UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO - CTC

ORGANIZAÇÃO DE COMPUTADORES I - INE5411-03208B

PROFESSOR: MARCELO DANIEL BEREJUCK

ALUNOS: GUSTAVO BATISTELL (12101228) E LUCAS DE S. MORO (23201136)

RELATÓRIO - LABORATÓRIO 2

Florianópolis, Outubro de 2024.

Introdução

Este relatório apresenta e explica a solução de implementação da atividade proposta no Laboratório 2 onde o display de 7-segmentos do simulador MARS 4.5 deve apresentar inicialmente a sequência de 0 a 9 e em segunda etapa os valores são fornecidos pelo usuário através de entrada via teclado alfanumérico também do próprio simulador.

Exercício 1

Para implementar essa primeira etapa, a dupla abriu o display do simulador em *Tools/Digital Lab Sim* e seguiu as orientações contidas no *help* desta janela do simulador, onde o display de 7-segmentos da direita, escolhido sem critério específico, "observa" o registrador com endereço 0xFFFF0010, conforme a Figura 1 abaixo.

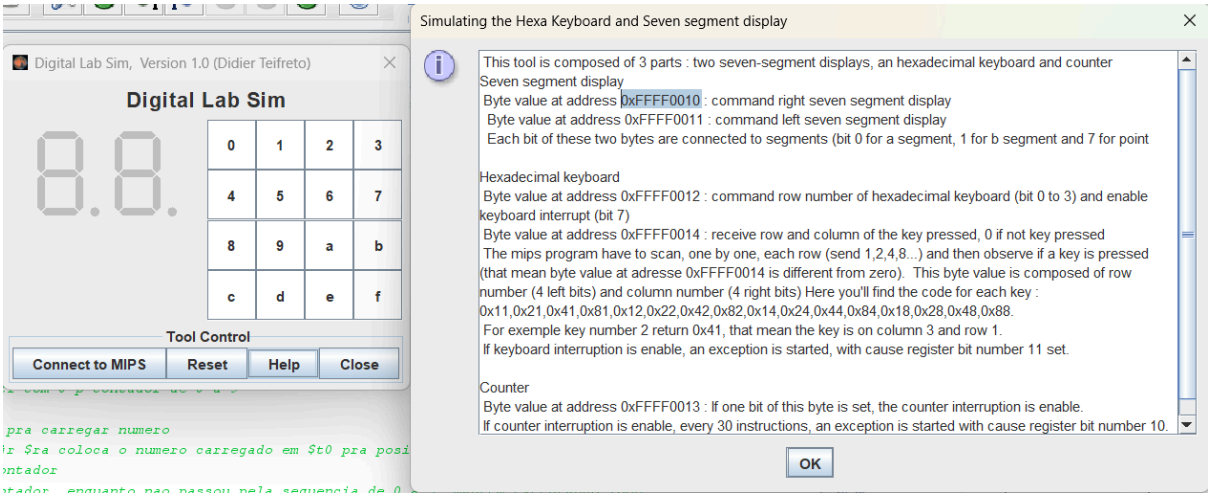

Figura 1 - *Display* e *Help* disponíveis no simulador

Os valores a serem visualizados no display de 7-segmentos foram codificados em binário e convertidos para decimal baseados na tabela elaborada pela dupla apresentada na Figura 2 abaixo.

leitura no display	segmento .	g	f	e	d	c	b	a	decimal
0	0	0	1	1	1	1	1	1	63
1	0	0	0	0	0	1	1	0	6
2	0	1	0	1	1	0	1	1	91
3	0	1	0	0	1	1	1	1	79
4	0	1	1	0	0	1	1	0	102
5	0	1	1	0	1	1	0	1	109
6	0	1	1	1	1	1	0	1	125
7	0	0	0	0	0	1	1	1	7
8	0	1	1	1	1	1	1	1	127
9	0	1	1	0	1	1	1	1	111
a	0	1	1	1	0	1	1	1	119
b	0	1	1	1	1	1	0	0	124
c	0	0	1	1	1	0	0	1	57
d	0	1	0	1	1	1	1	0	94
e	0	1	1	1	1	0	0	1	121
f	0	1	1	1	0	0	0	1	113

