

Laboratório 1

- Assembly MIPS –

Dayanne Fernandes da Cunha, 13/0107191

Lucas Mafra Chagas, 12/0126443

Marcelo Giordano Martins Costa de Oliveira, 12/0037301

Lucas Junior Ribas, 16/0052289

Caio Nunes de Alencar Osório, 16/0115132

Diego Vaz Fernandes, 16/0117925

¹Dep. Ciência da Computação – Universidade de Brasília (UnB)
CiC 116394 - OAC - Turma A

***Abstract.** This report corresponds to the Experiment 1 about Assembly MIPS.*

***Resumo.** Este relatório corresponde ao Experimento 1 sobre Assembly MIPS.*

1. Introdução

1.1. Objetivos

- Familiarizar o aluno com o Simulador/Montador MARS;
- Desenvolver a capacidade de codificação de algoritmos em linguagem Assembly MIPS;
- Desenvolver a capacidade de análise de desempenho de algoritmos em Assembly;

1.2. Ferramentas

- MARS v.4.5 Custom 7
- Cross compiler MIPS GCC
- Inkscape e GIMP

2. Procedimentos

2.1. Simulador/Montador MARS

Essa parte do relatório foi realizada com o intuito de familiarizar os alunos ao Simulador/Montador MARS.

No item 1.2 do relatório, foram pedidos os gráficos relacionados aos valores do vetor fornecido e ao tempo de execução da subrotina Sort fornecida pelo professor. Temos então o gráfico em Melhor caso 1 e Pior Caso 2 da rotina de ordenação Sort.

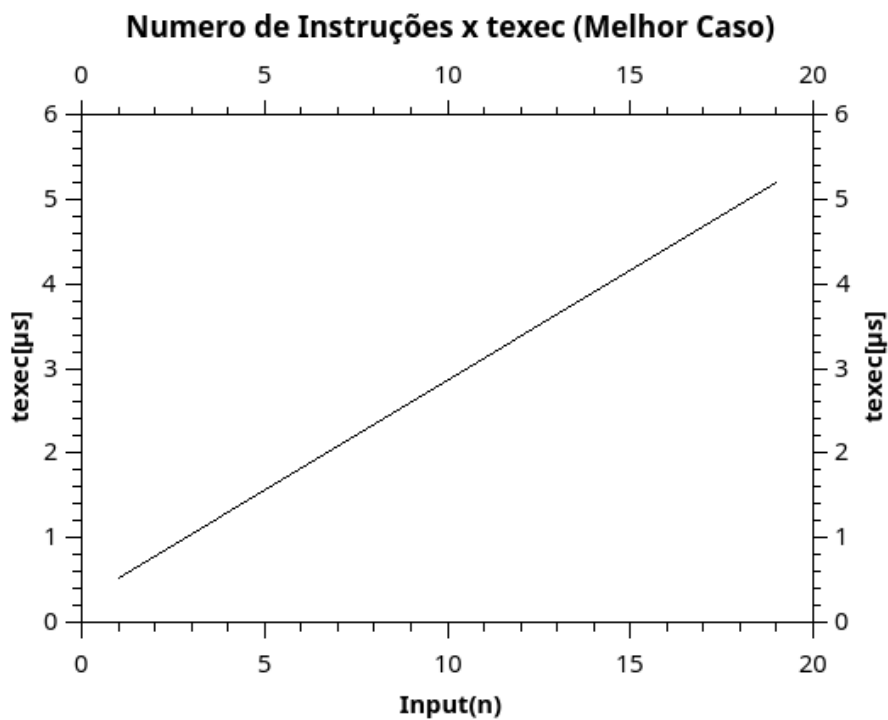


Figura 1. $n \times \text{texec}$ (Melhor Caso)

