YILDIZ TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM2021 ALT SEVİYE PROGRAMLAMA DERSİ

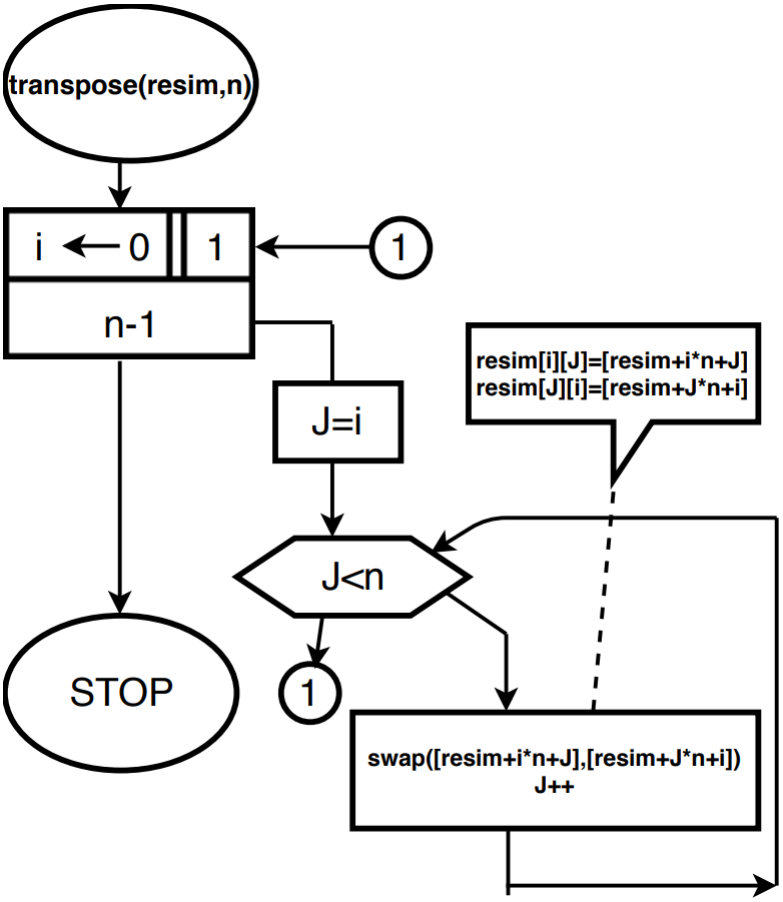
1.ÖDEV RAPORU

Hazırlayan: Mehmet Hayri Çakır - 16011023

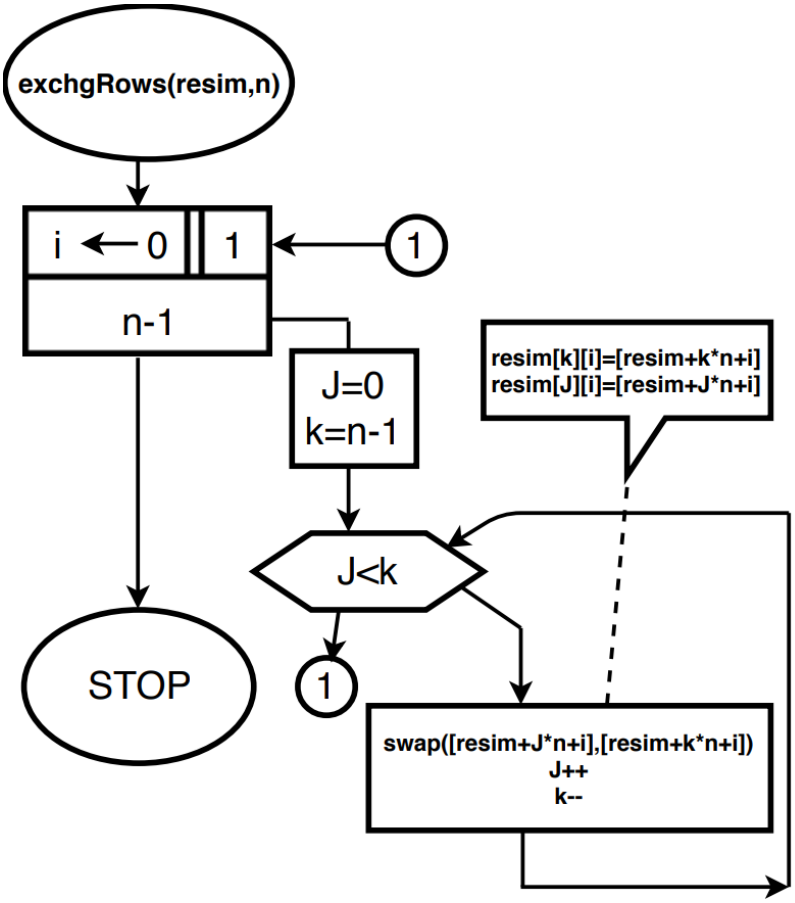
Sunulan/Dersin Yürütücüsü: Arş. Grv. Furkan Çakmak

İstanbul, 2018

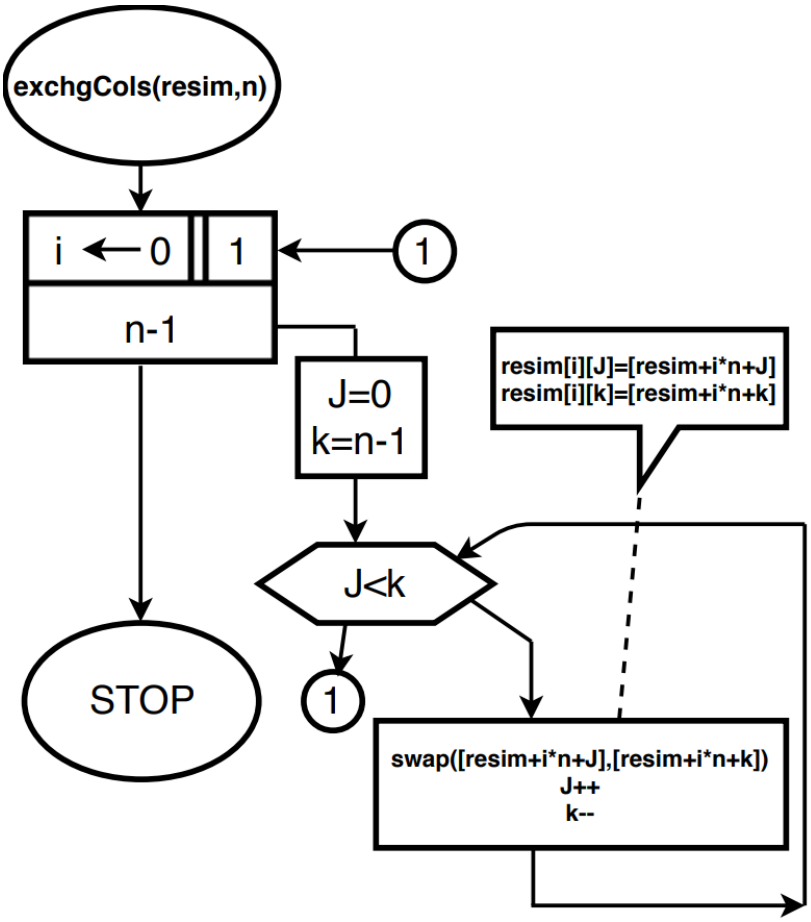
İÇİNDEKİLER



Akış-1(Transpoze)



Akış-2(SolaDöndür)



