Universidade de Brasília

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO - CIC

GRUPO 5

LABORATÓRIO 1 DE OAC

Assembly MIPS

Nome do Estudante	Matrícula
1. Iago Lobo Ribeiro de Moraes	14/0082921
2. Cristiano Krug Brust	15/0008058
3. José Marcos da Silva Leite	15/0038810
4. Yan Victor dos Santos	14/0033599
5. André Luiz de Moura Ramos Bittencourt	14/0130225

Professor(a):

Marcus Vinícius Lamar

Data : 29/09/2017

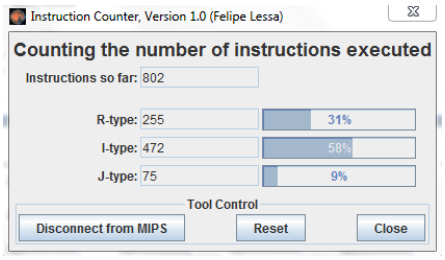
1 Simulador/Montador MARS

Instale em sua máquina o simulador/montador MARS v.4.5 Custom 7 disponível no Moodle.

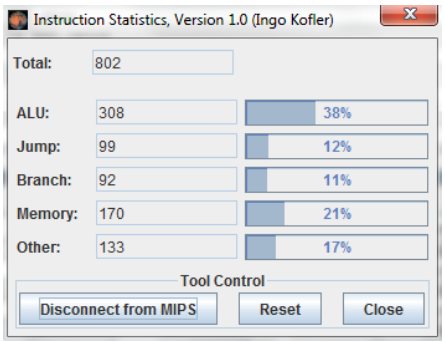
(0.0) 1.1) Dado o programa sort.s e o vetor: V[10]={5,8,3,4,7,6,8,0,1,9}, ordená-lo em ordem crescente e contar o número de instruções por tipo, por estatística e o número total exigido pelo algoritmo. Qual o tamanho em bytes do código executável?

(1.0) 1.2) Considere a execução deste algoritmo em um processador MIPS com frequência de clock de 50MHz que necessita 1 ciclo de clock para a execução de cada instrução (CPI=1). Para os vetores de entrada de n elementos já ordenados v_O[n]={1,2,3,4...n} e ordenados inversamente v_I[n]{n, n-1, n-2,...,2,1}, obtenha o número de instruções, calcule o tempo de execução para n={1,2,3,4,5,6,7,8,9,10,20, 30,40,50,60,70,80,90,100} e plote esses dados em um mesmo gráfico n x t exec. Comente os resultados obtidos.

Respostas:
1.1) Após escrever o código para ordenar em ordem crescente, foi realizada a contagem de instruções por tipo e estatítica, ferramentas disponibilizadas pela disciplina. O número total exigido pelo algoritmo foi 551. O tamanho em bytes do código executável é de 74 bytes.



(a) Instruction Counter



(b) Instruction Statistics

```
.eqv N 10

.data
vetor:   .word 5,8,3,4,7,6,8,0,1,9
newl:  .asciiz "\n"
tab:   .asciiz "\t"

.text
MAIN: la $a0,vetor
  li $a1,N
  jal show

  la $a0,vetor
  li $a1,N
  jal sort

  la $a0,vetor
  li $a1,N
```

```
19    jal show
20
21    li $v0,10
22    syscall
23
24  swap: sll $t1,$a1,2
25    add $t1,$a0,$t1
26    lw $t0,0($t1)
27    lw $t2,4($t1)
28    sw $t2,0($t1)
29    sw $t0,4($t1)
30    jr $ra
31  sort: addi $sp,$sp,-20
32    sw $ra,16($sp)
33    sw $s3,12($sp)
34    sw $s2,8($sp)
35    sw $s1,4($sp)
36    sw $s0,0($sp)
37    move $s2,$a0
38    move $s3,$a1
39    move $s0,$zero
40  for1: slt $t0,$s0,$s3
41    beq $t0,$zero,exit1
42    addi $s1,$s0,-1
43  for2: slti $t0,$s1,0
44    bne $t0,$zero,exit2
45    sll $t1,$s1,2
46    add $t2,$s2,$t1
47    lw $t3,0($t2)
48    lw $t4,4($t2)
49    slt $t0,$t4,$t3 # original(decrescente): slt $t0,$t3,$t4
50    beq $t0,$zero,exit2
51    move $a0,$s2
52    move $a1,$s1
53    jal swap
54    addi $s1,$s1,-1
55    j for2
56  exit2:  addi $s0,$s0,1
57    j for1
58  exit1:  lw $s0,0($sp)
59    lw $s1,4($sp)
60    lw $s2,8($sp)
61    lw $s3,12($sp)
62    lw $ra,16($sp)
63    addi $sp,$sp,20
64    jr $ra
65
66
67  show: move $t0,$a0
68    move $t1,$a1
69    move $t2,$zero
70
71  loop1:  beq $t2,$t1,fim1
72    li $v0,1
73    lw $a0,0($t0)
74    syscall
75    li $v0,4
76    la $a0,tab
77    syscall
78    addi $t0,$t0,4
```

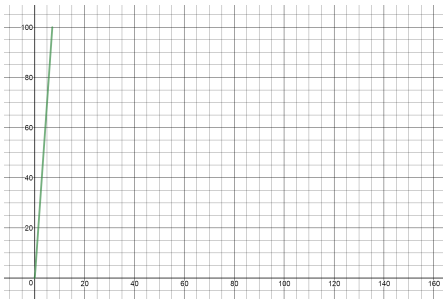
```
  addi $t2,$t2,1
  j loop1

fim1: li $v0,4
  la $a0,newl
  syscall
  jr $ra
```

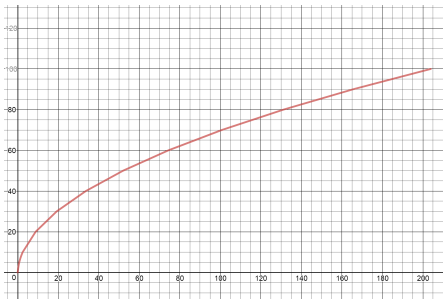
1.2) A fórmula para o cálculo do tempo de execução é:

$$t_{\text{exec}} = I \times CPI \times T$$

Com CPI = 1, nos importa saber o número de instruções e o período. Realizando algumas conversões, temos $T = 200 \times 10^{-3}$. O tempo de execução se encontra em milisegundos nos gráficos abaixo.