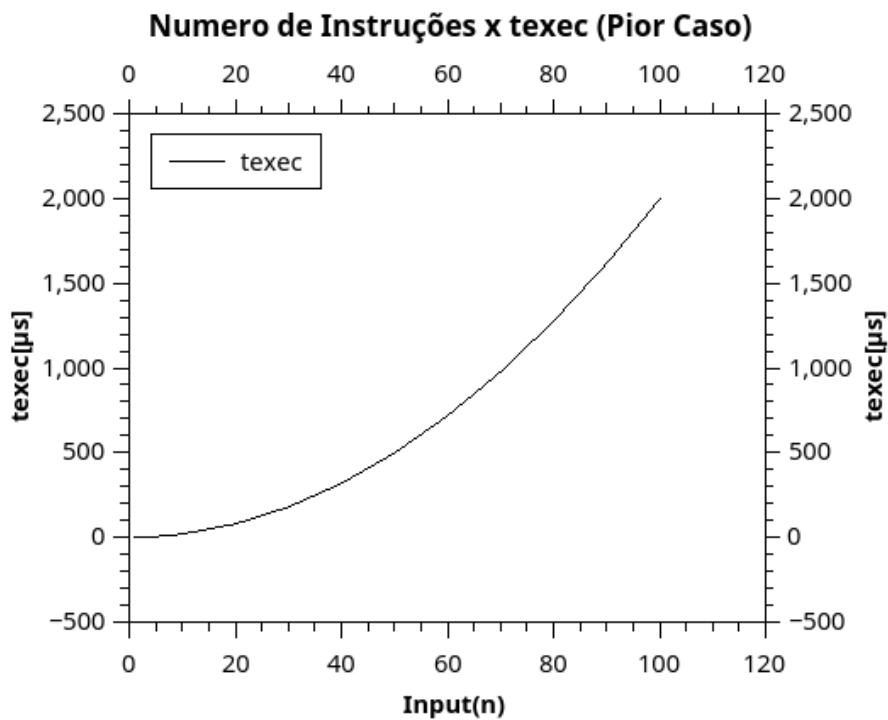


Figura 2. $n \times \text{texec}$ (Pior Caso)

Analisando somente os gráficos, percebemos que esse algoritmo tem tempo de

execução Linear em seu Melhor caso e em Pior caso, sendo representado por um braço de parábola, tem tempo de execução limitado por $O(n^2)$.

2.2. Compilador GCC

Para compilar código C em código Assembly foi utilizado o cross compiler [MIPS] GCC. Usando o comando `$ mips-elf-gcc -I./include -S testeX.c` ($X \in [0,8]$) foi testado a convenção para geração do código Assembly.

2.2.1. Diretivas

Diretivas são apenas comandos ao montador e não fazem parte do conjunto de instruções dos processadores *x86*. Todas as diretivas começam com `(.)` (*ASCII 0x2E*). Elas permitem a alocação de espaço para a declaração de variáveis `”.byte, .word ”`, definição de escopo `”.globl ”`, além de várias outras funções de gerenciamento como as listadas a seguir (as informações abaixo foram encontradas usando as fontes [ORACLE], [SourceWare c], [SourceWare a], [SourceWare b], [GNU], [MTU], [UAF] e [UNIBO]):

- `.file string` : Cria uma tabela de símbolos de entrada onde a *string* é o nome do símbolo e *STT_FILE* é o tipo deste símbolo, a *string* especifica o nome do arquivo fonte associado ao arquivo objeto.
- `.section section, attributes` : *Section* é montado como seção atual. *Attributes* é incluso se for a primeira vez que `.section` é especificado.
- `.mdebug` : Força a saída de depuração para entrar em uma seção `.mdebug` de estilo ECOFF em vez das seções padrão ELF `.stabs`.
- `.previous` : Troca esta seção pela que foi referenciada recentemente.
- `.nan` : Esta diretiva diz qual codificação *MIPS* será usada para ponto flutuante IEEE 754. A primeira, padrão 2008, diz para o montador utilizar a codificação IEEE 754-2008, enquanto a *legacy* utiliza a codificação original do *MIPS*.
- `.gnu_attribute tag, value` : Grava um atributo objeto *gnu* para este arquivo.
- `.globl symbol1, symbol2, ..., symbolN`: Torna global cada símbolo da lista. A diretiva torna o símbolo global no escopo mas não declara o símbolo.
- `.data` : Muda a seção atual para `.data` (dados estáticos do programa).
- `.type symbol[, symbol, ..., symbol], type[, visibility]`: Atribui tipo ao símbolo, podendo ser do tipo função, objeto, sem tipo e um objeto *TLS* (*Thread Local Storage*).
- `.size symbol, expr`: Resolve expressão e atribui tamanho em bytes ao *symbol*.
- `.word` : Armazena o valor listado como palavras de 32 bits no limite.
- `.rdata` : Adiciona dados apenas de leitura.
- `.align integer` : Ajusta o contador de locação para um valor múltiplo de 2.
- `.ascii ”string”`: Aloca espaço para cadeias de caracteres sem o `”\0”`.
- `.text` : Muda a seção atual para `.text` (instruções).
- `.ent name[,label]` : Marca o começo da função *name*.
- `.frame` : Descreve o quadro da pilha usada para chamar a função principal(*main*).
- `.set symbol, expression`: Resolve a expressão (*expression*) e atribui o valor ao símbolo (*symbol*).

