Relatório do Laboratório 02 - Organização de Computadores

Grupo:

- Henrique Mateus Teodoro - 23100472
- Rodrigo Schwartz - 23100471

Exercício 1

O exercício 1 consiste em mostrar os números de 0 até 9 de forma sequencial em um display, disponibilizado pela ferramenta Digital Lab Sim do Mars.

Inicialmente, foi feita a leitura do “manual” da ferramenta Digital Lab Sim, para que o funcionamento do display pudesse ser melhor compreendido. Através deste manual, foi possível encontrar os endereços dos displays:

> This tool is composed of 3 parts : two seven-segment displays, an hexadecimal keyboard and counter
> Seven segment display
> Byte value at address 0xFFFF0010 : command right seven segment display
> Byte value at address 0xFFFF0011 : command left seven segment display
> Each bit of these two bytes are connected to segments (bit 0 for a segment, 1 for b segment and 7 for point

Foi então decidido que o display da direita seria utilizado, e por conta disso, seria necessário manipular o endereço 0xFFFF0010, para que fosse possível passar os valores responsáveis por mostrar cada número (de 0 a 9) no display. Para isso, calculamos o valor em binário que ativaria cada número adequadamente no display de 7 segmentos, e então convertemos todos para decimal, visto que para mostrar no display, é necessário passar o correspondente do binário em decimal.

Assim, no segmento de variáveis, instanciamos cada número, com seu respectivo valor:

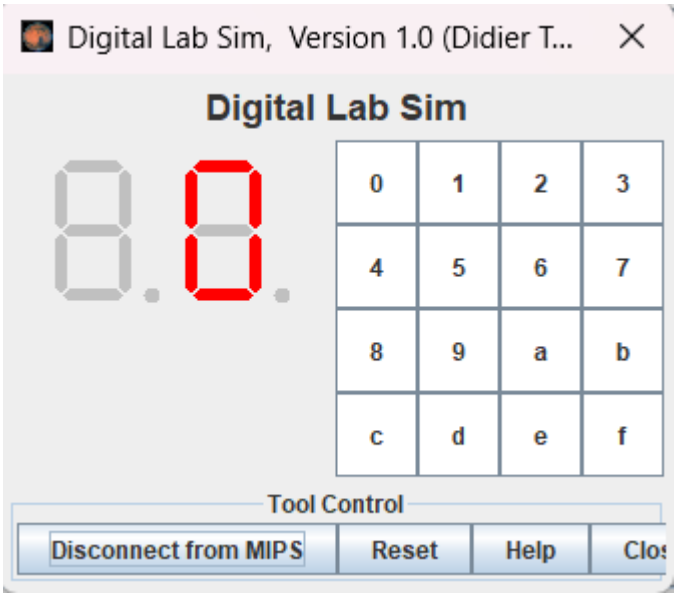
```
.data
        num0: .byte 63      # valor do byte para representar o número 0
        num1: .byte 6       # valor do byte para representar o número 1
        num2: .byte 91      # valor do byte para representar o número 2
        num3: .byte 79      # valor do byte para representar o número 3
        num4: .byte 102     # valor do byte para representar o número 4
        num5: .byte 109     # valor do byte para representar o número 5
        num6: .byte 125     # valor do byte para representar o número 6
        num7: .byte 7       # valor do byte para representar o número 7
        num8: .byte 127     # valor do byte para representar o número 8
        num9: .byte 111     # valor do byte para representar o número 9
```

Após essa etapa, compreendemos que seria necessário apenas carregar o endereço 0xFFFF0010 para um registrador, carregar o conteúdo de uma variável, como por exemplo num0, para um outro registrador, e então, passar o valor da

variável, para o endereço de memória que representa o display. Assim, o display deveria acender, mostrando o número que foi passado para ele. Exemplificando melhor, se for passado o valor 63 para o endereço de memória 0xFFFF0010, como o valor 63 liga os led necessários para mostrar o número 0 no display, o valor 0 deve ser mostrado no display. Para isso, o seguinte trecho de código é necessário:

```
.text
        li $s0, 0xFFFF0010 # carrega o endereço do display da direita em reg $s0

        la $t0, num0       # carrega o endereço de num0 no reg $t0
        lb $t1, 0($t0)     # carrega o conteúdo de num0 para $t0
        sb $t1, ($s0)      # passa o conteúdo de num0 para o endereço do display
```

E obtêm-se o número 0 sendo mostrado corretamente no display:



Por tanto, para mostrar os valores de 0 até 9, poderíamos apenas replicar o bloco de código anteriormente mais 9 vezes, passando o valor correspondente aos números que deveriam ser mostrados no display, que foram instanciados em .data.