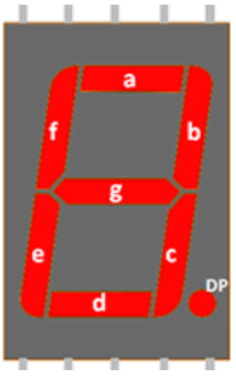

Figura 2 - Tabela e figura de display de 7 segmentos utilizadas para codificação

O código em assembly (encaminhado anexo) utilizou um loop e a chamada de uma função *carrega_numero*, que implementa uma estrutura parecida com *Jump Address Table*, bastante útil em case switch. A Figura 3 abaixo mostra a funcionalidade esperada.

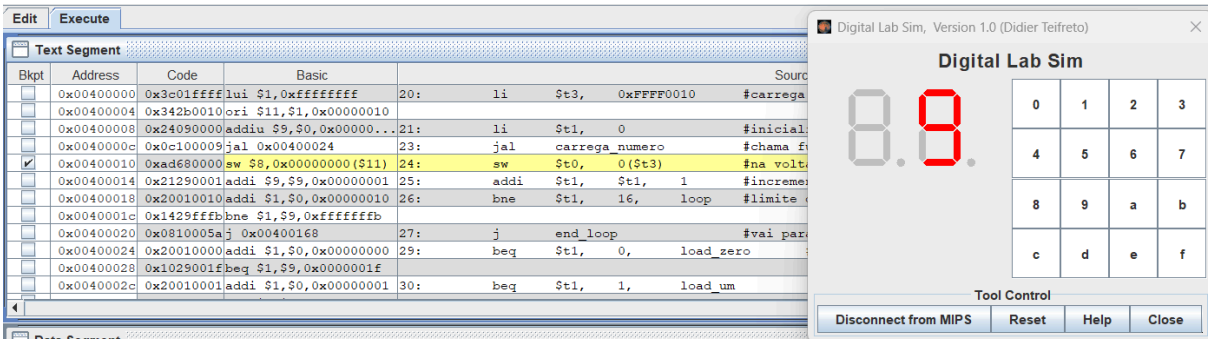

Figura 3 - Teste funcional com display

Neste primeiro código foi utilizado o formato *.word* para os dados, mas posteriormente, na execução da segunda parte do experimento, notou-se que poderia ser *.byte*. A estrutura semelhante a *JAT* também careceria de melhoria para evitar o uso repetitivo de *beq*.

Exercício 2

No segundo exercício, a proposta foi implementar, novamente na ferramenta Digital Lab Sim, a leitura dos valores a partir do teclado alfanumérico (de **0** até **f**), e exibir o valor da tecla pressionada no display de 7 segmentos. Para essa segunda etapa, foram utilizados os endereços de controle do teclado (0xFFFF0012) e da memória do código da tecla pressionada (0xFFFF0014), além do endereço do display usado no exercício anterior.

De início, carregamos os endereços dos dispositivos nos registradores apropriados, também inicializamos um registrador com o valor -1, registrador esse que atua como um marcador para armazenar o código da última tecla pressionada, evitando múltiplas atualizações no display.

```
.text
.globl main
main:
        li      $s0,    0xFFFF0010      #endereco do display de 7 segmentos
        li      $s2,    0xFFFF0012      #endereco teclado
        li      $s1,    0xFFFF0014      #endereco do codigo da tecla pressionada
        li      $t3,    -1              # inicializa $t3 com valor invalido
```