- `.mask mask offset` : Configura uma máscara que indica quais registradores de uso geral foram salvos na rotina atual. Esses valores são usados pelo montador para gerar a seção `.reginfo` do arquivo objeto dos processadores *MIPS*.
- `.fmask mask offset` : Configura uma máscara informando os registradores de ponto flutuante que a rotina atual salvou. Esses valores são usados pelo montador para gerar a seção `.reginfo` do arquivo objeto dos processadores *MIPS*.

2.2.2. Assembly no MARS

Algumas diretivas listadas acima não são reconhecidas pelo *MARS (Mips Assembly and Runtime Simulator)*, como por exemplo `.section`, `.previous`, `.nan`, *etc*, assim como alguns elementos como `@object`, `@function`, *etc*. Logo, as seguintes instruções foram retiradas:

- `@object`
 - `.type v, @object`
- `@function`
 - `.type show, @function`
 - `.type swap, @function`
 - `.type sort, @function`
 - `.type main, @function`
- `.section`
 - `.section .mdebug.abi32`
- `.previous`
 - `.previous`
- `.nan`
 - `.nan legacy`
- `.gnu_attribute`
 - `.gnu_attribute 4, 1`
- `.size`
 - `.size v, 40`
 - `.size show, .-show`
 - `.size swap, .-swap`
 - `.size sort, .-sort`
 - `.size main, .-main`
- `.rdata`
 - `.rdata`
- `.set`

- .set nomips16
 - .set nomicromips
 - .set noreorder
 - .set nomacro
 - .set reorder
 - .set macro
- .ent
 - .ent show
 - .ent swap
 - .ent sort
 - .ent main
- .frame
 - .frame \$fp,24,\$31 # vars= 0, regs= 2/0, args= 16, gp= 0
 - .frame \$fp,32,\$31 # vars= 8, regs= 2/0, args= 16, gp= 0
 - .frame \$fp,16,\$31 # vars= 8, regs= 1/0, args= 0, gp= 0
 - .frame \$fp,32,\$31 # vars= 8, regs= 2/0, args= 16, gp= 0
- .mask
 - .mask 0xc0000000,-4
- .fmask
 - .fmask 0x00000000,0
- .end
 - .end swap
 - .end sort
 - .end main
 - .end show
- .ident
 - .ident "GCC: (GNU) 4.8.1"

Há também outras instruções que o simulador emitiu alertas, como por exemplo sobre o *.align* não poder estar dentro de um segmento de texto. Portanto todos *.align* dentro de subrotinas foram retirados.

<pre>la \$2, <i>addr</i></pre>	→	<pre>lui at, %hi(<i>addr</i>) addiu \$2, at, %lo(<i>addr</i>)</pre>
<pre>la \$2, <i>addr</i>(\$3)</pre>	→	<pre>lui at, %hi(<i>addr</i>) addiu \$2, at, %lo(<i>addr</i>) addu \$2, \$2, \$3</pre>

Figura 3. Instruções equivalentes dos construtores %hi() e %lo(). Imagem retirada do livro [Sweetman 2005].

