R.A: 133691

R.A: 135522

R.A: 133656

Tutorial – Projeto Aspirador em Assembly M.I.P.S

São José dos Campos – Brasil

Novembro de 2019

R.A: 133691

R.A: 135522

R.A: 133656

Tutorial – Projeto Aspirador em Assembly M.I.P.S

Tutorial apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Arquitetura e Organização de Computadores

Alunos: Jhonathan Hiroo Eguchi

Lucas Gomes

Gustavo de Lima Lopes

Docente: Profa. Denise Stringhini

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos – Brasil

Novembro de 2019

Resumo

Este tutorial contêm todo o processo de desenvolvimento do projeto do aspirador em Assembly M.I.P. S (Microprocessor Without Interlocked Pipeline Stage) da disciplina de Arquitetura e Organização de Computadores. Foi-se usado no projeto o programa MARS (M.I.P.S Assembler and Runtime Simulator) para manipular as posições de memória e usar as instruções do M.I.P.S. Ademais, como parte do processo de visualização, tivemos o auxilio do Bitmap Display do MARS. Usamos um Bitmap Display de 16x16 unidades de altura e largura, bem como um display de 512x512 de altura e largura, nos resultando em um display de 32x32 unidades e um endereço base para o display 0x1004000 (Heap) para visualizarmos todo o projeto em funcionamento.

Palavras-chaves: MARS. Bitmap Display. M.I.P.S. Arquitetura e Organização de Computadores. Heap.

Lista de Ilustrações

Sumário

1 Introdução

Sabe-se que a tecnologia de hardware avança cada dia mais, porém, muitos ignoram os conhecimentos básicos e introdutórios de arquitetura e organização de computadores antes de querer adentrar o entendimento de áreas do momento como HPC (High-performance computing), desenvolvimento de Hardware etc., por isso, evidenciasse a importância desse projeto, pois nele aplicamos na integra os conhecimentos de Assembly e conhecimentos básicos para dominar um conjunto de instruções. Trata-se de um projeto em que temos um display de 512x512 e um pixel de 16x16, nos remetendo a um Bitmap Display de 32x32 pixels. O projeto foi realizado nesta matriz de memória usando-se instruções do M.I.P.S como o load word e o store word, principalmente para manipular endereços hexadecimais que representam cores, além de instruções como jumps e branchs para saber quando carregar ou não uma cor numa posição específica do display. Como um último projeto da disciplina de Arquitetura e Organização de Computadores esperasse usar a maioria dos conteúdos aprendidos na disciplina para dominar um conjunto de instruções, no caso o Assembly M.I.P.S. Espera-se que ao final do tutorial possa ser entendida a forma com que um conjunto de instruções pode ser usado na construção de jogos, pequenas aplicações e como base pro funcionamento de qualquer computador.

2 Objetivos

2.1 Gerais

Como solicitado pela Professora Denise e nas recomendações do projeto, o objetivo principal é aplicar conhecimentos de Arquitetura e Organização de Computadores, em especial todo o conhecimento de Assembly M.I.P.S, para desenvolver um programa em Assembly que nos permita visualizar, através do Bitmap Display do programa MARS, um aspirador fictício percorrendo e limpando uma casa.

2.2 Específicos

- Desenvolver um programa que nos permita entender com profundidade o mapeamento de memória na sua forma mais baixo nível possível.
- Implementar com eficiência, corretude e eficiência o projeto do aspirador em Assembly M.I.P.S no MARS..
- Explicitar neste tutorial o desenvolvimento do projeto e as ideias usadas.
- Explicitar as configurações do Bitmap Display do MARS e como ele foi utilizado.
- Esmiuçar como o código em Assembly foi organizado e segmentado.
- Conseguir mostrar de forma clara o aspirador percorrendo o BitMap e “limpando” a planta baixa da casa.

