UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Relatório do Lab 3

Disciplina de Sistemas Embarcados – Prof. Douglas Renaux

Autores: João Felipe Sarggin Machado e Waine Barbosa de Oliveira Junior

Versão: 12-Set-2022

1 Introdução

Durante o desenvolvimento de aplicações de sistemas embarcados, as linguagens e tecnologias utilizadas são, usualmente, de baixo nível (quando comparadas às de outras áreas, como Web). Isso permite, além do bom desempenho, um controle de memória adequado às arquiteturas de cada microcontrolador, com a alocação de recursos adequada à cada aplicação, e o uso dos GPIOs e outros periféricos mapeados na memória.

Para tais sistemas é comum o uso de C e C++, linguagens de alto nível se comparadas ao Assembly. O uso de tais facilita a escrita e compreensão do código, além de permitirem um nível de abstração muito grande nas aplicações. Na maioria dos casos, com as otimizações do compilador, o desempenho é praticamente o mesmo de um código de Assembly escrito à mão.

Porém em partes críticas de desempenho, como interrupções, trocas de contexto, ou rotinas que devem ser executadas o mais rápido possível, o desempenho de linguagens de alto nível pode não ser bom o suficiente. Em tais casos é comum reescrever a rotina em Assembly de uma maneira otimizada e utilizando conhecimentos do domínio do problema não conhecidos pelo compilador. Tendo em vista a proximidade entre C e Assembly, essa integração segue alguns padrões e é trivial de ser implementada.

Neste laboratório é desenvolvida uma aplicação com chamadas de C para Assembly, seguindo os padrões AAPCS (*Procedure Call Standard for Arm Architecture*). O principal objetivo é o aprendizado de como integrar uma aplicação em C com código em Assembly.

Para isso, o problema apresentado é o cálculo do histograma de uma imagem de 8 bits. A função em Assembly deve contar o número de pixels com cada valor, salvar em um vetor e retornar o número de pixels na imagem. Já o código em C deve preparar os dados para chamada da função, chamá-la e então imprimir o histograma resultante.

2 Planejamento das fases do processo de desenvolvimento

As atividades de desenvolvimento foram planejadas conforme sugerido pelo professor. A lista de atividades está abaixo:

1. Planejamento das fases do processo de desenvolvimento.
2. Definição do problema a ser resolvido.
3. Especificação da solução.
4. Estudo da plataforma de HW (placa Tiva e seu processador).
5. Projeto (design) da solução:
    a. Planejamento das estruturas de dados;
    b. Forma de passagem de parâmetros;
    c. Algoritmo e alocação de variáveis aos registradores.
6. Configuração do projeto na IDE (IAR).
7. Edição do código da solução.
8. Teste e depuração.

A documentação dos resultados foi feita conforme a realização de cada atividade.

3 Definição do problema a ser resolvido

Na área de processamento digital de imagens é muito comum o uso de histogramas e seus valores para calcular médias, medianas, picos, entre outras propriedades da imagem. Essas são utilizadas para diversos processamentos, como ajuste de brilho e contraste, checar e corrigir o balanço de cores da imagem, cálculo de thresholds, entre vários outros propósitos.

O problema especificado consiste no cálculo do histograma de uma imagem. Dada uma imagem de entrada em 8 bits e suas dimensões, calcular o número de ocorrência de cada valor, salvar num histograma e após isso retornar o número de pixels na imagem.

Computacionalmente deve ser desenvolvida uma função em Assembly com os parâmetros de entrada:

- *image width*: número de pixels em uma linha da imagem.
- *image height*: altura da imagem em pixels.
- *starting address:* endereço do primeiro pixel da imagem.
- *histogram*: endereço inicial de um vetor de tamanho 256. Cada posição do vetor armazena um inteiro sem sinal de 16-bits. Este vetor não possui dados válidos quando a função é chamada. Ele é usado apenas para o retorno da função.
- *Retorno*: inteiro sem sinal de 16 bits indicando o número de pixels processados.

As restrições do problema são:

- O tamanho máximo da imagem é de 64K (65.536) pixels.
    - A função retorna o valor 0 se o tamanho da imagem for superior a 64K.
- A declaração da função deve ser:
    - *uint16_t EightBitHistogram(uint16_t width, uint16_t height, uint8_t * p_image, uint16_t * p_histogram);*

4 Especificação da solução

A solução se dará por meio de um algoritmo simples escrito em instruções assembly ARM que será chamado a partir da função main escrita em C++. A passagem de argumentos para essa função se dará pelos registradores R0, R1, R2 e R3 e o retorno será feito pelo registrador R0, conforme especifica o Padrão de Chamada de Procedimento da Arquitetura ARM (AAPCS - ARM Architecture Procedure Call Standard).

