UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Relatório Laboratório 01

Maria Eduarda Teixeira Costa e Sofia Gazolla da Costa Silva

Departamento de Informática e Estatística - Universidade Federal de Santa Catarina

Organização de Computadores I

Professor Marcelo Daniel Berejuck

Setembro 2025

1 Introdução

As atividades propostas eram divididas em dois tipos diferentes: o primeiro sendo via console e outra utilizando a ferramenta Digital Lab Sim do MARS. Para cada tipo, haviam dois exercícios a serem desenvolvidos.

A primeira atividade via console consistia na implementação da fórmula de Bhaskara, com os valores de a, b e c previamente definidos e armazenados em variáveis na memória. Após o cálculo, os resultados deveriam ser guardados em variáveis na seção de dados. A segunda também consistia na implementação da fórmula de Bhaskara, mas desta vez com a, b e c sendo fornecidos pelo usuário. Além de armazenar os resultados na memória, o programa deveria imprimi-los no terminal.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Figure 1: Fórmula de Bhaskara

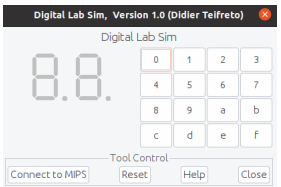

Figure 2: Digital Lab Sim

Já as tarefas no Digital Lab Sim envolveram o uso do display de sete segmentos e do teclado alfanumérico da ferramenta. O primeiro exercício pedia um programa que

exibisse, em sequência, os números de 0 a 9 no display. No segundo, o teclado deveria ser lido e conectado ao display, exibindo, assim, o valor da tecla que fosse pressionada.

2 Desenvolvimento

1. Atividades via Console:

(a) Atividade 1

Para o desenvolvimento da primeira atividade, que pedia a aplicação da fórmula de Bhaskara em valores já armazenados na memória, com o resultado final também sendo guardado, a primeira etapa consistiu na definição das variáveis na seção `.data`. Após declarar as cinco variáveis necessárias, com essas sendo os valores de `a, b` e `c` e, também, duas variáveis no momento nulas, chamadas `x0` e `x1`, as três primeiras, necessárias para o cálculo, foram carregadas em registradores *saved temporaries*. Essa foi a única etapa do programa que usou esse tipo de registrador, pois para todas as contas intermediárias foram usados registradores temporários.

Em seguida, foi realizado o cálculo de Δ ($b^2 - 4ac$), utilizando três operações de multiplicação e uma de subtração. Assim que o valor de Δ foi obtido, calculou-se sua raiz quadrada. Como o exercício lidava apenas com números inteiros, implementamos um laço que testa valores inteiros a partir de zero, elevando-os ao quadrado até encontrar ou ultrapassar o valor de delta, obtendo assim sua raiz inteira.

Por fim, o cálculo de $-b$ foi feito subtraindo b do registrador zero, e as duas raízes foram determinadas, uma somando o valor da raiz quadrada de delta à -b e outra subtraindo esse mesmo valor, com a divisão por $2a$ sendo realizada em seguida. O resultado inteiro dessa divisão foi movido para `$t0` (primeira raiz, que realiza a soma) e `$t1` (segunda raiz, que realiza a subtração). Finalmente, os valores foram armazenados em suas respectivas variáveis na memória.

(b) Atividade 2

A segunda atividade também pedia a implementação da fórmula de Bhaskara, entretanto, com valores de a, b e c inseridos pelo usuário. Além disso, os resultados

da conta deveriam ser armazenados na memória e, também, impressos no terminal. As duas principais diferenças estão no ínicio e no final do código. Tudo que ocorre entre o cálculo de Δ e o armazenamento das varíaveis, essas etapas inclusas, foi implementado exatamente da mesma maneira que na primeira versão da atividade.

A primeira diferença está na seção `.data`, pois nessa versão as variáveis `a, b` e `c` não são declaradas, já que elas são fornecidas pelo usuário e não precisam ser armazenadas na memória. As variáveis `x0` e `x1` são inicializadas nulas, assim como no exercício anterior e, além disso, há a criação de cinco strings, três dedicadas a solicitar que o usuário insira um valor para variável (`prompt1`, `prompt2`, `prompt3`) e as outras duas utilizadas no final do programa, para indicar qual das raízes está sendo impressa (`printx0`, `printx1`).

