Universidade de Évora

Arquitetura de Sistemas e Computadores

Relatório do Trabalho Prático

Discentes:

André Figueira – 31626

João Calhau – 31621

**Índice**

1. Objectivo
   1. Como é que este Objectivo vai ser atingindo?.............................................. Pág. 3
2. Descrição
   1. Esquema
   2. Conversão da imagem para formato RGB......................................................... Pág. 4
   3. O procedimento………………………………….Pág. 5
      1. .data segment…………………………….. Pág.5
      2. .text segment………………………… Pág6-11
   4. Conversão de formato .GRAY para formato .jpg (exemplo)……………………………………. Pág.12

3. Curiosidade………………………………………… …pág. 13

4. Conclusão………………………………………………..Pág.15

5. Bibliografia……………………………………..………Pág. 16

**1. Objectivo**

Pretende-se com este trabalho desenvolver um conjunto de funcões em assembly

Mips para detecção de contornos em imagem a cores. Dada uma imagem RGB, o

resultado final irá ser uma imagem em tons de cinzento, com fundo branco e com

traos escuros nos locais onde existem contornos na imagem original.

**1.1 Como e que este Objectivo vai ser atingindo?**

Este objectivo vai ser atingindo 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 ainda mais detalhada.

C:\Users\André\Downloads\lena512color.tiff

C:\Riot Games\lena_teste_113_final.tiff

**2. Descrição**

**2.1 Esquema**

main

READ\_RGB\_ IMAGE

RGB\_TO\_GRAY

CONVOLUTION

CONTOUR

WRITE\_GRAY\_ IMAGE

CICLO

CICLO2

MID

MARGENS

**2.1 Conversão da imagem para formato RGB**

Usando a linha de comando **sudo apt-get install ImageMagick** obtém-se

uma ferramenta capaz de conseguir atingir o objectivo de conversão para RGB.

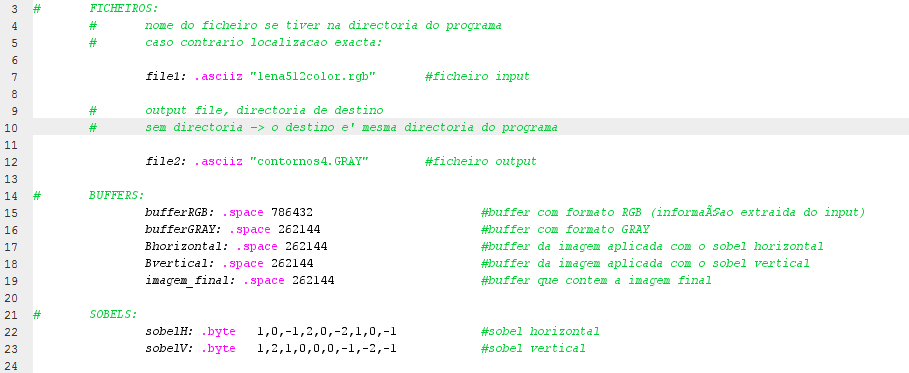
O próximo passo consiste em abrir o terminal e na directoria de onde se encontra a imagem que irá ser utilizada usar a linha de comando **convert** **example.tiff example.rgb**, sendo **.tiff** um formato de imagem (podendo ser este qualquer outra, por exemplo **.jpg**) como resultado obtemos uma imagem em formato RGB pronta a ser utilizada pelo programa.

**2.2 O procedimento**

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

**2.2.1 O .data segment**

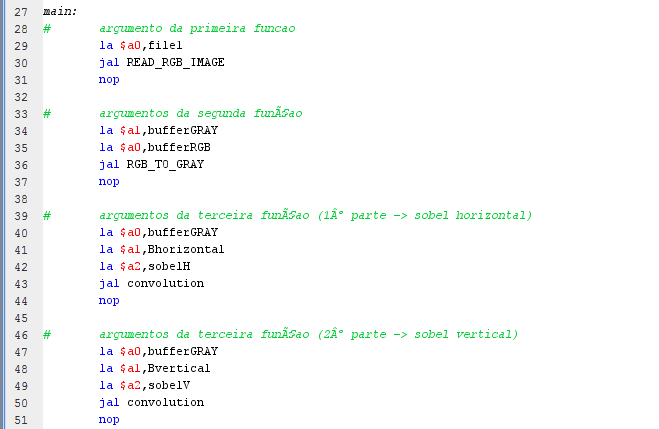
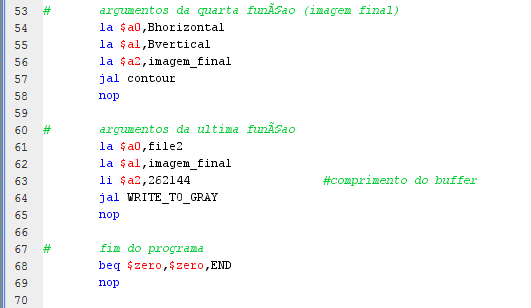
É nesta região da memória que se encontra todos os buffers utilizados, os sobels e contendo também localização do ficheiro de input e outputs. Como se pode ver pela imagem:



**2.2.2 O .text segment**

Neste segmento é onde 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**.