(a) Melhor caso (n x t_{exec})



(b) Pior caso (n x t_{exec})

Código utilizado para calcular esses valores:

```
.eqv N 100

.data
#vetor do pior caso poss vel
vetor1:   .word
    100,99,98,97,96,95,94,93,92,91,90,89,88,87,86,85,84,83,82,
81,80,79,78,77,76,75,74,73,72,71,70,69,68,67,66,65,64,63,62,61,60,
59,58,57,56,55,54,53,52,51,50,49,48,47,46,45,44,43,42,41,40,39,38,
37,36,35,34,33,32,31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16,
15,14,13,12,11,10,9,8,7,6,5,4,3,2,1
newl: .asciiz "\n"
tab:  .asciiz "\t"
msg1:   .asciiz "\nVetor original: "
msg2:   .asciiz "\nVetor ordenado: "

.text
MAIN:
  la $a0, msg1
  li $v0, 4
  syscall

  la $a0,vetor1
  li $a1,N
  jal show

  la $a0,vetor1
  li $a1,N
  jal sort

  la $a0, msg2
```

```
30   li $v0, 4
31   syscall
32
33   la $a0,vetor1
34   li $a1,N
35   jal show
36
37   li $v0,10
38   syscall
39
40 swap: sll $t1,$a1,2
41   add $t1,$a0,$t1
42   lw $t0,0($t1)
43   lw $t2,4($t1)
44   sw $t2,0($t1)
45   sw $t0,4($t1)
46   jr $ra
47
48 sort: addi $sp,$sp,-20
49   sw $ra,16($sp)
50   sw $s3,12($sp)
51   sw $s2,8($sp)
52   sw $s1,4($sp)
53   sw $s0,0($sp)
54   move $s2,$a0
55   move $s3,$a1
56   move $s0,$zero
57 for1: slt $t0,$s0,$s3
58   beq $t0,$zero,exit1
59   addi $s1,$s0,-1
60 for2: slti $t0,$s1,0
61   bne $t0,$zero,exit2
62   sll $t1,$s1,2
63   add $t2,$s2,$t1
64   lw $t3,0($t2)
65   lw $t4,4($t2)
66   slt $t0,$t4,$t3 # original(decrescente): slt $t0,$t3,$t4
67   beq $t0,$zero,exit2
68   move $a0,$s2
69   move $a1,$s1
70   jal swap
71   addi $s1,$s1,-1
72   j for2
73 exit2:  addi $s0,$s0,1
74   j for1
75 exit1:  lw $s0,0($sp)
76   lw $s1,4($sp)
77   lw $s2,8($sp)
78   lw $s3,12($sp)
79   lw $ra,16($sp)
80   addi $sp,$sp,20
81   jr $ra
82
83
84 show: move $t0,$a0
85   move $t1,$a1
86   move $t2,$zero
87
88 loop1:  beq $t2,$t1,fim1
89   li $v0,1
```

```
90    lw $a0,0($t0)
91    syscall
92    li $v0,4
93    la $a0,tab
94    syscall
95    addi $t0,$t0,4
96    addi $t2,$t2,1
97    j loop1
98
99  fim1: li $v0,4
100   la $a0,newl
101   syscall
102   jr $ra
```

2 Compilador GCC

Instale na sua máquina o cross compiler MIPS GCC disponível no Moodle. Forma de utilização: mips-sde-elf-gcc –S teste.c #diretiva –S para gerar o arquivo Assembly teste.s.

Inicialmente, teste com programas triviais em C para entender a convenção utilizada para a geração do código Assembly.

(0.5) 2.1) Dado o programa sortc.c, compile-o e comente o código em Assembly obtido indicando a função de cada uma das diretivas do montador usadas no código Assembly (.file .section .mdebug .previous .nan .gnu_attribute .globl .data .align .type .size .word .rdata .align .ascii .text .ent .frame .mask .fmask .set).

(0.5) 2.2) Indique as modificações necessárias no código Assembly gerado pelo gcc para poder ser executado no Mars.

(1.0) 2.3) Compile novamente o programa sortc.c e com a ajuda do Mars compare o número de instruções executadas e o tamanho em bytes dos códigos obtidos com os dados do item 1.1) para cada diretiva de otimização da compilação -O0, -O1, -O2, -O3, -Os.

Respostas: 2) Foi utilizado o cross Compiler MIPS GCC para gerar os arquivos.s através de arquivos.c. O comando utilizado foi: mips-sde-elf-gcc –S teste.c
2.1) As diretivas encontradas no código foram as seguintes:
.file: string associada ao arquivo fonte objeto.

.section: montado como a atual seção, setando atributos na primeira especificação.

.mdebug: Entra na seção de debug após ser forçado a sair da depuração.

.previous: Troca esta secão pela que foi referenciada recentemente.

.nan: indica a codificação MIPS a ser usada para ponto flutuante em padrão IEEE 754.

.gnu_attribute: este atributo é gravado para o arquivo.

.globl: Possui uma lista de símbolos que se tornarão globais.

.data: muda a seção atual para .data.

.align: contador de locação ajustado para múltiplo de dois.

.type: atribuição de tipo.

.word: armazena a entrada como palavra de 32 bit (limite).

.rdata: adiciona dados de leitura.

.ascii: alocação de espaço para cadeias de caracteres.

.text: A seção atual é mudada para .text

.ent: marca o começo para a função listada.

.frame: descreve o quadro da pilha usada para a função main.

.mask: a máscara indica registradores salvos na rotina atual.

.fmask: assim como mask, porém para ponto flutuante.

2.2) Para executar o arquivo gerado no MARS, foi necessário retirar muitas diretivas que não estavam sendo reconhecidas pelo MARS. Todas as que foram retiradas, foram comentadas no código .s gerado. O fluxo também teve de ser alterado para que a lógica do sort permanecesse.

