Processamento de imagens RGB em assembly

Fidelis Silva nº54761
Enoque Masasu nº53235

Maio
Ano Letivo 2022/2023

Arquitetura de Sistemas e Computadores I

Prof. Miguel Barão

Índice

1 Introdução

Este relatório apresenta o projeto de Arquitetura de Computadores 1, que tem como objetivo desenvolver um programa em linguagem Assembly para processamento de imagens RGB. O projeto visa demonstrar o entendimento dos conceitos teóricos de arquitetura de computadores, assim como a aplicação prática dos mesmos. O relatório consiste maioritariamente nas explicações dos passos envolvidos na execução de cada função ao longo do código.

2 Funções

2.1 Função strcmp

Esta função tem como objetivo comparar duas strings e verificar se ambas são iguais ou não. Essa função é utilizada futuramente na função ask_character. Ela utiliza os seguintes registos:

- a0, a1: Argumentos da função em que a0 é a string recolhida através do syscall 54, e a1 que é a segunda string que já foi definida nas variáveis globais
- t0, t1, t2: Registos temporários utilizados para fazer load de bytes e calculos intermédios.
- s0: Registo em que fica salvo o valor final da função (0 caso diferente, 1 caso igual)

A sequência de operações realizadas pela função são as seguintes:

1. Ao receber os argumentos, primeiramente (devido ao syscall 54) é preciso verificar se após a leitura da string, foi adicionado o caractere 'n', e caso sim, devemos removê-lo.
2. É iniciado um loop para que se ache o fim da string (caractere '0').
3. Após achado o último elemento, ajustamos o endereço de a0 para o caractere anterior a esse (que seria 'n'). Caso seja 'n', este é substituído por '0', caso contrário segue em frente. Utilizamos o código ascii para 'n', que no caso é 10.
4. Após removido o caractere, é começada a comparação entre as duas strings. É feito load de cada byte em ambas strings e estes são comparados. Caso sejam iguais, este procedimento continua, avançando os bytes, até que se ache o caractere '0'.
5. Caso as strings não sejam iguais em um dos bytes (em que a ordem destes é relevante), igualamos o registo s0 a 0. Caso ambas strings sejam iguais até que se encontre o caractere '0' de ambas, igualamos s0 a 1.
6. Ao retornar a função, o valor desejado está salvo em s0.

2.2 Função ask character

Função ask_character

A função ask_character tem como propósito verificar se a string inserida pelo usuário é um personagem válido ou não. Caso não seja válido, ela apresenta uma mensagem de erro e recolhe uma nova string do usuário, até que seja escolhido um personagem válido. Utilizamos os seguintes registos nesta função:

- `a0`: Único argumento da função, é a string recolhida do usuário.
- `a1`: contém os endereços da string correspondente aos personagens Yoda, Darth Maul e Mandalorian, em momentos diferentes da execução da função.
- `t3`: Registo temporário usado para comparações na função.
- `s0`: Registo onde é usado o valor do resultado da função strcmp.

Os registros são utilizados nas seguintes tarefas:

1. É adicionado no registo Stackpointer o nosso return address, já que utilizaremos outra função dentro desta.
2. Entramos num loop onde obtemos o valor em s0 do resultado de strcmp entre a string recebida e as strings Yoda, Darth Maul e Mandalorian respectivamente.
3. Caso uma delas seja igual a 1, redirecionamos o fluxo da função para Labels que vão atribuir os valores: 1 (caso seja Yoda), 2 (caso seja Darth Maul), e 3 (caso seja Mandalorian). E este valor é o retorno da função.
4. Caso s0 seja igual a 0 em todas as situações, é pedido uma nova string do utilizador, e juntamente apresentamos uma mensagem de erro. Este procedimento é repetido até que a string inserida seja válida e igual a uma das disponíveis para a escolha dos personagens.
5. Ao fim da função, é libertado o espaço antes ocupado no Stack pointer, e movemos o valor 1, 2 ou 3 para o registro a0, que é nosso registro de retorno.