Logo no início do código, os valores são solicitados. Isso acontece da mesma maneira para todos, com o comando para imprimir string sendo carregado em $v0 e o respectivo prompt sendo impresso. Após isso, o comando que lê inteiros é carregado em $v0 para que o usuário insira um inteiro. Finalmente, a pseudo-instrução `move` é utilizada para mover o valor fornecido de $v0 para um registrador *saved temporary*. Novamente, essa é a única sessão do código onde os registradores desse tipo são utilizados.

Após o armazenamento dos resultados na memória, ocorre a outra diferença dessa implementação. Novamente, o registrador $v0 é carregado com o comando de imprimir strings para que `printx0` possa ser impressa na tela, precedendo o valor da primeira raiz. Após isso, o comando carregado é o de imprimir inteiros, para que o resultado, que está armazenado em `x0`, seja impresso na tela, ao lado de `printx0`. A impressão de `x1` ocorre da mesma maneira.

(c) Quantidade de linhas

Para ambas as versões da implementação da fórmula de Bhaskara, verificamos a quantidade de linhas em basic e source, recebendo os seguintes valores:

Exercício	**Linhas Basic**	**Linhas Source**
01	31	23
02	63	50

Table 1: Quantidade de Linhas por Programa

Pode-se perceber que, para ambos os programas, a quantidade de linhas em basic foi maior que a quantidade de linhas em source. Para entender porque isso ocorre, devemos entender o que é cada tipo de linha. As linhas em source são aquelas escritas pelo programador. Já as linhas em basic são as instruções reais de máquina geradas pelo MARS após a montagem.

Sabendo disso, conclui-se que o motivo de o número de linhas em basic ser maior é que, muitas vezes, pseudo-instruções são utilizadas e, ao realizar a montagem, o montador precisa substituir ela por outras instruções. Alguns exemplos de ocorrências disso nos códigos da atividade são:

Pseudoinstrução (Source)	Expansão (Basic)
`li $v0, 4`	`addu $v0, $zero, 4`
`la $a0, prompt1`	`lui $at, high(prompt1)` `ori $a0, $at, low(prompt1)`
`move $s0, $v0`	`addu $s0, $v0, $zero`
`mul $t0, $s1, $s1`	`mult $s1, $s1` `mflo $t0`
`subi $t4, $t4, 1`	`addi $t4, $t4, -1`

Table 2: Pseudoinstruções utilizadas e como o MARS as reescreveu

2. **Atividades via Digital Lab:**

   (a) **Atividade 1**

   Para a implementação da primeira atividade do tipo via Digital Lab, que escreve os números de 0 a 9 no display sequencialmente, o primeiro passo foi criar um array de bytes na seção `.data`, contendo o padrão que desenha cada dígito no display de 7 segmentos.

   Em seguida, esse array é carregado em um registrador. Foi definido que utilizaríamos o display direito para a atividade, ao invés do esquerdo, e criamos, no registrador temporário `$t2`, um contador que servirá como índice para o laço.

   Dentro desse laço, primeiramente utilizamos, consecutivamente, um `load byte`, que pega padrão do dígito no array, e um `store byte`, que envia esse padrão para o display. Após essa etapa, é necessário utilizar o comando `sleep`, garantindo que cada número permaneça visível no display por tempo suficiente. Para esta atividade, definimos um tempo de 1 segundo.

   Depois que o número é exibido, `$t0` é incrementado para acessar o próximo

padrão do dígito no array `numeros`. O contador do índice do laço também é incrementado, registrando quantos números já foram exibidos (quantas vezes o laço foi executado). Por fim, a instrução `branch if less or equal` compara se o contador alcançou 9, pois assim que ele passar de 9, o laço será interrompido.