2.3) Para as otimizações, os códigos gerados variaram pouco, entre O0, 01, 02, O3 e Os. Mas, se comparados ao código não otimizado, sua estrutura foi reduzida consideravelmente.

```
.data
v:   .word 5
  .word 8
  .word 3
  .word 4
  .word 7
  .word 6
  .word 8
  .word 0
  .word 1
  .word 9

  .align  2
.text
.LC0:
  .ascii  "%d\011\000"
  .text
  .align  2
  .globl  show
  .set  nomips16
  .set  nomicromips
  .ent  show
  .type show, @function
show:
  .frame  $fp,32,$ra    # vars= 8, regs= 2/0, args= 16, gp= 0

  addiu $sp,$sp,-32
  sw  $ra,28($sp)
  sw  $fp,24($sp)
  move  $fp,$sp
  sw  $4,32($fp)
  sw  $5,36($fp)
  sw  $0,16($fp)
  b .L2
  nop

.L3:
  lw  $2,16($fp)
  sll $2,$2,2
  lw  $3,32($fp)
  addu  $2,$3,$2
  lw  $2,0($2)
  move  $5,$2
  lui $2,%hi(.LC0)
  addiu $4,$2,%lo(.LC0)
  jal printf
  nop

  lw  $2,16($fp)
  addiu $2,$2,1
  sw  $2,16($fp)
.L2:
  lw  $3,16($fp)
  lw  $2,36($fp)
  slt $2,$3,$2
  bne $2,$0,.L3
  nop
```

```
58
59    li   $4,10      # 0xa
60    jal  putchar
61    nop
62
63    nop
64    move   $sp,$fp
65    lw   $ra,28($sp)
66    lw   $fp,24($sp)
67    addiu  $sp,$sp,32
68    jr   $ra
69    nop
70  swap:
71
72    addiu  $sp,$sp,-16
73    sw   $fp,12($sp)
74    move   $fp,$sp
75    sw   $4,16($fp)
76    sw   $5,20($fp)
77    lw   $2,20($fp)
78    sll  $2,$2,2
79    lw   $3,16($fp)
80    addu   $2,$3,$2
81    lw   $2,0($2)
82    sw   $2,0($fp)
83    lw   $2,20($fp)
84    sll  $2,$2,2
85    lw   $3,16($fp)
86    addu   $2,$3,$2
87    lw   $3,20($fp)
88    addiu  $3,$3,1
89    sll  $3,$3,2
90    lw   $4,16($fp)
91    addu   $3,$4,$3
92    lw   $3,0($3)
93    sw   $3,0($2)
94    lw   $2,20($fp)
95    addiu  $2,$2,1
96    sll  $2,$2,2
97    lw   $3,16($fp)
98    addu   $2,$3,$2
99    lw   $3,0($fp)
100   sw   $3,0($2)
101   nop
102   move   $sp,$fp
103   lw   $fp,12($sp)
104   addiu  $sp,$sp,16
105   jr   $ra
106   nop
107
108
109 sort:
110   .frame   $fp,32,$ra     # vars= 8, regs= 2/0, args= 16, gp= 0
111   .mask 0xc0000000,-4
112   .fmask   0x00000000,0
113   .set   noreorder
114   .set   nomacro
115   addiu  $sp,$sp,-32
116   sw   $ra,28($sp)
117   sw   $fp,24($sp)
```

```
118   move  $fp,$sp
119   sw  $4,32($fp)
120   sw  $5,36($fp)
121   sw  $0,16($fp)
122   b .L6
123   nop
124
125 .L10:
126   lw  $2,16($fp)
127   addiu $2,$2,-1
128   sw  $2,20($fp)
129   b .L7
130   nop
131
132 .L9:
133   lw  $5,20($fp)
134   lw  $4,32($fp)
135   jal swap
136   nop
137
138   lw  $2,20($fp)
139   addiu $2,$2,-1
140   sw  $2,20($fp)
141 .L7:
142   lw  $2,20($fp)
143   bltz  $2,.L8
144   nop
145
146   lw  $2,20($fp)
147   sll $2,$2,2
148   lw  $3,32($fp)
149   addu  $2,$3,$2
150   lw  $3,0($2)
151   lw  $2,20($fp)
152   addiu $2,$2,1
153   sll $2,$2,2
154   lw  $4,32($fp)
155   addu  $2,$4,$2
156   lw  $2,0($2)
157   slt $2,$2,$3
158   bne $2,$0,.L9
159   nop
160
161 .L8:
162   lw  $2,16($fp)
163   addiu $2,$2,1
164   sw  $2,16($fp)
165 .L6:
166   lw  $3,16($fp)
167   lw  $2,36($fp)
168   slt $2,$3,$2
169   bne $2,$0,.L10
170   nop
171
172   nop
173   move  $sp,$fp
174   lw  $ra,28($sp)
175   lw  $fp,24($sp)
176   addiu $sp,$sp,32
177   j $ra
```

```
178   nop
179
180   .set  macro
181   .set  reorder
182   .end  sort
183   .size sort, .-sort
184   .align  2
185   .globl  main
186   .set  nomips16
187   .set  nomicromips
188   .ent  main
189   .type main, @function
190 main:
191   .frame  $fp,24,$ra    # vars= 0, regs= 2/0, args= 16, gp= 0
192   .mask 0xc0000000,-4
193   .fmask  0x00000000,0
194   .set  noreorder
195   .set  nomacro
196   addiu $sp,$sp,-24
197   sw  $ra,20($sp)
198   sw  $fp,16($sp)
199   move  $fp,$sp
200   li  $5,10     # 0xa
201   lui $2,%hi(v)
202   addiu $4,$2,%lo(v)
203   jal show
204   nop
205
206   li  $5,10     # 0xa
207   lui $2,%hi(v)
208   addiu $4,$2,%lo(v)
209   jal sort
210   nop
211
212   li  $5,10     # 0xa
213   lui $2,%hi(v)
214   addiu $4,$2,%lo(v)
215   jal show
216   nop
217
218   nop
219   move  $sp,$fp
220   lw  $ra,20($sp)
221   lw  $fp,16($sp)
222   addiu $sp,$sp,24
223   jr  $ra
224   nop
225
226   .set  macro
227   .set  reorder
228   .end  main
229   .size main, .-main
230   .ident  "GCC: (Sourcery CodeBench Lite 2016.05-7) 5.3.0"
```

Otimização -O1:

```
1 .data
2 v:  .word 5
3   .word 8
4   .word 3
5   .word 4
6   .word 7
```

```
  .word 6
  .word 8
  .word 0
  .word 1
  .word 9
.text
.LC0:
  .ascii  "%d\011\000"
  .text
  .align  2
  .globl  show
  .set  nomips16
  .set  nomicromips
  .ent  show
  .type show, @function
show:
  .frame  $sp,40,$31    # vars= 0, regs= 5/0, args= 16, gp= 0

  addiu $sp,$sp,-40
  sw  $31,36($sp)
  sw  $19,32($sp)
  sw  $18,28($sp)
  sw  $17,24($sp)
  blez  $5,.L2
  sw  $16,20($sp)

  move  $19,$5
  move  $16,$4
  move  $17,$0
  lui $18,%hi(.LC0)
  addiu $18,$18,%lo(.LC0)
.L3:
  lw  $5,0($16)
  jal printf
  move  $4,$18

  addiu $17,$17,1
  bne $19,$17,.L3
  addiu $16,$16,4

.L2:
  jal putchar
  li  $4,10     # 0xa

  lw  $31,36($sp)
  lw  $19,32($sp)
  lw  $18,28($sp)
  lw  $17,24($sp)
  lw  $16,20($sp)
  j $31
  addiu $sp,$sp,40


swap:
  .frame  $sp,0,$31   # vars= 0, regs= 0/0, args= 0, gp= 0

  sll $5,$5,2
  addu  $2,$4,$5
  lw  $3,0($2)
  addiu $5,$5,4
```

```
67    addu  $4,$4,$5
68    lw  $5,0($4)
69    sw  $5,0($2)
70    j $31
71    sw  $3,0($4)
72
73
74  sort:
75    .frame  $sp,48,$31    # vars= 0, regs= 8/0, args= 16, gp= 0
76
77    blez  $5,.L16
78    nop
79
80    addiu $sp,$sp,-48
81    sw  $31,44($sp)
82    sw  $22,40($sp)
83    sw  $21,36($sp)
84    sw  $20,32($sp)
85    sw  $19,28($sp)
86    sw  $18,24($sp)
87    sw  $17,20($sp)
88    sw  $16,16($sp)
89    move  $18,$4
90    move  $20,$4
91    addiu $22,$5,-1
92    move  $21,$0
93    b .L9
94    li  $19,-1      # 0xffffffffffffffff
95
96  .L12:
97    bltz  $16,.L10
98    nop
99
100   lw  $2,0($20)
101   lw  $3,4($20)
102   slt $2,$3,$2
103   beq $2,$0,.L10
104   move  $5,$16
105
106   move  $17,$20
107 .L17:
108   jal swap
109   move  $4,$18
110
111   addiu $16,$16,-1
112   beq $16,$19,.L10
113   nop
114
115   lw  $2,-4($17)
116   addiu $17,$17,-4
117   lw  $3,4($17)
118   slt $2,$3,$2
119   bne $2,$0,.L17
120   move  $5,$16
121
122 .L10:
123   addiu $21,$21,1
124   addiu $20,$20,4
125 .L9:
126   bne $22,$21,.L12
```

```
127   move  $16,$21
128
129   lw  $31,44($sp)
130   lw  $22,40($sp)
131   lw  $21,36($sp)
132   lw  $20,32($sp)
133   lw  $19,28($sp)
134   lw  $18,24($sp)
135   lw  $17,20($sp)
136   lw  $16,16($sp)
137   addiu $sp,$sp,48
138 .L16:
139   j $31
140   nop
141
142
143 main:
144   .frame  $sp,24,$31    # vars= 0, regs= 2/0, args= 16, gp= 0
145
146   addiu $sp,$sp,-24
147   sw  $31,20($sp)
148   sw  $16,16($sp)
149   li  $5,10     # 0xa
150   lui $16,%hi(v)
151   jal show
152   addiu $4,$16,%lo(v)
153
154   li  $5,10     # 0xa
155   jal sort
156   addiu $4,$16,%lo(v)
157
158   li  $5,10     # 0xa
159   jal show
160   addiu $4,$16,%lo(v)
161
162   lw  $31,20($sp)
163   lw  $16,16($sp)
164   j $31
165   addiu $sp,$sp,24
```