A mensagem de erro que é exibida no caso da escolha ser inválida, especifica a maneira que deve ser escrito o nome do personagem. O limite de caracteres que serão lidos pelo syscall 54 é de 15 (especificado no registo a2).

2.3 Função read_rgb_image

A função `read_rgb_image` tem como objetivo ler uma imagem no formato RGB a partir de um arquivo e armazenar os dados dessa imagem em um array. A função utiliza os seguintes registos:

- `s0`: Registo utilizado para armazenar o endereço do array onde a imagem será armazenada.
- `s1`: Registo utilizado para armazenar o descritor do arquivo.

- `t0`, `t1`, `t2`, `t3`, `t4`, `t5`: Registos temporários utilizados durante a execução da função.
- `a0`, `a1`, `a2`, `a3`: Registos (a) utilizados para os argumentos para as funções.

A sequência de operações realizadas pela função é a seguinte:

1. Prepara o *stack pointer* (`sp`) para armazenar os valores dos registos `s0` e `s1` empilhando-os.
2. Move o endereço do array (`a1`) para o registos `s0`.
3. Realiza uma chamada de sistema (`ecall`) para abrir o arquivo em modo de leitura utilizando a chamada de sistema 1024.
4. Salva o descritor do arquivo retornado em `a0` no registos `s1`.
5. Verifica se o arquivo foi aberto com sucesso. Caso contrário, salta para a etiqueta `read_rgb_image_error`.
6. Realiza uma chamada de sistema (`ecall`) para ler os dados da imagem do arquivo utilizando a chamada de sistema 63.
7. Carrega o descritor do arquivo (`s1`) em `a0` e o endereço do array de imagem (`s0`) em `a1`.
8. Define o tamanho máximo de bytes a serem lidos (172800, tamanho do arquivo `starwars.rgb`) em `a2`.
9. Realiza uma chamada de sistema (`ecall`) para ler os dados da imagem do arquivo.
10. Finaliza a leitura dos dados da imagem e fecha o arquivo. Para isso, realiza uma chamada de sistema 57 para fechar o arquivo utilizando o descritor em `s1`.
11. Limpa os registos `a0`, `a1`, `a2` e `a3` para evitar vazamento de informações.
12. Desempilha os valores dos registos `s0` e `s1` do *stack pointer*.
13. Retorna para a função chamadora.

Caso ocorra um erro na leitura do arquivo, a função exibe uma mensagem de erro utilizando a chamada de sistema 4 (`ecall`) e salta para a etiqueta `read_rgb_image_done`, finalizando a função.

2.4 Função Hue

A função 'hue' é responsável por calcular o valor de matiz (hue) com base nos valores de R, G e B fornecidos. Ela utiliza uma estrutura de múltiplos blocos condicionais para determinar a cor dominante e realizar os cálculos necessários. A função utiliza os seguintes registos:

- `t0`, `t1`, `t2`: Registos temporários que armazenam os valores de R, G e B, respectivamente.

- `t3`, `t4`, `t5`, `t6`: Registos temporários utilizados nos cálculos intermediários.
- `s0`: Registo utilizado para armazenar o valor final de matiz.

Os blocos condicionais verificam as relações entre os valores de R, G e B e executam cálculos específicos para cada caso. O resultado final é armazenado no registo 's0' e retornado pela função. Resumindo, a função 'hue' realiza as seguintes etapas:

1. Compara os valores de R, G e B para identificar a cor dominante.
2. Com base na cor dominante, realiza os cálculos adequados para obter o valor de matiz.
3. Armazena o valor de matiz calculado no registo 's0'.
4. Retorna o valor de matiz. Em resumo, a função 'hue' é responsável por determinar o matiz com base nos componentes de cor RGB.

