RELATÓRIO TÉCNICO

Controle de Display e Leitura de Teclado em Assembly MIPS

Diogo Borges Corso

Departamento de Computação

(48) 3721-6950

Campus Jardim das Avenidas, Rodovia Governador Jorge Lacerda, 3201 - Jardim das Avenidas - Araranguá - SC

UNIVERSIDADE FEDERAL DE SANTA CATARINA

SUMÁRIO

1 Introdução 4

2 Objetivos 5

3 Desenvolvimento 6

3.1 Exibição Sequencial no Display de 7 Segmentos 6

3.2 Leitura do Teclado Hexadecimal 8

4 Resultados 13

4.1 Exibição Sequencial no Display de 7 Segmentos 13

4.2 Leitura do Teclado Hexadecimal 15

5 Conclusão 17

Lista de Figuras

Figura 1 - Display de sete segmentos exibindo o dígito 4 14
Figura 2 - Display de sete segmentos exibindo o dígito 7 15
Figura 3 - Teclado Hexadecimal exibindo o dígito 5 16
Figura 4 - Teclado Hexadecimal exibindo o caracter A 16

1 Introdução

Este relatório tem como objetivo analisar e documentar o funcionamento de dois programas desenvolvidos em linguagem Assembly, utilizando a ferramenta Digital Lab Sim integrada ao simulador MARS. Ambos os códigos foram projetados para manipular dispositivos de entrada e saída típicos de sistemas embarcados, como displays de sete segmentos e teclado matricial.

O primeiro programa demonstra a exibição sequencial dos dígitos de 0 a 9 em um display de sete segmentos, utilizando um laço para percorrer um vetor com os padrões correspondentes a cada dígito. Já o segundo programa implementa a leitura de um teclado matricial, realizando a varredura das linhas e colunas para identificar a tecla pressionada e exibir o valor correspondente no display.

Este relatório abordará os detalhes de implementação dos códigos, o funcionamento dos periféricos simulados, as técnicas utilizadas para o controle do hardware e as principais conclusões obtidas a partir da análise dos programas.

2 OBJETIVOS

O presente trabalho tem como objetivos principais:

1. Desenvolver um programa em Assembly para o simulador MARS que realize a exibição sequencial dos números de 0 a 9 em um display de sete segmentos utilizando a ferramenta Digital Lab Sim.
2. Implementar um programa em Assembly que leia as entradas do teclado alfanumérico matricial disponível na ferramenta Digital Lab Sim e o faça exibir no display o valor da tecla pressionada, permitindo a leitura contínua por meio de um loop.
3. Demonstrar, por meio de prints de tela e análise do código, o correto funcionamento dos programas desenvolvidos, comprovando a interação adequada com os periféricos simulados.
4. Aplicar conceitos de programação em Assembly e controle de hardware digital, explorando técnicas de manipulação de memória e registros para interface com dispositivos externos.

3 DESENVOLVIMENTO

Neste trabalho, foram desenvolvidos dois programas em Assembly para o simulador MARS, utilizando a ferramenta Digital Lab Sim para simular dispositivos de entrada e saída comuns em sistemas embarcados, como displays de sete segmentos e teclados matriciais.

3.1 Exibição Sequência de 7 Segmentos

O primeiro programa foi projetado para mostrar, em sequência, os dígitos de 0 a 9 no display de sete segmentos, utilizando a ferramenta MARS com a extensão Digital Lab Sim. O programa utiliza um vetor chamado dígitos, que contém os padrões binários para cada número, indicando quais segmentos devem ser acesos para formar corretamente os dígitos no display.

Funcionamento detalhado:

- **Inicialização dos registradores:**
    - $t0 é carregado com o endereço base do display (0xFFFF0010), que é o endereço de memória mapeada correspondente à saída do display no Digital Lab Sim.
    - $t1 recebe o endereço inicial do vetor digitos, onde estão armazenados os padrões binários dos números de 0 a 9.
    - $t2 é utilizado como contador de iteração.

