-j-1] yani resimdizi[resim + (n*i) + n - j -1] değeri.

Bunun için öncelikle eax'e esı yani j atıyorum. Sonra aynı çarpma yöntemi ile n*i elde ediyorum. Ardından bu değere n eklemem lazım ve bunun için de yine çarpmadaki yöntemi kullanıp bir register aracılığıyla sayıma n ekliyorum.

Ardından elde ettiğim n*i + n sayısından j çıkarıp 1 eksiltmek için işlemlerimi yapıyorum ve n*i + n - j -1 elde edip shiftleft ile ikiyle çarpıyorum.

```
PUSH ESI
MOV ESI, EAX
MOV AX, WORD PTR[EBX + ESI]

PUSH EDI
MOV EDI, EDX
MOV WORD PTR[EBX + EDI], AX
POP EDI

MOV EDX, ESI
POP ESI
```

Burada oluşturduğumuz adreslerdeki değerlerin yerlerini değiştiriyoruz. İkinci olarak oluşturduğumuz adresteki değeri esı registeri aracılığıyla ax'e atıyoruz. Sonra edı registerini kullanarak da ilk oluşturduğumuz adrese yani edx'de tuttuğumuz adrese atıyoruz. Ardından esı'ya attığımız adresi yani ikinci oluşturduğumuz adresi edx' alıyoruz.Ardından esı'ya pop yaparak gerçek değerini alıyoruz.

```
POP AX
PUSH ESI
MOV ESI, EDX
MOV WORD PTR[ESI + EBX], AX
POP ESI

INC EDI

LOOP lopic2
POP CX
INC ESI
LOOP lopdis2
```

Son olarak tmp'deki değeri alıp ikinci oluşturulan adrese atacağız ve yer değiştirme işlemi tamamlanacak.Bunun iin pop ax yaparak en başta push yaptığımız tmp değerini ax'e çekiyoruz. Sonra esı registerini yine geçici olarak kullanıp ikinci adrese tmp değerini atıyoruz ve işlemler tamamlanıyor.

Döngü sonunda da iç döngü değişkeni olan j'yi yani edı'yı bir artırıyoruz.Sonra iç döngü loop ile dönmeye devam ediyor.İç döngü bitince de pop cx ile dış döngünün dönme sayısını alıyoruz ve dış döngü değişkeni olan esı yani i'yi bir artırıp loop ile tekrar başa dönüyoruz.

Soru 1- Sola Döndürme
Kullanılan algoritma

```
////////////////// transpoz ama ters yöne //////////////
for (i = 0; i < n; i++) {
        for (j = i; j < n; j++) {
            temp = mat[j][i];
            mat[j][i] = mat[i][j];
            mat[i][j] = temp;
        }
    }

////////////////// simetriğini yapma /////////////////
    for (i = 0; i < n; i++) {
        for (j = 0; j < (n / 2); j++) {
            temp = mat[j][i];
            mat[j][i] = mat[n - j-1][i];
            mat[n - j-1][i] = temp;
        }
    }
```

Resmimizi sola döndürürken sağa döndürme işlemi için kullandığımız algoritmanın aynısını kullanıyoruz. Fakat burada çevirme işlemi ters yönde olacağı için kullandığımız indisler de ilk algoritmadakilerin yer değiştirmiş hali oluyor.

Yine ilk olarak transpoz alma algoritması ile resmi çevirip ardından simetriğini alıyoruz.

```
__asm {
    MOV EBX, resim

    XOR EAX, EAX
    XOR EDX, EDX
    XOR ESI, ESI    //ESI=i EDI=j
    XOR ECX, ECX
    XOR EDI, EDI
    MOV CX, n
```

Burada öncelikle ebx registerimizde resim dizisinin başlangıç adresini tutmak için ebx'e resim'i atıyoruz. Ardından registerlerin içinde herhangi bir değer olması ihtimaline karşı xor ile registerlerin içini sıfırlıyoruz. Algoritmamdaki i degerine karşılık esı registerini ve j değişkenine karşılık edı değişkenini kullandım.