Os construtores `%hi()` e `%lo()` também não estão presentes em todos montadores MIPS, podendo ser substituídos como mostra a Figura 15. Logo, as seguintes instruções foram trocadas :

- `lui $v1,%hi($LC0)` e `addiu $a0,$v1,%lo($LC0)` : `la $a0, $LC0`
- `lui $v0,%hi(v)` e `addiu $a0,$v0,%lo(v)` : `la $a0, v`

Outro problema encontrado foi na instrução `j $31`, ou seja, `j $ra`. A instrução `j` pula para um endereço alvo, e `$ra` é um registrador, portanto esta instrução foi substituída por `jr $ra`.

Algumas subrotinas como `printf` e `putchar` não são encontradas no assembly gerado mas são chamadas com a instrução `jal`, portanto estas também foram retiradas.

Por fim, mesmo após estas modificações básicas de sintaxe para que o código possa ser executado no MARS, ainda assim o código possui o fluxo um pouco confuso, já que após o primeiro `loop` ele já iria sair do processo com a instrução `li $a0, 10` que chama o serviço `exit` (*terminate execution*). Portanto, algumas modificações foram realizadas para que o fluxo deste programa se inicie no `main`, chame as subrotinas `show` e `sort` e encerre no final da subrotina `main`.

2.2.3. Otimização do código Assembly

O arquivo `sortc.c` foi compilado novamente usando 5 diretivas de compilação, `-O0`, `-O1`, `-O2`, `-O3` e `-Os`. Todos os arquivos gerados, `sortc0.s`, `sortc1.s`, `sortc2.s`, `sortc3.s` e `sortcs.s` foram alterados de acordo com a subseção anterior para que fosse possível executá-los no MARS. Também foram removidos os códigos para mostrar os números da ordenação na tela de `debug` do MARS.

Com ajuda do *Mars* foram coletadas informações sobre o número de instruções executadas e a quantidade de memória utilizada nas otimizações `sortc0.s` e `sortc1.s` como é possível ver na Tabela 1. A contagem das instruções e informações sobre a memória foram conferidas no menu *Tools > Instruction Statistics*.

Arquivo	Instruções	Tamanho (Bytes)
<code>sortc.s</code>	1748	890
<code>sortc0.s</code>	1748	890
<code>sortc1.s</code>	172	57

Tabela 1. Códigos otimizados do arquivo `sortc.c` utilizando o *cross compiler mips-elf-gcc*.

Como podemos ver na Tabela 1 a otimização `-O0` do arquivo citado do item 1.1 da descrição do Laboratório 1, `sortc.s`, não altera em nada a quantidade de instruções e memória utilizada, porém a primeira otimização (`-O1`) já altera a quantidade de instruções executadas em uma proporção de aproximadamente 9.84 da original.

Não foi possível analisar as informações das otimizações `-O2`, `-O3` e `-Os` pois estes códigos emitiram erros de lógica e de instruções como é mostrado a seguir:

- Os arquivos *sortc2.s* e *sortc3.s* emitiram o seguinte erro de lógica : *line 105: Runtime exception at 0x0040010c: fetch address not aligned on word boundary 0xffffffffd.*
- O arquivo *sortcs.s* continha um pseudo código *bltzl 8,L10* que o conjunto de instruções do *Mars* não possui, portanto não foi possível executá-lo.

2.3. Sprites

Na resolução do exercício, utilizamos as sprites do Ryu para conseguir realizar a atividade de colisão proposta.

A colisão acontece quando duas hitbox de personagens distintos encostam entre si.

Vídeo mostrando a interação de colisão. <https://www.youtube.com/watch?v=kbTLQQuw6RA>

2.4. Cálculo das raízes da equação de segundo grau:

```
.data
    abc: .float 0, 0, 0
    four: .float 4
    two: .float 2
    negativoF: .float -1
    negativoW: .word -1
    cplex: .asciiz "i"
    mais: .asciiz "+"
    menos: .asciiz "-"
    quebralinha: .asciiz "\n"
    raiz1: .asciiz "R(1) = "
    raiz2: .asciiz "R(2) = "
    space: .asciiz " "
    aviso: .asciiz "Digite ' 1 ' para calcular outra funcao, se nao digite ' 0 ' para encerrar."
    msg: .asciiz "Digite o valor do coeficiente "
    coef: .asciiz "ABC"

.text

main:

continue:
    la, $t0, abc          # carrega o vetor abc vazio em t0
    li $t1, 0             # i = 0
    li $t2, 3             # flag = 3
    la $t4, coef          # char dos coeficientes

ler:
    li $v0, 4             #*****
    la $a0, msg           # mostra msg na tela
    syscall
```