- **Loop principal:**
    - O programa entra em um loop que percorre o vetor de padrões binários.
    - A instrução lb carrega o padrão do dígito atual.

- Em seguida, a instrução sb grava esse padrão no endereço do display, o que faz o número correspondente aparecer no display de sete segmentos.
- Para garantir que o usuário visualize a troca de número, é aplicado um atraso de 500 milissegundos utilizando a chamada de sistema syscall com o código 32.
- Tanto o ponteiro ($t1) quanto o contador ($t2) são incrementados para passar ao próximo dígito.
- O loop continua até que todos os 10 dígitos sejam exibidos.

- **Finalização:**

  - Após a exibição completa, o programa finaliza com uma chamada de sistema (syscall 10), encerrando a execução.

Este programa demonstra com clareza dois conceitos fundamentais em sistemas embarcados:

- O controle direto de hardware por meio de acesso a endereços de memória mapeados.
- O uso de vetores para armazenamento de padrões binários representando o comportamento visual de um dispositivo físico.

Código Fonte:

```
.data
DISPLAY: .word 0xFFFF0010

digitos: .byte 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F

.text
.globl main

main:
    li $t0, 0xFFFF0010
    la $t1, digitos
```

```
    li $t2, 0

loop:
    lb $t3, 0($t1)
    sb $t3, 0($t0)

    li $v0, 32
    li $a0, 500
    syscall

    addi $t1, $t1, 1
    addi $t2, $t2, 1
    li $t4, 10
    blt $t2, $t4, loop

    li $v0, 10
    syscall
```

3.2 Leitura do Teclado e Exibição Dinâmica

Este segundo programa foi desenvolvido para realizar a leitura das teclas pressionadas em um teclado hexadecimal 4x4, utilizando novamente a ferramenta MARS com a extensão Digital Lab Sim para simulação do hardware. O objetivo principal é detectar qual tecla foi pressionada no teclado e exibir seu valor correspondente em um display de sete segmentos.

Funcionamento detalhado:

- **Inicialização dos registradores:**
    - $t7 é utilizado como máscara para selecionar a coluna ativa no teclado, iniciando com o valor 1.
    - Os registradores $s0, $t1 e $t2 recebem os endereços mapeados para o display (0xFFFF0010), para seleção da linha do teclado (0xFFFF0012) e para leitura do teclado (0xFFFF0014), respectivamente.

- O registrador $t3 aponta para o vetor digitos, que contém os padrões binários para os números de 0 a 9 e as letras A a F, correspondendo aos valores hexadecimais possíveis.

- **Varredura das colunas (loop_scan):**

  - O programa escreve em KEYBOARD_ROW um valor que ativa uma coluna do teclado.
  - Em seguida, lê o valor do teclado via KEYBOARD_READ.
  - Se nenhum botão estiver pressionado ($t4 igual a zero), passa para a próxima coluna.

- **Identificação da coluna e linha pressionadas:**

  - O programa utiliza máscaras para identificar qual coluna (col0 a col3) e qual linha (row0 a row3) foram acionadas, baseado nos bits lidos.
  - Com as informações de linha e coluna, calcula o índice correspondente no vetor digitos para obter o padrão do dígito hexadecimal pressionado.
- **Exibição no display:**

  - O padrão binário do dígito é carregado do vetor e enviado para o display através do endereço mapeado.
  - Uma rotina de atraso (delay) é chamada para dar estabilidade à leitura e evitar repetições rápidas.

- **Continuidade do loop:**

  - A máscara de coluna ($t7) é deslocada para a esquerda para ativar a próxima coluna.
  - O ciclo se repete continuamente, permitindo a leitura constante do teclado.

Este programa demonstra uma técnica fundamental para leitura de teclados matriciais, que consiste em ativar colunas sequencialmente e ler as linhas para detectar qual tecla foi pressionada. A manipulação dos bits para identificar a posição exata da tecla e a exibição direta do valor correspondente no display evidenciam conceitos importantes em programação de sistemas embarcados.