É importante salientar que, não seria necessário instanciar as 10 variáveis na memória, carregar o conteúdo de cada uma para um registrador, e então passar esse valor para o endereço da memória do display para que o número fosse mostrado no display utilizando esse código. Seria possível utilizar um li, para carregar imediatamente o valor de cada número no registrador, e então passar esse valor do registrador para o endereço de memória do display. Entretanto, mesmo economizando algumas instruções, o código ficaria extenso e repetitivo, e foi aí que começamos a pensar em uma forma de utilizar um laço de repetição para que esse processo fosse repetido até todos os números serem mostrados no display. Assim, o código ficaria mais curto e fácil de ser compreendido.

Após algum tempo pensando, lembramos que uma palavra no processador MIPS é armazenada utilizando 4 bytes (32 bits). Como cada variável representa um número, e cada número é um byte, cada palavra armazenada na memória está armazenando 4 números. E como as variáveis foram armazenadas sequencialmente na memória, se a memória fosse percorrida de 1 em 1 byte, partindo do endereço da primeira variável declarada (nesse caso a variável num0), seria possível acessar o conteúdo de cada variável em um loop.

Para isso iniciamos o código carregando o endereço do display 0xFFFF0010 para um registrador, carregando o endereço da primeira variável (num0) para outro registrador (visto que será utilizado como endereço base / ponteiro para o loop), e instanciamos em dois outros registradores duas "variáveis" que foram utilizadas no loop, uma como contador que inicia em 0, e outra como comparador que inicia em 10:

```
.text
        li $s0, 0xFFFF0010 # carrega o endereço do display da direita em reg $s0

        la $t0, num0       # carrega o endereço de num0 no reg $t0
        li $t2, 0          # carrega o valor 0 para o reg $t2, que será o contador do loop
        li $t3, 10         # carrega o valor 10 para o reg $t3, que servirá como o comparador do loop
```

O loop foi então criado, com uma ideia simples:

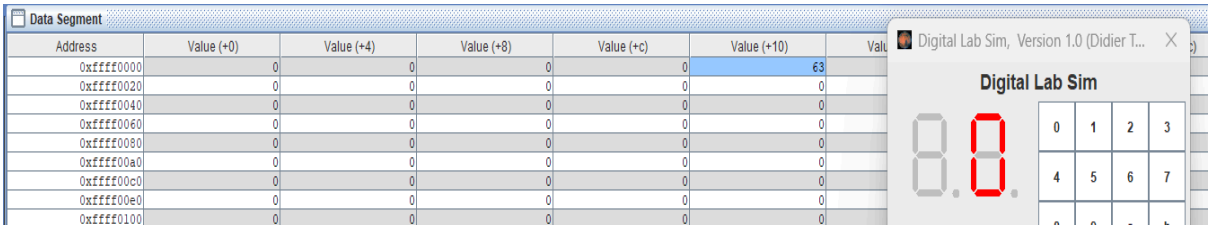
```
Loop:   lb $t1, 0($t0)     # carrega o conteúdo de num0 para $t0
        sb $t1, ($s0)      # passa o conteúdo de num0 para o endereço do display
        addi $t2, $t2, 1   # soma um no contador, ou seja, soma um em $t2
        addi $t0, $t0, 1   # soma um no endereço base, que é o endereço de num0;
        beq $t2, $t3, Exit # se o contador != comparador, ele refaz o loop, se não, da um jump para Exit
        j Loop

Exit:   nop
```