Figura 4 - Carregamento dos endereços

A partir daqui, o loop principal é iniciado, realizando uma varredura das linhas do teclado hexadecimal para detectar as teclas pressionadas. Esse processo envolve, a identificação da linha, a leitura do código da tecla e a decodificação para exibir esse valor no display de 7 segmentos.

```
loop:
        li      $t0,    0x01            #seleciona linha 1
        sb      $t0,    0($s2)          #escreve em $s2 para selecionar a linha
        lbu     $t1,    0($s1)          #carrega tecla pressionada da linha 1
        bne     $t1,    $zero,  processa_tecla

        li      $t0,    0x02            #seleciona linha 2
        sb      $t0,    0($s2)
        lbu     $t1,    0($s1)
        bne     $t1,    $zero,  processa_tecla

        li      $t0,    0x04            #seleciona linha 3
        sb      $t0,    0($s2)
        lbu     $t1,    0($s1)
        bne     $t1,    $zero,  processa_tecla

        li      $t0,    0x08            #seleciona linha 4
        sb      $t0,    0($s2)
        lbu     $t1,    0($s1)
        bne     $t1,    $zero,  processa_tecla
        j       loop                    # Repete o loop
```

Figura 5 - Varredura das linhas do teclado em loop

A varredura das linhas do teclado segue uma estrutura de repetição que verifica as quatro linhas disponíveis para identificar se alguma tecla foi pressionada. O procedimento começa carregando um valor que representa a seleção de uma linha do teclado (primeiramente, a linha 1, com o valor 0x01) e, em seguida, escreve esse valor no endereço do controlador do teclado, armazenado no registrador **$s0**. Após selecionar a linha, o código carrega o valor da tecla pressionada daquela linha, acessando a memória de código da tecla armazenada no endereço representado por **$s1**. Se nenhuma tecla foi pressionada, o valor retornado será zero, e o loop continua com a verificação da próxima linha.

Na Figura 6 abaixo é apresentado o funcionamento com as implementações já descritas acima.

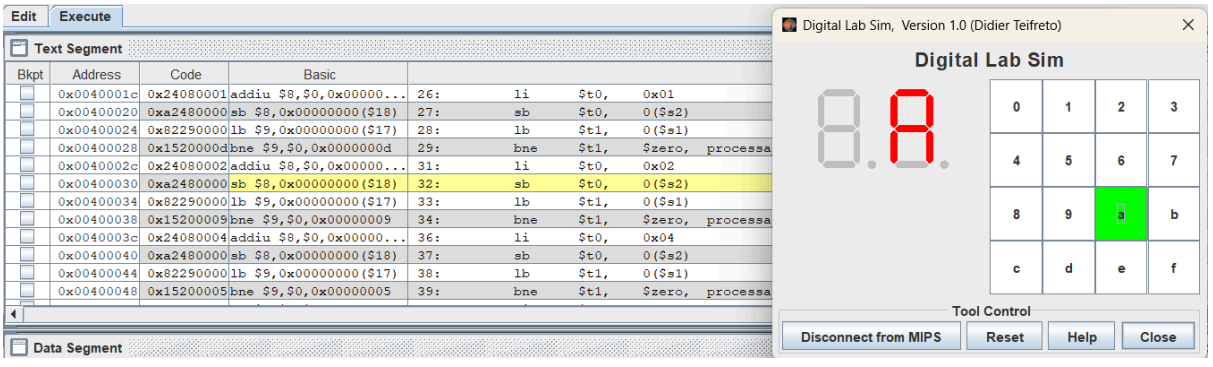

Figura 6 - Código em execução onde a tecla pressionada é exibida no display

Conclusão

Notou-se que era desnecessário alocar uma palavra inteira para cada dado, corrigindo para uso de byte na segunda etapa.

Notou-se também a falta de uso da convenção de chamada de funções, que apesar do exemplo simples, poderia ser implementada em melhoria futura.

Viu-se a necessidade do uso da instrução lbu, pois lb estava ocasionando em uma extensão de sinal, levando a possíveis resultados incorretos.