Figura 4. Imagens dos codigos ja comentados relacionados ao exercicio 4.

```

la $a0, mesg          # mostra msg na tela
syscall               #####

lb $t5, ($t4)         # carrega primeira posicao da string coef ( char dos coeficiente )

li $v0, 11            # codigo print de char
move $a0, $t5         # a0 = t5
syscall              # print

addi $t4, $t4, 1      # coef++ , anda 1 byte no vetor de coeficientes

li $v0, 4             #####
la $a0, quebralinha  #####
syscall              #####

li $v0, 6             # ler float do teclado e armazena em f0
syscall              #####

swc1 $f0, ($t0)       # armazena o valor lido no vetor abc ( t0 )
addi $t0, $t0, 4      # anda uma posicao (4bytes) em abc
addi $t1, $t1, 1      # i++

blt $t1, $t2, ler     # if ( i < flag ) go ler

li $v0, 4
la $a0, quebralinha
syscall

```

Figura 5. Imagens dos codigos ja comentados relacionados ao exercicio 4.

```

syscall

jal baskara           # chama a funcao baskara
move $a0, $v0         # move o retorno em v0 para passagem de argumento em a0
jal show              # chama a funcao show passando o a0

li $v0, 4             #####
la $a0, aviso         # printa aviso
syscall              #####

li $v0, 4
la $a0, quebralinha
syscall

li $v0, 5             # ler opcao calcular novamente ou nao
syscall              #####
move $t3, $v0         #####
beq $t3, 1, continue  # if ( t3 == 1 ) go continue, else o calculo termina

#jal baskara

#move $a0, $v0

#jal show

li $v0, 10            # return 0
syscall              # return 0

```

Figura 6. Imagens dos codigos ja comentados relacionados ao exercicio 4.


```

delta:
    la $t0, abc
    lwcl $f1, 0($t0)      # a
    lwcl $f2, 4($t0)      # b
    lwcl $f3, 8($t0)      # c

    mul.s $f2, $f2, $f2    # b^2
    la $t1, four          # 4
    lwcl $f4, ($t1)
    mul.s $f1, $f4, $f1    # 4*a
    mul.s $f3, $f1, $f3    # (4*a) * c
    sub.s $f3, $f2, $f3    # b^2 - 4*a*c

    #la $t0, negativoF
    #lwcl $f1, ($t0)
    #mul.s $f3, $f3, $f1

    mov.s $f0, $f3        # return delta (pelo f0)
    jr $ra

baskara:
    addi $sp, $sp, -4      # libera espaco para um word na pilha
    sw $ra, 4($sp)         # salva o endereco de retorno $ra na pilha
    jal delta              # chama a funcao que calcula o delta
    lw $ra, 4($sp)         # devolve o endereco de retorno da funcao atual
    addi $sp, $sp, 4       # libera o espaco alocado pela pilha

    li $t1, 0              # 0 em inteiro
    mtcl $t1, $f9          # movo 0 para o coproc 1
    cvt.s.w $f9, $f9       # converto 0 de int para float
    add.s $f8, $f9, $f0    # faco uma copia do delta em f0 para f8, COPIA
    # f0 = delta(float)
    # f8 = delta(float)

    cvt.w.s $f1, $f8       # converte delta de float para word
    mfc1 $t3, $f1          # movo delta do coproc 1 para os registradores de int
    # t3 = delta(int)

    bgtz $t3, next        # if (t3 > 0) go next, else multiplica por (-1)

    la $t1, negativoF      # f4 = -1
    lwcl $f4, ($t1)
    mul.s $f0, $f0, $f4    # delta = f0 * (-1)

next:

