Universidade de Évora

Arquitetura de Sistemas e Computadores

Relatório do Trabalho Prático

João Calhau - 31621
André Figueira - 31626

8 de Junho de 2014

Conteúdo

1 Objectivo

Pretende-se com este trabalho desenvolver um conjunto de funções em assembly Mips para etecção de contornos em imagems a cores. Dada uma imagem RGB, o resultado final iria ser uma imagem em tons de cinzento, com fundo branco e com traços escuros nos locais onde existem contornos na imagem original.

1.1 Como é que este objectivo vai ser atingido?

Este objectivo vai ser atingido através de várias funções criadas que irão dividir o problema em várias funções de forma a facilitar a resolução. Durante a construção destas funções notam-se também o chamamento de funções auxiliares para dividir o problema numa forma mais detalhada.

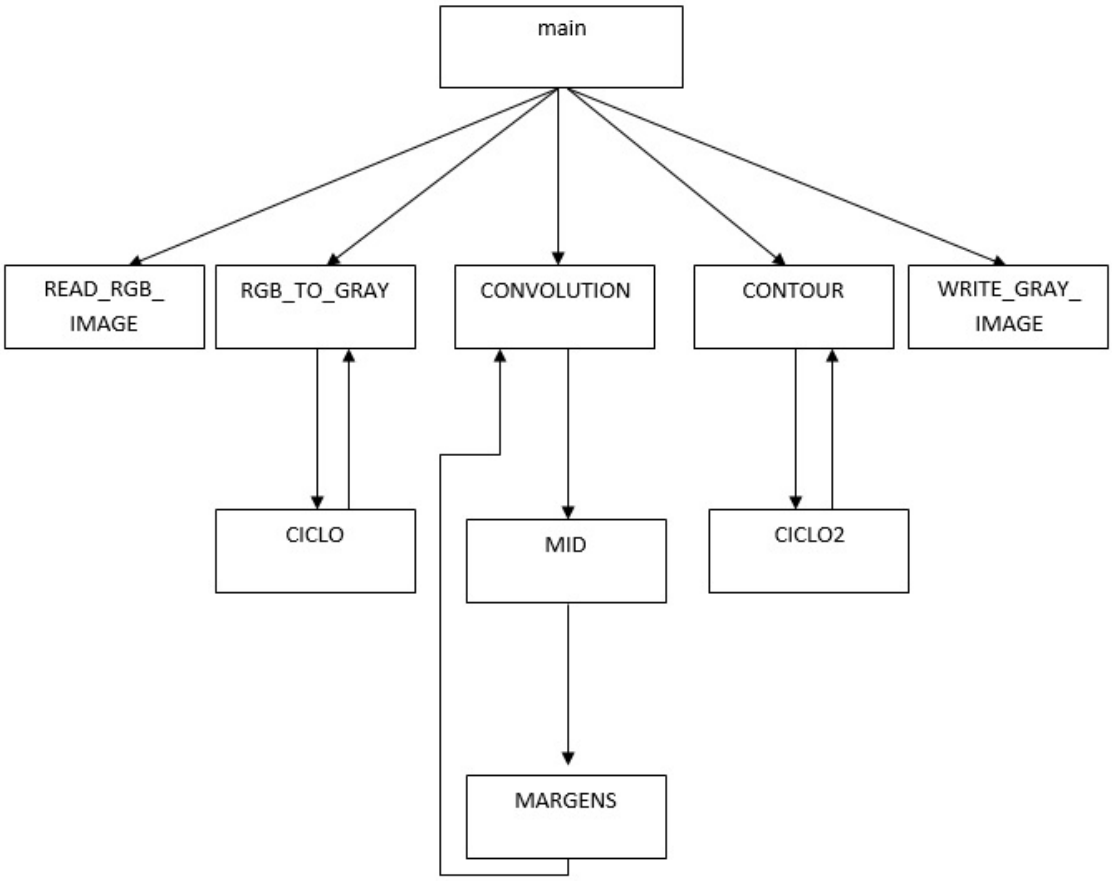
Figura 1: Lena

↓

Figura 2: Lena (Final)