3 Tutorial para execução do projeto

O primeiro passo para executar o projeto é saber a configuração do Bitmap Display do MARS 4.5, para isso, note que definimos as seguintes configurações:

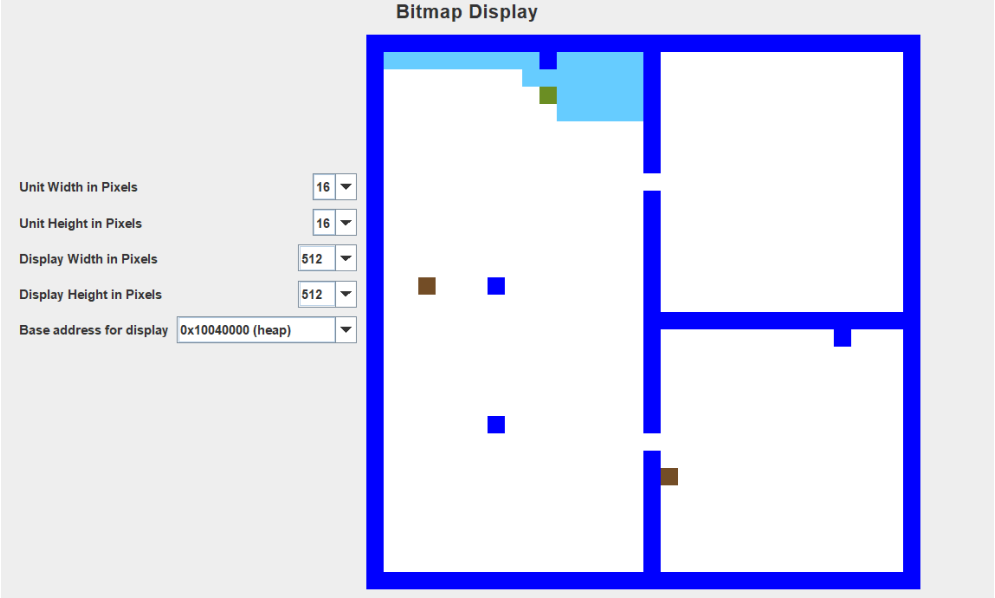

Figura 1 - Configuração do Bitmap Display. Fonte: Os autores

Para executar o projeto no MARS, siga o seguinte procedimento:

1. Na interface principal do MARS clique em Tools > Bitmap Display.
2. Configure o Bitmap Display como nas configurações da Figura 1.
3. No Bitmap Display clique em Connect to MIPS.
4. No MARS clique em Assemble the current file and clear breakpoints.
5. Clique em Run the current program.
6. O programa ira executar e você verá o aspirador no Bitmap Display.

No Bitmap Display, o ponto verde em movimento representa o aspirador, as linhas em azuis foram definidas como as paredes dos cômodos pedidos no projeto, os pontos azuis são os moveis e barreiras que o aspirador precisa contornar e os pontos em marrom são os moveis nos quais o aspirador consegue passar por baixo e aspirar, como mesas e cadeiras por exemplo.

4 Desenvolvimento do código em Assembly MIPS

Como uma boa prática de programação, dividimos o código do projeto em várias funções sendo que cada função tinha o seu respectivo papel no projeto. As primeiras 16 linhas do código e as 9 ultimas linhas do código resumem muito bem nossa estratégia de organização do projeto**.**

aspirador_mips.asm

```
1  .data
2          #Cores usadas no projeto
3
4          blue_clean:   .word 0x66ccff
5          white:    .word 0xFFFFFF
6          green:    .word 0x6B8E23
7          blue:     .word 0x0000ff
8          brown:    .word 0x734d26
9          brown_clean: .word 0x86592d
10
11         #vetor usado para armazenar a trajetoria do aspirador
12
13         vetor:    .space 4096
14
15 .text
16         j main
```

Figura 2 - Inicio do código em Assembly. Fonte: Os autores

A diretiva **.data** define o bloco de dados do programa, nela definimos os endereços hexadecimais das cores utilizadas no projeto. A diretiva **.text** define o inicio do bloco de instruções, já com um jump para a linha main no final do código, onde se encontra toda a organização do programa.

```
331        main:
332                jal set_cores
333                jal set_tela
334                jal planta_baixa
335                jal gerar_moveis
336                jal gerar_moveis_por_baixo
337                jal gera_aspirador
338                addi $t9, $zero, 0
339                jal limpa_casa
340
```

Figura 3 - Final do código em Assembly. Fonte: Os autores

O programa finaliza na linha 340, neste bloco main, note a forma como organizamos as funções principais do programa, que serão explicadas posteriormente neste relatório.

