Yıldız Teknik Üniversitesi
Elektrik-Elektronik Fakültesi
Bilgisayar Mühendisliği Bölümü

BLM2021 - Alt Seviye Programlama
Dr. Öğr. Üyesi Erkan USLU
ÖDEV-2

İsim: Berkan Eti
No: 21011001
E-posta: berkan.eti@std.yildiz.edu.tr

İçindekiler

1) Dilation

a) ASM Kodunun açıklanması

```
__asm {
        MOV ESI, resim_org.          → İlk olarak ESI registeri ile resim_org dizisine erişilir.
        XOR EAX, EAX // i            → EAX ile en dıştaki döngü kontrol edilir. n değerine kadar her
        DIS :                        döngüde 512 (bir satır aşağı gidecek şekilde) arttırılır.
        CMP EAX , n
        JNB DIS_SON
        XOR EBX ,EBX // j            → EBX içteki döngü için kullanılır. 512 değerine kadar her döngüde
        IC:                          1 arttırılır.( Satır içinde ilerleme EBX ile yapılır.)
        CMP EBX , 512
        JNB IC_SON
        XOR DI, DI // max            → Max değerlerini tutacağımız DI her iç döngüde 0'lanır.
        XOR ECX , ECX  // K
        PUSH EAX
        PUSH EDX                     → iç döngüde filtre kontrollerine başlanır. Filtre kontrolündeki satir
        PUSH EBX                       ilerlemesi ECX ile yapılır . ilk değer olarak (i - (filter_size / 2) * 512)
        MOV EAX , filter_size                atanır. (i + (filter_size / 2) * 512)'ye kadar döngü devam eder.
        SHR EAX , 1 // filter_size/2
        XOR EDX , EDX
        MOV EBX , 512
        MUL EBX                      → MUL ile çarpma işlemi yapılmış ve elde edilen sonucun NEG
        MOV ECX , EAX                kullanılarak negatifi alınmıştır. Daha sonra başta elde ettiğimiz
        NEG ECX                      EAX(satir) değeri ile toplanıp filtrenin başlangıç noktası bulunmuştur.
        POP EBX
        POP EDX
        POP EAX
        ADD ECX , EAX  // (i - (filter_size / 2) * 512)
        SATIR :
        MOV EDX, 3                   → Filtre uygulamasının iç döngüsü EDX ile kontrol edilir. Başlangıç
        SHR EDX, 1                   olarak (j - filter_size / 2) değeri verilir ve (j + filter_size / 2)'ye kadar
        NEG EDX                      devam eder.
        ADD EDX, EBX // (j - filter_size / 2)
        SUTUN :
        PUSH EBX
        PUSH AX
        XOR EBX , EBX
        ADD EBX , ECX
        ADD EBX , EDX                → EBX = ECX + EDX (Filtrenin kontrol edileceği nokta bulunur.)
        CMP EBX , 0                        → if(EBX >=0) -> devam et
        JL SUTUN_SON
        CMP EBX , n                  → if(EBX< n) -> devam et
        JG SUTUN_SON
        ADD EBX, EBX                 → WORD tipi erişim olduğu için EBX 2 katına çıkarılır ve değere erişilir.
        MOV AX ,WORD PTR  [ESI+EBX]
        CMP DI , AX                  → if(max  < dizi[EBX]) -> max = dizi[EBX]
        JA SUTUN_SON
        MOV DI , AX
        SUTUN_SON :
        POP AX
        POP EBX
```

```
        INC EDX          → döngü için EDX 1 arttırılır.(Filtrede anlık bulunan satırda ilerlenir.)
        PUSH EAX
        MOV EAX ,filter_size
        SHR EAX , 1
        ADD EAX , EBX
        CMP EDX , EAX            → if(EDX <= (j + filter_size / 2)) -> SUTUN döngüsüne devam et.
        POP EAX
        JNG SUTUN
        ADD ECX , 512    → döngü için ECX 512 arttırılır. (Filtrede alt satıra geçilir.)
        PUSH EBX
        PUSH EDX
        PUSH EAX
        MOV EAX , filter_size
        SHR EAX , 1
        XOR EDX , EDX
        MOV EBX , 512
        MUL EBX
        MOV EBX , EAX
        POP EAX
        POP EDX
        ADD EBX , EAX
        CMP ECX , EBX            → if(ECX <= (i + (filter_size / 2) * 512)) -> SATIR döngüsüne devam et.
        POP EBX
        JNG SATIR
        PUSH DI          → i,j noktası için oluşturulan filtre stack'e aktarılır.
        INC EBX
        JMP IC           → EBX (j) değeri arttırılır ve iç döngüye (satır içinde) devam edilir.
        IC_SON :
        ADD EAX , 512
        JMP DIS          → EAX(i) değeri 512 arttırılır ve alt satıra geçilir.
        DIS_SON :        → Dilation işlemi sonlanır.
        MOV ECX , n      → n defa stackten veri alınır.
        MOV EDI , n
        ADD EDI , EDI
        SUB EDI , 2  → veri almaya stack'in sonundan başlanır. (WORD tipinde olduğu için 2n-2)
        L1 :
        POP AX
        MOV WORD PTR [ESI + EDI], AX
        SUB EDI , 2
        LOOP L1                  → stack'deki veriler n defa döngüye girerek diziye aktarılır.
}
```