2.5 Função indicator

A função indicator recebe três parâmetros: a0 (valor de entrada), a1 (endereço de retorno) e a2 (personagem escolhido). Ela tem como objetivo determinar um indicador com base no valor de entrada e no personagem escolhido. A função utiliza os seguintes registo:

- `sp`: Ponteiro de pilha, utilizado para reservar espaço para o registo de retorno (ra) na pilha.
- `ra`: Registo de retorno, salvo na pilha para restauração posterior.
- `t1`, t2, t3, t4: Registos temporários, utilizados para armazenar valores constantes e comparar com o valor de s0 (valor retornado pela função hue).
- `s0`: Registo de valor retornado pela função hue, utilizado para comparação com os limites de cada personagem.
- `s1`: Registo de retorno, que armazena o indicador resultante da função.

A função realiza as seguintes etapas:

1. Reduz o espaço da pilha em 4 bytes para armazenar o registo de retorno (ra).
2. Salva o valor do registo de retorno (ra) na pilha.
3. Chama a função hue através da instrução jal hue, que não está presente no código fornecido.
4. Carrega valores constantes (t1, t2, t3, t4) para serem utilizados nas comparações posteriores.
5. Compara o valor do personagem escolhido (a2) com os valores constantes t1, t2 e t3 para determinar para qual personagem deve-se pular.
6. Para cada personagem, verifica se o valor retornado (s0) está dentro dos limites definidos.

7. Se o valor retornado (s0) está dentro dos limites, atribui o valor 1 ao registo t4. Caso contrário, atribui o valor 0. Pula para a label saida.

8. Nas labels yoda, darth_maul e mandalorian, caso o valor retornado (s0) esteja dentro dos limites, atribui o valor 1 ao registo t4. Caso contrário, o valor de t4 mantém-se 0. Salta para a label n_pertence.

9. Na label saida, carrega o valor do registo de retorno (ra) da pilha.

10. Aumenta o espaço da pilha em 4 bytes para liberar o espaço alocado para o registo de retorno (ra).

11. Move o valor do registo t4 para o registo de retorno s1.

12. Retorna o controle para a função chamadora.

2.6 Função Centro de Massa

A função 'centro_de_massa' calcula o centro de massa de um personagem em uma imagem. Ela recebe um endereço de memória ('a1') que aponta para os dados da imagem do personagem. A função utiliza os seguintes registo:

- `sp`: Ponteiro de pilha, utilizado para reservar espaço para o registo de retorno ('ra') na pilha.
- `ra`: Registo de retorno, salvo na pilha para restauração posterior.
- `s2`, s3, s4, s5, s6, s7, s8: Registo temporários utilizados para controlar os loops e realizar cálculos.
- `a1`: Registo que armazena o endereço dos dados da imagem do personagem.
- `a0`: Registo utilizado para carregar o endereço do array 'Ponto', que irá armazenar o resultado do centro de massa.
- `s4`, s5: Registo que acumulam a soma das coordenadas x e y, respectivamente.
- `s6`: Registo que conta quantos pixels pertencem ao personagem.

A função realiza as seguintes etapas:

1. Reduz o espaço da pilha em 4 bytes para armazenar o registo de retorno ('ra').

2. Salva o valor do registo de retorno ('ra') na pilha.

3. Inicializa os registo 's2', 's3', 's4', 's5', 's6', 's7' e 's8' com valores iniciais.

4. Inicia um loop externo para percorrer as linhas da imagem do personagem, verificando a altura ('s3') em relação à altura total ('s8').

5. Dentro do loop externo, inicia um loop interno para percorrer as colunas da imagem do personagem, verificando a largura ('s2') em relação à largura total ('s7').