4.1 Função Set Cores

Essa função talvez seja a mais simples de entender do projeto, simplesmente armazenamos os endereços hexadecimais, definidos na diretiva **.data**, das cores usadas no projeto nos registradores s1 a s6.

```
33          set_cores: #Salvar as cores em registradores
34                  lw $s1, brown
35                  lw $s3, blue_clean
36                  lw $s4, white
37                  lw $s5, blue
38                  lw $s7, green
39                  lw $s6, brown_clean
40                  jr $ra
```

Figura 4 – Atribuição das cores nos registradores. Fonte: Os autores

4.2 Função Set Tela

Essa função também tem um funcionamento muito simples, ela simplesmente pinta todo o Bitmap Display de branco, pois no registrador s4 está o endereço hexadecimal da cor branca. Aqui definimos um laço while que pula de 4 em 4 bits para atingir todos os pixels do Bitmap.

```
18      set_tela: #Inicia todos os valores para a tela
19              addi $t0, $zero, 65536 #65536 = (512*512)/4 pixels
20              add $t1, $t0, $zero #Adicionar a distribuiÃ§Ã£o de pixels ao endereco
21              lui $t1, 0x1004 #Endereco base da tela no heap, pode mudar se quiser
22              add $t0, $zero, $t1
23              addi $t2, $zero, 0
24              loop:
25                      sw $s4, ($t0) #Pinta o pixel na posicao $t0 com a cor de $s4
26                      addi $t0, $t0, 4 #Pulo +4 no pixel
27                      addi $t2, $t2, 1 #Contador +1
28                      beq $t2, 1024, exit
29                      j loop
```

Figura 5 - Funcionamento da setagem da tela para branco. Fonte: Os autores

4.3 Função Planta Baixa

Essa função pinta os limites azuis da planta baixa, ela tem o funcionamento parecido com a função anterior, simplesmente começando da primeira posição e pintando posição a posição em 7 blocos de código, indo de exit_1 a exit_7 até de fato terminar todos os limites do imóvel. A cada bloco exit zeramos o registrador temporário t2, depois simplesmente vemos onde começa a posição do Bitmap que queremos e simplesmente pintamos N posições a partir dela.

```
42          planta_baixa:
43                  add $t0, $zero, $t1
44                  addi $t2, $zero, 0
45                  loop_1:
46                          sw $s5, ($t0)
47                          addi $t0, $t0, 4
48                          addi $t2, $t2, 1 #Contador +1
49                          beq $t2, 31, exit_1
50                          j loop_1
51          exit_1:
52                  addi $t2, $zero, 0
53                  loop_2:
54                          sw $s5, ($t0)
55                          addi $t0, $t0, 128
56                          addi $t2, $t2, 1 #Contador +1
57                          beq $t2, 31, exit_2
58                          j loop_2
```

Figura 6 - Lógica da construção da planta baixa. Fonte: Os Autores

4.4 Função Gerar Móveis

A função gerar móveis executa um bloco de código 4 vezes afim de gerar 4 móveis das cores azuis. Aqui, usamos o syscall 42 para sortear um numero aleatório de 0 a 1024, depois, multiplicamos esse número por 4 pois a posição do bit map precisa ser divisível por 4 para que possamos carregar a cor do móvel nela. Além disso, verificamos se a cor é branca para saber se é uma posição possível de se ter móveis.

```
112         gerar_moveis:
113
114                 addi $t2,$zero, 0
115                 loop_9:
116                         li $a1, 1024
117                         li $v0, 42
118                         syscall
119                         move $t0, $a0
120                         mul $t0,$t0,4
121                         add $t0, $t0, $t1
122                         lw $s2, ($t0)
123                         bne $s2, $s4, loop_9
124                         sw $s5, ($t0)
125                         addi $t2, $t2, 1 #Contador +1
126                         beq $t2, 4, exit_8
127                         j loop_9
```