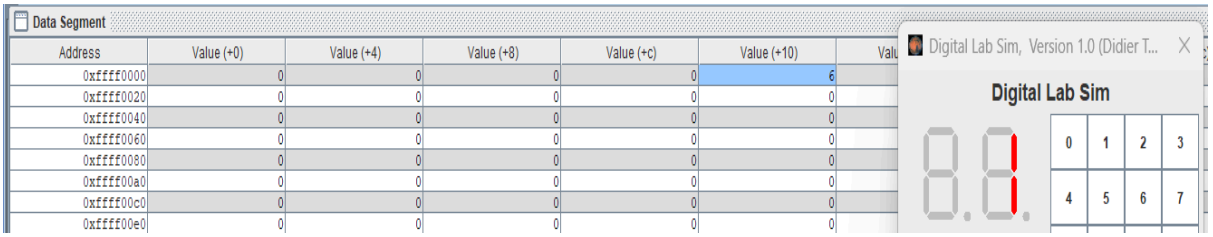
Inicialmente, o laço começa carregando em $t1 o conteúdo de $t0, que na primeira iteração, corresponde ao valor da variável num0. Após isso, o valor é passado para o endereço do display e o número é então mostrado no mesmo. Depois disso, o contador que está armazenado em $t2 é incrementado, para que seja possível saber em qual iteração se está. Nesse caso, acaba de ser feita a 1 iteração do loop. O endereço base armazenado em $t0 é então incrementado também, visto que deve apontar para a variável que está a 1 posição do endereço base (num0), que nesse caso é a variável num1.

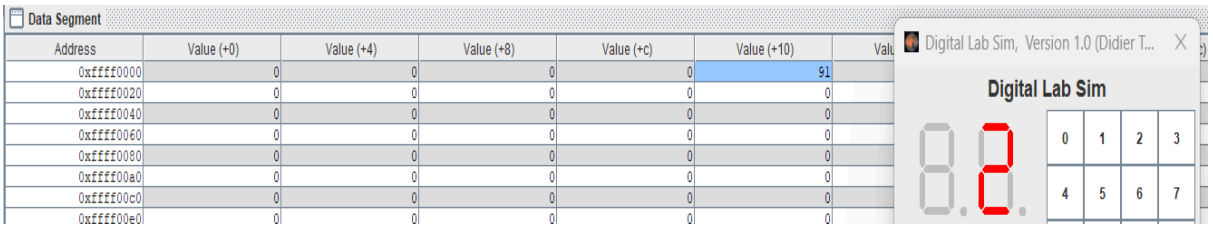
Depois disso, é feito uma comparação utilizando um branch on equal (beq), para saber se o contador ($t2) já chegou no valor do comparador ($t3). Se sim, significa que o todos os números já foram mostrados no display (ocorreram as 10 iterações), e então o programa encerra, indo para a label exit. Senão, volta para o

loop através de uma instrução jump, para que a próxima variável possa ser mostrada no display.

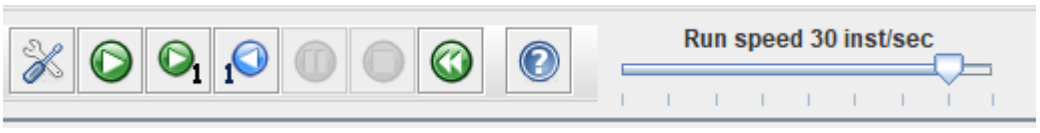
Ao rodar o programa linha por linha, é possível notar o correto funcionamento do código, como constam algumas imagens a seguir:







Para executar o programa todo de uma vez, e ver todas as iterações sequencialmente, é apenas necessário ajustar a velocidade de execução do código, para que seja possível ver as iterações. Para isso, basta deixar a configuração de velocidade da seguinte forma:



É possível ainda fazer a diminuição dessa velocidade, caso queira uma iteração mais lenta do programa.

Exercício 2

O exercício 2 consiste em, utilizando a ferramenta Digital Lab Sim, ler o valor pressionado em um teclado alfanumérico e exibi-lo em um display de 7 segmentos (de 0 até f). Como não há nenhum dado salvo na memória no início do código, o segmento .data estaria vazio, portanto não é necessário escrevê-lo no código. Inicialmente, carregamos alguns endereços referentes às ferramentas do Digital Lab Sim:

- O endereço 0xFFFF0012, salvo em $s0, faz referência ao espaço de memória utilizado para fazer o controle de linhas do teclado alfanumérico.
- O endereço 0xFFFF0014, salvo em $s1, faz referência ao espaço de memória que armazena a informação da tecla pressionada.
- Por fim, o endereço 0xFFFF0010, salvo em $s2, faz referência ao espaço de memória que controla os valores do display de sete segmentos mais à direita.

