Ezgi Sevi – 20011043
Bilgisayar Mühendisliği Bölümü
Alt Seviye Programla Ödevi Raporu
Furkan Çakmak

Dilation İşlemi:

İlk olarak kullanmak istediğim registerlerin değerlerini korumak için onları STACK'e PUSH'ladım.

Daha sonra Lena görselindeki renk skalasını binary' e indirgemek için 128 den büyük olanları 255 e küçük olanları 0 a eşitledim.

İşlem kolaylığı olması açısından n'İ 512'ye eşitledim. resim_org dizisindeki elemanlara iki farklı register (ESI, EDI) tarafından da ulaşmak istediğim için onları resim_org dizisinin pointer'ına eşitledim.

```
void Dilation(int n, int filter_size, short* resim_org) {

    __asm {

        PUSH ECX
        PUSH EDI
        PUSH ESI
        PUSH EAX
        PUSH EBX
        PUSH EDX

        MOV ECX, n
        MOV ESI, resim_org
        P1 : MOVSX EAX, WORD PTR[ESI]
        CMP EAX, 128
        JB SFR
        MOV WORD PTR[ESI], 255
        JMP P2
        SFR : MOV WORD PTR[ESI], 0
        P2 : ADD ESI, 2
        LOOP P1
        MOV n, 512

        MOV ESI, resim_org
        MOV EDI, resim_org
```

İşlemleri iki for döngüsünde gerçekleştirdim ilknin CX'sini ayaladım. İşlemler EAX registerini gerektirdiği için ilk önce onun üzerinde işlem yapıp daha sonra EAX 'i ECX'e aktardım. Böylece ilk for döngümün uzunluğunu ayarlamış oldum.

CX = (n-filter_size + 1) ^ 2

EBX register'ında hangi satırda olduğumun bilgisini saklayacak olan algoritmamdaki x'i sakladım. EBX her satır geçtiğinde bir artacak.

Daha önce CX ini ayarladığım ilk döngümde (i_dongusu) ilk işlem olarak ikinci döngümün (j_dongusu) CX bilgisini ayarlamak için ilk önce ilk döngünün CX ini PUSH ile STACK'e aktardım. Daha sonra ikinci döngümün CX' ini ayarladım.

CX = filter_size ^ 2

```
    MOV n, 512
    MOV EAX, n                          // for (i = 0; i < ((n - filter_size + 1)*(n - filter_size + 1)); i++)
    SUB EAX, filter_size
    INC EAX
    MUL EAX
    MOV ECX, EAX

    MOV EBX, 1                          // X in başlangıç değerinin ayarlanması

i_dongusu: MOV EAX, filter_size         //for (j = 0; j < (filter_size*filter_size); j++);
    MUL EAX
    XOR EDX, EDX                        // j döngüsünden önce j yi sıfırlama. j, dx de saklanacak
    PUSH ECX                            //j döngüsünün cx ine geçiş
    MOV ECX, EAX
```

İkinci döngümün ilk işleminde resim_orgun ESI ‘ncı elemanının en açık piksel olan 255’e eşit olup olmadığını kontrol ettim. Eğer eşitse filtre dahilindeki ilgili yeri, sol üst köşe yani konum bilgimi sakladığım register olan EDI’ nın gösterdiği yeri, 255’e eşitledim. Eğer eşit değilse filtrenin sıradaki elemanına geçerek aramaya devam ettirdim.

