Universidade de Brasília

INSTITUTO DE CIÊNCIAS EXATAS – IE
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO – CIC

Disciplina: CIC 116394 – Organização e Arquitetura de Computadores – Turma C
Semestre: 2019/2
Professor: Marcelo Grandi Mandelli

TRABALHO 1 – ASSEMBLY RISC-V

OBJETIVOS

- Familiarizar o aluno com o Simulador/Montador RARS;
- Desenvolver a capacidade de codificação de algoritmos em linguagem Assembly RISC-V.

METODOLOGIA

O trabalho deverá ser implementado utilizando o simulador RARS (disponível em https://github.com/TheThirdOne/rars/releases) para linguagem de programação em assembly RISC-V. Especificadamente, o trabalho deverá utilizar a ferramenta Bitmap Display do RARS (aba *"Tools → Bitmap Display"*).

O Bitmap Display é uma janela para exibição gráfica baseada em pontos. Suas principais características são as seguintes:

- Um ponto é conjunto de 1 ou mais *pixels*. O tamanho de um ponto é definido pelos parâmetros de *Unit Width in Pixels* e *Unit Heigth in Pixels* que definem, respetivamente, a largura do ponto em *pixels* e a altura do ponto em *pixels*.

- Resolução configurável, *default* 512 x 256 *pixels*.

- Cada ponto é representado por uma palavra de 32 bits (4 bytes), no formato RGB, um *byte* para cada cor. *Red*(vermelho) = 0x00FF0000, *Green*(verde) = 0x0000FF00, *Blue*(azul) = 0x000000FF. O *byte* mais significativo é ignorado.

- Essa ferramenta utiliza um *display* gráfico mapeado em memória, isto é, cada endereço de memória corresponde a um ponto.

- O endereço base de memória do Bitmap Display é configurável. Por exemplo, pode ser utilizado como endereço base de memória: *global data, global pointer, static data, heap, memory map.*

- Por *default*, o endereço base de memória aponta para o ponto superior esquerdo da tela.

- O desenho de um pixel na tela é realizado pela escrita de uma palavra contendo a descrição de sua cor RGB na posição de memória correspondente.

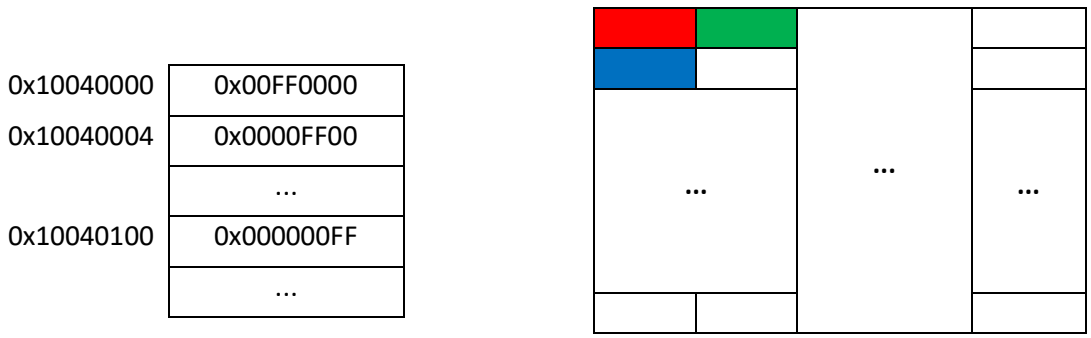
Para este trabalho, as configurações do Bitmap Display devem ser as seguintes:

- Configuração de memória (aba *"Settings → Memory Configuration"*): *default*

- Configuração do Bitmap Display:
  - Unidade de largura em pixels (*Unit Width in Pixels*): 4
  - Unidade de altura em pixels (*Unit Heigth in Pixels*): 4
  - Largura do display (*Display Width in Pixels*): 256
  - Altura do display (*Display Heigth in Pixels*): 256
  - Endereço base do display (*Base address for display*): 0x10040000 (*heap*)

Utilizando essas configurações, o Bitmap Display será uma matriz de 256x256 pixels. Além disso, como cada ponto tem tamanho de tamanho 4 pixels de altura por 4 pixels de largura, **o Bitmap Display será uma matriz de 64x64 pontos**. Para cada escrita de um valor RGB (como descrito anteriormente) em um endereço de memória (começando no endereço de base 0x10040000), desenhamos um ponto no *display*.