(b) **Atividade 2**

A segunda atividade via Digital Lab consistia em mostrar no display o valor da tecla clicada. Primeiro, foram criadas equivalências na seção `.eqv` para os endereços do display, da linha analisada e do valor da tecla. Em seguida, na seção `.data`, foram feitas duas tabelas: uma com o padrão hexadecimal dos dígitos para o display e outra com os códigos que o hardware retorna ao pressionar cada tecla.

Antes dos laços, escolheu-se um registrador (`$s7`) para guardar a última tecla pressionada, inicializado com valor inválido, para o programa saber que nenhuma tecla foi pressionada, e define-se que todos os segmentos do display começam apagados.

O loop principal define-se que a execução começa na linha um. Depois, o programa percorre as linhas, aplicando um delay, feito com um laço simples que apenas "gasta tempo" do programa, para as teclas conseguirem estabilizar.

Ao ler o valor da tecla, se nenhuma foi pressionada, o código avança. Se alguma for, inicia-se um laço que percorre a tabela de códigos, usando um ponteiro e um contador de índice, comparando os valores até achar o da tecla que foi selecionada, para saber o que deve ser exibido no display

Quando o valor é encontrado, o display é atualizado. Há ainda um laço que verifica se a tecla ainda está sendo pressionada.

Por fim, o programa volta a verificar as linhas, deslocando para a próxima por meio de um `sll` e ele repete esse processo até o usuário encerrar o programa.

3 Materiais e métodos

Para o desenvolvimento dessa atividade foi utilizada a linguagem `Assembly` e a IDE `MARS`, na qual foi realizada toda a escrita, execução e depuração de todos os programas implementados.

Para o segundo tipo de atividade, especificamente, foi utilizada a ferramenta `Digital Lab Sim`, integrada ao `MARS`. Essa ferramenta pode ser conectada ao `MIPS`, permitindo que ela responda durante a execução do seu programa.

Um de seus displays de sete segmentos e seu teclado alfanumérico foram utilizados nas atividades (o teclado alfanumérico *somente* na segunda).

4 Resultados

A execução do código transcorreu sem problemas, apresentando os resultados esperados. Durante o desenvolvimento, cada instrução foi cuidadosamente analisada, de modo a garantir que as operações, bem como a manipulação dos registradores e da memória, fossem realizadas conforme o planejado.

Para o cálculo das raízes da equação, consideramos os coeficientes $a = 1$ (armazenado no registrador `$s0`), $b = -5$ (armazenado no registrador `$s1`) e $c = 6$ (armazenado no registrador `$s2`).

O delta (Δ) foi calculado, resultando em $\Delta = 1$, sendo armazenado no registrador temporário `$t3`.

Na etapa de cálculo das raízes, obtemos os seguintes valores intermediários:

- Para x_0: o numerador $-b + \sqrt{\Delta} = 6$, armazenado em `$t6`.
- Para x_1: o numerador $-b - \sqrt{\Delta} = 4$, armazenado em `$t7`.
- O denominador $2a = 2$, armazenado em `$t5`.

Assim, as raízes são:

$$x_0 = \frac{6}{2} = 3 \quad \text{(armazenado em \texttt{\$t0})}$$

$$x_1 = \frac{4}{2} = 2 \quad \text{(armazenado em \texttt{\$t1})}$$

Por fim, os valores das raízes foram salvos na memória por meio da instrução `sw` (*store word*).

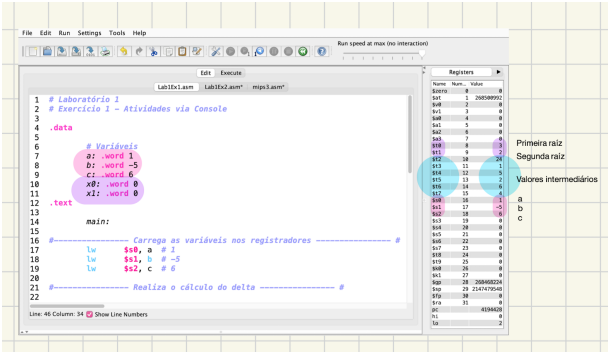

Figure 3: Execução do exercício 1.