Otimização -O2:

```
1  .data
2  v:   .word 5
3    .word 8
4    .word 3
5    .word 4
6    .word 7
7    .word 6
8    .word 8
9    .word 0
10   .word 1
11   .word 9
12   .ident  "GCC: (Sourcery CodeBench Lite 2016.05-7) 5.3.0"
13 .text
14 .LC0:
15   .ascii  "%d\011\000"
16   .text
17   .align  2
18   .globl  show
19   .set  nomips16
20   .set  nomicromips
```

```
  .ent  show
  .type show, @function
show:
  .frame  $sp,40,$31    # vars= 0, regs= 5/0, args= 16, gp= 0

  blez  $5,.L11
  nop

  addiu $sp,$sp,-40
  sw  $19,32($sp)
  lui $19,%hi(.LC0)
  sw  $17,24($sp)
  move  $17,$0
  addiu $19,$19,%lo(.LC0)
  sw  $18,28($sp)
  sw  $16,20($sp)
  move  $18,$5
  sw  $31,36($sp)
  move  $16,$4
.L3:
  lw  $5,0($16)
  addiu $17,$17,1
  move  $4,$19
  jal printf
  addiu $16,$16,4

  bne $18,$17,.L3
  lw  $31,36($sp)

  li  $4,10     # 0xa
  lw  $19,32($sp)
  lw  $18,28($sp)
  lw  $17,24($sp)
  lw  $16,20($sp)
  j putchar
  addiu $sp,$sp,40

.L11:
  j putchar
  li  $4,10     # 0xa


swap:
  .frame  $sp,0,$31   # vars= 0, regs= 0/0, args= 0, gp= 0

  sll $5,$5,2
  addiu $2,$5,4
  addu  $5,$4,$5
  addu  $4,$4,$2
  lw  $2,0($5)
  lw  $3,0($4)
  sw  $3,0($5)
  j $31
  sw  $2,0($4)


sort:
  .frame  $sp,0,$31   # vars= 0, regs= 0/0, args= 0, gp= 0

  blez  $5,.L28
```

```
81    move   $10,$0
82
83    addiu  $12,$5,-1
84    li     $9,-1      # 0xffffffffffffffff
85    beq    $10,$12,.L28
86    move   $3,$10
87
88  .L19:
89    bltz   $10,.L27
90    addiu  $11,$4,4
91
92    lw     $5,0($4)
93    lw     $6,4($4)
94    slt    $2,$6,$5
95    beq    $2,$0,.L24
96    nop
97
98    move   $2,$4
99    b .L18
100   move   $7,$11
101
102 .L25:
103   lw     $5,-4($2)
104   addiu  $7,$7,-4
105   lw     $6,0($2)
106   slt    $8,$6,$5
107   beq    $8,$0,.L17
108   addiu  $2,$2,-4
109
110 .L18:
111   addiu  $3,$3,-1
112   sw     $6,0($4)
113   sw     $5,0($7)
114   bne    $3,$9,.L25
115   addiu  $4,$4,-4
116
117 .L17:
118   addiu  $10,$10,1
119   move   $4,$11
120 .L26:
121   bne    $10,$12,.L19
122   move   $3,$10
123
124 .L28:
125   j $31
126   nop
127
128 .L24:
129 .L27:
130   addiu  $10,$10,1
131   b .L26
132   move   $4,$11
133
134
135 main:
136   .frame   $sp,24,$31     # vars= 0, regs= 2/0, args= 16, gp= 0
137
138   addiu  $sp,$sp,-24
139   li     $5,10      # 0xa
140   sw     $16,16($sp)
```

```
  lui $16,%hi(v)
  sw  $31,20($sp)
  jal show
  addiu $4,$16,%lo(v)

  addiu $4,$16,%lo(v)
  jal sort
  li  $5,10       # 0xa

  addiu $4,$16,%lo(v)
  lw  $31,20($sp)
  li  $5,10       # 0xa
  lw  $16,16($sp)
  j show
  addiu $sp,$sp,24
```

Otimização -O3:

```
.data
v:
  .word 5
  .word 8
  .word 3
  .word 4
  .word 7
  .word 6
  .word 8
  .word 0
  .word 1
  .word 9
.text
.LC0:
  .ascii  "%d\011\000"
  .text
  .align  2
  .globl  show
  .set  nomips16
  .set  nomicromips
  .ent  show
  .type show, @function
show:
  .frame  $sp,40,$31    # vars= 0, regs= 5/0, args= 16, gp= 0

  blez  $5,.L11
  nop

  addiu $sp,$sp,-40
  sw  $19,32($sp)
  lui $19,%hi(.LC0)
  sw  $17,24($sp)
  move  $17,$0
  addiu $19,$19,%lo(.LC0)
  sw  $18,28($sp)
  sw  $16,20($sp)
  move  $18,$5
  sw  $31,36($sp)
  move  $16,$4
.L3:
  lw  $5,0($16)
  addiu $17,$17,1
  move  $4,$19
  jal printf
```

```
45    addiu $16,$16,4
46
47    bne $18,$17,.L3
48    lw  $31,36($sp)
49
50    li  $4,10       # 0xa
51    lw  $19,32($sp)
52    lw  $18,28($sp)
53    lw  $17,24($sp)
54    lw  $16,20($sp)
55    j putchar
56    addiu $sp,$sp,40
57
58  .L11:
59    j putchar
60    li  $4,10       # 0xa
61
62  swap:
63    .frame  $sp,0,$31   # vars= 0, regs= 0/0, args= 0, gp= 0
64
65    sll $5,$5,2
66    addiu $2,$5,4
67    addu  $5,$4,$5
68    addu  $4,$4,$2
69    lw  $2,0($5)
70    lw  $3,0($4)
71    sw  $3,0($5)
72    j $31
73    sw  $2,0($4)
74
75  sort:
76    .frame  $sp,0,$31   # vars= 0, regs= 0/0, args= 0, gp= 0
77
78    blez  $5,.L28
79    move  $10,$0
80
81    addiu $12,$5,-1
82    li  $9,-1       # 0xffffffffffffffff
83    beq $10,$12,.L28
84    move  $2,$10
85
86  .L19:
87    bltz  $10,.L27
88    addiu $11,$4,4
89
90    lw  $3,0($4)
91    lw  $7,4($4)
92    slt $5,$7,$3
93    beq $5,$0,.L24
94    nop
95
96    move  $6,$4
97    b .L18
98    move  $5,$11
99
100 .L25:
101   lw  $3,-4($6)
102   addiu $5,$5,-4
103   slt $8,$7,$3
104   beq $8,$0,.L17
```

```
105   addiu $6,$6,-4
106
107 .L18:
108   addiu $2,$2,-1
109   sw  $7,0($4)
110   sw  $3,0($5)
111   bne $2,$9,.L25
112   addiu $4,$4,-4
113
114 .L17:
115   addiu $10,$10,1
116   move  $4,$11
117 .L26:
118   bne $10,$12,.L19
119   move  $2,$10
120
121 .L28:
122   j $31
123   nop
124
125 .L24:
126 .L27:
127   addiu $10,$10,1
128   b .L26
129   move  $4,$11
130
131 main:
132   .frame  $sp,24,$31    # vars= 0, regs= 2/0, args= 16, gp= 0
133
134   addiu $sp,$sp,-24
135   li  $5,10     # 0xa
136   sw  $16,16($sp)
137   lui $16,%hi(v)
138   sw  $31,20($sp)
139   jal show
140   addiu $4,$16,%lo(v)
141
142   addiu $4,$16,%lo(v)
143   jal sort
144   li  $5,10     # 0xa
145
146   addiu $4,$16,%lo(v)
147   lw  $31,20($sp)
148   li  $5,10     # 0xa
149   lw  $16,16($sp)
150   j show
151   addiu $sp,$sp,24
```