Akış-3(SağaDöndür)

1.1. Ön Açıklamalar

1.3, 1.4 ve 1.5. kısımlar ile ilgili bazı açıklamalar:

- Resim matrisi, aslında birbiri ardına gelen satırlar dizisi olarak tutulduğu için indis ile değil adres üzerinden eriştim. Örneğin; resim[i][j] yerine [resim+i*n+j] kullandım. Fakat anlatım kolaylığı olması açısından bazı yerlerde resim[i][j] gibi yazımlar da kullandım.
- Resim matrisi nxn boyutunda kare matris olduğu için n'den satır sayısı olarak bahsedilecektir.
- Akış-1'deki i için ESI, j için EDI,resim için EBX kullandım.
- Akış-2'deki (i+resim) için ESI kullandım. j için EDI, k için EBX kullandım. İndis hesaplarını bunlar üzerinden gerçekleştirdim. Matris elemanlarına erişmem gerektiğinde bu indisleri 2 ile çarptıktan(elemanlar Word boyutunda) sonra resmin başlangıç adresini bu indislere ekledim. Böylece gerekli adresi elde ettim ve istediğim elemana eriştim.
- Akış-3'teki (resim+i*n) için ESI, j için EDI, k için EBX kullandım.
- Bazı durumlarda ESI ve EDI yerine SI ve DI kullandım, o kısımlarda indisler en fazla n değerini alabileceği için ve n değeri de Word boyutunda olduğu için bunu yapmamda herhangi bir sakınca yok.

→Her iki sorunun kodunda da satır numaralandırmaları 16011023_main.cpp ve 16011023_soru2.asm dosyalarındaki satır numaralarına göre ayarlanmıştır.

1.2. Fonksiyonlar ve Neden Kullanıldıklarının Kısa Özeti

İlk sorunun çözümünde yazılan kodu 4 parçaya ayırabiliriz. Sorunun çözümü olan kodu inline assembly olarak yazmamız istendiği için, anlatım kolaylığı açısından 3 farklı fonksiyon kullanılmıştır gibi düşünebiliriz, bunlar **transpose**, **exchgRows** ve **exchgCols** fonksiyonlarıdır. Resmi sola döndürmek için **transpose** ve **exchgRows** fonksiyonları kullanılırken, resmi sağa döndürmek için ise **transpose** ve **exchgCols** fonksiyonları yazılan sırayla kullanılmalıdır. Örnek vermek gerekirse;

1 2 3	Transpose	1 4 7	exchgRows	3 6 9	
4 5 6	→	2 5 8	→	2 5 8	Sola döndürme tamamlandı.
7 8 9		3 6 9		1 4 7	

1.3. Transpose: Akışın ve Kodun Detaylı Açıklaması

```
83          XOR ECX,ECX             ;32-bitte çalıştığımız için 32 bitin döngü değişkeni olan ECX'i 0'lıyorum.
84          XOR ESI,ESI             ;ESI'yı i olarak kullanacağım, sıfırlıyorum
85          MOV EBX,resim           ;Register işlemleri DADDR'den daha hızlı olduğu için resim'i EBX'e aldım.--
                                    ;106 ve 107.satırlarda kullanacağım.
86          MOV CX,n                ;CX'i dönüş sayısı olarak kullanacağım, satır sayısını CX'e atıyorum.
87    L1:   MOV EDI,ESI             ;Akıştaki for döngüsü başlangıcı, EDI'yı j olarak kullanacağım, i'yi atıyor
88          MOV AX,n                ;While içinde i değişmediği için i*n ifadesini while içinde değil şimdiden
                                    ;hesaplayacağım ki while içinde her seferinde işlem yapılmasın.
89          CWDE                    ;32 bitlik çarpma için AX'i EAX'e genişletiyorum.
90          MUL ESI                 ;EAX*ESI  → Sonuç EDX:EAX'te.     (i*n)
91          MOV EDX,EAX             ;i*n'i EDX içinde saklayacağım. AX'i başka işlemler için kullanacağım.
92    W1:   CMP DI,n                ;For yerine while kullandığım için DI,n 'den küçükse dönmeye devam edecek.
93          JNB L1_INC              ;Değilse L1_INC label'ına atlayacak,ESI'yı artırıp for döngüsü (L1) dönecek
94          CMP SI,DI               ;SI DI'ya eşitse köşegen üzerindeyim demektir. Matriste köşegenin transpozu
95          JE DIAGNL               ;kendisine eşit olduğu için transpozunu almak mantıksız.DIAGNL labelinde
                                    ;EDI'yı artırıp while döngüsüne (W1) zıplayacak.
96          PUSH ESI                ;Swap işlemlerini yaparken i ve j değerlerini kaybetmemek için
97          PUSH EDI                ;ESI ve EDI'yı yığına atıyorum.
98          MOV AX,n                ;j*n ifadesini hesaplamak için AX'e eleman sayısı olan n'i alıyorum.
99          CWDE                    ;AX'i 32 bit çarpma işlemine tabii tutacağım için EAX'e genişletiyorum.
100         PUSH EDX                ;Çarpma işlemi sonucunda EDX'teki değeri kaybetmemek için yığına atıyorum.
101         MUL EDI                 ;EDI*EAX  → Sonuç EDX:EAX'te.     (j*n)
102         POP EDX                 ;EDX'i yığından çekiyorum.
103         ADD ESI,EAX             ;Resim matrisine ESI ve EDI ile erişeceğim için ESI=(j*n+i) yapıyorum.
104         ADD EDI,EDX             ;Aynı sebepten EDI'ya (i*n) ekliyorum → EDI=(i*n+j).
105         SHL ESI,1               ;Matris elemanları Word tipinde olduğu için kullandığım indisleri--
106         SHL EDI,1               ;2 ile çarpıyorum.
107         ADD ESI,EBX             ;Son olarak ESI ve EDI'ya resim matrisinin başlangıç adresini ekliyorum--
108         ADD EDI,EBX             ;böylece [resim+i*n+j] ve [resim+j*n+i] ifadelerini hazırlamış oldum.
109         MOV AX,WORD PTR[ESI]    ;AX aracılığıyla resim[i][j] ile resim[j][i]'yi yer değiştiriyorum. Yani
110         XCHG AX,WORD PTR[EDI]   ;Akış-1'deki while döngüsünün içindeki işlemi gerçekleştiriyorum burada.
111         MOV WORD PTR[ESI],AX    ;swap([resim+i*n+j],[resim+j*n+i]).
                                    ; XCHG DADDR,DADDR komutu mevcut olmadığı için AX aracılığıyla yaptım.
112         POP EDI                 ;İşlemlerimi tamamladım, artık güvenle yığından j değerini çekebilirim.
113         POP ESI                 ;Yığından i değerini çekiyorum.
114   DIAGNL:INC EDI                ;j indisini artırıyorum. Burası aynı zamanda köşegen matris üzerindeysem
                                    ;zıplayacağım label.
115         JMP W1                  ;While döngüsüne zıplıyorum ve j<n olduğu sürece W1 ile bu satır arasındaki
                                    ;işlemler tekrarlanacak.
116   L1_INC:INC ESI                ;j>=n olursa bu labela zıplayıp i yani ESI'yı artıracak.
117         LOOP L1                 ;For döngüsü n yani satır veya sütun sayısı kadar tekrarlanacak.
```

