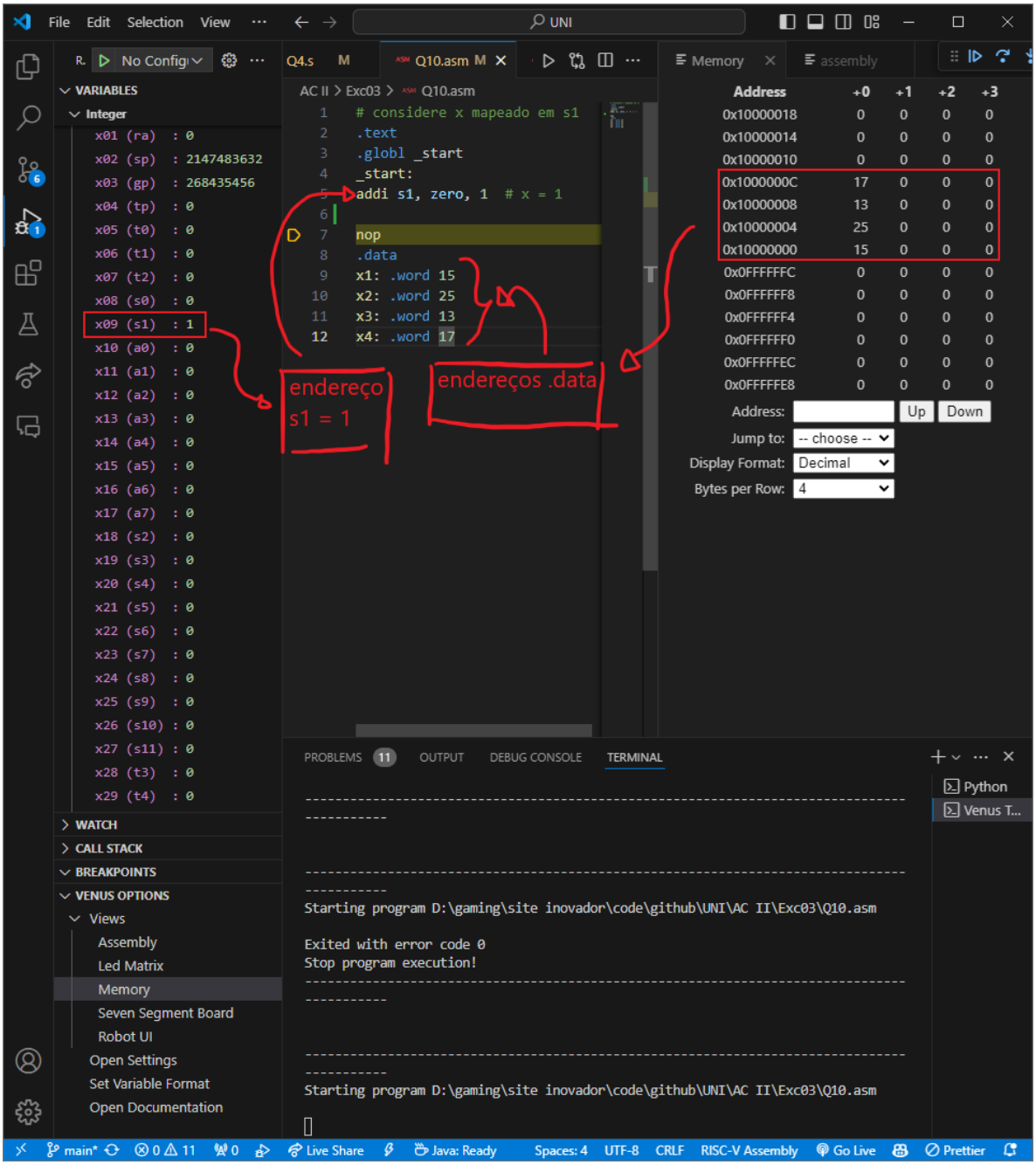
Exercício 03 - Arquitetura de Computadores 2

Henrique Oliveira da Cunha Franco

1 Programa 10



2 Programa 11

```
# considere x mapeado em s1
.text
.globl _start
_start:
addi s1, zero, 1  # x = 1

la t0, x1
la t1, x2
la t2, x3
la t3, x4

lw s0, 0(t0)
lw s1, 0(t1)
lw s2, 0(t2)
lw s3, 0(t3)

nop
.data
x1: .word 15
x2: .word 25
x3: .word 13
x4: .word 17
```

3 Programa 12

```
# considere x mapeado em s1
    .text
    .globl _start
_start:

    la      t0, x1
    la      t1, x2
    la      t2, x3
    la      t3, x4

    lw      s0, 0(t0)
    lw      s1, 0(t1)
    lw      s2, 0(t2)
    lw      s3, 0(t3)

    add     s4, s0, s1
    add     s5, s2, s3
    add     s6, s4, s5

    la      t4, soma

    sw      s6, 0(t4)

    nop
    .data
x1:
    .word   15
x2:
    .word   25
x3:
    .word   13
x4:
    .word   17
soma:
    .word   -1
```