Primeiramente o algoritmo do histograma deverá multiplicar o número de pixels da altura pela largura, para se obter o tamanho da imagem e verificar se esse tamanho é menor do que 64K (65.536) pixels, que é o primeiro valor acima do valor limite que uma variável do tipo *uint16_t* pode armazenar (2^{16} - 1 = 65.535), e nesse caso deve retornar 0. Essa especificação atende a primeira restrição do problema.

Em seguida, o algoritmo deverá limpar os 256 *half-words* que compõem o espaço de memória reservado para a contagem do histograma. Esse número (256) corresponde aos valores possíveis de pixels em uma imagem de tons de cinza de 8 bits por pixel (2^8 = 256). Essa etapa representa um laço e poderá ser realizada em 128 iterações, já que é possível limpar dois *half-words* de uma vez, utilizando escrita em *word*.

Na sequência o algoritmo deverá preencher o histograma, e para isso, deverá acessar o endereço de cada pixel na memória, obter o valor do pixel, utilizar esse valor como índice do histograma, obter a contagem de pixels daquele valor no histograma, somar 1 (um) a esse valor lido e escrever de volta no mesmo índice em que foi lido do histograma. Essa etapa representa um laço, e poderá ser realizada em "altura da imagem" * "largura da imagem" iterações.

Por fim, o algoritmo deverá retornar por meio do registrador R0 a quantidade de pixels processados.

Após o retorno da quantidade de pixels processada, a função main escrita em C++ irá se responsabilizar por apresentar os resultados, mostrando uma mensagem de erro caso o tamanho da imagem ultrapasse 64K pixels, ou mostrando o histograma de forma textual por meio do terminal do IAR na forma de duas colunas: valor do pixel e contagem de ocorrências.

5 Estudo da plataforma de HW

A placa Tiva TM4C1294NCPDT possui um processador de 32 bits que pertence à arquitetura ARMv7-M e utiliza o conjunto de instruções Thumb2. Esse conjunto mistura instruções de 16 (instruções *narrow*, que podem possuir sufixo .N) e 32 bits (instruções *wide*, que podem possuir sufixo .W) e é importante notar que todas as instruções devem estar alinhadas em *half-word*, ou seja, devem ser armazenadas em um endereço par.

Também é importante notar que uma vez que as instruções são armazenadas em endereços pares, o bit menos significativo do registrador PC (PC[0]) não é usado como um bit de endereço, ao invés disso, ele é copiado para o bit 24 do registrador EPSR (*Execution Program Status Register*) que é conhecido como bit T e é usado para indicar que o conjunto de instruções em uso é Thumb. Como a família Cortex-M possui apenas o conjunto de instruções Thumb, este bit deve estar sempre ligado, caso contrário, uma falha de instrução desconhecida ocorrerá e o processador entrará em um estado de trava, caso uma rotina de tratamento não seja escrita para resolver essa falha grave (*Hard Fault*).

6 Estudo da plataforma de SW

Certamente diversas instruções serão necessárias para a implementação da solução, podendo-se destacar algumas essenciais de antemão e dividi-las em grupos:

Instruções aritméticas:

MOV{S}{cond} Rd, Operand2 - Será necessário carregar valores iniciais para servir de índice nos laços que o algoritmo usará; Para carregar o valor 0 (zero) que será utilizado para limpar o espaço de memória correspondente ao histograma e por fim, para carregar os valores de retorno da função.

A instrução MOV custa 1 ciclo de clock neste caso, pois o registrador de destino não é PC.

ADD{S}{cond} {Rd,} Rn, Operand2 - Será necessário realizar somas para atualizar os índices de acesso à memória nos laços e também para somar 1 (um) à contagem de pixels de cada valor durante as iterações do laço de leitura da imagem.

A instrução ADD custa 1 ciclo de clock neste caso, pois o registrador de destino não é PC.

MUL{S}{cond} {Rd,} Rn, Rm - Será necessário multiplicar a altura e a largura da imagem para saber quantos pixels a imagem possui.

A instrução MUL custa 1 ciclo de clock.

Instruções de comparação e blocos condicionais:

CMP{cond} Rn, Operand2 - Será necessário realizar comparações entre o tamanho da imagem (quantidade de pixels) e o tamanho limite permitido (especificado pela restrição do problema). Também será necessário realizar comparações entre os índices utilizados nos laços e o valor limite deles, para que os laços não fiquem executando indefinidamente.

A instrução CMP custa 1 ciclo de clock.

IT{x{y{z}}} cond - Possivelmente será necessário utilizar blocos *If-Then* em conjunto com as funções CMP para retornar 0 (zero) pelo registrador R0 caso o tamanho da imagem seja maior do que 65.535 pixels ou dentro dos laços para atualizar índices e realizar saltos.

A instrução IT custa 1 ciclo de clock, mas pode custar 0 caso seja dobrada em uma instrução Thumb de 16 bits anterior.