Ardından dışdaki döngünün dönüş sayısı n olacağı için cx'i n yapıyoruz. N değeri word olduğu için ecx yerine cx kullanıyoruz.

```
lopdis: PUSH CX
        MOV EDI, ESI

        MOV CX, n
        SUB CX, SI
```

Bu işlemler tamamlandıktan sonra dis döngümüz olan lopdis isimli döngüye başlıyoruz.Burada cx i push ederek stackte tutuyoruz çünkü içteki döngüde tekrar cx i kullanmamız gerekecek ve cx in değeri değişecek. Sonrasında iç döngüdeki şartlarımızda olduğu gibi edı registerine esı yı atıyoruz çünkü içteki döngümüzdeki değişkenin başlangıç değeri dış döngünün değeri olmalı ki karşılıklı olarak yer değiştirme yapılabilsin. Ardından içteki döngünün dönüş sayısını ayarlamak için cx'e n atıyoruz ve cx'den SI çıkarıyoruz. Çünkü içteki for döngümüz i'den n'e kadar yani n-i kadar dönüyor.

```
lopic:  MOV EAX, EDI

        PUSH ESI
        XOR ESI, ESI
        MOV SI, n
        MUL ESI
        POP ESI

        ADD EAX, ESI
        SHL EAX, 1
```

Burada içteki döngümüz başlıyor. Önceki algoritmada yaptığım işlemlerin aynılarını kullanarak eax ile hedeflediğim değeri oluşturuyorum. Önce eax'e edı'yı atıp ardından esı registerini geçici olarak kullanıp eax'de j*n elde ediyorum. Sonra EAX + ESI ile de ilk adres değerim olan N*j + i yani mat[j][i] elde ediyorum ve bunu 2 ile çarpmak için eax'i bir kez sola shift ediyorum çünkü dizimizin elemanları word tanımlı.

```
PUSH EDI
MOV EDI, EAX
POP EDX
PUSH WORD PTR[EBX + EDI]

MOV EDI, EDX
MOV EDX,EAX
```

Ardından elde ettiğim bu değeri tmp olarak tutmak için yani en sonda tekrar kullanabilmek için stacke atıyorum. Bunu yapmak için de Edı registeri ile işlemleri gerçekleştiriyorum ve edı'nın gerçek değerini kaybetmemesi için de edx ile edı'nın değerini tutup, işlem bitince geri veriyorum. Sonra bu oluşturduğum değeri bir sonraki aşamada yer değiştirmede kullanacağım için bu adres değerimi edx'de tutmak için mov edx,eax yapıp bir sonraki adıma geçiyorum.

```
MOV EAX, ESI
PUSH EDX
PUSH ESI
XOR ESI, ESI
MOV SI, n
MUL ESI
POP ESI
POP EDX
ADD EAX, EDI
SHL EAX, 1
```

Burada ikinci adresimi yani mat[i][j] yi elde etmek için önce eax'e esı yani i atıyorum. Sonra n ile çarpmak için esı değişkenini geçici olarak kullanıyorum. Burada edx'i de push pop yapıyorum çünkü çarpma sonucunda edx de değişebilir ve benim ilk adres değerim edx'de tutulduğu için onu kaybetmemem lazım. Ardından çarpma işlemi sonucu elde ettiğim i*n değerine j eklemek için add eax,edı yapıyorum ve eax'i 2 ile çarpmak için shift left yapıp eax registerinde [n*i + j] yi yani ikinci adres değerimi elde ediyorum.

```
PUSH ESI
MOV ESI, EAX
MOV AX, WORD PTR[EBX + ESI]

PUSH EDI
MOV EDI, EDX
MOV WORD PTR[EDI + EBX], AX
POP EDI


MOV EDX,ESI
POP ESI
```

Adres değerlerimi elde ettiğim için yer değiştirme yapabilirim. İkinci adresteki değerimi ilk adrese taşımak için esı registerini geçici olarak kullanıp ax'e ikinci adresteki matris elemanının içindeki değeri atıyorum. Ardından edı registerini kullanarak da ax'deki değerimi ilk tuttuğum adrese atıyorum. Burada edı'ya edx'i atmamın sebebi ilk adres değerimi edx'de tutmuş olmam.