Código Fonte:

```
.data
DISPLAY_RIGHT: .word 0xFFFF0010
KEYBOARD_ROW:  .word 0xFFFF0012
KEYBOARD_READ: .word 0xFFFF0014

digitos: .byte 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x77, 0x7C, 0x39,
0x5E, 0x79, 0x71

.text
.globl main

main:
    li $t7, 1
    lw $s0, DISPLAY_RIGHT
    lw $t1, KEYBOARD_ROW
    lw $t2, KEYBOARD_READ
    la $t3, digitos

loop_scan:
    sb $t7, 0($t1)
    lb $t4, 0($t2)

    beqz $t4, next_row

    andi $t5, $t4, 0x0F

    li $t9, 0
    beq $t5, 1, col0
    beq $t5, 2, col1
    beq $t5, 4, col2
    beq $t5, 8, col3
    j next_row

col0:
    li $t9, 0
    j col_done

col1:
    li $t9, 1
```

```
    j col_done

col2:
    li $t9, 2
    j col_done

col3:
    li $t9, 3

col_done:
    andi $t6, $t4, 0xF0
    srl $t6, $t6, 4

    li $t8, 0
    beq $t6, 1, row0
    beq $t6, 2, row1
    beq $t6, 4, row2
    beq $t6, 8, row3
    j next_row

row0:
    li $t8, 0
    j row_done

row1:
    li $t8, 1
    j row_done

row2:
    li $t8, 2
    j row_done

row3:
    li $t8, 3

row_done:
    mul $t5, $t9, 4
    addu $t5, $t5, $t8

    addu $t6, $t3, $t5
    lb $t4, 0($t6)

    sb $t4, 0($s0)

    jal delay

next_row:
    sll $t7, $t7, 1
    li $t8, 16
    blt $t7, $t8, loop_scan
    li $t7, 1
```

```
    j loop_scan

delay:
    li $t0, 100000
delay_loop:
    addi $t0, $t0, -1
    bnez $t0, delay_loop
    jr $ra
```

4 RESULTADOS

Os dois programas desenvolvidos e simulados no ambiente MARS com a extensão Digital Lab Sim apresentaram o funcionamento esperado e demonstraram a interação eficaz entre o código Assembly e os dispositivos de hardware simulados (display de sete segmentos e teclado hexadecimal).

4.1 Exibição Sequencial de 7 Segmentos

No primeiro programa, foi possível verificar a exibição correta e sequencial dos dígitos de 0 a 9 no display de sete segmentos. Cada número permaneceu visível pelo tempo definido (500 milissegundos), permitindo uma visualização clara da sequência. O uso do vetor com os padrões binários provou ser eficiente para mapear os números ao controle dos segmentos, e a manipulação do endereço de memória mapeada funcionou conforme esperado.

As imagens da simulação evidenciaram a precisão na ativação dos segmentos para cada dígito, confirmando o entendimento da lógica de mapeamento e do controle de hardware via acesso direto à memória.

Imagens da Simulação:

Abaixo estão inseridas imagens da simulação no MARS e na ferramenta Digital Lab Sim, ilustrando tanto o código quanto a execução correta do programa:

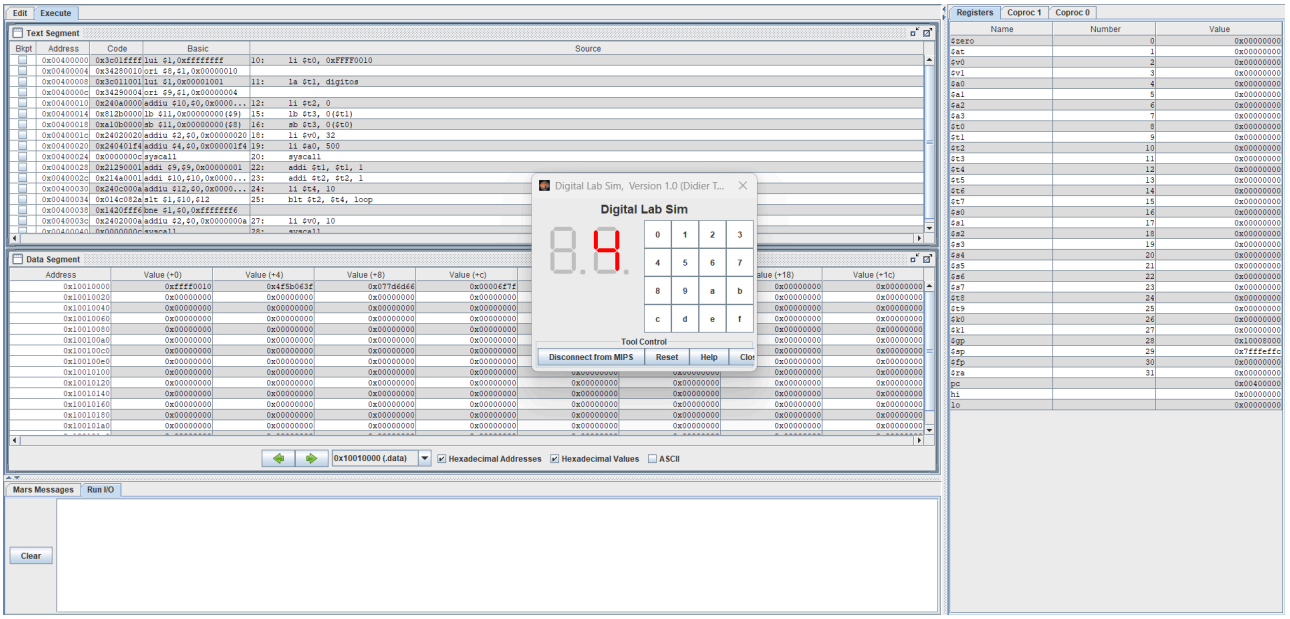

Figura 1 - Display de sete segmentos exibindo o dígito 4

Fonte: Imagem do autor.

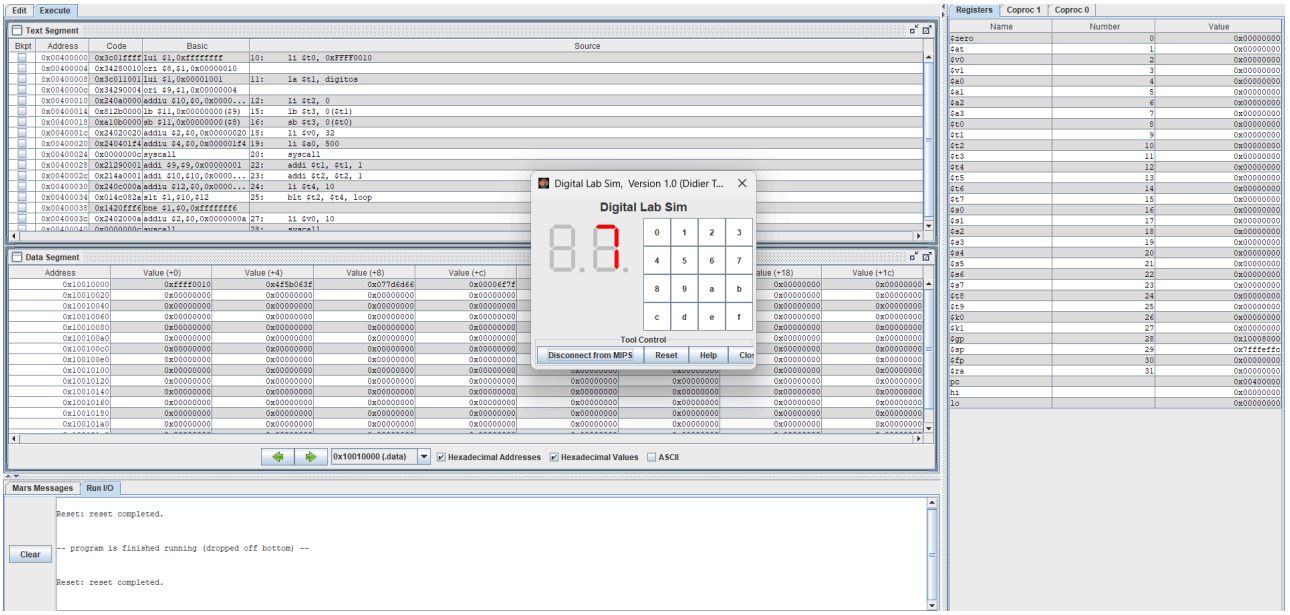

Figura 2 - Display de sete segmentos exibindo o dígito 7

Fonte: Imagem do autor.

