Negativo de uma imagem

Organização de Computadores – CC53B
UTFPR(Universidade Tecnológica Federal do Paraná), campus Ponta Grossa

Pedro Warmling Botelho
R.A.: 1839250

1 Introdução

Para este trabalho foi proposto o desenvolvimento de um programa escrito em Assembly do MIPS-32 que dada uma imagem PGM(Portable Gray Map) qualquer, apresente na saída outro arquivo PGM contendo o negativo da imagem.

Na fotografia, o negativo é uma imagem na qual as partes mais claras da fotografia aparecem escuras e as partes mais escuras aparecem claras. Uma imagem positiva é uma imagem normal, já uma imagem negativa é uma inversão total dela. Uma imagem colorida negativa é adicionalmente invertida na cor, com áreas vermelhas aparecendo ciano, verdes aparecendo magenta, azuis aparecendo amarelo e vice-versa. Essa inversão ocorre como resultado dos produtos químicos sensíveis a luz que um filme de câmera usa para capturar uma imagem que devido a exposição a luz, escurece. No caso das cores negativas, também são revertidas em suas respectivas cores complementares.

Para transformar a imagem da entrada em negativo foi considerado imagens em escala cinza onde 0 representa a cor preta e 255 representa a cor branca, sendo qualquer número entre 0 e 255 um tom de cinza diferente. Baseado nesse princípio, foi utilizado uma fórmula (equação 1) que transforma a cor presente na imagem em uma cor negativa.

Equação 1: y = -x + 255.

2 Desenvolvimento

2.1 Análise da estrutura de uma imagem PGM

```
P2
# feep.pgm
24 7
15
```

Figura 1: Cabeçalho de uma imagem PGM

Na primeira linha do arquivo PGM encontra-se o tipo, neste caso, P2. Na linha seguinte, há um comentário. Na terceira linha encontra-se as entradas X e Y que representam o tamanho da matriz que contém o código das cores, por exemplo 24 por 7, que em C seria *int matriz[24][7]*. Como em assembly MIPS-32 não podemos iniciar matrizes assim, poderíamos transformar esses dois números em um número só, sendo ele o número de código de cores que teremos que ler. Porém, ao invés de fazer isso, a matriz das cores será lida até ser o final do arquivo, pois facilitará a implementação.

Na quarta linha tem-se a regulagem de branco para saber o intervalo das cores, por exemplo, nessa imagem, teremos 0 para preto e 15 para branco, qualquer número entre 0 e 15 será um tom de cinza. Essas quatro linhas são lidas na entrada e reescritas na saída, porém são ignoradas em termos de utilização em código. O funcionamento será mostrado no seção 2.2.

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figura 2: Matriz das cores

A partir da quinta linha tem-se a matriz das cores, onde estão identificados os números que serão substituídos na equação 1 para que seja feita a conversão para o negativo da imagem.

2.2 Código em Assembly MIPS-32

A lógica do programa consiste basicamente em executar um laço de repetição na matriz das cores até chegar no fim do arquivo, lendo caractere por caractere, transformando o caractere em um inteiro, aplicando a fórmula e escrevendo o número final no arquivo de saída. A seguir será mostrado como isso foi feito.

Na figura 3 tem-se o *loop de descarte*, que, como dito anteriormente, irá descartar as quatro primeiras linhas do código pois não terão utilidade nenhuma na lógica do programa, estas quatro linhas serão apenas transportadas para o arquivo de saída sem qualquer modificação.

```
#le
move $a0, $t0
la   $a1, buffer_entrada
addi $a2, $zero, 1
addi $v0, $zero, 14
syscall
add $t3, $v0, $zero

#escreve
move $a0, $t1
la  $a1, buffer_entrada
add  $a2,  $zero, $t3
addi $v0, $zero, 15
syscall

lb $s4, buffer_entrada($zero)

beq $s4, $t7, fimDaLinha # é o fim da linha?
```

Figura 3: Loop de descarte

Agora chega-se em um ponto crucial do programa, que é a leitura dos códigos e a transformação para o negativo. Primeiramente, como a imagem PGM é uma imagem do tipo texto e manipularemos números inteiros, uma conversão de string para inteiro foi necessária. A lógica se estende ao seguinte, após a leitura do primeiro caractere, verica-se é um espaço, se não for, passa-se este caractere para a última posição de um vetor com 3 posições, pois o tamanho máximo de uma cor é um número com 3 digitos. Também foi necessário subtrair 48 de cada número lido, pois como há um caracter e precisamos de um inteiro, 48 é o número ascii onde se inicia os números decimais. O que foi explicado agora, está representado na figura 4.

```
# Lê o caracter
move $a0, $t0            # a0 recebe o arquivo aberto
la   $a1, buffer_entrada # a1 recebe o buffer de entrada
addi $a2, $zero, 1   # a2 recebe o tamanho maximo
addi $v0, $zero, 14      # syscall para ler do arquivo
syscall
add $t3, $v0, $zero # quantidade caracteres lidos
beq $t3, $zero, exit # é o fim do arquivo?

# Transforma o caracter lido em inteiro
lb $a1, buffer_entrada($zero)
beq $a1, $s4, ehEspaco # é um espaço?
beq $a1, $t7, ehEspaco # é \n?
addi $s3, $s3, 1
addi $a1, $a1, -48
```

Figura 4: Lê um caracter e transforma em um inteiro

Após esse processo estar concluído, o número inteiro que obteve-se por meio da subtração por 48 é passado para, no caso da primeira interação, a última posição do vetor de 3 posições para tratamento posterior. Como ilustra a figura 5.

```
# Insiro o caracter lido na posição do vetor
add $s5, $t4, $s5
sw $a1,0($s5)
addi $s6, $s6, -4
add $s5, $s6, $zero
```

Figura 5: Insere o caracter lido na posição do vetor

O laço é executado até que lê-se um espaço. Quando o espaço é lido, isso significa que temos o número a ser transformado salvo em um vetor de 3 posições, mesmo que uma ou duas das posições esteja vazia.