Otimização -Os:

```
1   #.file  1 "sortc.c"
2   #.section .mdebug.abi32
3   #.previous
4   #.nan legacy
5 # .module fp=32
6   #.module  oddspreg
7
8 .data
9   .globl  v
10  #.align 2
11  #.type  v, #@object
12  #.size  v, 40
```

```
13    v: .word 5, 8, 3, 4, 7, 6, 8, 0, 1, 9
14    #.rdata
15  # .align  2
16  .text
17  main:
18    #.frame $fp,24,$31    # vars= 0, regs= 2/0, args= 16, gp= 0
19    #.mask  0xc0000000,-4
20    #.fmask 0x00000000,0
21    #.set noreorder
22    #.set nomacro
23    addiu $sp,$sp,-24
24    sw  $31,20($sp)
25    sw  $fp,16($sp)
26    move  $fp,$sp
27    li  $5,10     # 0xa
28  # lui $2,%hi(v)
29  # addiu $4,$2,%lo(v)
30    la $2, v
31    jal show
32    nop
33
34    #li $5,10     # 0xa
35    #lui  $2,%hi(v)
36    #addiu  $4,$2,%lo(v)
37    la $2, v
38    jal sort
39    nop
40
41  # li  $5,10     # 0xa
42    #lui  $2,%hi(v)
43    #addiu  $4,$2,%lo(v)
44    la $2, v
45    jal show
46    nop
47
48    nop
49    move  $sp,$fp
50    lw  $31,20($sp)
51    lw  $fp,16($sp)
52    addiu $sp,$sp,24
53
54    li $v0,10
55    syscall
56
57    #.set macro
58    #.set reorder
59    #.end main
60    #.size  main, .-main
61    #.ident "GCC: (Sourcery CodeBench Lite 2016.05-7) 5.3.0"
62
63  .L3:
64    #lw $2,16($fp)
65    sll $2,$2,2
66    lw  $3,32($fp)
67    addu  $2,$3,$2
68    #lw $2,0($2)
69    move  $5,$2
70    #lui  $2,%hi(.LC0)
71    #addiu  $4,$2,%lo(.LC0)
72    la $2, .LC0
```

```
  #jal   printf
  nop

  lw   $2,16($fp)
  addiu $2,$2,1
  sw   $2,16($fp)

.LC0:
  #.ascii "%d\011\000"
  #.text
  #.align 2
  .globl  show
  .set  nomips16
  .set  nomicromips
  #.ent show
  #.type  show, @function
show:
  .frame  $fp,32,$31    # vars= 8, regs= 2/0, args= 16, gp= 0
  #.mask  0xc0000000,-4
  #.fmask 0x00000000,0
  #.set noreorder
  #.set nomacro
  addiu $sp,$sp,-32
  sw   $31,28($sp)
  sw   $fp,24($sp)
  move  $fp,$sp
  sw   $4,32($fp)
  sw   $5,36($fp)
  sw   $0,16($fp)
  b .L2
  nop

.L2:
  lw   $3,16($fp)
  lw   $2,36($fp)
  slt  $2,$3,$2
  bne  $2,$0,.L3
  nop

  li   $4,10      # 0xa
  #jal  putchar
  nop

  nop
  move  $sp,$fp
  lw   $31,28($sp)
  lw   $fp,24($sp)
  addiu $sp,$sp,32
  jr   $ra
  nop

  #.set macro
  #.set reorder
  #.end show
  #.size  show, .-show
  #.align 2
  #.globl swap
  #.set nomips16
  #.set nomicromips
  #.ent swap
```

```
133   #.type  swap, @function
134 swap:
135   #.frame $fp,16,$31    # vars= 8, regs= 1/0, args= 0, gp= 0
136   #.mask  0x40000000,-4
137   #.fmask 0x00000000,0
138   #.set noreorder
139   #.set nomacro
140   addiu $sp,$sp,-16
141   sw  $fp,12($sp)
142   move  $fp,$sp
143   sw  $4,16($fp)
144   sw  $5,20($fp)
145   lw  $2,20($fp)
146   sll $2,$2,2
147   lw  $3,16($fp)
148   addu  $2,$3,$2
149   lw  $2,0($2)
150   sw  $2,0($fp)
151   lw  $2,20($fp)
152   sll $2,$2,2
153   lw  $3,16($fp)
154   addu  $2,$3,$2
155   lw  $3,20($fp)
156   addiu $3,$3,1
157   sll $3,$3,2
158   lw  $4,16($fp)
159   addu  $3,$4,$3
160   lw  $3,0($3)
161   sw  $3,0($2)
162   lw  $2,20($fp)
163   addiu $2,$2,1
164   sll $2,$2,2
165   lw  $3,16($fp)
166   addu  $2,$3,$2
167   lw  $3,0($fp)
168   sw  $3,0($2)
169   nop
170   move  $sp,$fp
171   lw  $fp,12($sp)
172   addiu $sp,$sp,16
173   jr  $ra
174   nop
175
176   #.set macro
177   #.set reorder
178   #.end swap
179   #.size  swap, .-swap
180   #.align 2
181   #.globl sort
182   #.set nomips16
183   #.set nomicromips
184   #.ent sort
185   #.type  sort, @function
186 sort:
187   #.frame $fp,32,$31    # vars= 8, regs= 2/0, args= 16, gp= 0
188   #.mask  0xc0000000,-4
189   #.fmask 0x00000000,0
190   #.set noreorder
191   #.set nomacro
192   addiu $sp,$sp,-32
```

```
193   sw   $31,28($sp)
194   sw   $fp,24($sp)
195   move   $fp,$sp
196   sw   $4,32($fp)
197   sw   $5,36($fp)
198   sw   $0,16($fp)
199   b .L6
200   nop
201
202 .L10:
203   lw   $2,16($fp)
204   addiu $2,$2,-1
205   sw   $2,20($fp)
206   b .L7
207   nop
208
209 .L9:
210   lw   $5,20($fp)
211   lw   $4,32($fp)
212   jal swap
213   nop
214
215   lw   $2,20($fp)
216   addiu $2,$2,-1
217   sw   $2,20($fp)
218 .L7:
219   lw   $2,20($fp)
220   bltz   $2,.L8
221   nop
222
223   lw   $2,20($fp)
224   sll $2,$2,2
225   lw   $3,32($fp)
226   addu   $2,$3,$2
227   lw   $3,0($2)
228   lw   $2,20($fp)
229   addiu $2,$2,1
230   sll $2,$2,2
231   lw   $4,32($fp)
232   addu   $2,$4,$2
233   lw   $2,0($2)
234   slt $2,$2,$3
235   bne $2,$0,.L9
236   nop
237
238 .L8:
239   lw   $2,16($fp)
240   addiu $2,$2,1
241   sw   $2,16($fp)
242 .L6:
243   lw   $3,16($fp)
244   lw   $2,36($fp)
245   slt $2,$3,$2
246   bne $2,$0,.L10
247   nop
248
249   nop
250   move   $sp,$fp
251   lw   $31,28($sp)
252   lw   $fp,24($sp)
```

```
253   addiu $sp,$sp,32
254   jr   $ra
255   nop
256
257   #.set macro
258   #.set reorder
259   #.end sort
260   #.size   sort, .-sort
261   #.align 2
262   #.globl main
263   #.set nomips16
264   #.set nomicromips
265   #.ent main
266   #.type   main, @function
```