2 Descrição

2.1 Esquema



2.2 Conversão da imagem para formato RGB

Usando a linha de comando **sudo apt-get install ImageMagick** obtém-se uma ferramenta capaz de conseguir atingit o objectivo de conversão para RGB. O Próximo passo consiste em abrir o terminal e na directoria, onde se encontra a imagem que irá ser utilizada, usar a linha de comando **convert example.tiff example.rgb**, sendo **.tiff** um formato de imagem (podendo ser este qualquer outro, por exemplo **.jpg**) como resultado obtemos uma imagem em formato RGB pronta a ser utilizada pelo programa.

2.3 O procedimento

A explicação do procedimento é feita em duas partes, a primeira é em relação ao **.data segment** e a outra em ralação ao **.text segment**.

2.3.1 O .data segment

É nesta região da memória em que se encontram todos os buffers utilizados, contendo também localização do ficheiro de input e output, juntamente com as respectivas directorias. Como se pode ver pela figura 3:

```
#       FICHEIROS:
        #       nome do ficheiro se tiver na directoria do programa
        #       caso contrario localizacao exacta:

                file1: .asciiz "lena512color.rgb"       #ficheiro input

        #       output file, directoria de destino
        #       sem directoria -> o destino e' mesma directoria do programa

                file2: .asciiz "contornos4.GRAY"        #ficheiro output

#       BUFFERS:
                bufferRGB: .space 786432                        #buffer com formato RGB (informaÃSao extraida do input)
                bufferGRAY: .space 262144                       #buffer com formato GRAY
                Bhorizontal: .space 262144                      #buffer da imagem aplicada com o sobel horizontal
                Bvertical: .space 262144                        #buffer da imagem aplicada com o sobel vertical
                imagem_final: .space 262144                     #buffer que contem a imagem final

#       SOBELS:
                sobelH: .byte   1,0,-1,2,0,-2,1,0,-1            #sobel horizontal
                sobelV: .byte   1,2,1,0,0,0,-1,-2,-1            #sobel vertical

```

Figura 3: .data segment

2.3.2 O .text segment

Neste segmento é one se encontra a parte do código que irá resolver o problema de detectar os contornos da imagem. Para tal, o .text segment está dividido em vários **branches**.

O **branch** mais importante neste programa é sem dúvida o **main**,

```
main:
#       argumento da primeira funcao
        la $a0,file1
        jal READ_RGB_IMAGE
        nop

#       argumentos da segunda funÃ§ao
        la $a1,bufferGRAY
        la $a0,bufferRGB
        jal RGB_TO_GRAY
        nop

#       argumentos da terceira funÃ§ao (1Ã° parte -> sobel horizontal)
        la $a0,bufferGRAY
        la $a1,Bhorizontal
        la $a2,sobelH
        jal convolution
        nop

#       argumentos da terceira funÃ§ao (2Ã° parte -> sobel vertical)
        la $a0,bufferGRAY
        la $a1,Bvertical
        la $a2,sobelV
        jal convolution
        nop
#       argumentos da quarta funÃ§ao (imagem final)
        la $a0,Bhorizontal
        la $a1,Bvertical
        la $a2,imagem_final
        jal contour
        nop

#       argumentos da ultima funÃ§ao
        la $a0,file2
        la $a1,imagem_final
        li $a2,262144                   #comprimento do buffer
        jal WRITE_TO_GRAY
        nop

#       fim do programa
        beq $zero,$zero,END
        nop

```

Figura 4: Parte 'main' do .text segment

Este branch é o **branch central**, a partir deste tudo se controla, pois é este que efectua o chamamento de todas as outra funções que irão aingir, no fim, o objectivo desejado.