Depois disso, guardamos o valor 0 no registrador $t3, pois futuramente ele vai ser utilizado como comparador para verificar a repetição do clique de um botão, e assim, não ficar atualizando o display com o mesmo valor repetidas vezes. Dessa maneira o número não pisca no display. A seguinte imagem descreve as operações anteriores:

```
.text
        li $s0, 0xFFFF0012
        li $s1, 0xFFFF0014
        li $s2, 0xFFFF0010

        li $t3, 0
```

Após essas instruções serem realizadas, o loop principal é iniciado. A partir daqui, um processo para verificar se uma tecla foi pressionada é repetido para cada linha do teclado alfanumérico. Primeiro é necessário informar qual linha vamos fazer a verificação de tecla pressionada. Para isso devemos escrever no endereço 0xFFFF0012 o valor referente a determinada linha. Caso alguma tecla dessa linha for pressionada, um valor (referente à coluna e à linha da tecla) é escrito no endereço 0xFFFF0014. Caso nenhuma tecla dessa linha for pressionada, ele escreve o valor 0 nesse endereço. Inicialmente, é necessário fazer a verificação da linha 1. A primeira coisa a se fazer é armazenar o valor da linha que queremos verificar. Para ler a linha 1, é necessário escrever em 0xFFFF0012 o número 0x01 (em hexadecimal), pois o bit 0 desse número é 1. Para ler a linha 2, é necessário escrever no mesmo endereço o número 0x02, pois o bit 1 desse número é 1. Já para a linha 3 e 4, é necessário escrever o número 0x04 (pois o bit 2 desse número é 1) e 0x08 (pois o bit 3 desse número é 1), respectivamente. Como queremos primeiro ler a linha 1 e processar o resultado, o seguinte código é necessário:

```
loop_principal:
        li $t0, 0x01            # bit 0 = 1 -> linha 1
        sb $t0, 0($s0)          # linha 1 selecionada(escrevendo em 0xFFFF0012)
        lb $t1, 0($s1)          # $t1= código da tecla pressionada da linha 1 ($t1=0 se não for pressionada)
        bne $t1, $zero, linha1  # faz a verificação se alguma tecla da linha 1 foi pressionada
```

No código da imagem acima, o byte 0x01 é armazenado no registrador $t0. Como precisa-se verificar se alguma tecla da linha 1 foi pressionada, a instrução sb é utilizada, e escrevemos o byte do valor 0x01 no endereço 0xFFFF0012 (salvo em $s0). Assim, caso alguma tecla desta linha realmente tenha sido clicada, o endereço 0xFFFF0014 vai armazenar um valor baseado na coluna e linha do lugar pressionado. Caso nenhuma tecla seja apertada nessa linha, o valor 0 é escrito. Esse resultado é armazenado no registrador $t1, que é utilizado posteriormente na instrução bne para fazer a comparação com zero. Caso o valor seja diferente de 0, o código vai para um label referente às ações tomadas para descobrir qual tecla da linha 1 foi pressionada e, consequentemente, escrevê-la no display. Caso seja igual a zero, ele continua a execução normalmente e verifica se alguma tecla das próximas linhas foi pressionada. Assim, a estrutura desse código é replicada para cada linha, de acordo com suas necessidades (alterar valor da linha a ser lida ou label de destino, por exemplo). No final da verificação de todas linhas, utiliza-se um jump para voltar ao início do loop_principal, ficando preso nesse loop infinitamente.

```
loop_principal:
        li $t0, 0x01            # bit 0 = 1 -> linha 1
        sb $t0, 0($s0)          # linha 1 selecionada(escrevendo em 0xFFFF0012)
        lb $t1, 0($s1)          # $t1= código da tecla pressionada da linha 1 ($t1=0 se não for pressionada)
        bne $t1, $zero, linha1  # faz a verificação se alguma tecla da linha 1 foi pressionada

        li $t0, 0x02            # bit 1 = 1 -> linha 2
        sb $t0, 0($s0)          # linha 2 selecionada(escrevendo em 0xFFFF0012)
        lb $t1, 0($s1)          # $t1= código da tecla pressionada da linha 2 ($t1=0 se não for pressionada)
        bne $t1, $zero, linha2  # faz a verificação se alguma tecla da linha 2 foi pressionada

        li $t0, 0x04            # bit 2 = 1 -> linha 3
        sb $t0, 0($s0)          # linha 3 selecionada(escrevendo em 0xFFFF0012)
        lb $t1, 0($s1)          # $t1= código da tecla pressionada da linha 3 ($t1=0 se não for pressionada)
        bne $t1, $zero, linha3  # faz a verificação se alguma tecla da linha 3 foi pressionada

        li $t0, 0x08            # bit 3 = 1 -> linha 4
        sb $t0, 0($s0)          # linha 4 selecionada(escrevendo em 0xFFFF0012)
        lb $t1, 0($s1)          # $t1= código da tecla pressionada da linha 4 ($t1=0 se não for pressionada)
        bne $t1, $zero, linha4  # faz a verificação se alguma tecla da linha 4 foi pressionada

        j loop_principal
```