3 Sprites

Hoje em dia existem diversas bibliotecas nas mais diversas linguagens de programação de alto nível que permitem realizar, de maneira fácil, o desenho de um sprite em uma determinada posição da tela e detectar a colisão do mesmo com o cenário existente e mesmo outros sprites presentes através de hitboxes. Crie um procedimento (ou conjunto de procedimentos) que permita desenhar sprites na tela gráfica do Mars.

Considere que um Sprite é definido em um endereço (32 bits) na memória de dados que contém a estrutura:

- Tamanho h em pixels (1 byte)
- Tamanho w em pixels (1 byte)
- Bloco de h*w bytes contendo um retângulo que define as cores de cada pixel da imagem do Sprite
- lista de NumH vértices, tamanhos e tipos (todos de 1 byte), que definem as hitboxes do sprite

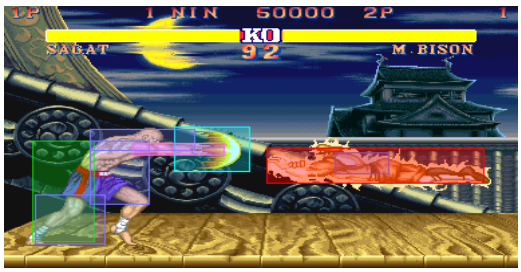


Figura 01 - Hitboxs do Sagat e M.Bison

(2.0) 3.1) int PrintSprite (int EnderecoSprite, int x, int y);

Imprime o sprite presente no endereço EnderecoSprite na posição (x,y) da tela e retorna 0 se não houver colisão ou o 'tipo' da hitbox que apresentou colisão. Grave vídeos demonstrativos e coloque os links no relatório.

A tela gráfica do Mars, acessível pelo BitMap Display Tool, possui resolução 320x240 e 8 bits/pixels para a codificação das cores. O pixel na posição (x,y) pode ser plotado através do comando sb $t0,0($t1):

$t1 = 0xFF000000+320*y+x é o endereço do pixel na memória de vídeo VGA

$t0 = 1 byte que define a cor no formato BBGGGRRR

Obs.: A cor 0xC7 (ou R=255 G=0 B=255) é considerada transparente pelo Mips no FPGA e pelo Mars Custom 7.

3.1) Vídeos: https://youtu.be/rlGKiUh4t2w - Sem colisão
https://youtu.be/-3syoSgDEU0 - Com colisão

Código feito para resolver a questão 3:

```
# Sprites para Lab1

.eqv VGA 0xFF000000

.data
SPRITE: .byte 85,62,0,0
BUFFER: .space 5300
HITBOX: .byte 2, 0, 2, 40, 81, 1, 41, 11, 24, 15, 2, 0 #colocamos
    aqui porque nao tinha no bin
```

```
9
10 COL: .asciiz "COLIDIU COLIDIU COLIDIU"
11 FILE1: .asciiz "menu.bin"
12 FILE2: .asciiz "ryu2.bin"
13
14 .text
15
16 # Preenche a tela de vermelho
17
18   la $t1,0xFF012C00
19   la $s2,0xFF000000
20   la $s1,0xF8F8F8F8
21 LOOP:    beq $s2,$t1,FORA
22   sw $s1,0($s2)
23   addi $s2,$s2,4
24   j LOOP
25 FORA:
26 # Abre o arquivo com tela de fundo
27   la $a0,FILE1
28   li $a1,0
29   li $a2,0
30   li $v0,13
31   syscall
32
33 # Le o arquivo para a memoria VGA
34   move $a0,$v0
35   la $a1,VGA
36   li $a2,76800
37   li $v0,14
38   syscall
39
40 #Fecha o arquivo
41   li $v0,16
42   syscall
43
44
45 # Abre o arquivo sprite
46 PULA: la $a0,FILE2
47   li $a1,0
48   li $a2,0
49   li $v0,13
50   syscall
51
52 # Le o sprite para a memoria BUFFER
53   move $a0,$v0
54   la $a1,BUFFER
55   li $a2,5270
56   li $v0,14
57   syscall
58
59 #Fecha o arquivo
60   li $v0,16
61   syscall
62
63
64   la $a0,SPRITE
65   li $a1,100
66   li $a2,100
67   jal PrintSprite
68
```

```
69    la $a0,SPRITE
70    li $a1,100
71    li $a2,50
72    jal PrintSprite
73
74    beq $v0, $zero, QWE
75    la $a0, COL
76    li $a3, 0x07c7
77    li $a1, 10
78    li $a2, 10
79    li $v0, 104
80    syscall
81
82  QWE:
83    li $v0,10
84    syscall
85
86  PrintSprite:
87  # imprime o sprite na tela
88    lb $t0,0($a0)    # TamX
89    lb $t1,1($a0)    # TamY
90
91    addi $a0,$a0,4
92    move $t2,$zero
93    move $t3,$zero
94
95    li $t5,320
96    mul $t4,$a1,$t5    # 320*x
97    add $t4,$t4,$a2    # +y
98    la $t6,VGA
99    add $t6,$t6,$t4     # endere o inicial de impress o do sprite
100   move $t8,$t6
101   addi $t9, $zero, 0xc7
102
103 LOOP1:   beq $t2,$t0,FORA1
104 LOOP2:   beq $t3,$t1,FORA2
105   lbu $t7,0($a0)
106   beq $t7, $t9, NAOTT
107   sb $t7,0($t6)
108 NAOTT:   addi $a0,$a0,1
109   addi $t6,$t6,1
110   addi $t3,$t3,1
111   j LOOP2
112 FORA2:   addi $t6,$t8,320
113   move $t8,$t6
114   addi $t2,$t2,1
115   move $t3,$zero
116   j LOOP1
117 FORA1:
118   la $t9, HITBOX
119   lb $t0,0($t9)    # NumH
120   addi $t9, $t9, 1
121   move $a3, $zero
122
123 LOOP33: beq $t0, $zero, FORA33
124   addi $t0, $t0, -1
125
126   lb $t4, 0($t9)          # x
127   lb $t5, 1($t9)          # y
128   lb $t6, 2($t9)          # tamx
```

```
129   lb $t7, 3($t9)          # tamy
130   lb $t8, 4($t9)    # tipo
131   mul $t8, $t8, 0x1B
132
133   addi $t9, $t9, 5
134
135   move $t2,$zero
136   move $t3,$zero
137
138   # a1 + t4; a2 + t5
139   add $t4, $t4, $a2
140   add $t5, $t5, $a1
141
142   li $t2, 320
143   mul $t1,$t2,$t5   # 320*linha
144   add $t1,$t1,$t4   # +coluna
145   la $t3,VGA
146   add $t3,$t3,$t1    # endere o inicial de impress o do sprite
147
148   move $t1, $t6
149
150   move $v0, $t3
151   mul $v1, $t2, $t7
152   add $v1, $v1, $t3
153
154 LOOP22: beq $t1, $zero, FORA22
155
156   addi $at, $zero 0
157   lbu $at, 0($v0)
158   addi $t2 $zero, 0x1B
159   bne $at, $t2, VRAUt
160   addi $a3, $zero, 1
161   j VRAU
162 VRAUt:  add $t2, $t2, $t2
163   bne $at, $t2, VRAU
164   addi $a3, $zero, 2
165 VRAU:
166   addi $at, $zero 0
167   lbu $at, 0($v0)
168   addi $t2 $zero, 0x1B
169   bne $at, $t2, VRAUt2
170   addi $a3, $zero, 1
171   j VRAU2
172 VRAUt2: add $t2, $t2, $t2
173   bne $at, $t2, VRAU2
174   addi $a3, $zero, 2
175 VRAU2:
176
177
178   sb $t8, 0($v0)
179   sb $t8, 0($v1)
180   addi $t1, $t1, -1
181   addi $v0, $v0, 1
182   addi $v1, $v1, 1
183   j LOOP22
184 FORA22:
185   move $v0, $t3
186   add $v1, $v0, $t6
187   addi $v1, $v1, -1
188
```

```
189   move $t1, $t7
190   add $t1, $t1, -1
191   addi $v0, $v0, 320
192   addi $v1, $v1, 320
193
194 LOOP11: beq $t1, $zero, FORA11
195
196   addi $at, $zero 0
197   lbu $at, 0($v0)
198   addi $t2 $zero, 0x1B
199   bne $at, $t2, VRAUt3
200   addi $a3, $zero, 1
201   j VRAU3
202 VRAUt3: add $t2, $t2, $t2
203   bne $at, $t2, VRAU3
204   addi $a3, $zero, 2
205 VRAU3:
206   addi $at, $zero 0
207   lbu $at, 0($v0)
208   addi $t2 $zero, 0x1B
209   bne $at, $t2, VRAUt21
210   addi $a3, $zero, 1
211   j VRAU21
212 VRAUt21:  add $t2, $t2, $t2
213   bne $at, $t2, VRAU21
214   addi $a3, $zero, 2
215 VRAU21:
216
217   sb $t8, 0($v0)
218   sb $t8, 0($v1)
219   addi $t1, $t1, -1
220   addi $v0, $v0, 320
221   addi $v1, $v1, 320
222   j LOOP11
223 FORA11:
224   j LOOP33
225 FORA33:
226
227
228   move $v0, $a3
229
230   jr $ra
231
232 FUNCAOZINHA:
233
234
235   jr $ra
```