4.2 Leitura do Teclado Hexadecimal

No segundo programa, a varredura das colunas do teclado foi implementada corretamente, possibilitando a identificação exata da tecla pressionada por meio da leitura dos sinais nas linhas. O programa exibiu no display o dígito hexadecimal correspondente, incluindo números de 0 a 9 e letras de A a F, de acordo com a tecla detectada.

A interação entre o teclado e o display demonstrou a correta aplicação da lógica de varredura e manipulação de bits, com o delay garantindo a estabilidade da leitura e evitando falsos acionamentos múltiplos. A simulação mostrou a resposta imediata do display ao toque nas teclas, validando o funcionamento do sistema.

Imagens da Simulação:

A seguir, estão apresentadas imagens da simulação, ilustrando:

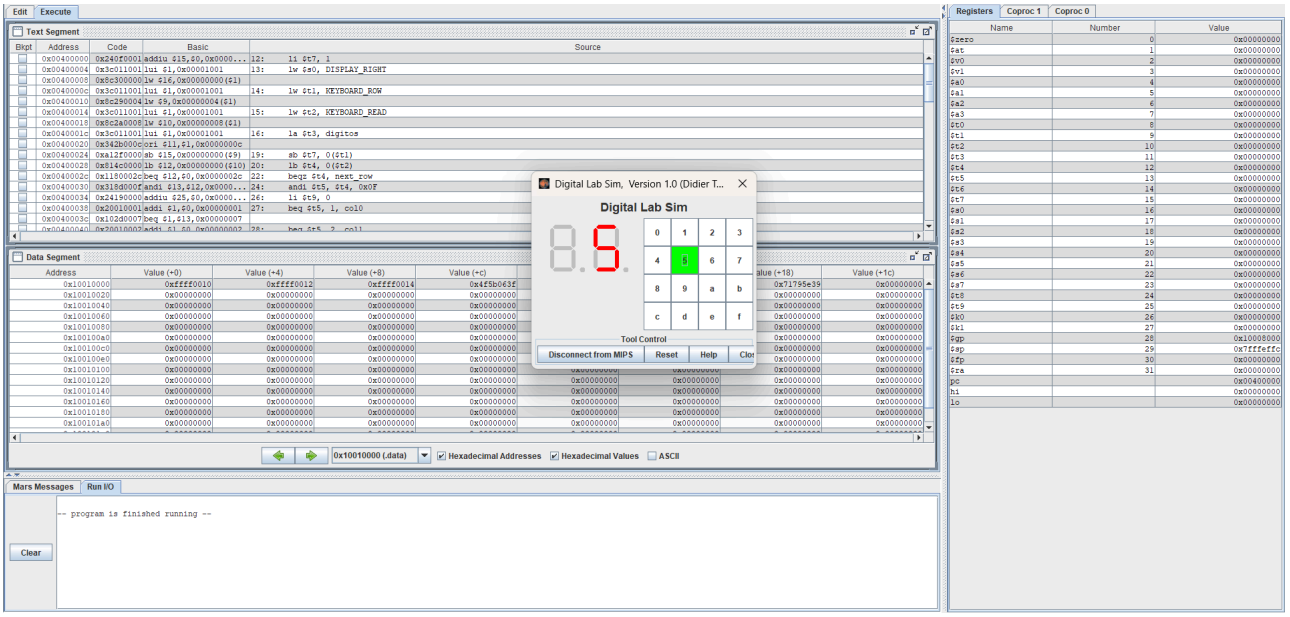

Figura 3 - Teclado Hexadecimal exibindo o dígito 5

Fonte: Imagem do autor.