Continuando com o exemplo da linha 1, após descobrir que uma tecla dessa linha foi pressionada, precisamos descobrir qual. Para isso é utilizado o valor escrito em 0xFFFF0014 e armazenado em $t1. O label que descreve a linha 1 é o seguinte:

```
linha1:
        beq $t1, $t3, loop_principal
        move $t3, $t1
        beq $t1, 17, zero
        beq $t1, 33, um
        beq $t1, 65, dois
        beq $t1, -127, tres
        j loop_principal
```

A primeira e segunda servem basicamente para comparar o valor digitado na iteração atual com o da iteração anterior. Caso os dois sejam iguais, significa que a mesma tecla está pressionada e não é necessário imprimi-la no display novamente, voltando para o laço principal. Caso contrário, o código segue sua sequência. O valor armazenado em $t1 contém informações sobre a linha e coluna pressionada, como descreve a ferramenta Digital Lab Sim:

```
Hexadecimal keyboard
 Byte value at address 0xFFFF0012 : command row number of hexadecimal keyboard (bit 0 to 3) and enable keyboard interrupt (bit 7)
 Byte value at address 0xFFFF0014 : receive row and column of the key pressed, 0 if not key pressed
 The mips program have to scan, one by one, each row (send 1,2,4,8...) and then observe if a key is pressed (that mean byte value at adresse 0xFFFF0014 is different from zero).  This byte value is composed of row number (4 left bits) and column number (4 right bits) Here you'll find the code for each key : 0x11,0x21,0x41,0x81,0x12,0x22,0x42,0x82,0x14,0x24,0x44,0x84,0x18,0x28,0x48,0x88.
 For exemple key number 2 return 0x41, that mean the key is on column 3 and row 1.
 If keyboard interruption is enable, an exception is started, with cause register bit number 11 set.
```

Por exemplo, a linha 1 e coluna 1 (número 0) é representada pelo valor 0x11, em hexadecimal. O número 1 é representado pelo valor 0x21, 2 é representado por 0x41 e 3 é representado 0x81. No código anterior, essa comparação é feita com a representação inteira desse byte, e não em hexadecimal, por isso os valores estão diferentes nos beq. No último número de cada linha do teclado (3, 7, b e f) ocorre um overflow com o valor de sua representação, pois isso a representação daquela tecla como inteiro se torna um número negativo. Comparando o valor de $t1 com cada retorno de tecla possível daquela linha, utilizamos um beq para ir ao label responsável por imprimir no display de sete segmentos o número selecionado. No final, para garantir o loop infinito, há um jump que vai para o início do loop_principal. A mesma estrutura de código é repetida para as demais linhas, como mostra a imagem a seguir:

```
linha1:
        beq $t1, $t3, loop_principal   # verifica se a tecla pressionada na iteração atual é a mesma que a da iteração anteriro, se sim, retorna para o loop principal
        move $t3, $t1                  # salva o valor que foi digitado, para não entrar aqui de novo repetidas vezes
        beq $t1, 17, zero              # verifica se o número da tecla pressionada equivale ao número zero, a partir do código da tecla
        beq $t1, 33, um                # verifica se o número da tecla pressionada equivale ao número um, a partir do código da tecla
        beq $t1, 65, dois              # verifica se o número da tecla pressionada equivale ao número dois, a partir do código da tecla
        beq $t1, -127, tres            # verifica se o número da tecla pressionada equivale ao número tres, a partir do código da tecla
        j loop_principal

linha2:
        beq $t1, $t3, loop_principal   # verifica se a tecla pressionada na iteração atual é a mesma que a da iteração anteriro, se sim, retorna para o loop principal
        move $t3, $t1                  # salva o valor que foi digitado, para não entrar aqui de novo repetidas vezes
        beq $t1, 18, quatro            # verifica se o número da tecla pressionada equivale ao número quatro, a partir do código da tecla
        beq $t1, 34, cinco             # verifica se o número da tecla pressionada equivale ao número cinco, a partir do código da tecla
        beq $t1, 66, seis              # verifica se o número da tecla pressionada equivale ao número seis, a partir do código da tecla
        beq $t1, -126, sete            # verifica se o número da tecla pressionada equivale ao número sete, a partir do código da tecla
        j loop_principal

linha3:
        beq $t1, $t3, loop_principal   # verifica se a tecla pressionada na iteração atual é a mesma que a da iteração anteriro, se sim, retorna para o loop principal
        move $t3, $t1                  # salva o valor que foi digitado, para não entrar aqui de novo repetidas vezes
        beq $t1, 20, oito              # verifica se o número da tecla pressionada equivale ao número oito, a partir do código da tecla
        beq $t1, 36, nove              # verifica se o número da tecla pressionada equivale ao número novo, a partir do código da tecla
        beq $t1, 68, A                 # verifica se o número da tecla pressionada equivale ao número A, a partir do código da tecla
        beq $t1, -124, B               # verifica se o número da tecla pressionada equivale ao número B, a partir do código da tecla
        j loop_principal

linha4:
        beq $t1, $t3, loop_principal   # verifica se a tecla pressionada na iteração atual é a mesma que a da iteração anteriro, se sim, retorna para o loop principal
        move $t3, $t1                  # salva o valor que foi digitado, para não entrar aqui de novo repetidas vezes
        beq $t1, 24, C                 # verifica se o número da tecla pressionada equivale ao número C, a partir do código da tecla
        beq $t1, 40, D                 # verifica se o número da tecla pressionada equivale ao número D, a partir do código da tecla
        beq $t1, 72, E                 # verifica se o número da tecla pressionada equivale ao número E, a partir do código da tecla
        beq $t1, -120, F               # verifica se o número da tecla pressionada equivale ao número F, a partir do código da tecla
        j loop_principal
```