Figura 7 - Lógica para gerar os móveis do projeto. Fonte: Os autores

4.5 Função Gerar Móveis por Baixo

Essa função tem funcionamento análogo a anterior, ela pinta os moveis marrons, os quais podemos passar em baixo.

```
129         gerar_moveis_por_baixo:
130
131                 addi $t2,$zero, 0
132                 loop_10:
133                         li $a1, 1024
134                         li $v0, 42
135                         syscall
136                         move $t0, $a0
137                         mul $t0,$t0,4
138                         add $t0, $t0, $t1
139                         lw $s2, ($t0)
140                         bne $s2, $s4, loop_10
141                         sw $s1, ($t0)
142                         addi $t2, $t2, 1 #Contador +1
143                         beq $t2, 2, exit
144                         j loop_10
```

Figura 8 - Lógica para gerar os móveis aspiráveis. Fonte: Os Autores

4.6 Função Gera Aspirador

Como temos um Bitmap Display 32x32 pixels, e como pra saltar precisamos ir somando de 4 em 4, simplesmente iniciamos o aspirador na posição 132, que seria a segunda coluna da segunda linha.

```
146         gera_aspirador:
147
148                         addi $t2, $zero, 132
149                         add $t2, $t2, $t1
150                         sw $s7, ($t2)
151                         j exit_8
```

Figura 9 - Logica para gerar a primeira posição do aspirador. Fonte: Os Autores

4.7 Função Limpa Casa

A cada posição em que o aspirador estiver nós demos load word do endereço das posições acima, direita, esquerda, e por ultimo em baixo, caso o endereço corresponda a cor marrom, ou seja, um móvel que podemos passar por baixo, pulamos para essa posição, a “aspiramos”, e depois fazemos a mesma coisa caso seja branco, agora, caso a posição seja um móvel que não podemos passar por baixo, ou seja, um ponto azul, simplesmente passamos pra próxima verificação. Para resolver o problema de caso não termos nenhuma posição branca ao redor do aspirador, nós implementamos uma lógica parecida com uma busca em profundidade, para isso, criamos um vetor de 4096 bytes, para armazenarmos a trajetória do aspirador, e voltamos nele até finalmente encontrar uma posição branca e prosseguir até terminar de limpar tudo, que no caso seria o contador contar até 853, que seria o número de posições limpáveis. Para controlar o tempo de cada passo e verificação, usamos o syscall 32, que da um sleep na execução do código usando uma thread do Java.

5 Conclusão

O projeto foi de suma importância para o entendimento total do funcionamento e do desenvolvimento de um projeto grande usando Assembly MIPS, bem como reforçar o entendimento dos conceitos de arquitetura e organização de computadores. Os resultados foram os esperados graças a uma boa projeção do código em Assembly e do entendimento do funcionamento do Bitmap do MARS. A maior dificuldade do projeto foi aprender em pouco tempo como funciona o bitmap do MARS, bem como bolar uma lógica de busca em profundidade pra posição atual do aspirador conseguir voltar nas posições ja passadas até encontrar uma posição não aspirada e terminar até ter terminar de pintar todas as posições não aspiradas Felizmente, como ficou explícito na apresentação do projeto em sala, nenhum problema foi encontrado e conseguiu-se cumprir o projeto proposto pela professora.

6 Referências Bibliográficas

1 MIPS Instruction Set. <https://www.dsi.unive.it/~gasparetto/materials/MIPS_Instruction_Set.pdfTOC>, Acessado em 24/10/2019.

2 MARS - Mips Assembly and Runtime Simulator. < https://courses.missouristate.edu/KenVollmar/MARS/ >. Acessado em 24/10/2019.