1ª Função:

O branch **READ_RGB_IMAGE**, recebe como argumento o ficheiro de input e transfere o conteúdo RGB para um buffer (**bufferRGB**) em memória.

```
4       #       nome do ficheiro se tiver na directoria do programa
5       #       caso contrario localizacao exacta:
6
7               file1: .asciiz "lena512color.rgb"       #ficheiro input
```

Figura 5: Ficheiro de input

2ª Função:

O branch **RGB_TO_GRAY**, recebe como argumentos dois buffers, um buffer que contem o **bufferRGB** e um buffer vazio (**bufferGRAY**), esta função efectua uma chamada de uma função auxiliar **CICLO** que irá percorrer um ciclo **while** que utilizando as instruções **lbu** (**load byte unassygned**), pois não é necessário a extensão de sinal, chama 3 bytes (ou seja um pixel em RGB) seguidos do bufferRGB que são aplicados na fórmula:

$$\mathbf{I = 0.30R + 0.59G + 0.11B}$$

Esta fórmula serve para nos fornecer o valor em gray do pixel correspondente à posição do pixel da imagem em formato RGB. Usando a instrução **sb** (**store byte**) guarda-se o resultado da fórmula no **bufferGRAY**, em seguida basta incrementar os contadores e buffers para seguirem para o próximo pixel. Este ciclo é repetido **786432 + 3** (+3 bytes, ou seja, mas um pixel, para não terminar antes de completar o ultimo pixel). Quando terminar, executa **jr $ra**, que volta para a função **RGB_To_GRAY** e a partir dai volta para a main pronta para passar para a proxima função.
O resultado da imagem deverá ser algo como:

Figura 6: Lena depois de aplicado a função RGB_TO_GRAY

3ª Função:

A função **convolution**, é a única que é chamada duas vezes a partir do **main**, isto porque o exige visto que a imagem guardada no bufferGRAy tem e ser aplicada com sobel, vertical e horizontal. Esta funcção tem dois chamamentos de funções auxiliares, visto que a **MID** (a principal) percorre o interior da imagem à excepção da primeira e ultima linha e das margens. Esta função usando a instrução **lbu (load byte unassygned)** chama um pixel (por ordem de posição no bufferGRAY) e usando a formula:

$$B(i,j) = \sum_{p\in\{-1,0,1\}} \sum_{q\in\{-1,0,1\}} A(i+p, j+q)\, S_h(2-p, 2-q)$$

Figura 7: Formula fornecida pelo docente

é usada em conjunto com as matrizes (1ª chamada da função convolution utiliza a matriz sobel horizontal e na 2ª a matriz sobel vertical):

$$S_h = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad S_v = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figura 8: matrizes Sobel vertical (Sv) e horizontal (Sh)

(Utilizando também um load byte mas desta vez apenas lb, porque é necessário haver extensão de sinal, valor negativo)

Permite então através de um ciclo **while** obter o valor do dito pixel aplicado com sobel e após ter sido utiizada a formula do somatório usar a seguinte formula:

$$B_h = \tfrac{1}{4}|S_h * A|$$
$$B_v = \tfrac{1}{4}|S_v * A|$$

Figura 9: Outra formula fornecida pelo docente

Usando a instrução **sb (store byte)** guarda o resultado num novo buffer (1ª chamada no main - bhorizontal, 2ª chamada no main - bvertical). Repetindo o cilco **261632 -2** (-2 para conseguir fazer o ultimo pixel da penúltima linha e não terminar antes de o fazer).

Esta fórmula apresenta um problema visto que não está definida para as margens, por isso é que o ciclo tem -512 ao valor original de 262144, para não percorrer a ultima linha (ficando assim a prento por default), e o contador (começando no 1) do ciclo começa em 514 para meter por default a primeira linha mais o primeiro elemento da segunda linha a 0 e começando na posição seguinte (posição 514).

Por isso foi criada uma função auxiliar **MARGENS** com o propósito de as detectar. Esta função é chamada pela função **MID** quando o contador $t0 cumpre uma restrição, esta restrição consiste em, o contador se for igual ao tamanho do valor da linha irá fazer com que avance 2 pixéis no contador e posições nos buffers para a frente:

```
228         MARGENS:
229                 addi $a1,$a1,2              #avanÃ§a 2 pixeis para a frente
230                 addi $a0,$a0,2              #avanÃ§a 2 pixeis para a frente
231                 addi $t1,$t1,512            #proxima restriÃ§ao para as margens -> +512 a restricao anterior
232                 addi $t0,$t0,2              #incrementa contador
233                 add $t4,$zero,$zero         #reset do contador t4
234                 bne $t1,261632,MID          #volta para a funÃ§ao, caso seja diferente, caso seja igual volta para main
235                 nop                         #e deixa a ultima linha (linha 512) a preto.
236
237                 jr $ra                      #volta pa convolution
238                 nop
239
```