1.4. exchgRows: Akışın ve Kodun Detaylı Açıklaması

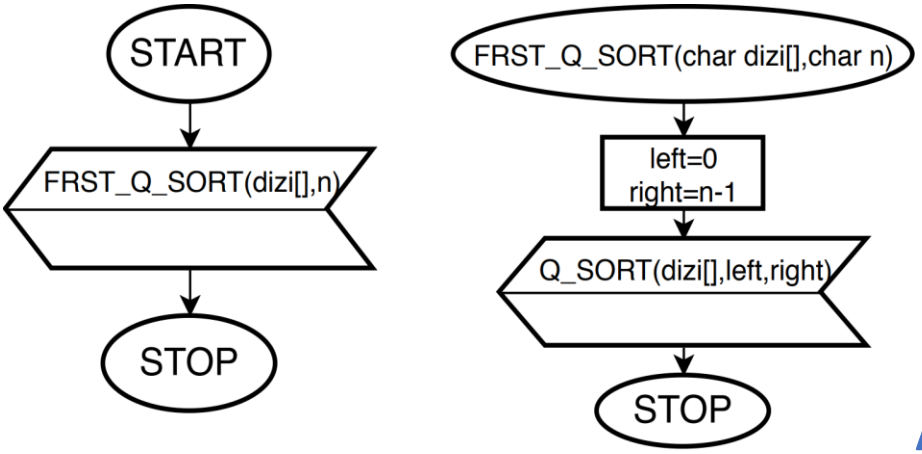
İ başlangıçta 0,j 0. satır i. elemandan, k (n-1). satır i. elemandan başlayacak yerleri değiştirilecek. Sonrasında j 1. satır i. elemanı, k (n-2).satır i. elemanı gösterecek ve onların yeri değiştirilecek. 0. sütun bu şekilde tamamlandıktan sonra i artırılıp 1.sütuna geçilecek ve tüm matrise bu işlem uygulanarak en sonunda 0. satırla (n-1). satır, 1.satırla (n-2).satır … (n/2).satırla (n/2+1).satır yer değiştirmiş olacak ve işlem tamamlanacak.

```
193         MOV CX,n                ;Önceden transpose kodunda for döngüsü bitince CX zaten 0'lanmış oluyor—
                                    ;Bu yüzden ECX'i tekrar XOR'lamama gerek yok. Satır sayısını CX'e atıyorum.
194         MOV ESI,resim           ;[resim+i]'yi ESI'da tutacağım çünkü işlemlerde registerlar DADDR'den--
                                    ;hızlı çalışır. 210,211 ve 220.satırlarda kullanacağım.
195 L2:     XOR EBX,EBX             ;k olarak kullanacağım indisin soldaki 16 bitinin içinde değer kalmasın.
196         XOR EDI,EDI             ;j olarak kullanacağım indisi sıfırlıyorum. For döngüsü başlangıcı.
197         MOV BX,n                ;Akış-2'deki gibi k'yı (n-1)'den başlatacağım. İşlem sonucunda BX=n
198         DEC EBX                 ;BX=n-1
199 W2:     CMP DI,BX               ;j<k olduğu sürece devam etmesini istiyorum. j=k olmuşsa Tüm satırların--
                                    ;yeri değiştirilmiş demektir. Ortadaki satırın yerini tekrar kendisiyle--
200         JNB L2_INC              ;değiştirmeme gerek yok, i'yi artırmak için L2_INC labelına atlıyorum.
201         PUSH EDI                ;j ve k ile işlemler yapacağım, değerlerini kaybetmemek için yığına--
202         PUSH EBX                ;atıyorum.
203         MOV AX,n                ;j*n işlemini gerçekleştirmek için gerekli hazırlıkları yapıyorum. AX'e--
204         CWDE                    ;satır sayısını alıp, EAX'e genişletiyorum.
205         MUL EDI                 ;j*n → Sonucu EAX'te.
206         MOV EDI,EAX             ;j*n işlemin sonucunu EDI'ya alıyorum, resme erişirken EDI'yı kullanacağım.
207         MOV AX,n                ;k*n işlemini gerçekleştirmek için gerekli hazırlıklar. j ve k'nın--
208         CWDE                    ;değerleri while içinde değiştiği için iki çarpma işlemini de while içinde-
209         MUL EBX                 ;yapıyorum.(Transpose kodunda birini while döngüsüne girmeden yapmıştım.)
210         MOV EBX,EAX             ;EBX ile de resme erişeceğim için k*n sonucunu EBX'e alıyorum.
211         SHL EDI,1               ;2 satır sonra resmin başlangıç adresini ekleyeceğim EDI ve EBX'teki--
212         SHL EBX,1               ;indisleri 2 ile çarpıyorum çünkü resim matrisi elemanlarım Word tipinde.
213         ADD EDI,ESI             ;Resme erişmeden önce son olarak başlangıç adresinden ne kadar ilerideki --
214         ADD EBX,ESI             ;elemana erişeceğimi söyleyen EDI ve EBX'e (resim+i)'yi ekliyorum.
215         MOV AX,WORD PTR[EBX]    ;AX aracılığıyla resim[j][i] ile resim[k][i]'nin yerlerini değiştiriyorum.
216         XCHG AX,WORD PTR[EDI]   ;XCHG DADDR,DADDR komutu mevcut olmadığı için [EBX]'i AX'e alıp [EDI] ile-
217         MOV WORD PTR[EBX],AX    ;AX'i yer değiştiriyorum. Sonrasında [EBX]'e AX'i([EDI])'yı atıyorum.
218         POP EBX                 ;Şimdilik EBX ve EDI ile işim bittiği için yığından eski değerleri olan--
219         POP EDI                 ;j ve k'yı yığından çekiyorum.
220         INC EDI                 ;j 0'dan, k ise n-1'den başlayıp gittikçe birbirlerine yaklaşacaklar.
221         DEC EBX                 ;Bu yüzden j'yi artırıp, k'yı azaltıyorum.
222         JMP W2                  ;While döngüsüne zıplıyorum. j<k olduğu sürece 196 ve 219.satırlar arası--
223 L2_INC:ADD ESI,2                ;tekrarlanacak. j>=k olursa bu label'a zıplanıp (resim+i) 2 artırılacak.
224         LOOP L2                 ;Çünkü for döngüsünde i artırılmalı fakat Word tipinde bir matris olduğu--
                                    ;için 2 artırıyorum. Sonrasında for döngüsüne(192.satırda L1) zıplıyorum.
```

