Universidade Tecnológica Federal do Paraná
Laboratório 3 - Sistemas Embarcados
Aluno: Felipe Alves Barboza

1 Introdução

A solução do laboratório deve gerar um histograma conforme os tons de cinza de uma imagem fornecida. A partir da imagem, é possível mapear os píxels em forma de matriz, em que cada pixel possui um valor entre 0 (preto) e 255 (branco), desta forma é possível realizar a contagem de cada valor e separá-lo em forma de histograma.

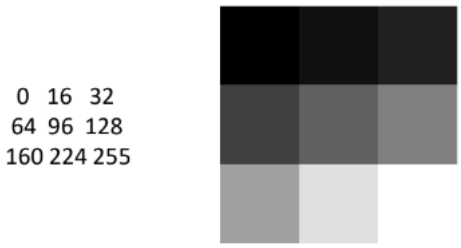

Figura 1: Escala de cinza

2 Estudo da plataforma

Como a maior parte da solução foi desenvolvida em código Assembly, o estudo foi focalizado em suas funções, do processador Cortex M4. Como esperado, seriam necessários acessos à memória, passagem de parâmetro por funções, ciclos, condicionais e branches. Com isso em mente, foram encontradas as seguintes especificações:

Operation		§	Assembler	S updates	Action	Notes
Add	Add		`ADD{S} Rd, Rn, <Operand2>`	N Z C V	Rd := Rn + Operand2	N
	with carry		`ADC{S} Rd, Rn, <Operand2>`	N Z C V	Rd := Rn + Operand2 + Carry	N
	wide	T2	`ADD Rd, Rn, #<imm12>`		Rd := Rn + imm12, imm12 range 0-4095	T, P
	saturating {doubled}	5E	`Q{D}ADD Rd, Rm, Rn`		Rd := SAT(Rm + Rn) doubled: Rd := SAT(Rm + SAT(Rn * 2))	Q

Figura 2: Adicionar

If-Then	If-Then	T2	`IT{pattern} {cond}`	Makes up to four following instructions conditional, according to pattern. pattern is a string of up to three letters. Each letter can be T (Then) or E (Else). The first instruction after IT has condition cond. The following instructions have condition cond if the corresponding letter is T, or the inverse of cond if the corresponding letter is E. See Table **Condition Field** for available condition codes.	T, U

Figura 3: Condicional

Branch	Branch		`B <label>`	PC := label. label is this instruction ±32MB (T2: ±16MB, T: –252 - +256B)	N, B
	with link		`BL <label>`	LR := address of next instruction, PC := label. label is this instruction ±32MB (T2: ±16MB).	
	and exchange	4T	`BX Rm`	PC := Rm. Target is Thumb if Rm[0] is 1, ARM if Rm[0] is 0.	N
	with link and exchange (1)	5T	`BLX <label>`	LR := address of next instruction, PC := label, Change instruction set. label is this instruction ±32MB (T2: ±16MB).	C
	with link and exchange (2)	5	`BLX Rm`	LR := address of next instruction, PC := Rm[31:1]. Change to Thumb if Rm[0] is 1, to ARM if Rm[0] is 0.	N
	and change to Jazelle state	5J	`BXJ Rm`	Change to Jazelle state if available	
	Compare, branch if (non) zero	T2	`CB{N}Z Rn,<label>`	If Rn [== or !=] 0 then PC := label. label is (this instruction + 4-130).	N,T,U
	Table Branch Byte	T2	`TBB [Rn, Rm]`	PC = PC + ZeroExtend(Memory(Rn + Rm, 1) << 1). Branch range 4-512. Rn can be PC.	T, U
	Table Branch Halfword	T2	`TBH [Rn, Rm, LSL #1]`	PC = PC + ZeroExtend(Memory(Rn + Rm << 1, 2) << 1). Branch range 4-131072. Rn can be PC.	T, U