Sonrasında edx'e ikinci adresi atıyorum çünkü bir sonraki adımda tmp'yi pop yapıp bu adrese atacağım.

```
POP AX
PUSH EDI
MOV EDI, EDX
MOV WORD PTR[EDI + EBX], AX
POP EDI

INC EDI
LOOP lopic
POP CX
INC ESI
LOOP lopdis
```

Son adımda da tmp değeri olarak en başta stacke yolladığım değeri ax'e pop ile çekiyorum. Bu değeri ikinci adresime yani mat[i][j] ye atmak için edx'de tuttuğum adresi kullanıp geçici olarak edı registerinden faydalanıyorum.

Atama işlemleri bitince de döngü değişkenlerini artırarak döngülerin dönmesiyle devam ediyorum.

```
;----|---ilk iç içe döngü bitti ikinci iç içe döngü başlıyor--

MOV EBX, resim

XOR EAX, EAX
XOR EDX, EDX
XOR ESI, ESI   //ESI=i EDI=j
XOR ECX, ECX
XOR EDI, EDI
MOV CX, n
```

İlk iç içe döngüm bitince ikinci iç içe döngüme başlamak için tekrar registerleri sıfırlayıp ebx'e resmin adresini atıyorum.Dış döngünün dönme sayısı olan n'i de cx'e atıyorum.

```
lopdis2: PUSH CX
         XOR EDI,EDI

         MOV CX, n
         SHR CX,1
```

İç döngümün dönme sayısı olan n/2 yi elde edeceğim için dış döngünün dönme sayısını korumam lazım, bu yüzden cx'i push yapıyorum. Ardından xor edı,edı yaparak edı'yı yani j'yi sıfırlıyorum çünkü iç döngü her yeni dış döngüde j=0 ile başlıyor. Sonra iç döngünün dönme sayısı olan n72 elde etmek için cx'e n atıp shift right ile 2 ye bölüyorum. Burada n/2 olmasının sebebi resmi çevirmek için ortasaına kadar gitmemiz gerekmesi. Eğer n olsaydı yer değiştirdiğimiz elemanlar tekrar yer değişikliği yapardı ve bir sonuç elde edemezdik.

```
lopic2: MOV EAX, EDI

         PUSH ESI
         XOR ESI,ESI
         MOV SI,n
         MUL ESI
         POP ESI

         ADD EAX,ESI
         SHL EAX,1
```

Sonra ilk adres değerim olan mat[j][i] elde etmek için eax'e j yani edı atıp n ile çarpma işlemi yapıyorum. Sonrasında i eklemek için esı ekleyip son olarak da iki ile çarpıyorum çünkü dizimiz word tipinde.

```
PUSH ESI
MOV ESI,EAX
POP EDX
PUSH WORD PTR[ESI + EBX]
MOV ESI,EDX

MOV EDX,EAX
```

Ardından elde ettiğim bu değeri tmp olarak tutmak için yani en sonda tekrar kullanabilmek için stacke atıyorum. Bunu yapmak için de Esı registeri ile işlemleri gerçekleştiriyorum ve esı'nın gerçek değerini kaybetmemesi için de edx ile esı'nın değerini tutup, işlem bitince geri veriyorum. Sonra bu oluşturduğum değeri bir sonraki aşamada yer değiştirmede kullanacağım için bu adres değerimi edx'de tutmak için mov edx,eax yapıp bir sonraki adıma geçiyorum.

```
XOR EAX,EAX
MOV AX,n
SUB EAX,EDI
DEC EAX

PUSH EDX
PUSH ESI
XOR ESI,ESI
MOV SI,n
MUL ESI
POP ESI
POP EDX

ADD EAX,ESI
SHL EAX,1
```

İkinci değerim olan mat[n - j - 1][i] adresi için ax'e n atıp j çıkarıp 1 eksiltiyorum. Benim adresim aynı zamanda resimdizi [resim + n*(n-j-1)+i] olduğu için bu değeri yukarılarda da kullandığım yöntemle n ile çarpıp i yani esı ekliyorum. Sonra da 2 ile çarpma için left shift yapıyorum.

```
PUSH EDI
MOV EDI,EAX
MOV AX,WORD PTR[EBX + EDI]

PUSH ESI
MOV ESI,EDX
MOV WORD PTR[EBX + ESI],AX
POP ESI

MOV EDX,EDI
POP EDI
```

```
POP AX
PUSH EDI
MOV EDI,EDX
MOV WORD PTR[EDI + EBX],AX
POP EDI

INC EDI

LOOP lopic2
POP CX
INC ESI
LOOP lopdis2
```

Adresleri elde ettikten sonra ilk döngülerimde kullandığım yöntem ile elemanların yerini değitirip döngülerimi tamamlıyorum.