```

Figura 7. Imagens dos codigos ja comentados relacionados ao exercicio 4.

```

    jr $ra

baskara:
    addi $sp, $sp, -4      # libera espaco para um word na pilha
    sw $ra, 4($sp)         # salva o endereco de retorno $ra na pilha
    jal delta              # chama a funcao que calcula o delta
    lw $ra, 4($sp)         # devolve o endereco de retorno da funcao atual
    addi $sp, $sp, 4       # libera o espaco alocado pela pilha

    li $t1, 0              # 0 em inteiro
    mtcl $t1, $f9          # movo 0 para o coproc 1
    cvt.s.w $f9, $f9       # converto 0 de int para float
    add.s $f8, $f9, $f0    # faco uma copia do delta em f0 para f8, COPIA
    # f0 = delta(float)
    # f8 = delta(float)

    cvt.w.s $f1, $f8       # converte delta de float para word
    mfc1 $t3, $f1          # movo delta do coproc 1 para os registradores de int
    # t3 = delta(int)

    bgtz $t3, next        # if (t3 > 0) go next, else multiplica por (-1)

    la $t1, negativoF      # f4 = -1
    lwcl $f4, ($t1)
    mul.s $f0, $f0, $f4    # delta = f0 * (-1)

next:

```

Figura 8. Imagens dos codigos ja comentados relacionados ao exercicio 4.

```

next:

la $t5, negativoW      # carregando o (-1) int
lw $t6, ($t5)          # t6 = -1

la $t0, abc
lwcl $f1, 0($t0)        # a
lwcl $f2, 4($t0)        # b
lwcl $f3, 8($t0)        # c

la $t1, negativoF
lwcl $f4, ($t1)         # f4 = -1
mul.s $f2, $f4, $f2     # (-1)*b
sqrt.s $f5, $f0         # f5 = raiz(f0) = raiz(delta)
la $t2, two
lwcl $f6, ($t2)         # f6 = 2
mul.s $f1, $f6, $f1     # 2 * a

slt $t4, $t3, $zero     # if ( t3 < 0 ) t4 = 1 , else t4 = 0

beqz $t4, reais

complexas:
    # f18 = x1
    # f20 = x2

    div.s $f18, $f2, $f1 # -b / 2 * a
    div.s $f20, $f5, $f1 # raiz(delta) / 2 * a
    addi $sp, $sp, -8    # aloca 2 espacos na pilha
    swcl $f18, 4($sp)    # armazena a raiz na pilha
    swcl $f20, 0($sp)    # armazena a raiz na pilha

    j endReais          # salta para o fim da funcao para retorno complexo

reais:
    # f18 = x1
    # f20 = x2
    add.s $f7, $f2, $f5 # f7 = -b + raiz(delta)
    div.s $f18, $f7, $f1 # f7 / 2 * a

    sub.s $f7, $f2, $f5 # f7 = -b - raiz(delta)
    div.s $f20, $f7, $f1 # f7 / 2 * a

    addi $sp, $sp, -8    # libera 2 espacos na pilha
    swcl $f18, 4($sp)    # armazena a raiz na pilha
    swcl $f20, 0($sp)    # armazena a raiz na pilha

    li $t0, 1           # t0 = 1
    move $v0, $t0       # return t0 = 1
    jr $ra              # return

```

Figura 9. Imagens dos codigos ja comentados relacionados ao exercicio 4.

```

complexas:
    # f18 = x1
    # f20 = x2

    div.s $f18, $f2, $f1 # -b / 2 * a
    div.s $f20, $f5, $f1 # raiz(delta) / 2 * a
    addi $sp, $sp, -8    # aloca 2 espacos na pilha
    swcl $f18, 4($sp)    # armazena a raiz na pilha
    swcl $f20, 0($sp)    # armazena a raiz na pilha

    j endReais          # salta para o fim da funcao para retorno complexo

reais:
    # f18 = x1
    # f20 = x2
    add.s $f7, $f2, $f5 # f7 = -b + raiz(delta)
    div.s $f18, $f7, $f1 # f7 / 2 * a

    sub.s $f7, $f2, $f5 # f7 = -b - raiz(delta)
    div.s $f20, $f7, $f1 # f7 / 2 * a

    addi $sp, $sp, -8    # libera 2 espacos na pilha
    swcl $f18, 4($sp)    # armazena a raiz na pilha
    swcl $f20, 0($sp)    # armazena a raiz na pilha

    li $t0, 1           # t0 = 1
    move $v0, $t0       # return t0 = 1
    jr $ra              # return