Figura 4: Branch

Single data item loads and stores		§	Assembler	Action if <op> is LDR	Action if <op> is STR	Notes
Load or store word, byte or halfword	Immediate offset		`<op>{size}{T} Rd, [Rn {, #<offset>}]{!}`	Rd := [address, size]	[address, size] := Rd	1, N
	Post-indexed, immediate		`<op>{size}{T} Rd, [Rn], #<offset>`	Rd := [address, size]	[address, size] := Rd	2
	Register offset		`<op>{size} Rd, [Rn, +/-Rm {, <opsh>}]{!}`	Rd := [address, size]	[address, size] := Rd	3, N
	Post-indexed, register		`<op>{size}{T} Rd, [Rn], +/-Rm {, <opsh>}`	Rd := [address, size]	[address, size] := Rd	4
	PC-relative		`<op>{size} Rd, <label>`	Rd := [label, size]	Not available	5, N
Load or store doubleword	Immediate offset	5E	`<op>D Rd1, Rd2, [Rn {, #<offset>}]{!}`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	6, 9
	Post-indexed, immediate	5E	`<op>D Rd1, Rd2, [Rn], #<offset>`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	6, 9
	Register offset	5E	`<op>D Rd1, Rd2, [Rn, +/-Rm {, <opsh>}]{!}`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	7, 9
	Post-indexed, register	5E	`<op>D Rd1, Rd2, [Rn], +/-Rm {, <opsh>}`	Rd1 := [address], Rd2 := [address + 4]	[address] := Rd1, [address + 4] := Rd2	7, 9
	PC-relative	5E	`<op>D Rd1, Rd2, <label>`	Rd1 := [label], Rd2 := [label + 4]	Not available	8, 9

Figura 5: Store and load

3 Design da solução

Com esse embasamento teórico, foi possível criar o design da solução. Inicialmente a criação do vetor para o histograma composto de inteiros sem-sinal de 16 bits. Após isso, foi necessário a codificação da função em assembly, a qual seguiu os seguintes passos:

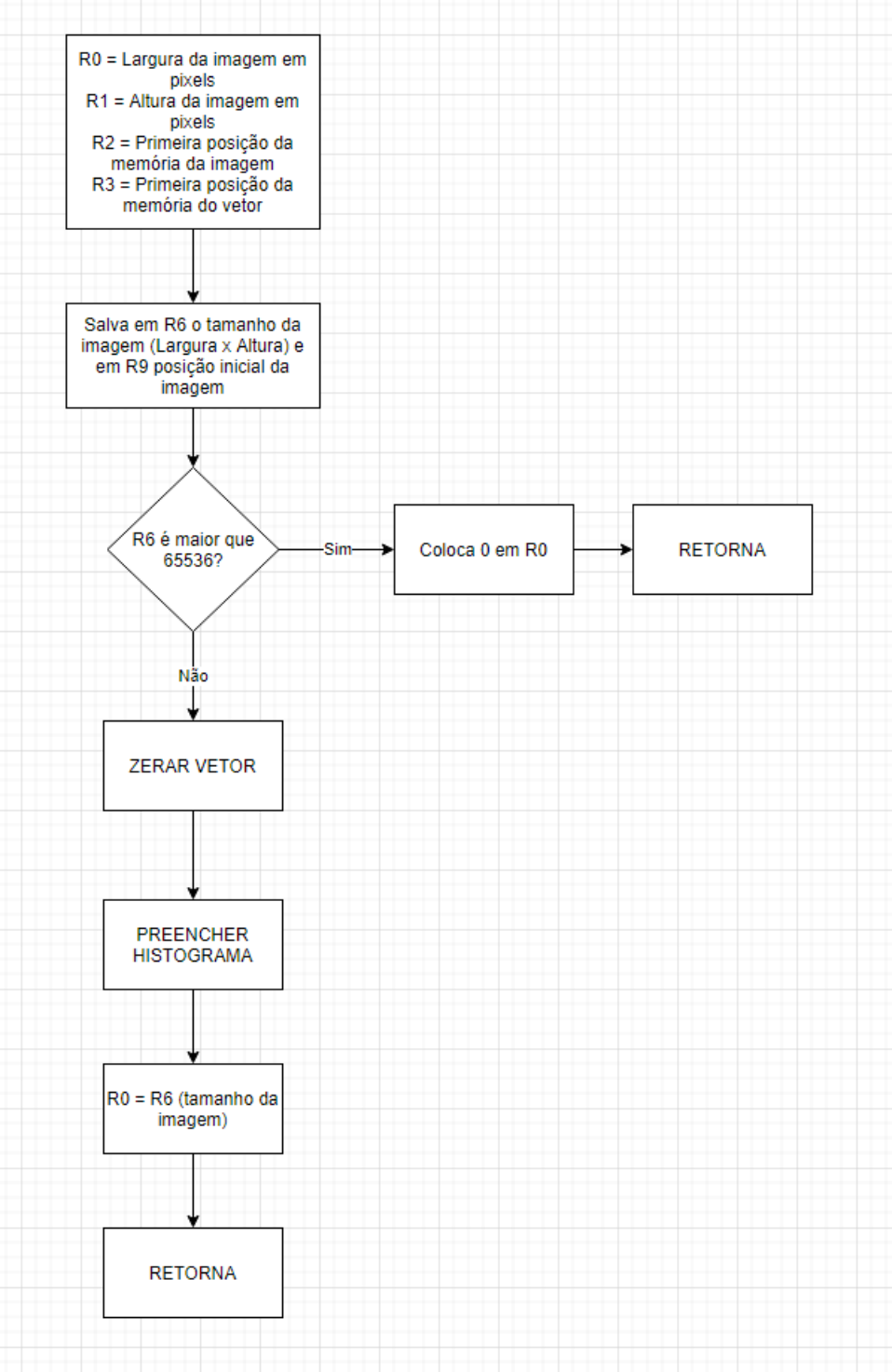

Figura 6: Fluxo completo do algorítmo

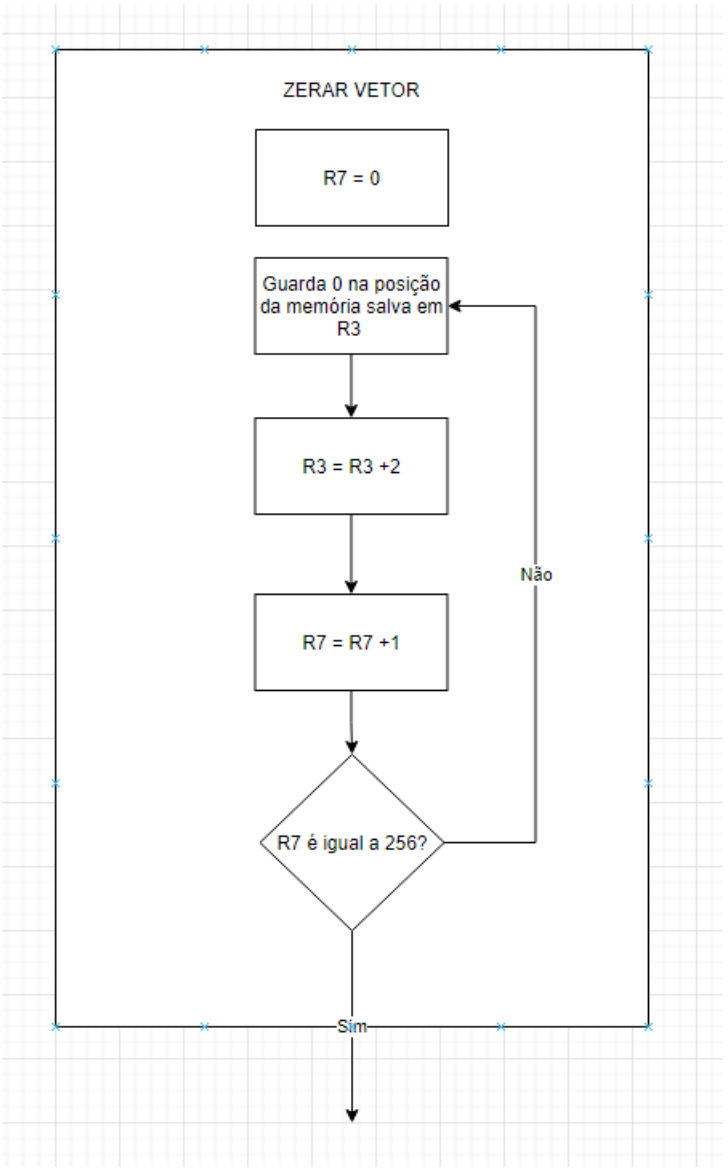

Figura 7: Zera o vetor do histograma

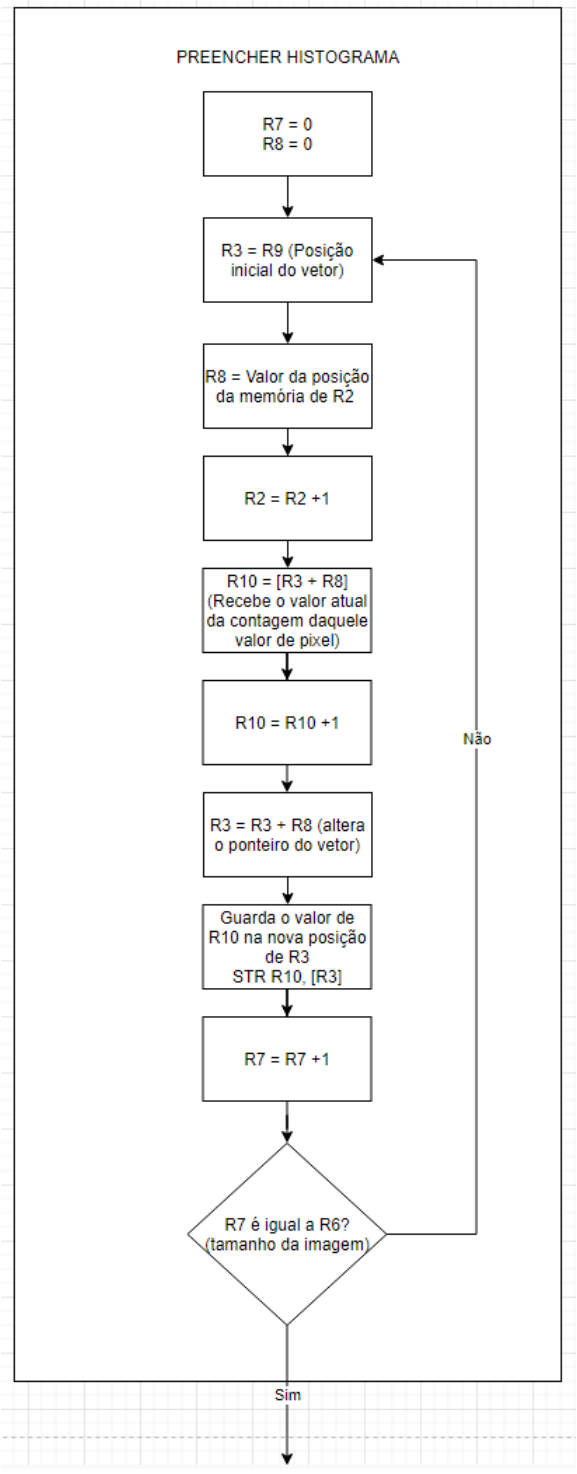

Figura 8: Preenche o vetor conforme imagem

4 Resultados