(Not: Algoritma gereği aynı işlemleri tekrar tekrar yaptığım için bazı yerlerde ayrıntılı açıklama eklemedim.İlk algoritmadaki ayrıntılı açıklamaları tekrar etmeden kısaca açıklık getirmeye çalıştım.)

Soru 2 Quıck Sort

```
myss    SEGMENT PARA STACK 'yıgın'
        DW 100 DUP(?)
myss    ENDS

myds    SEGMENT PARA 'veri'
CR      EQU 13
LF      EQU 10
HATA    DB  CR,LF,'sayi girmediniz yeniden giriniz:',0
dizi    DB  100 DUP(?)
n       DB 0 ;ELEMAN SAYISI
LOWPTR  DB 0 ;SOLDAKİ POİNTERİM
HIGHPTR DB ? ;SAĞDAKİ POİNTERİM
Mesaj1      DB  CR,LF,'lutfen eleman sayisini giriniz: ',0
Mesaj2  DB  CR,LF,'lutfen elemanlari giriniz: ',0
myds ENDS
```

Programın başlangıcında gerekli olan tanımlamaları ve segmentleri ayarlıyorum. Burada data segmentte cr yani carriage return (satır başı) için ascii değeri olan 13 ü ve lf yani line feed(satır besleme) için de ascii değeri olan 10 değerini tanımlıyorum.

Ardından hata mesajım için ekrana gelecek yazıyı, dizi için 100 elemanlık yer tanımını, eleman sayısını tutacağım n değişkenini, pivotun solunda ve sağında kullanacağım indis değerlerini, eleman sayısı alırken ve elemanlarıisterken kullanacağım mesajları tanımlıyorum.

Mesajların başında cr ve lf kullanarak yeni satıra geçerek yazmasını sağlıyoruz.
Son olarak data segmentimi kapatıyorum.

```
mycs    SEGMENT PARA 'kod'
        ASSUME CS:mycs, DS:myds, SS:myss
BASLA PROC FAR
        PUSH DS
        XOR AX,AX
        PUSH AX
        MOV AX,myds
        MOV DS,AX
```

Kod segmentimi oluşturup altında kesim yazmaçları ile oluşturduğum kesimleri eşleştiriyorum.

Sonrasında programa başlayıp 'BASLA' adında prosedür oluşturuyorum ve data segmentimi DS ye atmak için işlemlerimi gerçekleştiriyorum.

```
        LEA AX,Mesaj1
        CALL PUT_STR
        CALL GETN
        MOV n,AL
        DEC AL
        MOV HIGHPTR,AL

        XOR DI,DI
        XOR CX,CX
        MOV CL,n


        JMP zıpla
tekrar: LEA AX,HATA
        CALL PUT_STR

zıpla:  LEA AX,Mesaj2
        CALL PUT_STR
;--------elemanları alıyoruz
EL_AL:  CALL GETN
        CMP AX,-128
        JL tekrar
        CMP AX, 127
        JG tekrar
        MOV dizi[DI],AL
        INC DI
        LOOP EL_AL
;----------------------------
        CALL QUICKSORT
```

Sonrasında AX registerime Mesaj1'in yani eleman sayısını isteyeceğim mesajın adresini LEA komutu ile alıyorum. PUT_STR ile ax 'de adresini tuttuğum mesajı yazdırıyorum ve GETN ile kullanıcının gireceği eleman sayısını AL'ye alıp n değişkenime atıyorum. Sonrasında al'yi bir azaltıp pivotun sağ tarafını belirtmek için kullanacağım HIGHPTR değişkenine atıyorum. Burada n-1 yapmamın sebebi indislerin 0 dan başlaması.