```

Figura 10. Imagens dos codigos ja comentados relacionados ao exercicio 4.

```

endReais:
    li $t0, 2                # t0 = 2
    move $v0, $t0            # return t0 = 2
    jr $ra                   # return

show:
    move $t0, $a0            # t0 = a0 argumento passado
    beq $t0, 2, showC        # if ( t0 == 2 ) go showC, else showR

showR:
    lwcl $f18, 4($sp)        # retira a raiz x1 da pilha
    lwcl $f20, 0($sp)        # retira a raiz x2 da pilha
    addi $sp, $sp, 8         # libera o espaco alocado pela pilha

    li $v0, 4                #####
    la $a0, raiz1            #####
    syscall                  #####

    li $v0, 2                #####
    mov.s $f12, $f18         #####
    syscall                  #####
    # area de print
    li $v0, 4                #####
    la $a0, quebralinha     #####
    syscall                  #####
    #####
    li $v0, 4                #####
    #####
    #####

```

Figura 11. Imagens dos codigos ja comentados relacionados ao exercicio 4.

```

    la $a0, quebralinha     #####
    syscall                  #####
    #####
    li $v0, 4                #####
    la $a0, raiz2           #####
    syscall                  #####
    #####
    li $v0, 2                #####
    mov.s $f12, $f20        #####
    syscall                  #####
    #####

    li $v0, 4                #####
    la $a0, quebralinha     #####
    syscall                  #####
    li $v0, 4                #####
    la $a0, quebralinha     #####
    syscall                  #####

    j end

showC:
    lwcl $f18, 4($sp)        # retira a raiz complexa positiva da pilha
    lwcl $f20, 0($sp)        # retira a raiz complexa negativa da pilha
    addi $sp, $sp, 8         # libera espaco alocado pela pilha

    li $v0, 4                #####
    la $a0, raiz1            #####
    syscall                  #####
    #####
    # area de print R1
    #####

```

Figura 12. Imagens dos codigos ja comentados relacionados ao exercicio 4.

```

li $v0, 4          #####
la $a0, raiz1      #####
syscall           #####
                  # area de print R1
li $v0, 2          #####
mov.s $f12, $f18   #####
syscall           #####
li $v0, 4          #####
la $a0, space      #####
syscall           #####
li $v0, 4          #####
la $a0, mais       #####
syscall           #####
li $v0, 4          #####
la $a0, space      #####
syscall           #####
li $v0, 2          #####
mov.s $f12, $f20   #####
syscall           #####
li $v0, 4          #####
la $a0, cplex      #####
syscall           #####

```

Figura 13. Imagens dos codigos ja comentados relacionados ao exercicio 4.

```

li $v0, 4          #####
la $a0, quebralinha
syscall           #####
li $v0, 4          #####
la $a0, raiz2      #####
syscall           #####
                  # area de print R2
li $v0, 2          #####
mov.s $f12, $f18   #####
syscall           #####
li $v0, 4          #####
la $a0, space      #####
syscall           #####
li $v0, 4          #####
la $a0, menos      #####
syscall           #####
li $v0, 4          #####
la $a0, space      #####
syscall           #####
li $v0, 2          #####
mov.s $f12, $f20   #####
syscall           #####

```

Figura 14. Imagens dos codigos ja comentados relacionados ao exercicio 4.

```

34      li $v0, 4          #####
35      la $a0, space      #####
36      syscall            #####
37                          #####
38      li $v0, 2          #####
39      mov.s $f12, $f20    #####
40      syscall            #####
41                          #####
42      li $v0, 4          #####
43      la $a0, cplex       #####
44      syscall            #####
45                          #####
46      li $v0, 4          #####
47      la $a0, quebralinha #####
48      syscall            #####
49      li $v0, 4          #####
50      la $a0, quebralinha #####
51      syscall            #####
52
53
54      end:
55      jr $ra
56
57
58
59
60
61

```

Figura 15. Imagens dos codigos ja comentados relacionados ao exercicio 4.