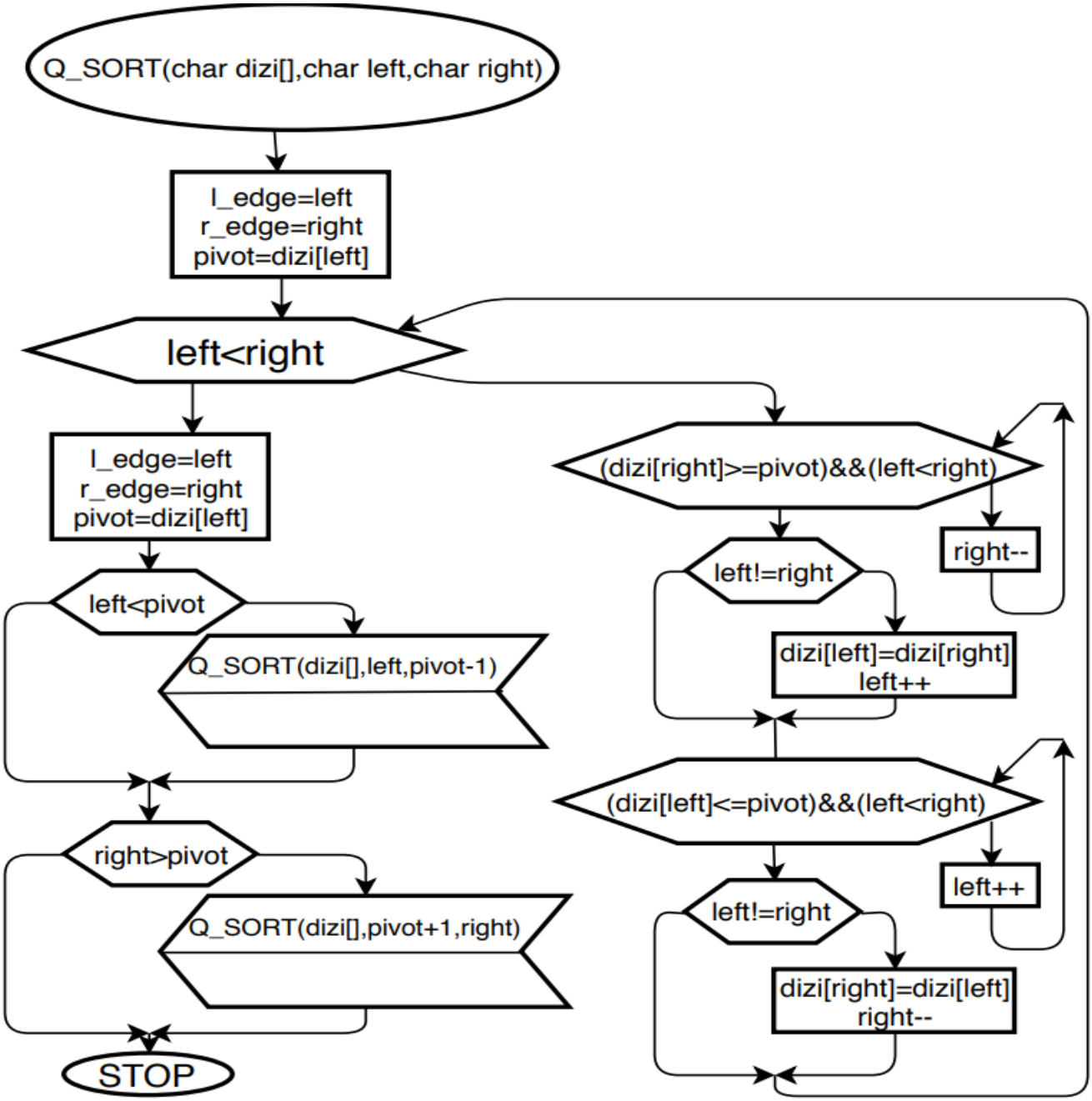
1.5. exchgCols: Akışın ve Kodun Detaylı Açıklaması

i başlangıçta 0, j i. satır 0. elemandan, k i. satır (n-1). elemandan başlayarak yerleri değiştirilecek. Sonrasında j i. satır 1. elemanı, k i. satır 1. elemanı gösterecek ve onların yeri değiştirilecek. 0. satır bu şekilde tamamlandıktan sonra i artırılıp 1.satıra geçilecek ve tüm matrise bu işlem uygulanarak en sonunda 0. sütunla (n-1). sütun, 1.sütunla (n-2).sütun … (n/2).sütun (n/2+1).sütun yer değiştirmiş olacak ve işlem tamamlanacak.

```
120         MOV CX,n                ;Önceden transpose kodunda for döngüsü bitince CX zaten 0'lanmış oluyor—
                                    ;Bu yüzden ECX'i tekrar XOR'lamama gerek yok. Satır sayısını CX'e atıyorum.
121         MOV ESI,resim           ;[resim+i*n]'i ESI'da tutacağım çünkü işlemlerde registerlar DADDR'den--
                                    ;hızlı çalışır. 132 ve 133 satırlarda kullanacağım.(i*n)'i 145.satırda-
                                    ;ekliyorum. Çünkü başlangıçta i=0.
122 L3:     XOR EBX,EBX             ;k olarak kullanacağım indisin soldaki 16 bitinin içinde değer kalmasın.
123         XOR EDI,EDI             ;j olarak kullanacağım indisi sıfırlıyorum.
124         MOV BX,n                ;Akış-3'teki gibi k'yı (n-1)'den başlatacağım. İşlem sonucunda BX=n
125         DEC EBX                 ;BX=n-1
126 W3:     CMP DI,BX               ;j<k olduğu sürece devam etmesini istiyorum. j=k olmuşsa Tüm sütunların--
                                    ;yeri değiştirilmiş demektir. Ortadaki sütunun yerini tekrar kendisiyle--
127         JNB L3_INC              ;değiştirmiyorum, (resim+i*n)'i güncellemek için L3_INC labelına atlıyorum.
128         PUSH EDI                ;j ve k ile işlemler yapacağım, değerlerini kaybetmemek için yığına--
129         PUSH EBX                ;atıyorum.
130         SHL EDI,1               ;2 satır sonra resmin başlangıç adresini ekleyeceğim EDI ve EBX'teki--
131         SHL EBX,1               ;indisleri 2 ile çarpıyorum çünkü resim matrisi elemanlarım Word tipinde.
132         ADD EDI,ESI             ;Resme erişmeden önce son olarak başlangıç adresinden ne kadar ilerideki --
133         ADD EBX,ESI             ;elemana erişeceğimi söyleyen EDI ve EBX'e (resim+i*n)'i ekliyorum.
134         MOV AX,WORD PTR[EBX]    ;AX aracılığıyla resim[i][j] ile resim[i][k]'nın yerlerini değiştiriyorum.
135         XCHG AX,WORD PTR[EDI]   ;XCHG DADDR,DADDR komutu mevcut olmadığı için [EBX]'i AX'e alıp [EDI] ile-
136         MOV WORD PTR[EBX],AX    ;AX'i yer değiştiriyorum. Sonrasında [EBX]'e AX'i([EDI])'yı atıyorum.
137         POP EBX                 ;Şimdilik EBX ve EDI ile işim bittiği için yığından eski değerleri olan--
138         POP EDI                 ;j ve k'yı yığından çekiyorum.
139         INC EDI                 ;j 0'dan, k ise n-1'den başlayıp gittikçe birbirlerine yaklaşacaklar.
140         DEC EBX                 ;Bu yüzden j'yi artırıp, k'yı azaltıyorum.
141         JMP W3                  ;While döngüsüne zıplıyorum. j<k olduğu sürece 126. ve 141.satırlar arası--
                                    ;tekrarlanacak.
142 L3_INC:MOV AX,n                 ;(resim+i*n) ifadesi her for(L3) çevriminde i*n*2(Word tipinde olduğu için)
143         CWDE                    ;kadar artmalıdır. Bunu da AX'e n'i alıp, 1 kez sola shift ederek--
144         SHL AX,1                ;gerçekleştirdim. Sonrasında (resim+i*n)'i tutan ESI'ya EAX'i ekledim.
145         ADD ESI,EAX             ;Bu sayede yavaş çalışan MUL komutunu kullanmadan işimi halletmiş oldum.
146         LOOP L3                 ;For döngüsüne zıplıyorum.
```


START
FRST_Q_SORT(dizi[],n)
STOP
FRST_Q_SORT(char dizi[],char n)
left=0
right=n-1
Q_SORT(dizi[],left,right)
STOP

Akış-4


Q_SORT(char dizi[],char left,char right)
l_edge=left
r_edge=right
pivot=dizi[left]
left<right
l_edge=left
r_edge=right
pivot=dizi[left]
left<pivot
Q_SORT(dizi[],left,pivot-1)
right>pivot
Q_SORT(dizi[],pivot+1,right)
STOP
(dizi[right]>=pivot)&&(left<right)
right--
left!=right
dizi[left]=dizi[right]
left++
(dizi[left]<=pivot)&&(left<right)
left++
left!=right
dizi[right]=dizi[left]
right--

Akış-5

2.1. Yordamlar ve Neden Kullanıldıklarının Özeti

2. Sorunun çözümünde toplam 10 adet yordam kullanılmıştır. Bunlar; **MAIN, FRST_Q_SORT, Q_SORT, READ_ARR, PRINT_ARR, GETC, PUTC, GETN, PUTN, PUT_STR**'dir. Programın kodu DOSBox'ta derlenmiş ve link edilmiştir. DOSBox içinden çalıştırılmalıdır.