Sonra DI registerimi sıfırlıyorum çünkü aşağısında for döngüsü ile elemanlarımı alırken indis olarak DI yı kullanacağım. CL'ye n atarak döngü sayımı belirleyip ax aracılığıyla yine ekrana elemanları isteme mesajımı yazdırıyorum.

```
        JMP zıpla
tekrar: LEA AX,HATA
        CALL PUT_STR

zıpla:  LEA AX,Mesaj2
        CALL PUT_STR
;--------elemanları alıyoruz
EL_AL:  CALL GETN
        CMP AX,-128
        JL tekrar
        CMP AX, 127
        JG tekrar
        MOV dizi[DI],AL
        INC DI
        LOOP EL_AL
;----------------------------
        CALL QUICKSORT
```

Başlattığım döngü ile her seferinde GETN fonksiyonuyla AL registerime değer okuyup bu değerleri dizinin elemanı olması için DI'yı kullanarak diziye atıyorum. Her seferinde DI yı 1 arttırarak sıradaki indise geçip sırayla elemanlarımı alıyorum.Bunları yapmadan önce aldığım elemanı cmp kullanarak -128'den küçük ve 127'den büyük olmaması için kontrol ediyorum. Eğer kriterlere uymayan sayı girilirse tekrar noktasına zıplayıp ekrana hata mesajımı yazdırıyorum ve eleman almaya devam ediyorum.Yukarıda jmp zıpla yapmamın sebebi ise ilk başta ekrana hata yazdırmaması gerekli. Bu yüzden orada atlama yapıyorum. Bu işlem bitince de Quıcksort fonksiyonumu call ile çağırıp sıralamaya başlıyorum.(Devamındaki işlemleri ileride anlatacağım.)

Ana fonksiyonumun altına quıck sort fonksiyonumu tanımlıyorum. Quıcksort algoritmasında bir pivot seçiliyor ve bu pivotun sağındaki ve solundaki elemanlar önce pivottan büyük veya küçük olmasına göre pivotun sağına ve soluna yerleştiriliyor. Ardından bu oluşan dizinin sağına ve soluna da tekrar recursive olarak quıck sort yapılarak dizi sıralanmış oluyor. Algoritmada pivotu hangi eleman seçtiğinizin bir önemi yok. Ben burada ortadaki elemanı pivot seçip işlemler yapacağım.

```
;------------------------------QUICKSORT FONKSİYONU--
QUICKSORT    PROC NEAR
             XOR BX,BX
             MOV BL,LOWPTR
             MOV DI,BX

             MOV BL,HIGHPTR
             MOV SI,BX

             MOV AX,DI    ;AX E DI VE SI YI ATARAK N-1 Y
             ADD AX,SI
             SHR AX,1

             MOV BX,AX
             MOV DL,dizi[BX] ;BURADA DİZİNİN ORTASINDAK
```

Burada önce pivotumun sol tarafındaki ucun değerini tutması için LOWPTR değişkenimi BL registeri aracılığıyla DI ya atıyorum.Dı registeri benim soldan pivota doğru gelmemde indis görevi görecek. Aynı şekilde sağ ucu tutan değer olan HIGHPTR değerimi de BL aracılığıyla SI ya atıyorum.SI da benim pivotun sağından pivota doğru gelmemde indis görevi görecek.(SI ve DI dan sağ ve sol indis diye bahsedeceğim.)

Pivotumu belirttiğim gibi dizinin ortasındaki eleman olarak belirledim. Bunun için AX registerine DI yı yani soldaki indis elemanımı atıyorum. Ardından SI ekleyerek sağdaki indis elemanımla topluyorum ve shift right ile elde ettiğim değeri ikiye bölüyorum. Sonuçta pivotumun indisi yani (n-1)/2 elde etmiş oluyorum. Pivot olarak seçtiğim indisteki elemanı da DL registeri ile tutuyorum çünkü herhangi bir bölme veya çarpma işlemi yapmayacağım için dx boşta kalıyor ve onu bu şekilde kullanıyorum.

```
L1:     CMP DI,SI
        JA SOLUSIRALA

L2:     CMP dizi[DI],DL
        JGE ATLA1
        INC DI
        JMP L2

ATLA1:  CMP dizi[SI],DL
        JLE ATLA2
        DEC SI
        JMP ATLA1

ATLA2:  CMP DI,SI
        JA  L1
        MOV DH,dizi[DI]
        XCHG DH,dizi[SI]
        MOV dizi[DI],DH
        INC DI
        CMP SI,0
        JE L1
        DEC SI
        JMP L1
```

Sağ ve sol indislerimi ve pivotumu oluşturduktan sonra işlemlere başlıyorum. İlk olarak yapacağım işlem pivotun sağındaki ve solundaki elemanları pivot ile karşılaştırıp ona göre yer değiştirmek. Bunun için ilk önce sol ve sağdaki indislerimi karşılaştıryorum. Çünkü işlemler yapıldıkça soldaki indisim sağdakinden büyük olabilir ve bu demek olur ki pivotun solundaki elemanları tamamı pivottan küçük olacak şekilde işlemler yapıldı, sol tarafı kendi içinde sıralayabiliriz.