4 Cálculo das raízes da equação de segundo grau

Dada a equação de segundo grau: a.x$^2 + b.x + c = 0$

(1.0) 4.1) Escreva um procedimento int baskara(float a, float b, float c) que retorne 1 caso as raízes sejam reais e 2 caso as raízes sejam complexas conjugadas, e coloque na pilha os valores das raízes.

(0.5) 4.2) Escreva um procedimento void show(int t) que receba o tipo (t=1 raízes reais, t=2 raízes complexas), retire as raízes da pilha e as apresente na tela, conforme os modelos abaixo:

Para raízes reais:	Para raízes complexas:
R(1)=1234.0000	R(1)=1234.0000 + 5678.0000 i
R(2)=5678.0000	R(2)=1234.0000 – 5678.0000 i

(0.5) 4.3) Escreva um programa main que leia do teclado os valores float de a, b e c, execute as rotinas baskara e show e volte a ler outros valores.

(1.0) 4.4) Escreva as saídas obtidas para os seguintes polinômios [a, b, c] e, considerando um processador MIPS de 1GHz, onde instruções tipo-J são executadas em 1 ciclo, tipo-R em 2 ciclos, tipo-I em 3 ciclos, tipo-FR e FI em 4 ciclos de clock, calcule os tempos de execução da sua rotina baskara (otimizada).

a) [1, 0, -9.86960440]
b) [1, 0, 0]
c) [1, 99, 2459]
d) [1, -2468, 33762440]
e) [0, 10, 100]

4.4)
a) 102×10^{-6}

b) 102×10^{-6}

c) 104×10^{-6}

d) 104×10^{-6}

e) 96×10^{-6}

Código feito para responder os itens da questão 4:

```
.data

mais: .asciiz " + "
i: .asciiz " i"
r1: .asciiz "R(1)="
r2: .asciiz "R(2)="
quebralinha: .asciiz "\n"

.text
j MAIN

```

```
BASKARA:

  mtc1 $a0, $f0 #a
  mtc1 $a1, $f1 #b
  mtc1 $a2, $f2 #c

  #delta = b*b - 4*a*c
  #x = (-b +- sqrt(delta))/(2*a)

  mul.s $f3, $f1, $f1 # b*b
  mul.s $f4, $f0, $f2 # a*c
  add.s $f4, $f4, $f4 # 2*a*c
  add.s $f4, $f4, $f4 # 4*a*c

  sub.s $f3, $f3, $f4 # b*b - 4*a*c

  # se delta < 0, raizes complexas; caso contrario raizes reais
  mtc1 $zero, $f5
  c.lt.s $f3, $f5
  sub.s $f1, $f5, $f1 # b = -b
  add.s $f0, $f0, $f0 # a = 2*a
  bc1t COMPLEXO


  sqrt.s $f3, $f3 # sqrt(delta)

  add.s $f6, $f1, $f3 # -b + sqrt(delta)
  div.s $f6, $f6, $f0 # (-b + sqrt(delta)) / (2*a)

  sub.s $f7, $f1, $f3 # -b - sqrt(delta)
  div.s $f7, $f7, $f0 # (-b - sqrt(delta)) / (2*a)

  addi $sp, $sp, -8

  mfc1 $t0, $f6
  mfc1 $t1, $f7

  sw $t0, 4($sp)
  sw $t1, 0($sp)

  li $v0, 1
  jr $ra
COMPLEXO:


  sub.s $f3, $f5, $f3 # delta = abs(delta)

  sqrt.s $f3, $f3 # sqrt(delta)
  div.s $f3, $f3, $f0 # sqrt(delta) / (2*a)

  div.s $f1, $f1, $f0 # -b / (2*a)

  # x1 = $f1 + i*$f3; x2 = $f1 + i * (- $f3)

  addi $sp, $sp, -16

  mfc1 $t0, $f1
  mfc1 $t1, $f3
  sub.s $f3, $f5, $f3
  mfc1 $t2, $f3
```