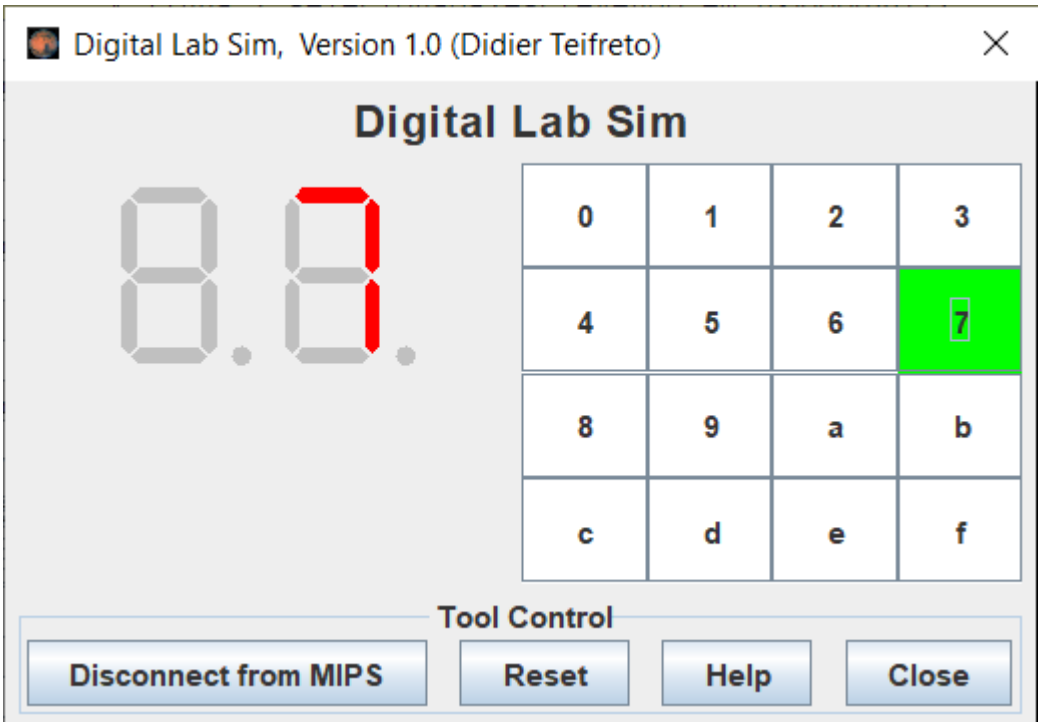
Há labels específicos para imprimir todos os possíveis números no display de sete segmentos. Vamos considerar o exemplo sendo a tecla 0 pressionada, caindo no seguinte label:

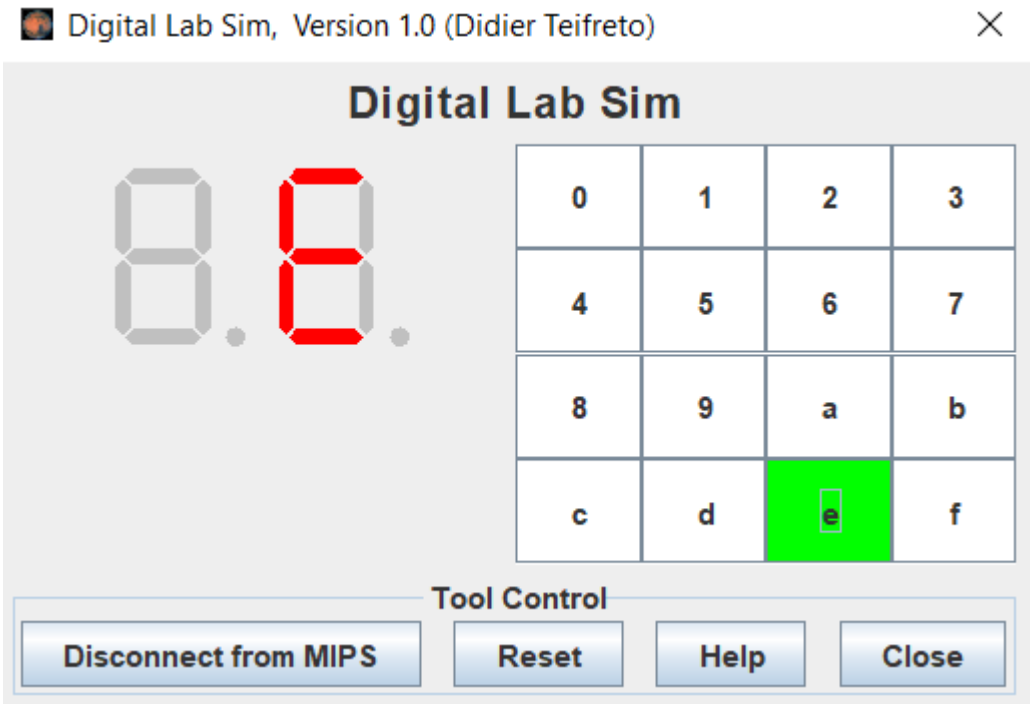
```
zero:
        li $t2, 63          #valor do byte para representar o número 0
        sb $t2, 0($s2)      #passa o valor que representa o número 0 para o endereço do display
        j loop_principal
```

Carregamos para $t2 o valor decimal do byte para representar o número 0 no display, que nesse caso é 63 (00111111). Após isso, basta escrever o byte com sb no endereço 0xFFFF0010 (endereço do display da direita), que está salvo em $s2, imprimindo a tecla pressionada no display. Como o laço precisa executar infinitamente, há um jump no final voltando para o laço principal. Essa mesma estrutura é replicada para os outros números, variando somente seu valor de representação. Por exemplo, para os números 9 e A:

```
nove:
        li $t2, 111          #valor do byte para representar o número 9
        sb $t2, 0($s2)      #passa o valor que representa o número 9 para o endereço do display
        j loop_principal

A:
        li $t2, 119         #valor do byte para representar a letra A
        sb $t2, 0($s2)      #passa o valor que representa a letra A para o endereço do display
        j loop_principal
```

Por fim, o laço de principal continua infinitamente e os valores pressionados são exibidos no display. Segue alguns prints para demonstrar a execução das instruções:





Apesar da execução correta das instruções e o aparentemente bom funcionamento, notamos um bug que pensamos ser da própria plataforma MARS. Ao executar o código com run speed máximo, após entrar no loop infinito e pressionar algumas teclas seguidas vezes, o MARS para de funcionar. Acreditamos que seja um problema do próprio MARS, ocorrendo pela entrada em um laço infinito com velocidade máxima. Esse problema deixa de acontecer quando diminuímos o run speed, no canto superior da tela.

Diminuindo a barra da velocidade em uma unidade, o código já executa perfeitamente.