Yaptığım kontrolde eğer soldaki indis sağdakini geçmişse sol tarafla işim bittiği için orayı sıralamam gerekiyor ve SOLUSIRALA noktasına zıplıyorum. (ileriki sayfalarda bu bölüm anlatılacaktır.)

Eğer sol indis hala sağdakinden küçükse kod devam ediyor. L2 noktasında sol indisteki eleman ile pivottaki elemanı karşılaştırıp eğer sol indisin bulunduğu noktadaki eleman pivottan küçükse DEC DI diyerek soldan pivota doğru bir adım yaklaşıyorum.(Devamı sonraki sayfada.)

```
L1:         CMP DI,SI
            JA SOLUSIRALA

L2:         CMP dizi[DI],DL
            JGE ATLA1
            INC DI
            JMP L2

ATLA1:      CMP dizi[SI],DL
            JLE ATLA2
            DEC SI
            JMP ATLA1

ATLA2:      CMP DI,SI
            JA  L1
            MOV DH,dizi[DI]
            XCHG DH,dizi[SI]
            MOV dizi[DI],DH
            INC DI
            CMP SI,0
            JE L1
            DEC SI
            JMP L1
```

Eğer sol indisin bulunduğu yerdeki dizi elemanı pivottan büyük ise ATLA1 noktasına atlıyorum ve sol tarafı bırakıp sağ indisin bulunduğu yere işleme başlıyorum. Burada da sağ indisteki eleman pivottan büyükse ellemeden bir sola kayıp pivota yaklaşarak karşılaştırma yapmaya devam ediyorum. Pivottan küçük elemana rastladığımda ATLA2 noktasına zıplıyorum ve burada solda bulmuş olduğum ''pivottan büyük eleman'' ile sağ indiste bulmuş olduğum ''pivottan küçük elemanı'' yer değiştiriyorum.

Bunu yaparken yine sol ve sağ indis tutan elemanlarımı karşılaştırıp eğer soldaki sağdakini geçtiyse solu pivotun solunu sıralaması için zıplıyorum. Geçmediyse de yer değiştirmek için boşta duran DH registerinden faydalanıp soldaki indisimdeki dizi elemanını DH ye atıyorum. Ardından sağdaki indisimdeki dizi elemaıyla xchg yaparak yer değiştiryorum ve sağdaki indisteki elemandan gelen değeri de soldaki indisteki dizi elemanına atıyorum.

Bunlar bitince de sol indisimi 1 artırıp pivota yaklaşıyorum. Sağ indisimi azaltmadan önce sağ indisin 0 olup olmaması kontrolü yapıyorum.Çünkü eğer sağdaki indisim en başa gelmişse ve ben azaltmaya çalışırsam yanlış sonuçlanır. Bu kontrol sonucu sağ indis sıfır ise L1 noktasına zıplayıp devam ediyorum.Sıfır değil ise 1 azaltarak pivota bir adım daha yaklaşıp L1 e zıplıyorum.

```
SOLUSIRALA: XOR AX,AX
            MOV AL,LOWPTR
            CMP AX,SI
            JAE SAGISIRALA
            MOV AX,SI
            MOV HIGHPTR,AL
            CALL QUICKSORT

SAGISIRALA: MOV AL,n
            DEC AL
            MOV HIGHPTR,AL
            XOR AX,AX
            MOV AL,HIGHPTR
            CMP DI,AX
            JAE BİTİS
            MOV AX,DI
            MOV LOWPTR,AL
            CALL QUICKSORT

BİTİS:      RET
QUICKSORT   ENDP
```