No segundo exercício, o usuário fornece os coeficientes da equação por meio do *prompt*. No exemplo da imagem, foram digitados os valores $a = 1$ (registrador `$s0`), $b = -21$ (registrador `$s1`) e $c = 110$ (registrador `$s2`).

A lógica utilizada para o cálculo das raízes foi a mesma do exercício anterior. Como resultado, obteve-se $x_0 = 11$, armazenado no registrador temporário `$t8`, e $x_1 = 10$, armazenado no registrador temporário `$t9`.

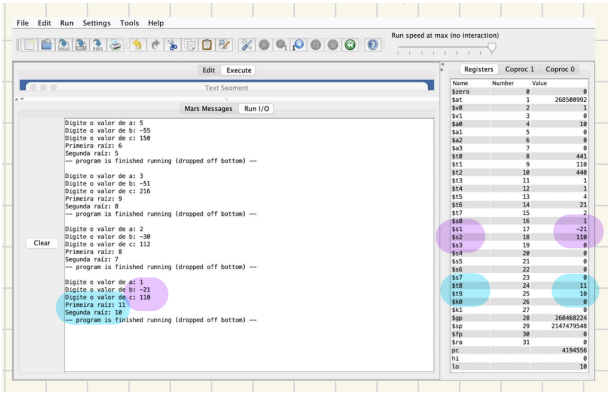

Figure 4: Execução do exercício 2.

Na primeira atividade proposta no *Digital Lab*, foi implementado um programa que escreve, sequencialmente, os números de 0 a 9 em um dos displays de sete segmentos da ferramenta *Digital Lab* .

A execução foi realizada com sucesso, e os dígitos de 0 a 9 foram exibidos corretamente no display.

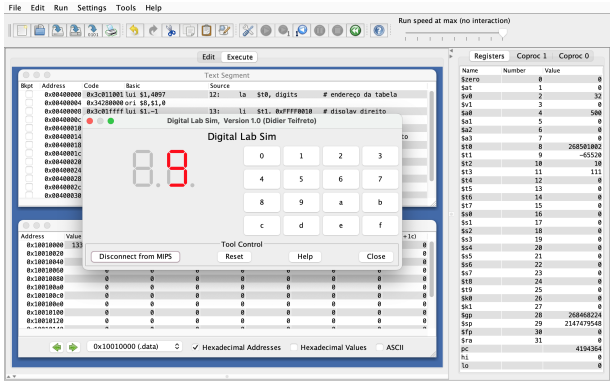

Figure 5: Execução no Digital Lab – Atividade 1 (Figura 1).

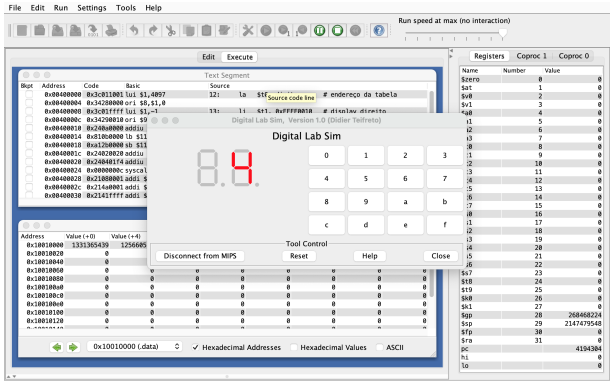

Figure 6: Execução no Digital Lab – Atividade 1 (Figura 2).

Na segunda atividade, utilizando a ferramenta *Digital Lab*, foi desenvolvido um programa que realiza a leitura do teclado alfanumérico e exibe no display de sete segmentos o valor correspondente à tecla pressionada (de 0 até f).

A execução também foi concluída com sucesso, e os dígitos pressionados no teclado foram corretamente mostrados no display.

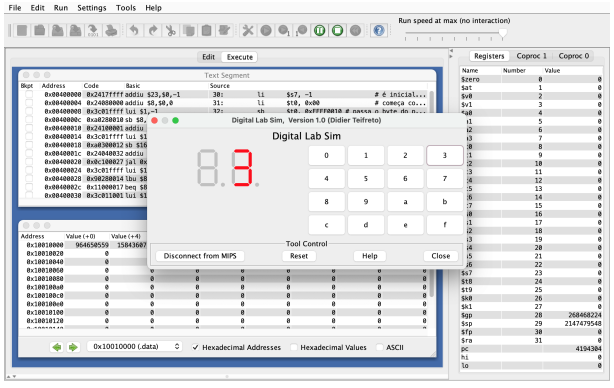

Figure 7: Execução no Digital Lab – Atividade 2 (Figura 1).