NOT : EAX : i
EBX : j / ESI ile beraber veri okuma için kullanılmıştır.
ECX : k
EDX : l
DI : max

b) Pseudo Code ile Gösterimi

```
i = 0
while(i < n) :
        j = 0
        while(j<512) :
                max = 0
                k = i - (filter_size/2)*512)
                while(k<= i + (filter_size/2)*512) :
                        l = j - filter_size/2
                        while(l<= j + filter_size/2) :
                                if(max < resim[k+l]) max = resim[k+l]
                        END
                END
        push(resim[k+l])
        END
END

c = N
d = N-1
for(c -> 0:-1) :  // N'den 0'a kadar birer birer azalarak döngüye girecek
        pop(x)
        resim[d] = x
        d = d - 1
END
```

NOT : Pseudo Code'da her elemanın dizide bir indise karşılık geldiği kabul edilmiştir.

c) Örnekler

lena.pgm

filter_size = 3

filter_size = 5

filter_size = 7

filter_size = 25

2) Erosion

a) ASM Kodunun açıklanması

NOT : Erosion ve Deliaton arasındaki fark min/max'dan kaynaklanmaktadır.

```
__asm {
        MOV ESI, resim_org.        → İlk olarak ESI registeri ile resim_org dizisine erişilir.
        XOR EAX, EAX // i          → EAX ile en dıştaki döngü kontrol edilir. n değerine kadar her
        DIS :                      döngüde 512 (bir satır aşağı gidecek şekilde) arttırılır.
        CMP EAX , n
        JNB DIS_SON
        XOR EBX ,EBX // j          → EBX içteki döngü için kullanılır. 512 değerine kadar her döngüde
        IC:                        1 arttırılır.( Satır içinde ilerleme EBX ile yapılır.)
        CMP EBX , 512
        JNB IC_SON
        MOV DI, 255 // min         → Min değerlerini tutacağımız DI her iç döngüde 255'e eşitlenir.
        XOR ECX , ECX  // K
        PUSH EAX
        PUSH EDX                   → iç döngüde filtre kontrollerine başlanır. Filtre kontrolündeki satir
        PUSH EBX                     ilerlemesi ECX ile yapılır . ilk değer olarak (i - (filter_size / 2) * 512)
        MOV EAX , filter_size              atanır. (i + (filter_size / 2) * 512)'ye kadar döngü devam eder.
        SHR EAX , 1 // filter_size/2
        XOR EDX , EDX
        MOV EBX , 512
        MUL EBX                    → MUL ile çarpma işlemi yapılmış ve elde edilen sonucun NEG
        MOV ECX , EAX              kullanılarak negatifi alınmıştır. Daha sonra başta elde ettiğimiz
        NEG ECX                    EAX(satir) değeri ile toplanıp filtrenin başlangıç noktası bulunmuştur.
        POP EBX
        POP EDX
        POP EAX
        ADD ECX , EAX  // (i - (filter_size / 2) * 512)
        SATIR :
        MOV EDX, 3                 → Filtre uygulamasının iç döngüsü EDX ile kontrol edilir. Başlangıç
        SHR EDX, 1                 olarak (j - filter_size / 2) değeri verilir ve (j + filter_size / 2)'ye kadar
        NEG EDX                    devam eder.
