Universidade Federal do Rio Grande do Norte
Departamento de Engenharia de Computação e Automação

Relatório da Primeira Avaliação:

Obtendo o histograma de uma imagem em níveis de cinza usando Assembly para MIPS32

Aluno: Victor Emanuel Ribeiro Silva
Professor orientador: Diogo Pinheiro Fernandes Pedrosa
Disciplina: Arquitetura de Computadores

Maio
2018

Conteúdo

1 Introdução ao Problema

Neste trabalho está documentado o método desenvolvido pelo autor para se obter o histograma de uma imagem em níveis de cinza. O programa foi escrito em Assembly para arquitetura MIPS32 e testado no simulador MARS.

Imagens digitais são representadas no computador como vetores bidimensionais, matrizes. Cada elemento da matriz contém a informação de um pixel da imagem. Geralmente em imagens coloridas, adota-se o padrão: RGB (Red Green Bue) ou CMY (Cyan Magenta Yellow).

Uma imagem analógica de apenas uma cor pode ser representado matematicamente como uma função de duas variáveis $f(x, y)$ cujo valor no ponto (x, y) é proporcional ao brilho da imagem. Como trabalharemos em uma imagem em níveis de cinza, cada ponto está associado com uma intensidade dessa cor. Usaremos valores que ocupam 1 Byte na memória para representar todos os níveis possíveis, ou seja, é como se existissem 256 intensidades. Também adotaremos que 0 significa preto e 255 branco.

Para ser processada no computador, que é um sistema digital, a imagem deve ser transformada em um sinal digital. Isso ocorre através dos processos de Amostragem (discretização espacial) e Quantização (discretização em amplitude). A amostragem bidimensional transfere os valores de pontos no plano contínuo para uma matriz MxN. Já a Quantização, discretiza os valores em cada elemento em níveis em definidos, em nossa aplicação todos são inteiros positivos de 1 Byte. Nesse trabalho, a imagem já digitalizada, é uma matriz de $64x81$ com valores variando de 0 a 255.

O próximo passo foi construir o histograma com a distribuição de níveis de cinza. A alternativa mais simples foi criar um vetor cujas posições que devem ser preenchidas com a quantidade de pixels que apresenta determinada cor. Para isso, foi convencionado que a cor a ser representada por uma posição do vetor é dada por seu próprio índice. Isto é, a quantidade de pixels de cor I foi armazenada na posição I do vetor.

2 O Programa

2.1 Seção de Dados

Na seção de dados são declarados:

1. A matriz com os níveis de cinza de cada pixel da imagem variando de 0 a 255
2. O vetor H com 256 posições indexadas de 0 a 255
3. Lmax, inteiro com a quantidade de níveis de cinza
4. Dois valores contendo a dimensão da imagem em pixels, 64 x 81
5. Duas Strings contendo ":"e "\n"para deixar a exibição do histograma esteticamente melhor

```
H:      .word 0:256
Lmax:   .word 255
M:      .word 64
N:      .word 81
dp:     .asciiz ": "
nl:     .asciiz "\n"
```

2.2 Seção de Texto

O programa consiste de apenas três funções: a *main*, *imprimir_vetor* e *criar_hist*.

```
main:
        jal criar_hist
        jal imprime_vetor

        li $v0, 10
        syscall
```

Sobre a *main* não há muito o que se falar. Nessa aplicação, seu objetivo se resume a chamar as outras duas funções e finalizar o programa usando um *syscall*.

```
imprime_vetor:
        la $t1, H
        lw $t2, Lmax
        move $t3, $zero
        addi $t2, $t2, 1
        sll $t2, $t2, 2
        add $t2, $t2, $t1

        loop_print:
                li $v0, 1
                move $a0, $t3
                syscall
                li $v0, 4
                la $a0, dp
                syscall
                li $v0, 1
                lw $a0, 0($t1)
                syscall

                addi $t1, $t1, 4
                slt $t0, $t1, $t2
                beq $t0, 0, fim_print
                li $v0, 4
                la $a0, nl
                syscall
                addi $t3, $t3, 1

                j loop_print

        fim_print:
                jr $ra
```

A *imprimir_vetor* inicializa os registradores pondo em $\$t1$ o endereço base do vetor H e em $\$t2$ o endereço do último elemento de H mais quatro Bytes. Em outras palavras, como cada conjunto de 4 Bytes em H é indexado de 0 a 255, é como se $\$t2$ guardasse o endereço do elemento com índice 256 (que na verdade não existe). O motivo de ser guardado um endereço na memória localizada após o término do vetor será explicado com a instrução **BEQ** (Branch If Equal) mais adiante. O valor em $\$t2$ é dado pela fórmula: $\$t2 = (Lmax + 1) * 4 + addrBase$