A Figura 1 mostra um exemplo de desenho de pontos em um Bitmap Display de 64x64 pontos (tamanho do ponto de 4x4 pixels). A Figura 1 (a) mostra a memória, apresentando ao lado esquerdo um endereço de memória, e ao lado direito o valor contido nesse endereço. A Figura 1 (b) apresenta um exemplo Bitmap Display, onde cada retângulo representa um ponto de tamanho 4x4 pixels.



a) Memória

b) Bitmap Display

Figura 1 – Exemplo de desenho de pontos em um Bitmap Display

No exemplo da Figura 1, quando escrevemos o valor RGB 0x00FF0000 no endereço de memória 0x10040000 (endereço de base) estamos definindo que o ponto superior esquerdo do Bitmap Display será vermelho.

Se escrevermos no endereço seguinte de memória 0x10040004 (endereço de base + 4), estaremos desenhando no ponto imediatamente à direita ao ponto do endereço de base (ponto verde na Figura 1 (b)).

Se escrevermos no endereço de memória 0x10040100 (endereço de base + 256 (4x64)) estaremos escrevendo no ponto imediatamente abaixo ao ponto do endereço de base (ponto azul na Figura 1(b)).

DEFINIÇÃO

Este trabalho consiste no desenvolvimento de um programa em código assembly RISC-V, especificadamente na variação RV32I, que implementa funções de desenho e manipulação de imagens no Bitmap Display do simulador RARS.

Para este trabalho, assuma as seguintes definições:

- Como mencionado anteriormente, o Bitmap Display utilizado será uma matriz de 64x64 pontos, onde cada ponto tem tamanho 4x4 pixels.

- Cada ponto do Bitmap Display pode ser endereçado por uma linha **x** e uma coluna **y**.

- O Bitmap Display possui 64 linhas, indexadas de 0 a 63 de baixo para cima. Por exemplo, a linha mais abaixo da tela terá índice x=0 e a mais acima na tela terá índice x=63.

- O Bitmap Display possui 64 colunas, indexadas de 0 a 63 da esquerda para direita. Por exemplo, a coluna mais à esquerda da tela terá índice y=0 e a mais à direita na tela terá índice y=63.

- A posição onde x=0 e y=0 será a posição inferior esquerda do Bitmap Display. Na Figura 1: o ponto vermelho está na posição x=63 e y=0; o ponto verde está na posição x=63 e y=1; e o ponto azul está na posição x=62 e y=0.

- Todos os parâmetros das funções a serem implementadas neste trabalho devem ser solicitados a um usuário, que entrará valores via teclado.

- Os parâmetros devem ser solicitados ao usuário na ordem definida pelos nomes das chamadas de funções descritos a seguir, da esquerda para à direita. Por exemplo, para a função *get_point(int x, int y)*, primeiro é solicitado ao usuário o valor de *x* e posteriormente o valor de *y*.

As seguintes funções devem ser implementadas neste trabalho:

- **Obtém ponto – *get_point(int x, int y)***
    - Essa função tem como parâmetros uma posição dada por *x* e *y*. Como retorno, a função imprime na console os valores dos componentes R, G e B do ponto indicado pela posição dada por *x* e *y*.

- **Desenha ponto - *draw_point(int x, int y, int val)***
    - Essa função tem como parâmetros uma posição dada por *x* e *y*, além de um valor *val* em RGB. Para a formação do valor *val*, deve ser solicitado ao usuário separadamente os valores das componentes R, G e B (através de números em decimal entre 0 e 255). Como retorno, a função desenha um ponto de cor definida pelo valor *val* na posição denotada por *x* e *y* do Bitmap Display.