Filtre dizimi bir matris gibi hayal edecek olursa filtrede satır sonuna gelip gelmediğimi kontrol etmek için filtrede bulunduğum konumun(j), bir fazlasını alarak bunun filtre boyutuna tam bölünüp bölünemediğine baktım. J bilgisini EDX register’i içinde sakladım.

```
105     j_dongusu: MOVSX EAX, WORD PTR[ESI]
106         CMP EAX, 255
107         JNE Label1
108         MOV WORD PTR [EDI], 255
109
110     Label1: PUSH EDX                            // bölme işleminden önce dx in değerlerini koruma
111         INC EDX                                 //if ((j + 1) % filter_size == 0)
112         MOV EAX, EDX
113         XOR EDX,EDX
114         DIV filter_size                         // kalan dx de
```

Eğer bölünebiliyorsa filtremde satır sonunda olduğumu anlarım ve bir alt satıra geçmek için gerekli işlemleri filtrenin içindeyken genel dizide güncel konum bilgimi tutan k’ değişkenine uygularım. Ancak resim_org dizisinin Word tanımlı olduğunu unutmayarak eklediğim her değeri 2 ile çarparım.

$$k = k + ((n – filter\_size) +1) * 2$$

K değişkenimdeki bilgiyi ise ESI registerinde saklarım. Eğer tam bölünemiyorsa k değerini 2 arttırıp , 2 olmasının sebebi resim_org dizisinin Word tanımlı olması, güncek k değerini yine ESI ‘ya aktarıp gezinmeye devam ederim.

Bu işlemlerden sonra ikinci döngüme j -> EBX değerini arttırıp devam ederim.

```
115         CMP EDX, 0
116         JNZ Label2
117
118         MOV EAX, n                              //n
119         SUB EAX, filter_size                    //n - filter_size
120         INC EAX                                 //(n - filter_size) + 1
121         SHL EAX, 1                              //32 bit bellekte hareket ettiği için
122         ADD ESI, EAX                            //k = k + (n - filter_size) + 1
123         JMP Label3
124
125     Label2: ADD ESI, 2                              //k=k+1
126
127     Label3: POP EDX
128         INC EDX                                 //j+1
129
130         LOOP j_dongusu
```

İkinci döngünün içinden çıktıktan sonra ilk döngümün CX register'ini geri POP'larım. Bir alt satıra geçmem gerekip gerekmediğini resim_org dizisinde konum bilgimi saklayan EDI registerim sayesinde belirlerim. Gerekli İf koşulumun içinde

İf(x*n-konum == filter_size) ifadesinin doğruluğunu kontrol ederim.

x-> dizi matris gibi düşünülürse satır bilgisini tutar. EBX registerinde saklanır.

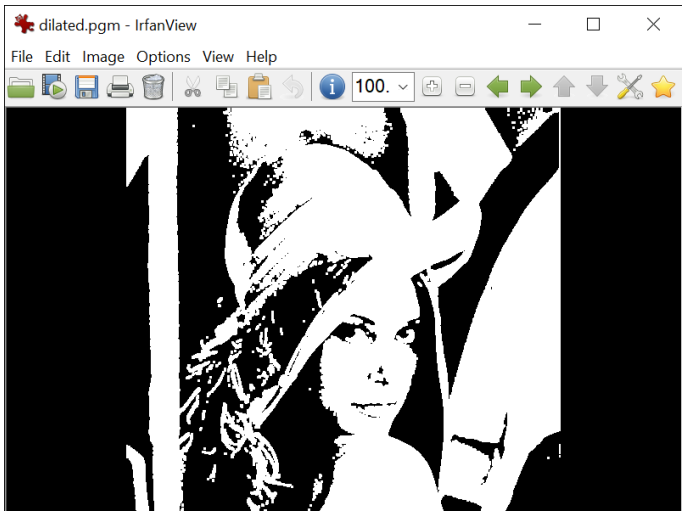
konum-> EDI registeri içinde saklanır filtreyi hareket ettirmede kullanılır.