Além disso foi elaborado um algorítmo em python para gerar o histograma conforme o resultado do código e comparado com o histograma fornecido, concluindo que a prática foi concluída de forma correta.

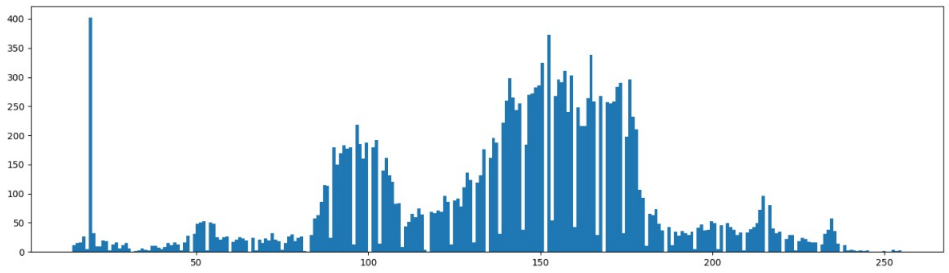

Figura 9: Resultado do algorítmo

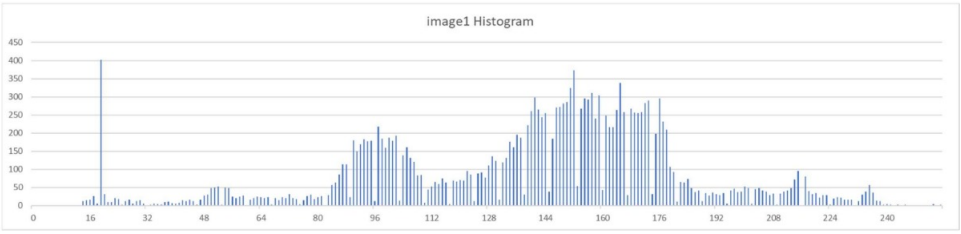

Figura 10: Base fornecida