Instruções de salto:

BX{cond} Rm - Será necessário realizar saltos para retorno da chamada de função utilizando o registrador LR.

A instrução BX custa 1 + P ciclos de clock, onde P é o número de ciclos necessários para preencher o pipeline, que pode ser de 1 a 3, dependendo do alinhamento da instrução alvo (local do salto) e se o processador consegue predizer o salto antecipadamente com sucesso.

B{cond} label - Será necessário realizar saltos condicionais para o início dos laços, para que as operações necessárias sejam repetidas na próxima iteração.

A instrução B custa 1 + P ciclos de clock, conforme explicado acima.

Instruções Load/Store:

STR{type}{cond} Rt, [Rn, Rm {, LSL #n}] - Será necessário escrever 0 (zero) nos espaços de memória pertencentes ao histograma antes de gerá-lo. Também será necessário escrever os valores atualizados na memória (somar 1 em cada valor) das contagens de cada valor de pixel no histograma.

A instrução STR custa 2 ciclos de clock para as variantes *Byte*, *Half-Word* e *Word*.

LDR{type}{cond} Rt, [Rn, Rm {, LSL #n}] - Será necessário ler os valores de cada pixel da imagem na memória e também de ler os valores da contagem dos pixels no histograma antes de atualizá-los (somar 1 em cada valor).

A instrução LDR custa 2 ciclos de clock para as variantes *Byte*, *Half-Word* e *Word*.

PUSH{cond} reglist - Talvez seja necessário salvar o valor de alguma variável na pilha caso a função utilize mais de 5 registradores (Já que os registradores R0, R1, R2, R3 e R12 são registradores de "scratch" que são armazenados na pilha automaticamente quando uma rotina é chamada).

A instrução PUSH custa 1 + N, onde N é o número de registradores envolvidos.

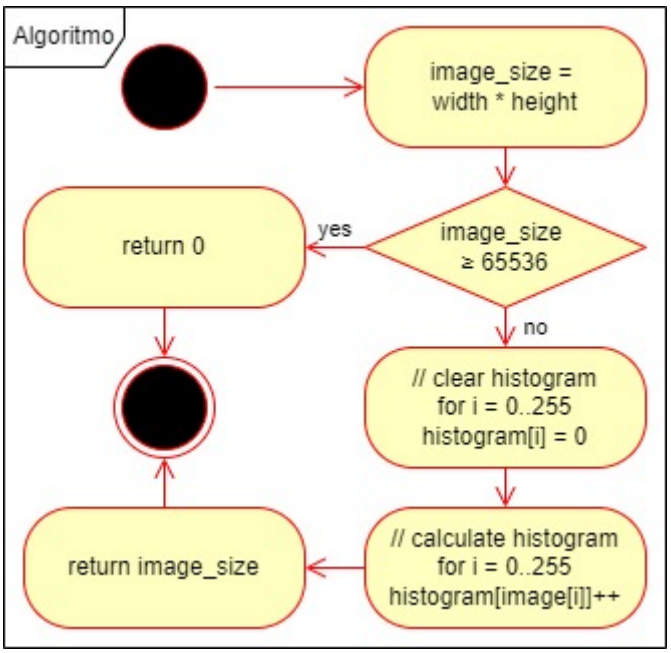
POP{cond} reglist - Caso mais de 5 registradores sejam utilizados na função, antes do retorno da chamada de função, seus valores originais devem ser restaurados.

A instrução POP custa 1 + N, onde N é o número de registradores envolvidos. Se PC estiver na lista de registradores então o custo é 1 + N + P, onde P é o número de ciclos necessários para preencher o pipeline, conforme explicado nas instruções de salto.

Obs.: Todas considerações sobre o número de clocks utilizados consideram o pipeline cheio, exceto quando explicitamente dito o contrário