Então, entra-se na função que soma essas posições para que se forme um número e para que isso aconteça, as posições precisam ser multiplicadas. A primeira posição será multiplicada por 1 (figura 6), a segunda posição será multiplicada por 10 (figura 7) e a terceira posição será multiplicada por 100 (figura 8).

```
mult100:
        lw $s6, 8($t4)
        mul $s6, $s6, 100
        add $t5, $t5, $s6
        addi $s4, $s4, 1
        addi $s7, $s7, 1
        j somaPosicoes
```

Figura 6: Multiplicação por 1

```
mult10:
        lw $s6, 4($t4)
        mul $s6, $s6, 10
        add $t5, $t5, $s6
        addi $s4, $s4, 1
        addi $s7, $s7, 1
        j somaPosicoes
```

Figura 7: Multiplicação por 10

```
mult1:
        lw $s6, 0($t4)
        #mul $s6, $s6, 1
        add $t5, $t5, $s6
        addi $s4, $s4, 1
        addi $s7, $s7, 1
        j somaPosicoes
```

Figura 8: Multiplicação por 100

O resultado da multiplicação é somado e guardado em um registrador que utilizaremos agora para a equação 1, como é mostrado na figura 9.

```
formula:
        sub $t5, $t6, $t5
```

Figura 9: Executa a fórmula do negativo

Por fim, temos agora um número inteiro que está transformado em negativo. Porém, esse número ainda precisa ser convertido para caractere para que seja lido em uma imagem PGM.

Feito isso, o próximo passo é escrever o número no arquivo de saída para que assim se forme o negativo da imagem a cada interação do laço.

Este processo será executado até o fim do arquivo.

3 UTILIZAÇÃO

Na pasta onde se encontra o *codigo.asm* deve-se ter uma imagem do tipo PGM com o nome *entrada.pgm* como é mostrado na figura 10.

Figura 10: Pasta do programa

Tendo isso, o simulador MARS deve ser aberto e o arquivo *codigo.asm* executado normalmente.

4 RESULTADOS

Para entradas nas quais a aplicação de cada pixel na fórmula não resulta em um número com o último digito igual a zero ou números menores que 101, temos o negativo da entrada na imagem de saída. Esse processamento pode ser visto nas imagens seguintes.

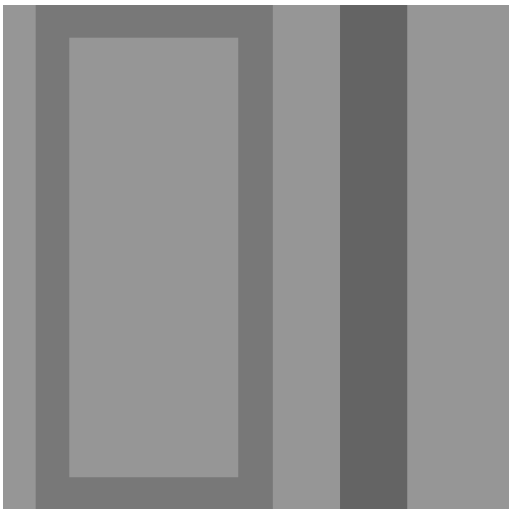

Figura 11: entrada.pgm

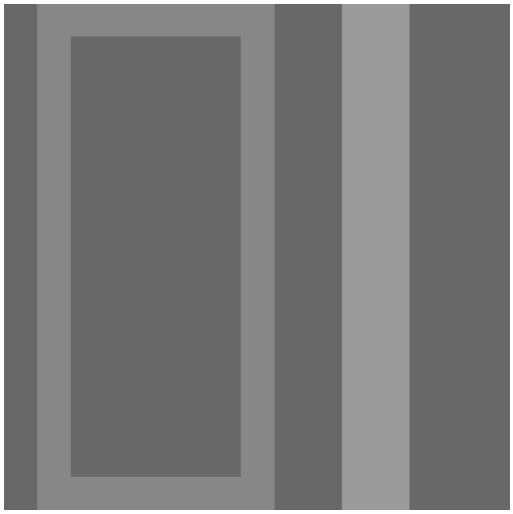

Figura 12: saida.pgm

Em relação aos problemas mencionados, as figuras seguintes demonstram os erros.

Figura 13: entrada.pgm

Figura 14: saidaerrada.pgm

REFERÊNCIAS

[1] Netpbm PGM. http://netpbm.sourceforge.net/doc/pgm.html

[2] Netpbm PPM. http://netpbm.sourceforge.net/doc/ppm.html

[3] Tony Brock, Syscall funtions available in MARS. https://courses.missouristate.edu/KenVollmar/mars/Help/SyscallHelp.html

[4] Charles Gordon, Going from C to MIPS Assembly. https://courses.cs.washington.edu/courses/cse378/00au/ctomips2.pdf

[5] Hobby electronics, ascii table for MIPS. http://www.hobbytronics.co.uk/ascii-character-set

[6] Prof. Dr. Paulo Miranda. http://www.vision.ime.usp.br/~pmiranda/mac2166_1s17/aulas/P3/pgm_ppm.pdf

[7] ASCII PGM Files http://people.sc.fsu.edu/~jburkardt/data/pgma/pgma.html