O laço de repetição dentro da função faz o trabalho equivalente à estrutura FOR nas linguagens de alto nível. O iterador é o $\$t1$, que é incrementado sempre em quatro unidades, e a condição de parada é o valor em $\$t2$. Quando $\$t1 = \$t2$ significa que todos os elementos já foram impresso na tela e o curso da execução é desviado pelo BEQ para o *fim_print*, que por sua vez desvia-o para a função chamadora.

```
criar_hist:
        la $s0, H
        la $s1, imagem
        lw $t0, Lmax
        lw $t1, M
        lw $t2, N
        subi $t8, $t1, 1
        subi $t9, $t2, 1

        mul $t4, $t8, $t1
        add $t4, $t4, $t9
        sll $t4, $t4, 2
        add $t4, $t4, $s1

        for:

                lw $t5, 0($s1)
                sll $t5, $t5, 2
                add $t5, $t5, $s0

                lw $t3, ($t5)
                addi $t3, $t3, 1
                sw $t3, ($t5)

                addi $s1, $s1, 4
                blt $s1, $t4, for
        jr $ra
```

A *criar_hist* é a responsável por percorrer toda a matriz e preencher o vetor H com a frequência de cada nível de cinza. Como de costume, a primeira coisa a se fazer é preparar os registradores com os valores iniciais necessários. O endereço base da matriz está em $\$s0$ e o do vetor H em $\$s1$. Nos registradores temporários ($\$t0$ a $\$t9$) estão valores usados em cálculos de endereços ou servem só como variável auxiliar.

Função de cada registrador temporário:

- $\$t0$: Tamanho do vetor H
- $\$t1$: Número de linhas da imagem, M
- $\$t2$: Número de colunas da imagem, N
- $\$t3$: Valor de $H[f(x, y)]$

- $\$t4$: Endereço do último elemento da matriz mais 4 Bytes
- $\$t5$: $f(x, y)$
- $\$t8$: M-1
- $\$t9$: N-1

O código na última imagem implementa um laço de repetição contada cujo iterador é o registrador $\$s0$ e a condição de parada é o conteúdo de $\$s1$. Dentro do *for*, é carregado em $\$t5$ o nível de cinza de um pixel através da instrução **LW** (load word). O valor adquirido é usado para calcular o endereço na memória na qual está localizada a posição de H cujo valor deve ser incrementado em uma unidade. Isto é, a cada iteração do laço queremos ler um inteiro I que é a cor de um pixel, calcular em qual endereço da memória está a posição com índice I do vetor H e, por fim, incrementar seu valor. Para encontrar o endereço citado, foi usada a fórmula:

$$addrVetor = corPixel * 4 + addrBaseVetor$$

Como a maneira na qual a matriz é percorrida não importa, foi escolhida a mais facilmente implementável. A ordem na qual os elementos são acessados é a mesma na qual eles estão disposto na memória, linearmente. Para isso acontecer, o registrador iterador $\$s1$ é acrescido de 4 unidades a cada execução do laço. O que precisaria de dois laços aninhados em linguagens de programação de alto nível, aqui temos apenas um. Para finalizar o *for*, é realizada a comparação lógica $\$s1 < \$t4$. A instrução **BLT** (Branch If Less Then), desvia o curso da execução para a tag escrita em seu terceiro argumento se o resultado da comparação for verdadeiro.