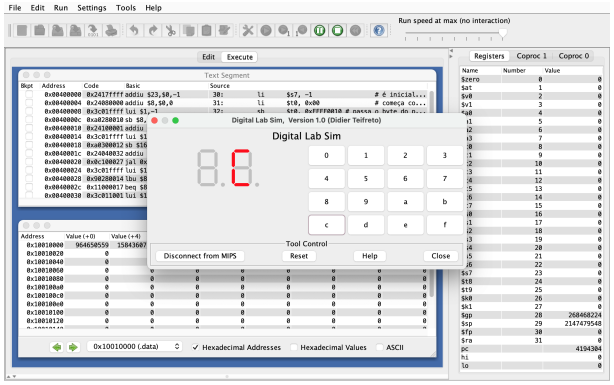

Figure 8: Execução no Digital Lab – Atividade 2 (Figura 2).

5 Dificuldades

Para as atividades via console, nossa principal dificuldade foi conseguir fazer a raiz quadrada lidando somente com inteiros, já que só encontramos o comando que o realiza com números e registradores de ponto flutuante. Para resolver esse problema, implementamos o laço que foi descrito detalhadamente na seção 2.1(a) do relatório.

Já para as atividades via **Digital Lab** tivemos mais dificuldades, por não termos nenhuma experiência com a ferramenta.

Nosso problema foi especificamente com a **segunda atividade**, onde o teclado alfanumérico deveria ser lido e o valor da tecla pressionada deveria ser exibido em um dos `displays`.

Inicialmente estávamos tendo o problema de o `MARS` travar sempre que uma nova tecla fosse selecionada, por um erro que estava acontecendo na lógica do *loop*.

Após a resolução desse problema, percebemos que a **quarta coluna de teclas** (3, 7, B e F) não estava funcionando. Esse problema foi extremamente difícil de solucionar, pois só depois de muita pesquisa descobrimos qual era o erro. Não havia nenhum tempo de espera entre a seleção da linha e a leitura do dado. Por conta disso, os valores da 4ª coluna nunca apareciam, pois a leitura acontecia antes do registrador ser atualizado. Para resolver isso, adicionamos um pequeno *delay*, o qual garante que haja tempo para a estabilização do sinal antes da leitura. Esse delay é feito por um laço simples que incrementa um contador. Tentamos implementar o comando *sleep*, mas mesmo testando ele com diversos parâmetros diferentes, não conseguimos fazer com que o código funcionasse dessa maneira.

O único detalhe que não conseguimos resolver é que é necessário clicar duas vezes na tecla que você deseja que seja exibida no `display`. Caso você clique em duas teclas diferentes, uma vez em cada uma, o valor da última tecla clicada será exibido.

6 Conclusão

Esse laboratório e suas quatro atividades foram essenciais para que pudessemos ampliar nosso entendimento sobre a linguagem Assembly e o funcionamento do MARS e da arquitetura MIPS.

Além disso, ele foi especialmente útil para que pudessemos entender mais sobre quando usar qual tipo de registrador, sobre tipos diferentes de instrução, pseudo-instruções e como elas são realizadas pelo montador, estruturas de laço e vários outros conceitos que com certeza serão extremamente necessários para os próximos laboratórios e atividades desenvolvidas nessa matéria.

Outro ponto interessante é que, como já mexemos com displays de sete segmentos em disciplinas anteriores, como Circuitos e Técnicas Digitais e Sistemas Digitais, utilizar ferramentas que envolvessem isso nesse laboratório foi muito útil para que pudessemos entender e explorar melhor as similaridades e diferenças entre VHDL e Assembly, organizando alguns conceitos cuja compreensão definitivamente facilitará o nosso estudo nessa disciplina.