2.2. Stack Segment ve Data Segment

```
1   sseg        SEGMENT PARA STACK 'yigin'  ;Stack segmentimin boyutunu ayarlıyorum.
2               DW 20 DUP(0)                ;20 Word boyutunda bir alan maksimum 100 elemanlı bir dizi--
3   sseg        ENDS                        ;için yeterli olacaktır. Stack segmentimi sonlandırıyorum.
4                                           ;
5   dseg        SEGMENT PARA 'veri'         ;Data segmentimi açıyorum.
6   count       DB 1                        ;Ekran mesajları için kullanacağım.(count. Elemanı girin:)
7   dizi        DB 100 DUP(0)               ;100 elemanlı diziyi oluşturup içine 0 dolduruyorum.
8   CR          EQU 13                      ;Carriage Return ve Line Feed; yeni satıra geçmek ve ekrandaki
9   LF          EQU 10                      ;yazıların düzenli gözükmesi için gereklidir.(\n)
10  HT          EQU 9                       ;Horizontal Tab; Diziyi ekrana basarken kullanılacaktır.
11  mesaj1      DB 'Dizi boyutunu veriniz: ',0 ;READ_ARR yordamında kullanılacaktır.
12  mesaj2      DB '.elemani giriniz: ',0   ;READ_ARR yordamında diziyi okumak için kullanılacaktır.
13  hata1       DB CR,LF,'Dikkat! Sayi girmediniz!!',CR,LF,0 ;Girilen karakterin rakam olmaması durumunda
                                            ;ekrana basılacak uyarıdır.(READ_ARR'da)
14  hata2       DB 'Dikkat! Sayi -128 den kucuk veya 127 den buyuk olamaz!!',CR,LF,0 ;Girilen sayının--
                                            ;-128'den küçük veya 127'den büyük olması durumunda--
                                            ;kullanıcıyı uyarır.
15  given_arr   DB CR,LF,'Girilen Dizi:',HT,0 ;READ_ARR ile okunan dizi PRINT_ARR ile ekrana basılır.
16  sorted_arr  DB CR,LF,'Siralanmis Dizi:',0 ;Q_SORT ile sıralanan dizi PRINT_ARR ile ekrana basılır.
17  space       DB ' ',0                    ;Dizi elemanları yazdırılırken ekrana boşluk veya yatay tab--
                                            ;karakteri basılabilir.
18  dseg        ENDS                        ;Data segmentimi sonlandırıyorum.
```

2.3. MAIN Yordamı ve Açıklaması

```
20  cseg    SEGMENT PARA 'kod'              ;Code segmentimi açıyorum.
21          ASSUME DS:dseg, SS:sseg, CS:cseg ;Data, Code ve Stack Segment için hangi segmentleri---
22  MAIN    PROC FAR                        ;kullanacağımı söylüyorum ve MAIN yordamını başlatıyorum.
23                                          ;
24          PUSH DS                         ;DS'ye kendi data segmentim olan dseg'i atacağım, değerini--
25          XOR AX,AX                       ;kaybetmemek için AX'le beraber yığına atıyorum. Sonrasında--
26          PUSH AX                         ;AX aracılığıyla DS'ye kendi dseg'imi atıyorum.
27          MOV AX,dseg                     ;Bu 5 satır kod EXE tipi programlarda standart olarak --
28          MOV DS,AX                       ;yazılması gereken koddur.
29                                          ;
30          CALL READ_ARR                   ;Klavyeden eleman sayısını ve diziyi okuyacağım yordamı--
31          MOV AX,OFFSET given_arr         ;çağırdım. AX'i kullanarak ekrana 16.satırdaki given_arr'da
32          CALL PUT_STR                    ;yer alan('Girilen dizi: ') ifadesini PUT_STR ile yazdırdım.
33          CALL PRINT_ARR                  ;Sonrasında girilen diziyi ekrana yazdırdım.
34          CALL FRST_Q_SORT                ;Girilen diziyi sıralamak üzere FRST_Q_SORT yordamını çağırdım
35          MOV AX,OFFSET sorted_arr        ;AX'i kullanarak ekrana 17.satırdaki sorted_arr'da yer alan--
36          CALL PUT_STR                    ;('Siralanmis dizi: ') ifadesini PUT_STR ile yazdırdım.
37          CALL PRINT_ARR                  ;FRST_Q_SORT ve Q_SORT yordamlarıyla sıralanmış olan diziyi--
38                                          ;ekrana yazdırdım.
39          RETF                            ;
40  MAIN    ENDP                            ;MAIN yordamını sonlandırıyorum.
```

2.4. READ_ARR Yordamı ve Açıklaması

```
READ_ARR      PROC NEAR              ;Yordamın başlangıcı.
              MOV AX,OFFSET mesaj1   ;mesaj1'in offsetini AX'e alıp, PUT_STR yordamı ile ekrana–
              CALL PUT_STR           ; 'Dizi boyutunu veriniz: ' metnini yazdırıyorum.
              CALL GETN              ;GETN yordamı ile dizi boyutunu AL register'ına alıyorum.
              CBW                    ;AH'da değer varsa 0 yapmak için AL'yi AX'e genişletiyorum.
              MOV CX,AX              ;CX'te eleman sayısını tutacağım. Yordamların başında PUSH--
              PUSH CX                ;bitiminde POP yaparak değerini koruyacağım.
              XOR DI,DI              ;DI'yı dizi indisi olarak kullanacağım için sıfırlıyorum.
ARR_LOOP:     MOV AL,count           ;'count. elemanı giriniz: ' metnini ekrana basmak için AL'ye count'u
              CBW                    ;alıyorum. Count byte tipinde olduğu için AL'yi AX'e genişlettim.
              CALL PUTN              ;PUTN yordamı ile count'u ekrana yazdırıyorum.
              MOV AX,OFFSET mesaj2   ;Sonrasında AX'e mesaj2'nin offsetini alıp, PUT_STR yordamı ile--
              CALL PUT_STR           ;'.elemanı giriniz: ' stringini ekrana yazdırıyorum. Sonrasında--
              CALL GETN              ;dizinin ilgili elemanını klavyeden okuyorum. -128 ve 127 aralığında
              CMP AX,-128            ;olup olmadığının kontrolünü yapıyorum. Eğer -128'den küçük veya--
              JL ERROR2              ;127'den büyük bir sayı girilmişse; ERROR2 labeline atlıyorum. hata2
              CMP AX,127             ;içinde yer alan hata mesajını ekrana yazdırıyorum.
              JG ERROR2              ; 'Dikkat! Sayi -128 den kucuk veya 127 den buyuk olamaz!!'
              INC count              ;Sayı geçerli aralıktaysa count'u artırıyorum ve dizinin ilgili--
              MOV dizi[DI],AL        ;yerine AL'deki değeri yazıyorum. Çünkü dizi byte tanımlı.
              INC DI                 ;dizi indisini artırıyorum.
              LOOP ARR_LOOP          ;For döngüsünü dönüyorum. Eleman sayısı kadar(CX) dönecek.
              JMP FIN_READ_ARR       ;Okumayı bitirdiysem FIN_READ_ARR labelına atlıyorum.
ERROR2:       MOV AX,OFFSET hata2    ;Girilen sayı istenen aralıkta değilse [-128,127] ekrana hata mesajı
              CALL PUT_STR           ;yazdırılacak.
              JMP ARR_LOOP           ;CX azalmasın diye de LOOP yerine JMP komutu kullanıyorum, çünkü--
FIN_READ_ARR: POP CX                 ;hatalı giriş yapıldı, tekrar giriş yapılmalı.  CX'e yığından eleman
              RET                    ;sayısını çekiyorum ve RETURN yaptıktan sonra
READ_ARR      ENDP                   ;READ_ARR yordamını sonlandırıyorum.
```