Sağa ve sola atma işlemleri bittikten sonra sırada oluşan sol ve sağ taraftaki dizileri sıralamak var. Yukarılarda bahsettiğim karşılaştırma sonucu yani sol indis sağ indisten büyük olunca SOLUSIRALA' ya zıplayıp sol tarafta oluşan diziyi sıralayacağım.Bunu yaparken yine quıck sort fonksiyonumdan faydalanacağım için sol taraf için yeni indis değerleri oluşturuyorum. Bunu yaparken de önce AX registerimi kullanacağım için sıfırlıyorum. Ardından AL' ye lowptr yani ilk baştaki dizinin başını temsil eden değeri atıyorum ve bu değeri SI ile karşılaştıryorum. Çünkü eğer SI AX'e yani lowptr'ye eşitse sadece SI en başa kadar gelmiştir ve sağdaki elemanların sıralanması gerekir. Bunun için SAGSIRALA noktasına zıplayacağım.(Bir sonraki sayfada anlatacağım.)

Eğer SI başa kadar gelmemişse pivotun solunda oluşan diziyi sıralarken sağ indis olarak SI'yı kullanacağım. Bu şekilde indis ataması yapıp tekrar Quıck sort fonksiyonunu çağırarak pivotun solunu kendi içinde sıralıyorum.

```
SOLUSIRALA: XOR AX,AX
            MOV AL,LOWPTR
            CMP AX,SI
            JAE SAGISIRALA
            MOV AX,SI
            MOV HIGHPTR,AL
            CALL QUICKSORT

SAGISIRALA: MOV AL,n
            DEC AL
            MOV HIGHPTR,AL
            XOR AX,AX
            MOV AL,HIGHPTR
            CMP DI,AX
            JAE BİTİS
            MOV AX,DI
            MOV LOWPTR,AL
            CALL QUICKSORT

BİTİS:      RET
QUICKSORT   ENDP
```

SAGSIRALA bölümünde ise sol tarafında yapılması gereken işlem kalmadığı zaman sıralama yapıyorum. Bunun için de sağda kalan diziye özel indisler oluşturmam lazım. Bunu yaparken önce diziin sonunu HIGHPTR yapmak için AX'e n atıp 1 eksiltiyorum. Sonra bunu HIGHPTR ye atıyorum. Şimdi bir karşılaştırma yapmam lazım. Eğer benim asıl dizimde sol indis yeni oluşturduğum sağ indisten büyükse zaten dizi sıralanmış demek oluyor. Bu yüzden tekrar AL ye HIGHPTR atıyorum ve karşılaştırmamı yapıyorum.

Karşılaştırma sonucu sol indisim yeni HIGHPTR den küçük çıkarsa sol indisimi de yeni sol ucu tutan eleman olarak atıyorum ve tekrar quıcksort fonksiyonu çağırarak pivotun sağında kalan yerleri sıralıyorum.

İşlemler bu şekilde recursive olarak gerçekleştiriliyor ve diziler kendi içlerinde sıralanıyor. Sonuçta ana dizimiz de sıralanmış oluyor.

(İşlemlerde karşılaştırma yaparken ABOVE-BELOW kullanmamızın sebebi dizide negatif elemanların da olması ihtimalidir.)

```
;----------------------------
        CALL QUICKSORT

        XOR CX,CX
        MOV CL,n
        XOR AX,AX
        XOR DI,DI

EL_YAZ: MOV AL,dizi[DI]
        CALL PUTN
        MOV AL,32
        CALL PUTC
        INC DI
        LOOP EL_YAZ


        RETF
BASLA   ENDP
```

Sıralama işlemi bittikten sonra diziyi ekrana yazdırmam lazım. Bunun için CX'imi sıfırlayıp CL'sine n atarak döngü sayımı belirliyorum. Ardından elemanları yazdırırken fonksiyonlar AX registeri üzerinden işlem yapacağı için AX'i sıfırlıyorum ve indis olarak da DI'yı kullanmak için DI'yı sıfırlıyorum.

EL_YAZ döngüsü ile elemanlarımı ekrana yazdıracağım. İlk elemandan başlayarak sırayla elemanları AL ye alıp putn ile ekrana yazdırıyorum. Sonra AL ye 32 atıp PUTC ile elemanlar arasına boşluk bırakıyorum. 32 yapmamın sebebi ascii tablosunda boşluk karakterinin 32 ile tutulması. Bu değeri de karakter olduğu için putc ile ekrana basıyorum.