- **Desenha retângulo com preenchimento – *draw_full_rectangle(int xi, int yi, int xf, int yf, int val)***
    - Essa função tem como parâmetros duas posições, sendo uma dada por *xi* e *yi* e outra dada por *xf* e *yf*; além de um valor *val* em RGB. Para a formação do valor *val*, deve ser solicitado ao usuário separadamente os valores das componentes R, G e B (através de números em decimal entre 0 e 255). Como retorno deverá ser desenhado um retângulo com preenchimento de cor *val*. Um dos quatro cantos do retângulo desenhado deverá ser o ponto de posição xi e yi; e outro canto, o ponto de posição xf e yf.
    - Por exemplo, a Figura 3 mostra um exemplo de retângulo desenhado por essa função, onde xi=60, yi=60, xf=20, yf=45 e val = 0x00FF0000. Na Figura 2, o canto superior direito do retângulo é dado pelo ponto de posição xi=60 e yi=60, e o canto inferior esquerdo do retângulo é dado pelo ponto de posição xf=20 e yf=45.

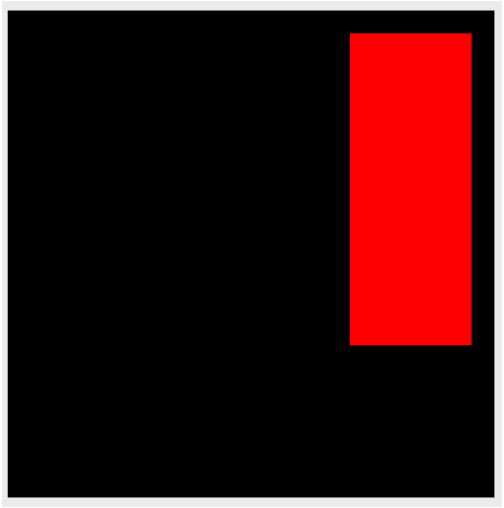


Figura 3 – Retângulo desenhado pela função *draw_full_rectangle(xi, xf, yi, yf, val),* onde xi=60, yi=60, xf=20, yf=45 e val = 0x00FF0000. A cor de fundo é preta.

- **Desenha retângulo sem preenchimento – *draw_empty_rectangle(int xi, int yi, int xf, int yf, int val)***
    - Essa função tem como parâmetros duas posições, sendo uma dada por *xi* e *yi* e outra dada por *xf* e *yf*; além de um valor *val* em RGB. Para a formação do valor *val*, deve ser solicitado ao usuário separadamente os valores das componentes R, G e B (através de números em decimal entre 0 e 255). Como retorno deverá ser desenhado um retângulo somente com uma borda de tamanho um ponto e de cor *val*. O interior do retângulo deve permanecer a cor de fundo de quando foi chamada a função. Um dos quatro cantos do retângulo desenhado deverá ser o ponto de posição xi e yi; e outro canto, o ponto de posição xf e yf.
    - Por exemplo, a Figura 2 mostra um exemplo de retângulo desenhado por essa função, onde xi=10, yi=10, xf=60, yf=20 e val = 0x00FF0000. Na Figura 2, o canto inferior esquerdo do retângulo é dado pelo ponto de posição xi=10 e yi=10, e o canto superior direito do retângulo é dado pelo ponto de posição xf=60 e yf=20.

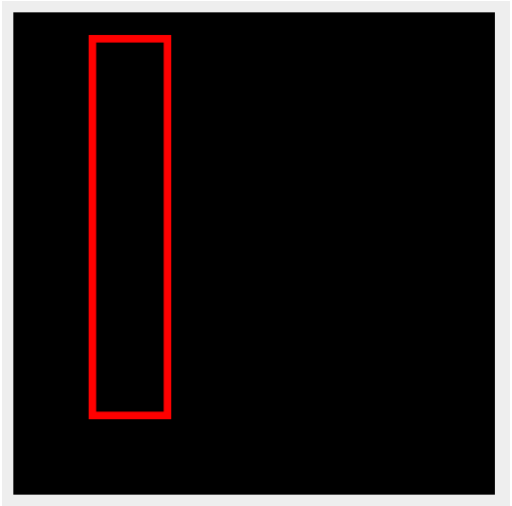


Figura 2 – Retângulo desenhado pela função *draw_empty_rectangle(xi, xf, yi, yf, val),* onde xi=10, yi=10, xf=60,yf=20 e val = 0x00FF0000. A cor de fundo é preta.