Figura 10: Função **MARGENS**

Se o valor da restrição for igual ao valor da penúltima linha, a função volta para a função convolution e daí volta para a main. Em seguida será repetir esta função outra vez, para aplicar o sobel vertical à imagem.

Nota: visto que existe chamamento de função dentro de função é necessário guardar o valor do registo **$ra** usando a pilha para se conseguir voltar para a função **main** e continuar o programa e não entrar num cilco infinito.

O resultado deverá ser algo como:

Figura 11: Lena com o Sobel Horizontal aplicado

Figura 12: Lena com o Sobel Vertical aplicado

4ª Função:

A função **contour** recebe como argumentos os dois buffers aplicados com os sobels vertical e horizontal (bhorizontal e bvertical), efectua uma chamada de uma função auxiliar **CICLO2** que através de um ciclo while fazendo a soma de cada ponto, aplicando a seguinte fórmula:

$$C = \frac{1}{2}(B_h + B_v).$$

Figura 13: Ainda outra formula fornecida pelo docente

E depois aplicando a seguinte formula para inverter as cores a cada ponto:

$$D(i,j) = 255 - C(i,j)$$

Figura 14: Ainda outra formula fornecida pelo docente

Através destas duas formulas guarda-se o resultado no buffer **imagem_final**.

Para optimização de código escolheu aplicar-se directamente durante o ciclo while às duas formulas. Permitindo assim quando se efectua o chamamento dos pontos dos dois buffers (bhorizontal e bvertical) usando a instruçãp **lb** para que o ponto que entrasse no buffer imagem_final fosse o ponto final sem necessidade de passos adicionais. Ao terminar executa o **jr $ra** que volta para a função **contour** e dai volta para a função main.

5ª Função:

O branch **WRITE_TO_GRAY**, tem o único propósito de escrever num ficheiro (output), que neste caso é transferir a informação contida no buffer **imagem_final** para o ficheiro output. Ao terminar executa **jr $ra** onde volta para main e termina o programa.

```
9       #       output file, directoria de destino
10      #       sem directoria -> o destino e' mesma directoria do programa
11
12              file2: .asciiz "contornos4.GRAY"        #ficheiro output
13
```

Figura 15: Ficheiro de Output

2.4 Conversão de formato .GRAY para formato .jpg (exemplo)

Usando a linha de comando na directoria da imagem final (5º Passo), **convert -size 512x512 -depth 8 imagem.gray imagem.jpg** é nos permitido converter a imagem para a seguinte:

Figura 16: Lena (Resultado final)

3 Curiosidades

O Programa executa 39 017 632 instruções, tendo em conta que o processador tem 2.13 Ghz, excuta o programa em 0.0183181371 segundos

A imagem presente no enunciado do trabalho não se parece com a demonstrada anteriormente, portanto após varios testes, descobriu-se que para ficar igual à imagem aprensentada no enunciado tinha de haver uma modificação na formula:

$$C = \frac{1}{2}(B_h + B_v).$$

Esta modificação consiste apenas em multiplicar-se a formula acima por 2, a imagem, assim, seria:

Figura 17: Lena com a formula modificada aplicada

Que por si, é muito semelhante à do enunciado.

4 Conclusão

No final do programa podemos concluir que foi um sucesso, o programa faz o que tem a fazer sem erros ou bugs detectados. Conclui-se também que através deste trabalho ao aplicar todos os conhecimentos adquiridos em Arquitectura de Sistemas e Computadores contribuiu para um maior conhecimento e entendimento.

5 Bibiografia

Bibliografia usada:

Syscalls
http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html

ImageMagick
http://www.imagemagick.org/

obel Operator
http://en.wikipedia.org/wiki/Sobel_operator