7 Projeto (design) da solução

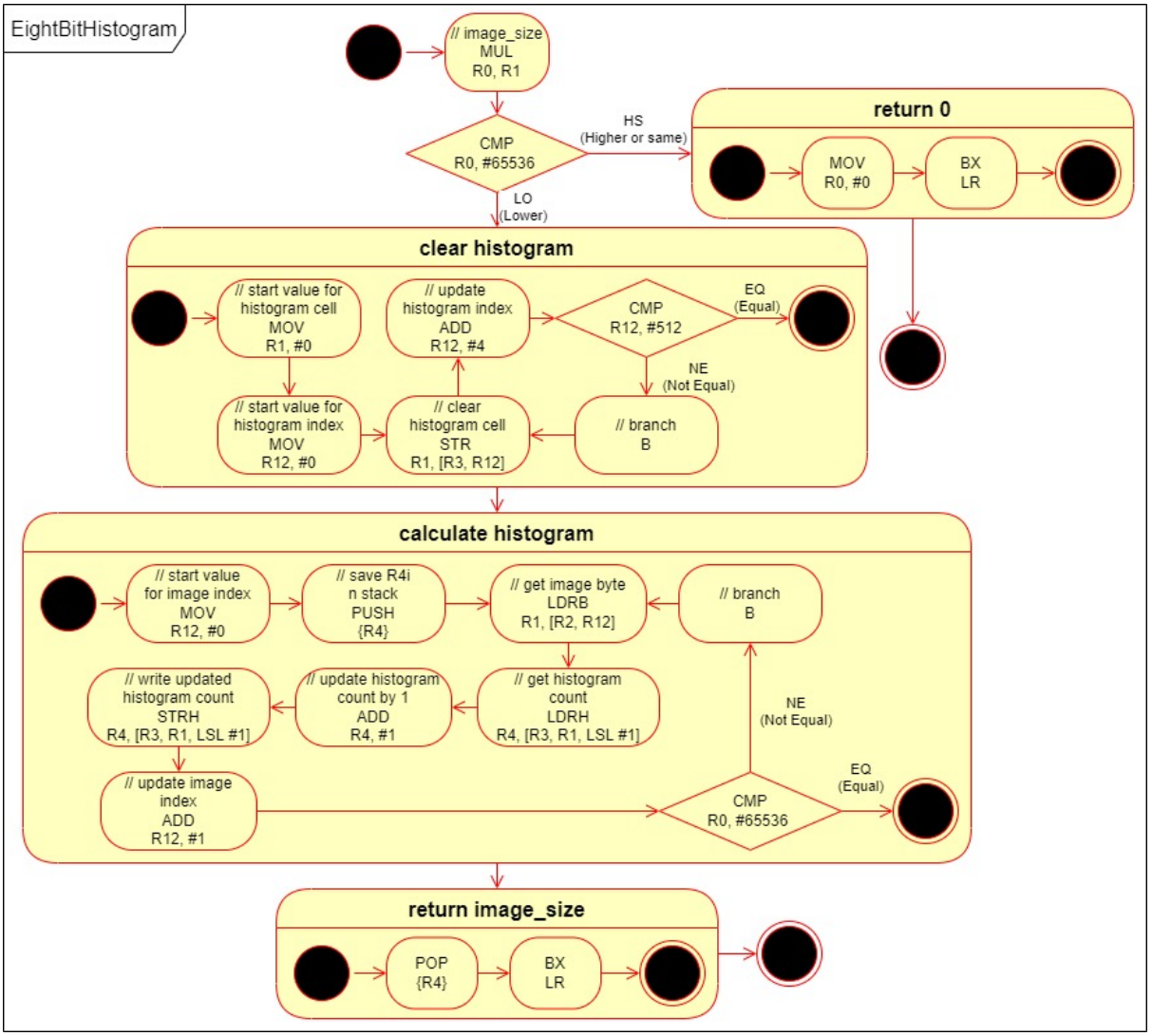
A solução será implementada conforme o seguinte algoritmo:



A passagem de parâmetros para a função, segundo os padrões AAPCS (*Procedure Call Standard for Arm Architecture*) se dará da seguinte forma:

Registrador	**Tipo**	**Variável**	**Comentário**
R0	uint16_t	width	Largura da imagem (em pixels)
R1	uint16_t	height	Altura da imagem (em pixels)
R2	uint8_t const*	p_image	Endereço do primeiro pixel da imagem
R3	uint16_t*	p_histogram	Endereço do primeiro pixel do histograma

Por fim, planeja-se uma sequência de instruções assembly em cada uma das etapas do algoritmo da seguinte forma:



8 Configuração do projeto na IDE (IAR).

As configurações básicas incluem a seleção do dispositivo "Texas Instruments TM4C1294NCPDT", o "VFPv4 single precision" como versão da unidade de ponto flutuante (FPU) e a configuração completa ("Libc++") da biblioteca de tempo de execução C/C++17.

As formatações das funções "printf" e "scanf" estão definidas como "ful", ou seja, formatação total. A seleção de Heap está no modo automático.

Já o compilador está definido para a linguagem C++ e em conformidade com a norma padrão com extensões do IAR. A otimização está definida para o nível baixo.

9 Teste e depuração.

Para validação da função escrita em assembly, será escrita uma função com a mesma finalidade em C++, seguindo o algoritmo apresentado no documento de descrição do laboratório 3 e será comparado ambas as saídas, espera-se que elas sejam idênticas.

Adicionalmente será escolhido alguns valores de pixels aleatoriamente e por meio de um editor de texto que possua a ferramenta de procura por expressões com contagem, será verificado se a contagem apresentada pelo editor de texto (no documento image.c)

corresponde com a contagem para aquele mesmo valor de pixel apresentado pelo algoritmo em assembly.