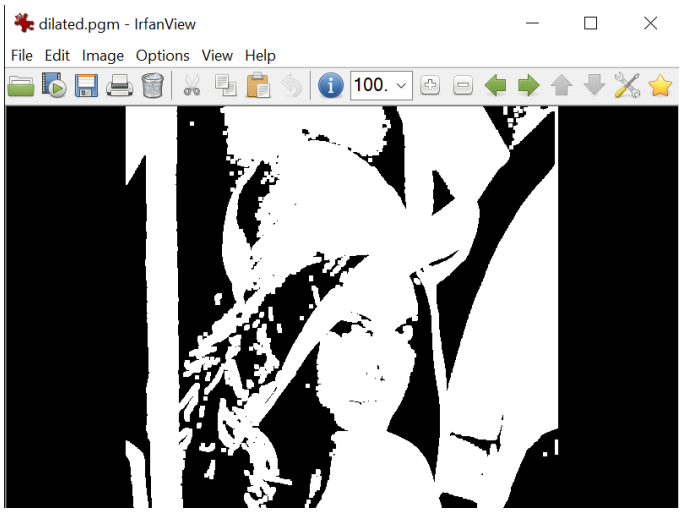
Eğer doğruysa bir alt satıra geçmem gerekmektedir. Bunu: konum = konum + filter_size işlemi ile gerçekleştiririm. Eğer yanlışsa yine resim_org dizisinin Word tanımlı olduğunu unutmayak EDI' Ya iki ekleyerek bir sonraki işlemler grubuna geçmeye hazırlanırım. En sonunda k(filte içindeki aranmalar sırasındaki konumu gösterir, ESI registeri içinde saklanır.) değişkenime konum değişkenimi atayarak filtremi hareket ettirmiş olurum.

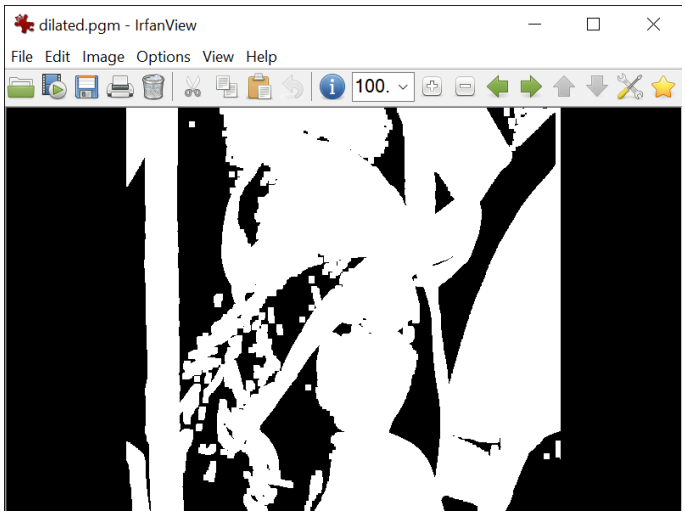
```
131
132         MOV EAX, EBX                        // X şu an ax in içinde
133         MUL n
134         SUB EAX, EDI
135         POP ECX                             // i döngüsünün cx ine tekrar geçiş
136
137         CMP EAX, filter_size
138         JNE Label4
139         MOV EAX, filter_size
140         SHL EAX,1
141         ADD EDI, EAX                         //konum = konum + filter_size;
142         INC EBX                              //x = x + 1;
143         JMP Label5
144
145       Label4: ADD EDI, 2                         //konum = konum + 1;
146       Label5: MOV ESI, EDI                       //k = konum;
147         LOOP i_dongusu
```

Değerlerini kaybetmemeleri için STACK'e attığım register'leri geri POP etme işlemi.

```
149         POP EDX
150         POP EBX
151         POP EAX
152         POP ESI
153         POP EDI
154         POP ECX
155       }
156       printf("\nDilation islemi sonucunda resim \"dilated.pgm\" ismiyle olusturuldu...\n");
157   }
158
```


dilated.pgm - IrfanView
File Edit Image Options View Help
100.


dilated.pgm - IrfanView
File Edit Image Options View Help
100.


dilated.pgm - IrfanView
File Edit Image Options View Help
100.