2.5. PRINT_ARR Yordamı ve Açıklaması

```

PRINT_ARR     PROC NEAR              ;PRINT_ARR yordamının başlangıcı.
              PUSH CX                ;CX'teki eleman sayısı değerini kaybetmemek için yığına atıyorum.
              XOR DI,DI              ;dizi indisi olarak kullanacağım DI'yı sıfırlıyorum.
PUT_ARR:      MOV AL,dizi[DI]        ;AL'ye dizinin o anki elemanını atıyorum.
              CBW                    ;AH'da değer kalmasın diye AL'yi AX'e genişletiyorum.
              CALL PUTN              ;AX'teki değeri PUTN yordamı ile ekrana yazdırıyorum.
              MOV AX,OFFSET HT       ;Kolay okunması ve hizalanması açısından elemanlar arasına yatay tab
              CALL PUTC              ;karakteri koyuyorum. (PUTC yordamını kullanarak.)
              INC DI                 ;Sonraki elemana erişmek için dizi indisimi artırıyorum.
              LOOP PUT_ARR           ;Eleman sayısı kadar for döngüsü kullanarak diziyi yazdırıyorum.
              POP CX                 ;Yığından eleman sayısını CX'e çekiyorum.
              RET                    ;RETURN ve sonrasında da ENDP ile PRINT_ARR yordamını sonlandırdım.
PRINT_ARR     ENDP                   ;Artık ekrana dizi yazdırmam gerekirse CALL PRINT_ARR yazabilirim.
```

2.6. FRST_Q_SORT Yordamı ve Açıklaması

Bu yordamı, MAIN yordamının içinde kod kalabalığı oluşturmamak için kullandım. İşlevi, SI'ya ilk değer olarak 0; DI'ya ise CX-1 atamak ve sonrasında sıralama yordamı olan Q_SORT'u çağırmaktır. Akış-4'teki left için SI, right için DI registerlarını kullandım.

```
42  FRST_Q_SORT         PROC NEAR       ;FRST_Q_SORT yordamının başlangıcı.
43                      XOR SI,SI       ;left parametresi olarak kullanacağım SI register'ını sıfırlıyorum.
44                      MOV DI,CX       ;right parametresi olarak kullanacağım DI register'ına CX'i aldım.
45                      DEC DI          ;Sonra bir azalttım. (CX'in içinde eleman sayısı tutuluyor.)
46                      CALL Q_SORT     ;İlk parametreleri ayarladım, diziyi sıralayacak olan Q_SORT ---
47                      RET             ;yordamını çağırıyorum. RET ve sonrasında da ENDP komutlarıyla --
48  FRST_Q_SORT         ENDP            ;yordamı sonlandırıyorum.
```

2.7. Q_SORT Yordamı ve Açıklaması

Akış-5'i inceleyelim:

- Kodun açıklamalarında yer alan l_edge, r_edge gibi ifadeler Akış-5'e göre yazılmıştır. Bu değişkenler için mevcut registerlar kullanılmıştır. Ek değişken tanımlanmamıştır. Değişkenler için kullanılan registerlar:
  - l_edge=DX
  - r_edge=BX
  - left=SI
  - right=DI
  - pivot=AL
- l_edge, left'ten; r_edge, right'tan başlayacak.
- pivot, bizim referans alacağımız eleman. dizi[left] olarak ayarlıyoruz. Yani quick sort algoritmasını ilk elemanı pivot seçecek şekilde yazdım. (right pivot, middle pivot da yaygın olarak kullanılıyor, hatta sıralanmış diziler için daha optimize çalışıyor fakat ilk elemanı pivot alarak yapmayı bildiğim için bunu kullandım.)
- l_edge ve r_edge, gittikçe birbirine yaklaşacak. Onların arasında kalan elemanlar üzerinden işlem yapacağız, dışlarında kalan elemanlar sıralanmış oluyor.
- pivot'u ilk elemandan başlatıp, son elemanla kıyaslıyoruz. Eğer pivot büyük değilse, son elemanı işaret eden right'ı azaltıp bir önceki elemanla kıyaslıyoruz. Ta ki, pivottan küçük bir eleman bulana kadar, bulduysak ve left ve right aynı elemana işaret etmiyorsa, pivota yani dizi[left]'e küçük olan elemanı(dizi[right]) atıyoruz. left=right olmuşsa daha küçük eleman bulamadık ve aradaki makas kapanmış demektir. LOOP1_END label'ına atlamam gerekir. left<right olduğu sürece bu işlemleri tekrarlıyorum ve left>=right olursa LOOP1_END label'ına gidiyorum. pivottaki değeri kaybetmeden dizi[left]'e veriyorum.

```
50  Q_SORT          PROC NEAR           ;Q_SORT yordamının başlangıcı.
51                  PUSH CX             ;CX'in değerini(eleman sayısı) kaybetmemek için yığına atıyorum.
52                  MOV DX,SI           ;l_edge=left işlemini gerçekleştiriyorum. DX=l_edge, SI=left
53                  MOV BX,DI           ;r_edge=right işlemini gerçekleştiriyorum. BX=r_edge, DI=right
54                  MOV AL,dizi[SI]     ;pivot=dizi[left] işlemi. AL=pivot
55  LOOP1:          CMP SI,DI           ;left<right mı diye kontrol ediyorum, değilse LOOP1_END label'ına--
56                  JNB LOOP1_END       ;atlıyorum. İşlemleri yapıp Q_SORT'u recursive olarak çağıracağım.
57  LOOP2:          CMP dizi[DI],AL     ;dizi[right]>=pivot değilse LOOP2_END1'e atlayıp left!=right mı diye
58                  JL LOOP2_END1       ;bakacağım. False ise sonraki while döngüsüne bakacağım.
59                  CMP SI,DI           ;left<right mı diye bakıyorum, değilse left!=right mı diye bakmadan-
60                  JNB LOOP2_END2      ;direkt olarak sonraki while döngüsüne bakacağım.
61                  DEC DI              ;Her iki şart da sağlandıysa, right'ı azaltıyorum.
62                  JMP LOOP2           ;55 veya 57. Satırdaki şartlardan biri sağlanmayana kadar dönecek.
63  LOOP2_END1:     CMP SI,DI           ;dizi[right]<pivot ise buraya atlıyorum. left!=right mı diye sordum.
```

```
                JE LOOP2_END2       ;Cevap hayırsa, dizi[left]'ten küçük bir dizi[right] bulamadım,--
                                    ;LOOP2_END2 label'ına atlayıp diğer while döngüsüne bakacağım.
                                    ;Cevap evetse, dizi[left]'ten küçük bir dizi[right] buldum demektir.
                MOV CL,dizi[DI]     ;CL aracılığıyla dizi[left]'e dizi[right]'ı atıyorum. dizi[left]'in-
                MOV dizi[SI],CL     ;değeri güvende çünkü pivot'un içinde tutuluyor.
                INC SI              ;dizi[left]'i geçip, sonraki elemana bakmak için left'i artırıyorum.
LOOP2_END2:     CMP dizi[SI],AL     ;dizi[left]<=pivot mu diye bakıyorum. Değilse LOOP3_END1'teki--
                JG LOOP3_END1       ;if kontrolünü yapacağım.
                CMP SI,DI           ;Üstteki kontrol sağlandıysa left<right mı diyorum. Cevap hayırsa--
                JNB LOOP3_END2      ;74. Satırdaki kontrole gerek kalmadan LOOP3_END2'ye gidip oradan da
                                    ;55. Satırdaki en dıştaki while döngüsüne zıplıyorum.
                INC SI              ;Üstteki iki şart sağlandıysa left'i artırıyorum. Aslında buradaki--
                                    ;amacım, pivottan büyük bir dizi[left] bularak dizi[right]'ın eski--
                                    ;yerine dizi[left]'i koymak. Tüm bu işlemler bittiğindeyse en son--
                                    ;boş kalan yere pivottaki sayıyı yerleştirmek ve dizi tamamen--
                                    ;sıralanana kadar recursive olarak Q_SORT çağırmak.
                JMP LOOP2_END2      ;68 ve 70. Satırdaki şartlardan biri bozulana kadar dönecek.
LOOP3_END1:     CMP SI,DI           ;68.satırdaki şart bozulursa buraya zıplayacak. left!=right kontrolü
                JE LOOP3_END2       ;yapılacak. False ise LOOP3_END2 üzerinden LOOP1'e zıplayacak.
                                    ;True ise pivottan büyük bir dizi[left] buldum demektir ve artık--
                MOV CL,dizi[SI]     ;CL yardımıyla dizi[right]'a dizi[left]'i atayabilirim.right'ı da --
                MOV dizi[DI],CL     ;azaltıyorum çünkü oraya atama yaptım ve şimdilik işim bitti, bir--
                DEC DI              ;önceki elemana bakacağım. left ve right arası gittikçe daralıyor.
LOOP3_END2:     JMP LOOP1           ;70 veya 74. Satırdaki şartlardan biri false olursa buraya zıplıyor.
                                    ;Buradan ise en dıştaki while döngüsüne zıplıyorum. left<right ise--
                                    ;dönmeye devam edecek LOOP1.
LOOP1_END:      MOV dizi[SI],AL     ;left<right şartı bozulursa bu label'a atlıyorum ve dizi[left]'teki-
                                    ;boşluğa pivottaki elemanı atıyorum.
                MOV AX,SI           ;pivot'a ise kaldığım son noktayı, yani left'i atıyorum.
                MOV SI,DX           ;left'e l_edge'i atıyorum, right'a r_edge'i atıyorum. Eğer dizi --
                MOV DI,BX           ;sıralandıysa zaten l_edge'in içinde left, r_edge'in içinde right--
                                    ;olacağı için bir şey değişmeyecek.
                CMP SI,AX           ;left<pivot mu diye bakıyorum. True ise sol kısımda sıralama--
                JNB SECOND_IF       ;bitmemiş demektir. False ise right>pivot doğru mu diye bakacağım.
                PUSH SI             ;left<pivot true oldu, right'ı pivot-1 yapıp Q_SORT'u çağıracağım--
                PUSH DI             ;ama çağırdığım Q_SORT bittiğinde left ve right'ın eski değerlerini-
                MOV DI,AX           ;kaybetmemek için yığına atıyorum. right=pivot yaptım.
                DEC DI              ;right'ı azaltıp right=pivot-1 parametresini elde ettim.
                CALL Q_SORT         ;left'i aynı değeriyle yollayacağım. Q_SORT'u çağırıyorum.
                POP DI              ;Çağırdığım recursive Q_SORT bitti, left ve right'ı yığından çektim.
                POP SI              ;
SECOND_IF:      CMP DI,AX           ;84. Satırdaki if kontrolü false olduysa buraya zıplıyor, true--
                                    ;olduysa da 90.satırda çağrılan recursive Q_SORT tamamlanınca bu--
                                    ;kontrol yapılacak. right<=pivot ise sağ kısımda sıralama bitmiş--
                JNA ENDING          ;demektir. Hem solda hem sağda sıralama bitmiş oldu. Böylece dizi--
                                    ;sıralı hale geldi ve ENDING label'ına zıplıyorum.
                PUSH SI             ;right>pivot ise bu kısma geliyorum. Q_SORT çağıracağım. Değerleri--
                PUSH DI             ;kaybetmemek için left ve right'ı yığına atıyorum.
                MOV SI,AX           ;left parametresini pivot+1 olarak ayarlıyorum.
                INC SI              ;
                CALL Q_SORT         ;Q_SORT'u çağırıyorum. Çağırdığım Q_SORT sonlanınca bir alt satırdan
                POP DI              ;devam edecek ve left'le right'ı yığından çekecek.
                POP SI              ;
ENDING:         POP CX              ;Yordamın başında yığına attığım eleman sayısını CX'e çekiyorum.
                RET                 ;ENDING label'ına geldiysek, dizi sıralandı demektir. RET ve ENDP--
Q_SORT          ENDP                ;komutlarıyla yordamı sonlandırıyorum.
```

2.8. GETC Yordamı ve Açıklaması

Klavyeden basılan karakteri AL yazmacına alır ve ekranda gösterir. İşlem sonucunda sadece AL etkilenir.

```
GETC            PROC NEAR       ;Yordam başlangıcı.
                MOV AH,1H       ;Bu satırda AH'a 1 verdikten sonra
                INT 21H         ;bu satır geliyorsa; 21.Interrupt'ın 01H nolu fonksiyonunu kullanıyorum--
                RET             ;demektir. O fonksiyon da klavyeden karakter okuma fonksiyonu.
GETC            ENDP            ;Ders kitabının 307. Sayfasında ilgili tablo bulunmaktadır.
```

2.9. PUTC Yordamı ve Açıklaması

AL yazmacındaki değeri ekranda gösterir.

```
PUTC            PROC NEAR       ;Yordam başlangıcı.
                PUSH AX         ;AH değişeceği için AX'teki değeri kaybetmemek için yığına atıyorum.
                PUSH DX         ;DL değişeceği için DX'teki değeri kaybetmemek için yığına atıyorum.
                MOV DL,AL       ;Interrupt ekrana basarken DL'yi kullandığı için(?) DL'ye AL'yi alıyoruz.
                MOV AH,2        ;Interrupt 21'in 02H nolu fonksiyonunu kullanacağız.
                INT 21H         ;
                POP DX          ;Yordam başlangıcında yığına attığım DX ve AX'i çekiyorum.
                POP AX          ;
                RET             ;RET ve ENDP ile yordamı sonlandırıyorum.
PUTC            ENDP            ;
```