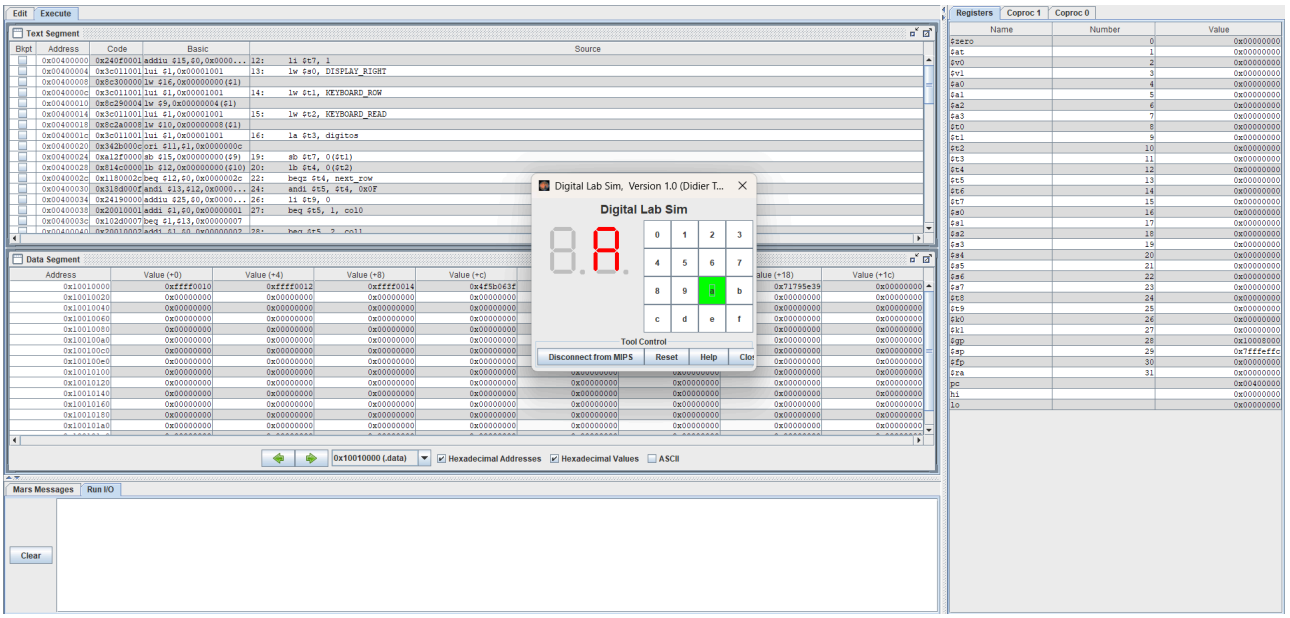

Figura 4 - Teclado Hexadecimal exibindo o caracter A

Fonte: Imagem do autor.

5 Conclusão

A implementação dos dois programas em Assembly MIPS utilizando o simulador MARS e o Digital Lab Sim proporcionou uma compreensão prática dos mecanismos de entrada e saída mapeados em memória. No primeiro programa, a exibição dos números de 0 a 9 no display de 7 segmentos exigiu a criação de uma tabela de codificação segmentada, o que reforçou o entendimento sobre representação binária de caracteres e controle direto de hardware.

Já o segundo programa, que envolveu a leitura contínua do teclado alfanumérico e a exibição do valor correspondente no display, apresentou um desafio adicional: a necessidade de realizar leituras constantes e responder apenas a entradas válidas, filtrando o valor -1 retornado quando nenhuma tecla está pressionada. Essa lógica é fundamental em sistemas embarcados reais, onde a eficiência do loop principal pode afetar o desempenho geral do sistema.

Além disso, o trabalho demonstrou como manipular diferentes instruções MIPS, como carregamento de byte (lb), escrita em endereços específicos de I/O, comparação de valores e controle de laços, promovendo um domínio mais profundo sobre a linguagem Assembly.