Erosion İşlemi:

İlk olarak kullanmak istediğim registerlerin değerlerini korumak için onları STACK'e PUSH'ladım.

Daha sonra Lena görselindeki renk skalasını binary' e indirgemek için 128 den büyük olanları 255 e küçük olanları 0 a eşitledim.

İşlem kolaylığı olması açısından n'İ 512'ye eşitledim. resim_org dizisindeki elemanlara iki farklı register (ESI, EDI) tarafından da ulaşmak istediğim için onları resim_org dizisinin pointer'ına eşitledim.

```
159  void Erosion(int n, int filter_size, short* resim_org) {
160      __asm {
161          PUSH ECX
162          PUSH EDI
163          PUSH ESI
164          PUSH EAX
165          PUSH EBX
166          PUSH EDX
167
168
169          MOV ECX, n
170          MOV ESI, resim_org
171          F1 : MOVSX EAX, WORD PTR[ESI]
172          CMP EAX, 128
173          JB SFR
174          MOV WORD PTR[ESI], 255
175          JMP F2
176          SFR : MOV WORD PTR[ESI], 0
177          F2 : ADD ESI, 2
178          LOOP F1
179          MOV n, 512
180
181          MOV ESI, resim_org
182          MOV EDI, resim_org
```

İşlemleri iki for döngüsünde gerçekleştirdim ilknin CX'sini ayaladım. İşlemler EAX registerini gerektirdiği için ilk önce onun üzerinde işlem yapıp daha sonra EAX 'i ECX'e aktardım. Böylece ilk for döngümün uzunluğunu ayarlamış oldum.

CX = (n-filter_size + 1) ^ 2

EBX register'ında hangi satırda olduğumun bilgisini saklayacak olan algoritmamdaki x'i sakladım. EBX her satır geçtiğinde bir artacak.

Daha önce CX ini ayarladığım ilk döngümde (i_dongusu) ilk işlem olarak ikinci döngümün (j_dongusu) CX bilgisini ayarlamak için ilk önce ilk döngünün CX ini PUSH ile STACK'e aktardım. Daha sonra ikinci döngümün CX' ini ayarladım.

CX = filter_size ^ 2

```
183
184          MOV EAX, n                                  // for (i = 0; i < ((n - filter_size + 1)*(n - filter_size + 1)); i++)
185          SUB EAX, filter_size
186          INC EAX
187          MUL EAX
188          MOV ECX, EAX
189
190          MOV EBX, 1                                  // X in başlangıç değerinin ayarlanması
191          i_dongusu: MOV EAX, filter_size                     //for (j = 0; j < (filter_size*filter_size); j++);
192          MUL EAX
193          PUSH ECX                                    //j döngüsünün cx ine geçiş
194          MOV ECX, EAX
195          XOR EDX, EDX                                // j döngüsünden önce j yi sıfırlama. j, dx de saklanacak
196
```

İkinci döngümün ilk işleminde resim_orgun ESI ‘ncı elemanının en koyu piksel olan 0’a eşit olup olmadığını kontrol ettim. Eğer eşitse filtre dahilindeki ilgili yeri, sol üst köşe yani konum bilgimi sakladığım register olan EDI’ nın gösterdiği yeri, 0’e eşitledim. Eğer eşit değilse filtrenin sıradaki elemanına geçerek aramaya devam ettirdim.