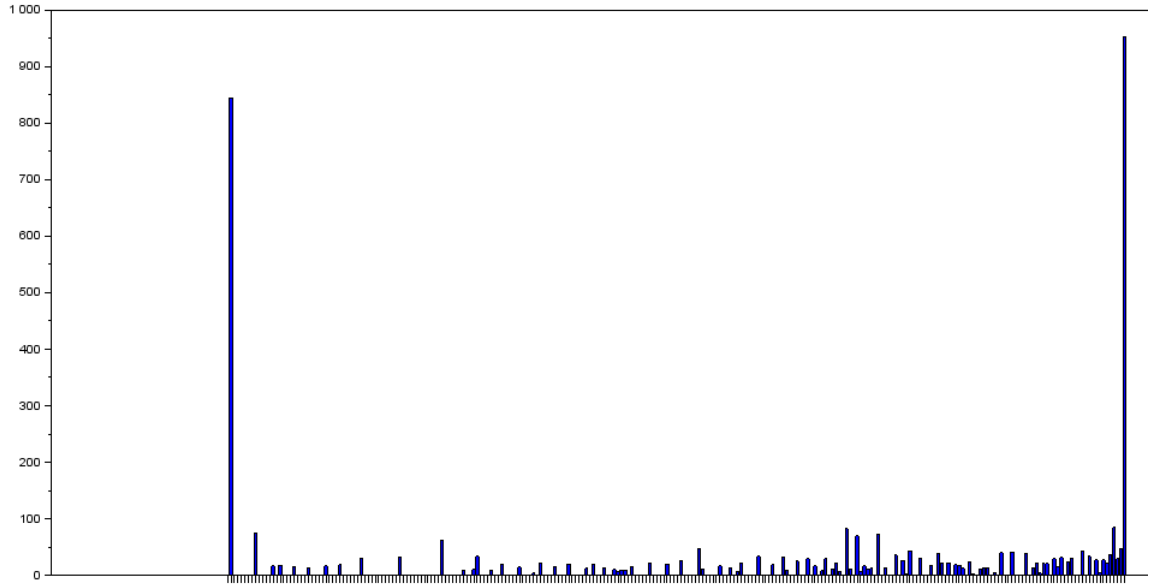
3 Análise dos Resultados

Histograma Final:

0: 0
1: 845
2: 0
3: 0
4: 0
5: 0
6: 0
7: 0
8: 75
9: 0
10: 0
11: 0
12: 0
13: 17
14: 0
15: 18
16: 0
17: 0
18: 0
19: 16
20: 0
21: 0
22: 0
23: 14
24: 0
25: 0
26: 0
27: 0
28: 17
29: 0
30: 0
31: 0
32: 19
33: 0
34: 0
35: 0
36: 0
37: 0
38: 30
39: 0
40: 0
41: 0
42: 0
43: 0
44: 0
45: 0
46: 0
47: 0
48: 0
49: 33
50: 0
51: 0
52: 0
53: 0
54: 0
55: 0
56: 0
57: 0
58: 0
59: 0
60: 0
61: 62
62: 0
63: 0
64: 0
65: 0
66: 0
67: 9
68: 0
69: 0
70: 10
71: 34
72: 0
73: 0
74: 0
75: 9
76: 0
77: 0
78: 19
79: 0
80: 0
81: 0
82: 0
83: 15
84: 0
85: 0
86: 0
87: 4
88: 0
89: 22
90: 0
91: 0
92: 0
93: 15
94: 0
95: 0
96: 0
97: 20
98: 0
99: 0
100: 0
101: 0
102: 12
103: 0
104: 19
105: 0
106: 1
107: 14
108: 0
109: 0
110: 10
111: 7
112: 9
113: 9
114: 0
115: 16
116: 0
117: 0
118: 0
119: 0
120: 22
121: 0
122: 0
123: 0
124: 0
125: 20
126: 0
127: 0
128: 1
129: 26
130: 0
131: 0
132: 0
133: 0
134: 46
135: 11
136: 0
137: 0
138: 0
139: 0
140: 17
141: 0
142: 0
143: 13

144: 0
145: 6
146: 22
147: 0
148: 0
149: 0
150: 0
151: 34
152: 0
153: 0
154: 0
155: 19
156: 0
157: 0
158: 32
159: 9
160: 0
161: 0
162: 25
163: 0
164: 0
165: 29
166: 0
167: 17
168: 0
169: 8
170: 29
171: 0
172: 11
173: 21
174: 7
175: 0
176: 83
177: 11
178: 0
179: 70
180: 7
181: 17
182: 11
183: 12
184: 0
185: 72
186: 0
187: 13
188: 0
189: 0
190: 36
191: 0
192: 26
193: 2
194: 43
195: 0
196: 0
197: 31
198: 0
199: 0
200: 18
201: 0
202: 39
203: 22
204: 0
205: 22
206: 0
207: 19
208: 18
209: 12
210: 0
211: 24
212: 2
213: 0
214: 10
215: 13
216: 14
217: 0
218: 5
219: 0
220: 40
221: 0
222: 1
223: 41
224: 0
225: 0
226: 0
227: 38
228: 0
229: 13
230: 21
231: 4
232: 21
233: 21
234: 0
235: 29
236: 16
237: 32
238: 0
239: 24
240: 29
241: 0
242: 0
243: 43
244: 0
245: 34
246: 0
247: 27
248: 4
249: 27
250: 21
251: 37
252: 85
253: 29
254: 46
255: 952

O histograma construído em Scilab pode ser visto na próxima imagem. A barra mais à esquerda representa a quantidade de 0 e, no lado oposto, a com quantidade de 255.


1 000
900
800
700
600
500
400
300
200
100
0