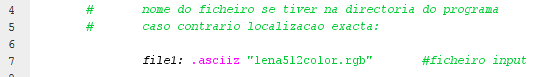
O **branch** mais importante neste programa é sem dúvida o **main,**



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

**1º Função:**

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



Ficheiro de input

**2º 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:

**I = 0.30R + 0.59G + 0.11B**

Esta fórmula serve para nos fornecer o valor em gray do pixel correspondente a 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 próximo pixel. Este ciclo é repetido **786432 + 3** (+3 bytes, ou seja mais 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 main pronta para passar para a próxima função.

O resultado da imagem deverá ser algo como:

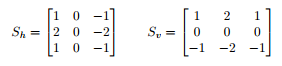


**3º Função:**

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

**

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



(utilizando também um load byte mas desta vez lb, porque é necessario haver extensão de sinal valor negativo)

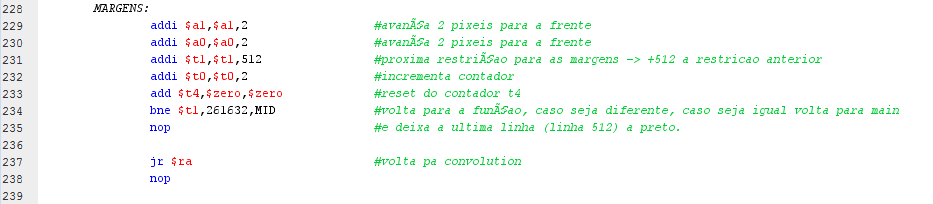
Permite então através de um ciclo **while** obter o valor do dito pixel aplicado com sobel e após ter sido utilizado a fórmula do somatório usar a seguinte formula:



que usando a instrução **sb (store byte)** guarda o resultado num novo buffer(1º chamada no main – bhorizontal, 2º chamada no main - bvertical). Repetindo o ciclo 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 preto por default), e 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:



Com esta função permite com que os valores na matriz bhorizontal e bvertical (nas margens) fiquem a preto por default.

Se o valor da restrição for igual ao valor da penúltima linha, a função volta para a função convolution e daí volta para main.

Em seguida será repetir esta função outra vez, para aplicar o sobel vertical a 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 em ciclo infinito.

O resultado deverá ser algo como:

C:\Riot Games\lena_bhorizontal.tiff

Imagem aplicada com sobel horizontal.

**C:\Riot Games\lena_bvertical.tiff**

Imagem aplicada com sobel vertical.

**4º Função:**

O branch  **contour,** esta função 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 faz a soma de cada ponto, aplicando a seguinte fórmula:



,

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

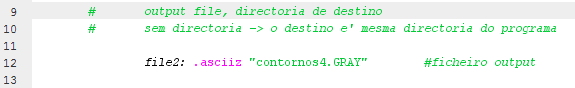


Através destas duas formulas guarda-se o resultado no buffer **imagem\_final**.

Para optimização de código escolheu-se aplicar directamente durante o ciclo while as duas fórmulas. Permitindo assim quando se efectua-se o chamamento dos pontos dos dois buffers (bhorizontal e bvertical) usando a instruço **lb** que o ponto que entra-se 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.

**5º 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 de output. Ao terminar executa **jr $ra** onde volta para main e termina o programa.

****

**2.3 Conversão de formato .GRAY para formato .jpg (exemplo)**

Usando a linha de comando na directoria da imagem final (em relação ao 6º Passo):



Com esta linha de comando irá permitir visualizar a imagem, que será:

C:\Riot Games\lena_teste_113_final.tiff

**3. Curiosidade**

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



Esta modificação consiste em que se multiplica-se a fórmula por 2, a imagem seria:

C:\Riot Games\lena_teste_112_final.tiff

Que por si, e 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. Bibliografia**

**Bibliografia usada:**

Syscalls

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

ImageMagick

<http://www.imagemagick.org/>

Sobel Operator

<http://en.wikipedia.org/wiki/Sobel_operator>