- **Converte para negativo da imagem – *convert_negative()***
    - Essa função não possui parâmetros. Como retorno, essa função deve apresentar o negativo da imagem contida no Bitmap Display. O negativo de uma imagem é obtido através da inversão de cores de uma imagem normal. Para fazer isso, deve-se obter o componente R, G e B de cada ponto do Bitmap Display e subtrair de 255 (O novo valor RGB de um ponto será 255 – R, 255 – G e 255 – B).

- **Converte para tons de vermelho – *convert_redtones()***
    - Essa função não possui parâmetros. Como retorno, essa função deve apresentar a imagem contida no Bitmap Display somente em tons de vermelho. Para fazer isso, deve-se manter somente o componente R de cada ponto da imagem, colocando-se o valor 0 nos componentes G e B.

Além dessas funções que devem ser implementadas, será fornecido o código da seguinte função que deverá ser incorporada ao programa:

- **Carrega imagem – *load_image(string image_name, int address, int buffer, int size)***
    - Essa função tem como parâmetros o nome de uma imagem *image_name* a ser carregada no display, o endereço base de memória do display *address*, o endereço de uma palavra na memoria para utilizar como buffer e o tamanho *size* da imagem em pixels. A função retorna a exibição da imagem no bitmap display.

Todas as funções implementadas nesse trabalho devem estar contidas no mesmo arquivo de código assembly. Além disso, deverá ser realizada uma interface inicial com o usuário, utilizando chamadas de sistema. A interface inicial do usuário deverá conter um cardápio de opções, apresentando as possíveis funções que podem ser chamadas. A Figura 4 apresenta um exemplo de interface inicial com o usuário.

```
• Defina o número opção desejada:
   1. Obtém ponto
   2. Desenha ponto
   3. Desenha retângulo com preenchimento
   4. Desenha retângulo sem preenchimento
   5. Converte para negativo da imagem
   6. Converte imagem para tons de vermelho
   7. Carrega imagem
   8. Encerra
```

Figura 4 – Exemplo de interface inicial com o usuário

CÓDIGO FONTE

- Neste trabalho, você deverá entregar apenas o código fonte assembly (arquivo .asm), o qual deve conter:
    - **Identificação:** utilizando comentários de código, crie um cabeçalho no topo do código com título do trabalho, nomes e matrículas dos alunos integrantes do grupo.
    - **documentação de função:** utilizando comentários de código, crie um cabeçalho para cada função implementada, detalhando o seu funcionamento, modo de usar e uma descrição geral de como foi implementada. Você pode incluir um (pseudo)código em C da função para complementar a explicação. Lembre-se que todas as funções devem estar em um mesmo arquivo.
    - **comentários relevantes para o entendimento do programa**: comente as linhas que você achar necessário para explicar o funcionamento do programa. Por exemplo, defina que registrador é usado para cada variável do programa, explique porque um *loop* é utilizado, etc.

ENTREGA

Entregar o código fonte no moodle (aprender.unb.br) até às 23h55 do dia 18/09/2019

O nome do arquivo deve conter as matrículas dos integrantes do grupo:
matriculaaluno1_matriculaaluno2_matriculaaluno3.asm

ENTREGA EM ATRASO: o número de pontos descontados por dia em atraso na entrega do trabalho é dado pela equação: $2^n+(n-1)$, sendo n o número de dias em atraso. (1 dia → -2 pontos, 2 dias → -5 pontos, 3 dias → -10 pontos)

GRUPOS

O trabalho deverá ser realizado em grupos de 2 ou 3 alunos.

CRITÉRIOS DE AVALIAÇÃO

Funcionamento do código – 60% da nota

- Função Obtém ponto – 5%
- Função Desenha ponto – 7%
- Função Desenha retângulo com preenchimento – 10%
- Função Desenha retângulo sem preenchimento – 13%
- Função Converte para negativo da imagem – 8%
- Função Converte imagem para tons de vermelho – 8%
- Interface com o usuário – 9%

Documentação do código – 40% da nota

- Documentação de função – 20%
- Comentários relevantes no código – 20%

Esta especificação pode ser atualizada para se efetuar correções do texto ou alterações para se deixar mais claro o que está sendo pedido.

Caso a especificação sofrer uma atualização, os alunos serão informados via moodle (aprender.unb.br).

Última atualização: 03/09/2019 às 11:15