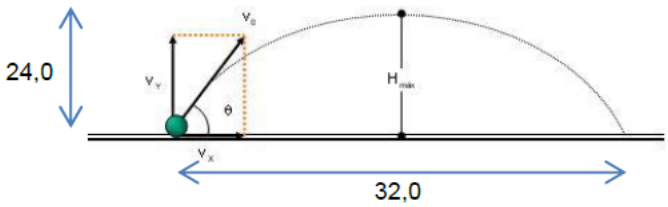
```
72
73
74    sw $t0, 0($sp)
75    sw $t1, 4($sp)
76    sw $t0, 8($sp)
77    sw $t2, 12($sp)
78
79    li $v0, 2
80    jr $ra
81
82 SHOW:
83    li $t0, 1
84    beq $a0, $t0, SHOW_REAL
85
86    li $v0, 4
87    la $a0, r1
88    syscall # print R(1) =
89
90    li $v0, 2
91    lw $t0, 0($sp)
92    mtc1 $t0, $f12
93    syscall #print parte real primeira raiz
94
95    li $v0, 4
96    la $a0, mais
97    syscall # print +
98
99    li $v0, 2
100   lw $t0, 4($sp)
101   mtc1 $t0, $f12
102   syscall #print parte imaginaria primeira raiz
103
104   li $v0, 4
105   la $a0, i
106   syscall # print i
107
108   li $v0, 4
109   la $a0, quebralinha
110   syscall # print \n
111
112   li $v0, 4
113   la $a0, r2
114   syscall # print R(2) =
115
116
117   li $v0, 2
118   lw $t0, 8($sp)
119   mtc1 $t0, $f12
120   syscall #print parte real primeira raiz
121
122   li $v0, 4
123   la $a0, mais
124   syscall # print +
125
126   li $v0, 2
127   lw $t0, 12($sp)
128   mtc1 $t0, $f12
129   syscall #print parte imaginaria primeira raiz
130
131   li $v0, 4
```

```
132   la $a0, i
133   syscall # print i
134
135   li $v0, 4
136   la $a0, quebralinha
137   syscall # print \n
138
139   addi $sp, $sp, 16 # desaloca o espa o usado para guardar as
        raizes
140
141   j END_SHOW
142
143 SHOW_REAL:
144   li $v0, 4
145   la $a0, r1
146   syscall # print R(1) =
147
148   li $v0, 2
149   lw $t0, 4($sp)
150   mtc1 $t0, $f12
151   syscall #print primeira raiz
152
153   li $v0, 4
154   la $a0, quebralinha
155   syscall # print \n
156
157
158   li $v0, 4
159   la $a0, r2
160   syscall # print R(2) =
161
162   li $v0, 2
163   lw $t0, 0($sp)
164   mtc1 $t0, $f12
165   syscall #print segunda raiz
166
167   li $v0, 4
168   la $a0, quebralinha
169   syscall # print \n
170
171
172   addi $sp, $sp, 8 # desaloca o espa o usado para guardar as raizes
173
174 END_SHOW:
175   jr $ra
176
177 MAIN:
178
179   addi $v0, $zero, 6
180   syscall # le a
181   mfc1 $a0, $f0
182
183   addi $v0, $zero, 6
184   syscall # le b
185   mfc1 $a1, $f0
186
187   addi $v0, $zero, 6
188   syscall # le c
189   mfc1 $a2, $f0
190
```

```
191     jal BASKARA
192     move $a0, $v0
193
194     jal SHOW
195
196     j MAIN
```

5 Trajetórias

Uma constante na área de simulação computacional é o cálculo de trajetória de pontos/objetos considerando as leis da física. Dada a aceleração da gravidade 9.8 m/s^2, que o ar não tenha resistência (sem atrito) e que a tela possui um tamanho de $32,0 \times 24,0$ metros conforme o desenho abaixo:



(1.5) 5.1) Realize a simulação do lançamento de uma bola de canhão de tamanho 1 pixel, desenhando a sua trajetória em tempo real com base na velocidade inicial $\vec{V}0$ (com componentes vertical V_{0y}, horizontal V_{0x} e ângulo θ).

Leia os valores necessários do teclado e faça uma animação gráfica do disparo. Considere que a bola possa sair da região visível da tela. Grave vídeos demonstrativos com diversos casos e coloque os links no relatório.

(0.5) 5.2) Indique os valores de θ e $|\vec{V}0|$ para que um lançamento atinja os limites superior e lateral direito da tela.
Obs.: Os pulos do Dhalsim não seguem a física...

5.1) Vídeo: https://youtu.be/HT5wo6jWIVM

Código feito para responder a questão 5:

```
.data

C: .float -49
BN: .asciiz "\n"
COR: .byte 0x07

.text

MAIN:
  li $t9, 10
  mtc1 $t9, $f20
  cvt.s.w $f20, $f20


  li $v0, 6
  syscall
  mul.s $f0, $f0, $f20
  mfc1 $a1, $f0

  syscall
  mul.s $f0, $f0, $f20
  mfc1 $a2, $f0


  li $s0, 0
  li $s1, 320
  lb $t3, COR
```

```
28
29 LOOP: beq $s0, $s1, FIM
30
31   li $a0, 5
32   li $v0, 32
33   syscall
34
35   move $a0, $s0
36   jal F
37   nop
38
39   mtc1 $v0, $f12
40   mtc1 $v0, $f10
41
42   cvt.w.s $f10, $f10
43   mfc1 $t0, $f10
44
45   #x = $s0, y = $t0
46
47   blt $t0, 0, PULA
48   bgt $t0, 239, PULA
49
50   li $t1, 239
51   li $t2, 320
52
53   sub $t1, $t1, $t0
54
55   mul $t1, $t1, 320
56
57   add $t1, $t1, $s0
58   la $t2, 0xff000000
59   add $t2, $t2, $t1
60
61
62   sb $t3, 0($t2)
63 PULA:
64   #move $a0, $t0
65   #li $v0, 1
66   #syscall
67
68   #la $a0, BN
69   #li $v0, 4
70   #syscall
71
72   addi $s0, $s0, 1
73   j LOOP
74 FIM:
75   #j MAIN
76   li $v0, 10
77   syscall
78
79
80 F:  # returna Vy*x/Vx - 4.9*x*x/(Vx*Vx); a0 = x, a1 = Vx, a2 = Vy
81   mtc1 $a0, $f0 #x
82   mtc1 $a1, $f1 #Vx
83   mtc1 $a2, $f2 #Vy
84
85   cvt.s.w $f0, $f0
86
87   mul.s $f3, $f2, $f0
```

```
88   div.s $f3, $f3, $f1
89
90   lwc1 $f4, C #-4.9
91
92   mul.s $f4, $f4, $f0
93   mul.s $f4, $f4, $f0
94   div.s $f4, $f4, $f1
95   div.s $f4, $f4, $f1
96
97   add.s $f12, $f3, $f4
98   mfc1 $v0, $f12
99   jr $ra
100
101 F2: # returna Vy*x/Vx - 4.9*x*x/(Vx*Vx); a0 = x, a1 = Vx, a2 = Vy
102  mtc1 $a0, $f0 #x
103  mtc1 $a1, $f1 #Vx
104  mtc1 $a2, $f2 #Vy
105
106  cvt.s.w $f0, $f0
107
108  mul.s $f3, $f2, $f0
109  div.s $f3, $f3, $f1
110
111  lwc1 $f4, C #-4.9
112
113  mul.s $f4, $f4, $f0
114  mul.s $f4, $f4, $f0
115  div.s $f4, $f4, $f1
116  div.s $f4, $f4, $f1
117
118  add.s $f12, $f3, $f4
119  mfc1 $v0, $f12
120  jr $ra
```

5.2) Os valores são os seguintes: $V_x = 7.22956891292$ e $V_y = 21.6887067388$. Em radianos, seria: $= 1.24904577$ rad e $|V| = 22.861904266$.