        ADD EDX, EBX // (j - filter_size / 2)
        SUTUN :
        PUSH EBX
        PUSH AX
        XOR EBX , EBX
        ADD EBX , ECX
        ADD EBX , EDX              → EBX = ECX + EDX (Filtrenin kontrol edileceği nokta bulunur.)
        CMP EBX , 0                      → if(EBX >=0) -> devam et
        JL SUTUN_SON
        CMP EBX , n                → if(EBX< n) -> devam et
        JG SUTUN_SON
        ADD EBX, EBX               → WORD tipi erişim olduğu için EBX 2 katına çıkarılır ve değere erişilir.
        MOV AX ,WORD PTR  [ESI+EBX]
        CMP DI , AX                → if(min  > dizi[EBX]) -> max = dizi[EBX]
        JB SUTUN_SON
        MOV DI , AX
```

```
SUTUN_SON :
POP AX
POP EBX
INC EDX          → döngü için EDX 1 arttırılır.(Filtrede anlık bulunan satırda ilerlenir.)
PUSH EAX
MOV EAX ,filter_size
SHR EAX , 1
ADD EAX , EBX
CMP EDX , EAX          → if(EDX <= (j + filter_size / 2)) -> SUTUN döngüsüne devam et.
POP EAX
JNG SUTUN
ADD ECX , 512    → döngü için ECX 512 arttırılır. (Filtrede alt satıra geçilir.)
PUSH EBX
PUSH EDX
PUSH EAX
MOV EAX , filter_size
SHR EAX , 1
XOR EDX , EDX
MOV EBX , 512
MUL EBX
MOV EBX , EAX
POP EAX
POP EDX
ADD EBX , EAX
CMP ECX , EBX          → if(ECX <= (i + (filter_size / 2) * 512)) -> SATIR döngüsüne devam et.
POP EBX
JNG SATIR
PUSH DI          → i,j noktası için oluşturulan filtre stack'e aktarılır.
INC EBX
JMP IC           → EBX (j) değeri arttırılır ve iç döngüye (satır içinde) devam edilir.
IC_SON :
ADD EAX , 512
JMP DIS          → EAX(i) değeri 512 arttırılır ve alt satıra geçilir.
DIS_SON :        → Dilation işlemi sonlanır.
MOV ECX , n      → n defa stackten veri alınır.
MOV EDI , n
ADD EDI , EDI
SUB EDI , 2  → veri almaya stack'in sonundan başlanır. (WORD tipinde olduğu için 2n-2)
L1 :
POP AX
MOV WORD PTR [ESI + EDI], AX
SUB EDI , 2
LOOP L1          → stack'deki veriler n defa döngüye girerek diziye aktarılır.
}
```

NOT : EAX : i
EBX : j / ESI ile beraber veri okuma için kullanılmıştır.
ECX : k
EDX : l
DI : min

b) Pseudo Code ile Gösterimi

```
i = 0
while(i < n) :
        j = 0
        while(j<512) :
                min = 255
                k = i - (filter_size/2)*512)
                while(k<= i + (filter_size/2)*512) :
                        l = j - filter_size/2
                        while(l<= j + filter_size/2) :
                                if(min > resim[k+l]) min = resim[k+l]
                        END
                END
        resim[i+j] = min
        END
END

c = N
d = N-1
for(c -> 0:-1) :  // N'den 0'a kadar birer birer azalarak döngüye girecek
        pop(x)
        resim[d] = x
        d = d - 1
END
```

NOT : Pseudo Code'da her elemanın dizide bir indise karşılık geldiği kabul edilmiştir.

c) Örnekler

lena.pgm

filter_size = 3

filter_size = 5

filter_size = 7

filter_size = 25