Filtre dizimi bir matris gibi hayal edecek olursa filtrede satır sonuna gelip gelmediğimi kontrol etmek için filtrede bulunduğum konumun(j), bir fazlasını alarak bunun filtre boyutuna tam bölünüp bölünemediğine baktım. J bilgisini EDX register’i içinde sakladım.

```
197     j_dongusu : MOVSX EAX, WORD PTR[ESI]
198     CMP EAX, 0
199     JNE L1
200     MOV WORD PTR[EDI], 0
201
202     L1 : PUSH EDX                           // bölme işleminden önce dx in değerlerini koruma
203     INC EDX                                 //if ((j + 1) % filter_size == 0)
204     MOV EAX, EDX
205     XOR EDX, EDX
206     DIV filter_size                         // kalan dx de
207     CMP EDX, 0
208     JNZ L2
209
```

Eğer bölünebiliyorsa filtremde satır sonunda olduğumu anlarım ve bir alt satıra geçmek için gerekli işlemleri filtrenin içindeyken genel dizide güncel konum bilgimi tutan k’ değişkenine uygularım. Ancak resim_org dizisinin Word tanımlı olduğunu unutmayarak eklediğim her değeri 2 ile çarparım.

$$k = k + ((n – filter\_size) +1) * 2$$

K değişkenimdeki bilgiyi ise ESI registerinde saklarım. Eğer tam bölünemiyorsa k değerini 2 arttırıp , 2 olmasının sebebi resim_org dizisinin Word tanımlı olması, güncek k değerini yine ESI ‘ya aktarıp gezinmeye devam ederim.

Bu işlemlerden sonra ikinci döngüme j -> EBX değerini arttırıp devam ederim.

```
209
210     MOV EAX, n                              //n
211     SUB EAX, filter_size                    //n - filter_size
212     INC EAX                                 //(n - filter_size) + 1
213     SHL EAX, 1                              //32 bit bellekte hareket ettiği için
214     ADD ESI, EAX                            //k = k + (n - filter_size) + 1
215     JMP L3
216
217     L2 : ADD ESI, 2                         // k=k+1
218     L3 :POP EDX
219     INC EDX                                 //j+1
220     LOOP j_dongusu
221
```

İkinci döngünün içinden çıktıktan sonra ilk döngümün CX register'ini geri POP'larım. Bir alt satıra geçmem gerekip gerekmediğini resim_org dizisinde konum bilgimi saklayan EDI registerim sayesinde belirlerim. Gerekli İf koşulumun içinde

İf(x*n-konum == filter_size) ifadesinin doğruluğunu kontrol ederim.

x-> dizi matris gibi düşünülürse satır bilgisini tutar. EBX registerinde saklanır.

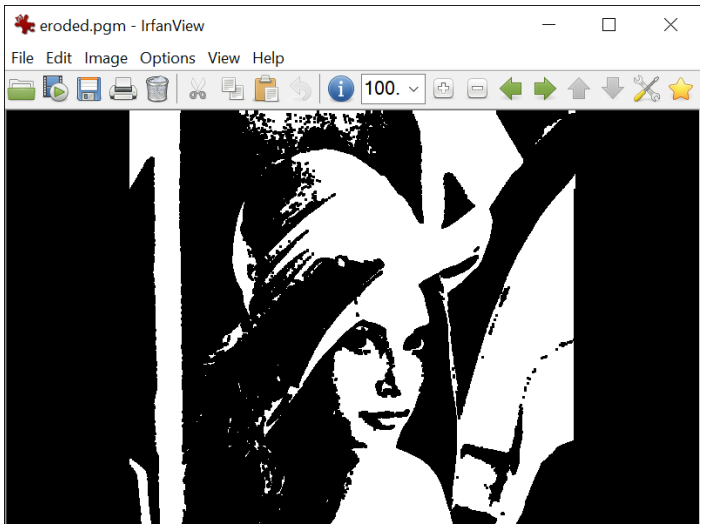
konum-> EDI registeri içinde saklanır filtreyi hareket ettirmede kullanılır.