Sonrasında sıradaki elemana geçmek için indisimi yani DI'yı artırıp tekrar başa dönüyorum ve sıradaki elemanı yazdıra yazdıra işlemleri tamamlıyorum. En sonda ENDP ile de BASLA prosedürümü kapatıyorum.

```
GETN    PROC NEAR
        PUSH BX
        PUSH CX
        PUSH DX
GETN_START:
        MOV DX,1
        XOR BX,BX
        XOR CX,CX
NEW:
        CALL GETC
        CMP AL,CR
        JE FIN_READ
        CMP AL, '-'
        JNE CTRL_NUM
NEGATIVE:
        MOV DX,-1
        JMP NEW
CTRL_NUM:
        CMP AL,'0'
        JB error
        CMP AL,'9'
        JA error
        SUB AL, '0'
        MOV BL,AL
        MOV AX,10
        PUSH DX
        MUL CX
        POP DX
        MOV CX,AX
        ADD CX,BX
        JMP NEW
error:
        MOV AX, OFFSET HATA
        CALL PUT_STR
        JMP GETN_START
FIN_READ:
        MOV AX,CX
        CMP DX,1
        JE FIN_GETN
        NEG AX
FIN_GETN:
        POP BX
        POP CX
        POP DX
        RET
GETN    ENDP
```

GETN

Getn fonksiyonu klavyeden girilen sayıyı okuyarak AX yazmacı üzerinde döndürür.

Klavyeden sayı girildikten sonra enter tuşlanırsa okumayı bitirir. Bu sırada sayı alırken girilen karakterlerin rakam olup olmadığını kontrol ederek eğer rakam girilmezse ekrana hata mesajı yazdırır. Ayrıca sayının negatif pozitif olup olmadığını anlamak için işarete bakar.

```
PUTN    PROC NEAR
        PUSH CX
        PUSH DX
        XOR DX,DX
        PUSH DX
        MOV CL,10
        CMP AL,0
        JGE CALC_DIGITS
        NEG AL
        PUSH AX
        MOV AL, '-'
        CALL PUTC
        POP AX
CALC_DIGITS:
        DIV CL
        ADD AH, '0'
        MOV DL,AH
        PUSH DX
        XOR AH,AH
        CMP AL,0
        JNE CALC_DIGITS
DISP_LOOP:
        POP AX
        CMP AL,0
        JE END_DISP_LOOP
        CALL PUTC
        JMP DISP_LOOP
END_DISP_LOOP:
        POP DX
        POP CX
        RET
PUTN    ENDP
```

PUTN

PUTN fonksiyonu AX registerinde tutulan sayıyı hane hane yazdırmak için kullanılır. Bu işlem yapılırken bölme işleminden faydalanılacağı için DX sonucu etkilemesin diye 0 yapılır. Ardından sayı negatif ise işareti ekrana basılır ve pozitif hali yazılır.

```
PUTC    PROC NEAR
        PUSH AX
        PUSH DX
        MOV DL,AL
        MOV AH,2
        INT 21H
        POP DX
        POP AX
        RET
PUTC    ENDP

GETC    PROC NEAR
        MOV AH,1h
        INT 21H
        RET
GETC    ENDP
;-----------------------------
PUT_STR PROC NEAR
        PUSH BX
        MOV BX,AX
        MOV AL,BYTE PTR [BX]
PUT_LOOP:
        CMP AL,0
        JE PUT_FIN
        CALL PUTC
        INC BX
        MOV AL,BYTE PTR [BX]
        JMP PUT_LOOP
PUT_FIN:
        POP BX
        RET
PUT_STR ENDP
```

PUTC, GETC VE PUT_STR

PUTC fonksiyonu AL yazmacındaki değeri ekranda göstermek için kullanılır. DL ve AH değişeceği için AX ve DX registerleri push pop ile korunur.

GETC fonksiyonu klavyeden girilen karakteri AL yazmacına alır ve işlemin sonucunda sadece AL etkilenir.

PUT_STR fonksiyonu ise AX'de adresi tutlan ve sonunda NULL yani 0 olan diziyi karakter karakter ekrana yazdırır. Bu işlem sırasında dizinin indisi olarak BX kullanılır. Bu yüzden BX'in değerini korumak için push pop yapılır.