2.4.1. Teste de raizes reais e complexas

Criamos o procedimento bhaskara especificado na figura 6, e chamamos delta cujo o codigo esta comentado na figura 5 para fazer o calcula e retornar o resultado e dependendo se ele era positivo (resultados reais) ou negativo (resultados imaginarios) e retornariamos 1 ou 2 respectivamente.

2.4.2. Show

Utilizamos a função show figura 8, que por sua vez dependendo se a mesma recebeu 2 ou 1 chama showC (função pra printar as raizes complexas), ou deixa prosseguir para showR (cujo o procedimento é printar as raizes reais) respectivamente. comentado e apresentados nas figuras 9,10,11,12.

2.4.3. Main

Inicializamos o codigo printando uma mensagem pedindo os coeficientes que utilizaremos na equação de segundo grau(usando os comandos de syscall e os respectivos registradores), após o recebimento das mesmas chamamos a instrução bhaskara que por sua vez chama delta e assim subsequentemente chamando show, no final imprimos os resultados e perguntamos ao usuario se ele continuara utilizando o programa e caso sim chamasse continue que retorna ao começo do arquivo. mostrados respectivamente com seus processos em todas as figuras desde a 2.

2.4.4. Respostas da questão 4.4

Nessa parte, precisamos calcular o texec para vários valores, como nosso codigo não tem nenhum loop as Instruções FR e FI são utilizadas em um número constante de

vezes, já os outros tipos podem sofrer pequenas diferenças na quantidade de instruções. Foi fornecido que o código rodaria em um processador MIPS de 1GHZ. Utilizamos $t_{exec} = C * T$ e chegamos aos resultados:

- a) [1, 0, -9.86960440]
 $t_{exec} = 211\mu s$
- b) [1, 0, 0]
 $t_{exec} = 222\mu s$
- c) [1, 99, 2459]
 $t_{exec} = 219\mu s$
- d) [1, -2468, 33762440]
 $t_{exec} = 219\mu s$
- e) [0, 10, 100]
 $t_{exec} = 211\mu s$

3. Análise dos Resultados

4. Conclusão

Referências

- [GNU] GNU, G. Emulated tls. <https://gcc.gnu.org/onlinedocs/gccint/Emulated-TLS.html>. [Online; acessado 18-Setembro-2017].
- [MIPS] MIPS. <https://aur.archlinux.org/packages/cross-mips-elf-gcc/>. [Online; acessado 15-Setembro-2017].
- [MTU] MTU, M. T. U. Mips assembler directives of gnu assembler. <http://pages.mtu.edu/~mmkoksai/blog/?x=entry:entry120116-130646>. [Online; acessado 18-Setembro-2017].
- [ORACLE] ORACLE. Assembler directives. https://docs.oracle.com/cd/E26502_01/html/E28388/eoiyg.html. [Online; acessado 18-Setembro-2017].
- [SourceWare a] SourceWare. Gnu attribute. https://sourceware.org/binutils/docs-2.26/as/Gnu_005fattribute.html#Gnu_005fattribute. [Online; acessado 18-Setembro-2017].
- [SourceWare b] SourceWare. Microblaze directives. <https://sourceware.org/binutils/docs/as/MicroBlaze-Directives.html>. [Online; acessado 18-Setembro-2017].
- [SourceWare c] SourceWare. Mips nan encodings. <https://sourceware.org/binutils/docs-2.27/as/MIPS-NaN-Encodings.html>. [Online; acessado 18-Setembro-2017].
- [Sweetman 2005] Sweetman, D. (2005). *See MIPS Run*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edition.
- [UAF] UAF, U. o. A. F. C compiler. <https://www.cs.uaf.edu/2000/fall/cs301/notes/notes/node97.html>. [Online; acessado 18-Setembro-2017].
- [UNIBO] UNIBO, U. o. B. Mips assembly language programmer's guide asm-01-doc. http://www.cs.unibo.it/~solmi/teaching/arch_2002-2003/AssemblyLanguageProgDoc.pdf. [Online; acessado 18-Setembro-2017].