6. Chama a função ‘indicator‘, cujo resultado é armazenado no registo ‘s1‘.
7. Verifica se o valor de ‘s1‘ é igual a zero, indicando que o pixel não pertence ao personagem.
8. Se o pixel pertence ao personagem, incrementa o contador ‘s6‘, soma a coordenada x (‘s2‘) ao registo ‘s4‘ e soma a coordenada y (‘s3‘) ao registo‘s5‘.
9. Incrementa o endereço em ‘a1‘ para apontar para o próximo pixel na imagem.
10. Incrementa o contador ‘s2‘ para percorrer a próxima coluna.
11. Retorna para o início do loop interno para percorrer as colunas.
12. Após percorrer todas as colunas, incrementa o contador ‘s3‘ para percorrer a próxima linha.
13. Retorna para o início do loop externo para percorrer as linhas.
14. Após percorrer todas as linhas, realiza o cálculo do centro de massa dividindo as somas ‘s4‘ e ‘s5‘ pelo contador ‘s6‘.
15. Carrega o endereço do array ‘Ponto‘ em ‘a0‘.
16. Armazena o valor calculado do centro de massa das coordenadas x e y nos endereços ‘0(a0)‘ e ‘4(a0)‘, respectivamente.
17. Restaura o valor do registo de retorno (‘ra‘) da pilha.
18. Aumenta o espaço da pilha em 4 bytes para liberar o espaço alocado para o registos de retorno (‘ra‘).
19. Retorna para a função chamadora.

2.7 Função Draw Cross

A função ‘draw_cross‘ é responsável por desenhar uma cruz em uma imagem. Aqui está uma explicação do que a função faz e quais registos são utilizados:

- Define os valores iniciais para os registos ‘t0‘, ‘t1‘ e ‘s0‘.
- Calcula o valor de ‘x_start‘ como ‘cx - 5‘, armazenando o resultado em ‘t2‘.
- Calcula o valor de ‘x_end‘ como ‘cx + 5‘, armazenando o resultado em ‘t3‘.
- Calcula o valor de ‘y_start‘ como ‘cy - 5‘, armazenando o resultado em ‘t4‘.
- Calcula o valor de ‘y_end‘ como ‘cy + 5‘, armazenando o resultado em ‘t5‘.

  A partir daqui, inicia-se um loop para desenhar a cruz na direção vertical:

  1. Verifica se o valor de ‘t4‘ é maior que ‘t5‘. Se for, encerra o loop vertical (‘end_loop_y‘).

2. Calcula o índice do pixel a ser modificado utilizando a fórmula ‘cx + y * largura‘, armazenando o resultado em ‘s1‘.
3. Realiza alguns deslocamentos (‘slli‘) no registo ‘s1‘ para obter o deslocamento adequado para o array de imagem.
4. Adiciona ‘a0‘ a ‘s1‘ para obter o endereço de memória correto para o pixel.
5. Define o valor 255 (‘s2‘) no canal de cor vermelho (R) do pixel.
6. Armazena o valor zero no canal de cor verde (G) e azul (B) do pixel seguinte.
7. Atualiza ‘s1‘ para o próximo pixel.
8. Incrementa ‘t4‘ para avançar para a próxima linha vertical.
9. Retorna ao início do loop vertical (‘loop_y‘).
   Após desenhar a cruz na direção vertical, a função continua para desenhar a cruz na direção horizontal:
10. Define os valores iniciais para os registos ‘s1‘ e ‘s2‘.
11. Verifica se o valor de ‘t2‘ é maior que ‘t3‘. Se for, encerra o loop horizontal (‘end_draw_cross‘).
12. Calcula o índice do pixel a ser modificado utilizando a fórmula ‘x + cy * largura‘, armazenando o resultado em ‘s1‘.
13. Realiza alguns deslocamentos (‘slli‘) no registos ‘s1‘ para obter o deslocamento adequado para o array de imagem.
14. Adiciona ‘a0‘ a ‘s1‘ para obter o endereço de memória correto para o pixel.
15. Define o valor zero no canal de cor vermelho (R) do pixel.
16. Define o valor 255 (‘s2‘) no canal de cor verde (G) do pixel seguinte.
17. Armazena o valor zero no canal de cor azul (B) do pixel seguinte.
18. Atualiza ‘t2‘ para avançar para a próxima coluna horizontal.
19. Retorna ao início do loop horizontal (‘loop_x‘). Após desenhar a cruz na direção horizontal, a função retorna (‘ret‘).