3.0. GETN Yordamı ve Açıklaması

Klavyeden basılan sayıyı okur. Sonucu AX yazmacı üzerinden döndürür. DX: sayının işaretli olup olmadığını belirler. BL: hane bilgisini tutar. CX: okunan sayının işlenmesi sırasındaki ara değeri tutar. AL klavyeden okunan karakteri tutar (ASCII). Dönüş değeri olan AX dışındaki kullanılan yazmaçların önceki değerleri korunmalıdır.

```
GETN            PROC NEAR       ;
                PUSH BX         ;Kullandığım yazmaçların değerini kaybetmemek için yığına atıyorum.
                PUSH CX         ;
                PUSH DX         ;
GETN_START:     MOV DX,1        ;Sayının şimdilik pozitif olduğunu varsayalım.
                XOR BX,BX       ;Okuma yapmadı, hane 0 olur.
                XOR CX,CX       ;ara toplam değeri de 0'dır.
NEW:            CALL GETC       ;Klavyeden ilk değeri AL'ye okur.
                CMP AL,CR       ;Enter tuşuna basılmış ise,
                JE FIN_READ     ;okuma biter.
                CMP AL,'-'      ;AL , '-' mi geldi ?
                JNE CTRL_NUM    ;gelen 0-9 arasında bir sayı mı?
NEGATIVE:       MOV DX,-1       ; - basıldı ise sayı negatif, DX=-1 olur.
                JMP NEW         ;yeni haneyi al
CTRL_NUM:       CMP AL,'0'      ;sayının 0-9 arasında olduğunu kontrol et
                JB ERROR        ;değil ise gerekli hata mesajı ekrana basılacak.
                CMP AL,'9'      ;
                JA ERROR        ;
                SUB AL,'0'      ;rakam alındı haneyi toplama dahil et.
                MOV BL,AL       ;BL'ye okunan haneyi koy
                MOV AX,10       ;Haneyi eklerken *10 yapılacak
                PUSH DX         ;MUL komutu DX'i bozar, işaret için saklanmalı. Yığına atıyorum.
                MUL CX          ;AX * CX → Sonuç  DX:AX
                POP DX          ;işareti yığından çekiyorum.
                MOV CX,AX       ;CX'deki ara değer *10 yapıldı.
                ADD CX,BX       ;okunan haneyi ara değere ekle
                JMP NEW         ;klavyeden yeni basılan değeri al
```

```
ERROR:          MOV AX,OFFSET hata1;
                CALL PUT_STR  ;       Hatalı giriş yapılması durumunda ekrana hata mesajını yazdır.
                MOV AL,count  ;Hatalı giriş yapılırsa tekrar count‘. Elemanı giriniz’ mesajını yazdır
                CBW           ;
                CALL PUTN     ;
                MOV AX,OFFSET mesaj2 ;
                CALL PUT_STR  ;
                JMP GETN_START; O ana kadar okunanları unut yeniden sayı almaya başla.
FIN_READ:       MOV AX,CX     ; sonuç AX üzerinden dönecek
                CMP DX,1      ; İşarete göre sayıyı ayarlamak lazım
                JE FIN_GETN   ;
                NEG AX        ; AX= -AX
FIN_GETN:       POP DX        ; Değerini korumak için yığına attığım yazmaçları çekiyorum.
                POP CX        ;
                POP DX        ;
                RET           ;
GETN            ENDP          ;
```

3.1. PUTN Yordamı ve Açıklaması

AX’te bulunan sayıyı onluk tabanda hane hane yazdırır. CX: haneleri 100’a bölerek bulacağız. CX=10 olacak. DX: 32 bit bölmede işleme dahil olacak. Sonucu etkilemesin diye 0 olmalı.

```
PUTN            PROC NEAR     ;
                PUSH CX       ;
                PUSH DX       ;
                XOR DX,DX     ;DX 32 bit bölmede sonucu etkilemesin diye 0 olmalı.
                PUSH DX       ;Haneleri ASCII karakter olarak yığında saklayacağız
                              ;Kaç haneyi alacağımızı bilmediğimiz için yığına 0 değeri koyup--
                              ;onu alana kadar devam edeceğiz.
                MOV CX,10     ;CX=10
                CMP AX,0      ;
                JGE CALC_DIGITS;
                NEG AX        ;Sayı negatifse AX pozitif yapılır.
                PUSH AX       ;AX’i saklamak için yığına atıyoruz.
                MOV AL,'-'    ;işareti ekrana yazdır
                CALL PUTC     ;
                POP AX        ;AX’i yığından çekiyoruz.
CALC_DIGITS:    DIV CX        ;AX/CX →Sonuç DX:AX   AX=Bölüm DX=Kalan
                ADD DX,'0'    ;kalan değerini ASCII olarak bul
                PUSH DX       ;yığına at
                XOR DX,DX     ;DX=0
                CMP AX,0      ;bölen 0 kaldıysa sayının işlenmesi bitti demektir.
                JNE CALC_DIGITS; işlemi tekrarla
                              ;Yazılacak tüm haneler yığında. En anlamlı hane üstte, en az anlamlı hane--
                              ;altta ve onun altında da sona vardığımızı anlamak için konan 0 değeri var.
DISP_LOOP:      POP AX        ;Sırayla değerleri yığından çekiyoruz.
                CMP AX,0      ;AX=0 olursa sona geldik demektir.
                JE END_DISP_LOOP;
                CALL PUTC     ;AL’deki ASCII değeri yaz
                JMP DISP_LOOP ;işleme devam et
END_DISP_LOOP:POP DX          ;
                POP CX        ;
                RET           ;
PUTN            ENDP          ;
```

3.2. PUT_STR Yordamı ve Açıklaması

AX'te adresi verilen, sonunda 0 olan dizgeyi karakter karakter yazdırır. BX dizgeye indis olarak kullanılır.

```
240
241 PUT_STR     PROC NEAR           ;
242             PUSH BX             ;BX'teki değeri kaybetmemek için yığına atıyorum.
243             MOV BX,AX           ;Adresi BX'e al
244             MOV AL,BYTE PTR[BX] ;AL'de ilk karakter var.
245 PUT_LOOP:   CMP AL,0            ;
246             JE PUT_FIN          ;0 geldiyse dizge sona erdi demek
247             CALL PUTC           ;AL'deki karakteri ekrana yazar
248             INC BX              ;bir sonraki karaktere geç
249             MOV AL,BYTE PTR[BX] ;
250             JMP PUT_LOOP        ;yazdırmaya devam
251 PUT_FIN:    POP BX              ;BX'in eski değerini yığından çekiyorum
252             RET                 ;
253 PUT_STR     ENDP                ;
```

Yararlanılan Kaynaklar ve Bağlantılar

- Yrd. Doç. Dr. Ahmet Tevfik İnan – 80x86 Assembly Dili 2.Baskı
- https://en.wikipedia.org/wiki/Quicksort#Choice_of_pivot
- http://www.aivosto.com/visustin/sample/quicksort.html
- https://medium.com/basecs/pivoting-to-understand-quicksort-part-1-75178dfb9313
- https://en.wikipedia.org/wiki/Newline#Representations
- https://www.geeksforgeeks.org/rotate-matrix-90-degree-without-using-extra-space-set-2/
- https://www.draw.io
- ftp://ftp.univap.br/pub/manutencao/Drivers/CD_Rec_HDs_e_Outros/Cursos%20e%20Apostilas/MTL-OPTI/MTL-OPTI.pdf