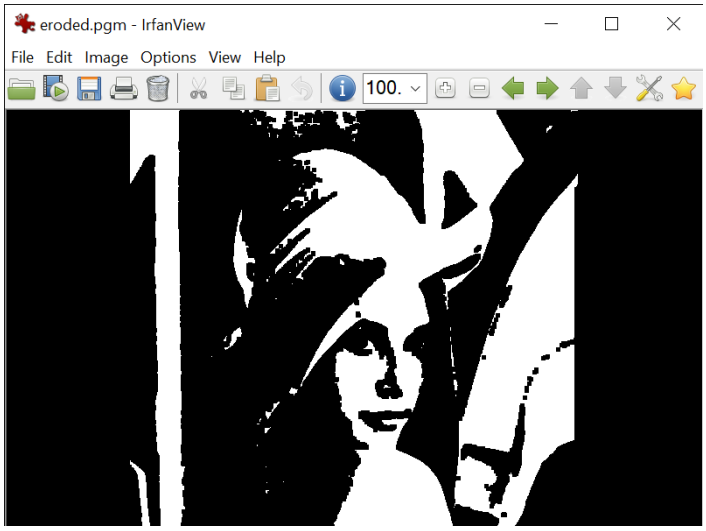
Eğer doğruysa bir alt satıra geçmem gerekmektedir. Bunu: konum = konum + filter_size işlemi ile gerçekleştiririm. Eğer yanlışsa yine resim_org dizisinin Word tanımlı olduğunu unutmayak EDI' Ya iki ekleyerek bir sonraki işlemler grubuna geçmeye hazırlanırım.

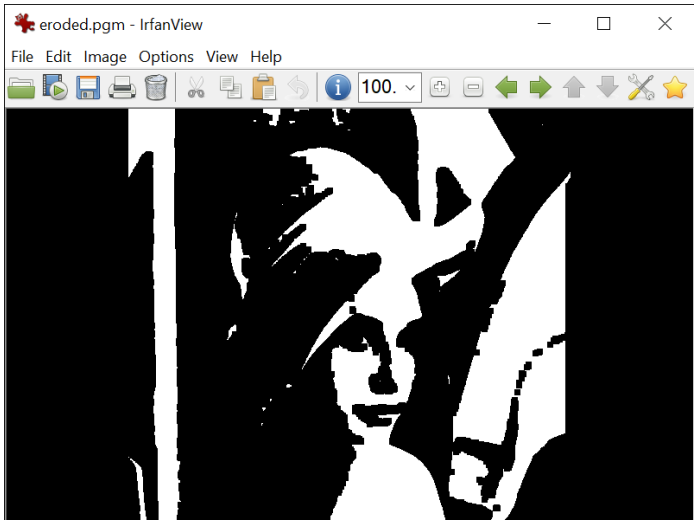
```
222         MOV EAX, EBX                                // X şu an ax in içinde
223         MUL n
224         SUB EAX, EDI
225         POP ECX                                     // i döngüsünün cx ine tekrar geçiş
226         CMP EAX, filter_size
227         JNE L4
228         MOV EAX, filter_size
229         SHL EAX, 1
230         ADD EDI, EAX                                 //konum = konum + filter_size;
231         INC EBX                                      //x = x + 1;
232         JMP L5
233         L4 : ADD EDI, 2                             //konum = konum + 1;
```

En sonunda k (filte içindeki aranmalar sırasındaki konumu gösterir, ESI registeri içinde saklanır.) değişkenime konum değişkenimi atayarak filtremi hareket ettirmiş olurum. Değerlerini kaybetmemeleri için STACK'e attığım register'leri geri POP etme işlemi.

```
234         L5 : MOV ESI, EDI                                   //k = konum;
235         LOOP i_dongusu
236
237         POP EDX
238         POP EBX
239         POP EAX
240         POP ESI
241         POP EDI
242         POP ECX
243     }
244     printf("\nErosion islemi sonucunda resim \"eroded.pgm\" ismiyle olusturuldu...\n");
245 }
246
```


eroded.pgm - IrfanView
File Edit Image Options View Help
100.


eroded.pgm - IrfanView
File Edit Image Options View Help
100.


eroded.pgm - IrfanView
File Edit Image Options View Help
100.