2.8 Função Write_image_rgb

A função ‘write_rgb_image‘ é responsável por escrever os dados de uma imagem RGB em um arquivo. Aqui está uma explicação do que a função faz e quais registos são utilizados:

- s0 e s1 para armazenar temporariamente os valores dos registos ‘a0‘ e ‘a1‘ para uso posterior.
- a0 para armazenar o descritor de arquivo e o endereço da mensagem de erro.
- a1 para armazenar o endereço do array de dados RGB da imagem.
- a2 para armazenar o tamanho máximo dos dados RGB a serem escritos.
- a7 para controlar a chamada do sistema.

1. Prepara o stack pointer (‘sp‘) para armazenar os valores dos registos ‘s0‘ e ‘s1‘, empilhando-os e alocando espaço no stack (‘addi sp, sp, -8‘, ‘sb s0, 0(sp)‘, ‘sb s1, 4(sp)‘).

2. Move o valor de ‘a1‘ para ‘s0‘, que contém o endereço do array de dados RGB da imagem.

3. Prepara o registos ‘a7‘ com o valor 1024, que representa a chamada do sistema para abrir um arquivo.

4. Define o valor 1 em ‘a1‘, indicando que o arquivo deve ser aberto para escrita.

5. Faz a chamada do sistema (‘ecall‘) para abrir o arquivo. O descritor de arquivo é retornado em ‘a0‘.

6. Move o descritor de arquivo (‘a0‘) para ‘s1‘ para ser usado posteriormente.

7. Verifica se o arquivo foi aberto com sucesso (‘beqz a0, write_rgb_image_error‘). Se o valor em ‘a0‘ for zero, significa que houve um erro na abertura do arquivo, e o fluxo é desviado para a etiqueta ‘write_rgb_image_error‘.

8. Prepara o registos ‘a7‘ com o valor 64, que representa a chamada do sistema para escrever em um arquivo.

9. Move o descritor de arquivo (‘s1‘) para ‘a0‘, para indicar o arquivo em que os dados serão escritos.

10. Move o endereço do array de dados RGB (‘s0‘) para ‘a1‘, indicando os dados que serão escritos.

11. Define o valor 172800 em ‘a2‘, que representa o tamanho máximo em bytes dos dados RGB.

12. Faz a chamada do sistema (‘ecall‘) para escrever os dados RGB no arquivo.

13. Prepara o registos ‘a7‘ com o valor 57, que representa a chamada do sistema para fechar um arquivo.

14. Move o descritor de arquivo (‘s0‘) para ‘a0‘, para indicar o arquivo que será fechado.

15. Faz a chamada do sistema (ecall) para fechar o arquivo.

16. Retorna da função (‘ret‘).

    Caso ocorra um erro na abertura do arquivo, o fluxo é desviado para a etiqueta ‘write_rgb_image_error‘:

17. Carrega o endereço da mensagem de erro em ‘a0‘.

18. Prepara o registos ‘a7‘ com o valor 4, que representa a chamada do sistema para imprimir uma string.

19. Faz a chamada do sistema (‘ecall‘) para imprimir a mensagem de erro.

20. Desempilha os valores dos registos ‘s0‘ e ‘s1‘ do stack (‘addi sp, sp, 8‘, ‘lb s0, 0(sp)‘, ‘lb s1, 4(sp)‘).

21. Retorna da função (‘ret‘).

3 Observações

- O ficheiro que contém o output do nosso programa chama-se ”Newstarwars.rgb”, que aparece após a execução do programa.

- Na função centro_de_massa, optamos por utilizar store word ao invés de store byte pois uma das coordenadas do centro era superior a 255, que passa o limite possível para armazenar em um byte.

- Ao executar o programa, notamos um pequeno pixel no canto superior esquerdo da imagem que é alterado em cor que acreditamos ser devido ao syscall 54 sobrepor o que estava anteriormente escrito.