4 Programa 13

```
# Considere a seguinte expressao: y = 127x - 65z + 1
    .text
    .globl _start
_start:

    la      t0, x
    la      t1, z
    la      t2, y

    lw      s0, 0(t0)
    lw      s1, 0(t1)
    lw      s2, 0(t2)

    addi    t3, zero, 127
    addi    t4, zero, 65
    addi    t5, zero, 1

    mul     s3, s0, t3
    mul     s4, s1, t4
    sub     s5, s3, s4
    add     s2, s5, t5

    sw      s2, 0(t2)

    nop

    .data
x:
    .word   5
z:
    .word   7
y:
    .word   0
# esse valor (y) devera ser sobrescrito apos a execucao do programa
```

5 Programa 14

```
    .text
    .globl _start
_start:

# Laco para calcular A[i] = B[i] * (i + 1) + A[i], onde i vai de 0 a 4
# i = 0
    la      s0, A           # Endereco de A
    la      s1, B           # Endereco de B

    lw      t0, 0(s0)       # Carrega A[0] em t0
    lw      t1, 0(s1)       # Carrega B[0] em t1
    addi    t2, zero, 1     # i + 1
    mul     t1, t1, t2      # B[i] * (i + 1)
    add     t0, t0, t1      # B[i] * (i + 1) + A[i]
    sw      t0, 0(s0)       # Armazena o resultado em A[0]

# i = 1
    lw      t0, 4(s0)       # Carrega A[1] em t0
    lw      t1, 4(s1)       # Carrega B[1] em t1
    addi    t2, zero, 2     # i + 1
    mul     t1, t1, t2      # B[i] * (i + 1)
    add     t0, t0, t1      # B[i] * (i + 1) + A[i]
    sw      t0, 4(s0)       # Armazena o resultado em A[1]

# i = 2
    lw      t0, 8(s0)       # Carrega A[2] em t0
    lw      t1, 8(s1)       # Carrega B[2] em t1
    addi    t2, zero, 3     # i + 1
    mul     t1, t1, t2      # B[i] * (i + 1)
    add     t0, t0, t1      # B[i] * (i + 1) + A[i]
    sw      t0, 8(s0)       # Armazena o resultado em A[2]

# i = 3
    lw      t0, 12(s0)      # Carrega A[3] em t0
    lw      t1, 12(s1)      # Carrega B[3] em t1
    addi    t2, zero, 4     # i + 1
    mul     t1, t1, t2      # B[i] * (i + 1)
    add     t0, t0, t1      # B[i] * (i + 1) + A[i]
    sw      t0, 12(s0)      # Armazena o resultado em A[3]

# i = 4
    lw      t0, 16(s0)      # Carrega A[4] em t0
    lw      t1, 16(s1)      # Carrega B[4] em t1
    addi    t2, zero, 5     # i + 1
    mul     t1, t1, t2      # B[i] * (i + 1)
    add     t0, t0, t1      # B[i] * (i + 1) + A[i]
    sw      t0, 16(s0)      # Armazena o resultado em A[4]
    nop
    .data
A:
    .word   1, 3, 5, 7, 9   # int A[] = {1,3,5,7,9};
B:
    .word   2, 4, 6, 8, 10  # int B[] = {2,4,6,8,10};
```

6 Programa 15

```
    .text
    .globl _start
_start:
    la      s0, x
    la      s1, y
    la      s2, m

    lw      t0, 0(s0)
    lw      t1, 0(s1)
    lw      t2, 0(s2)

    bgt     t0, t1, if
else:
    sw      t1, 0(s2)
    j       end
if:
    sw      t0, 0(s2)

end:
    nop

    .data
x:
    .word   7
y:
    .word   23
m:
    .word   0
```

7 Programa 16

```
#int a = um_valor_inteiro_qualquer;
#int b = um_valor_inteiro_qualquer;
#int x = 0;
#.if ( a >= 0 && b <= 50 )
# x = 1;
    .text
    .globl _start
_start:


    la      s0, a
    la      s1, b
    la      s2, x

    lw      s3, 0(s0)
    lw      s4, 0(s1)

    blt     s3, zero, end
    addi    t0, zero, 50
    bgt     s4, t0, end

    addi    t0, zero, 1
    sw      t0, 0(s2)

end:
    nop

    .data
a:
    .word   25
b:
    .word   51
x:
    .word   0
```

8 Programa 17

```
    .text
    .globl _start

_start:
    la      s0, x
    lw      s1, 0(s0)

    beq     s1, zero, case0

    addi    t0, zero, 1
    beq     s1, t0, case1

    addi    t0, zero, 2
    beq     s1, t0, case2

    addi    t0, zero, 3
    beq     s1, t0, case3

    j       default

case0:
    addi    s1, zero, 17
    j       end
case1:
    addi    s1, zero, 29
    j       end
case2:
    addi    s1, zero, 41
    j       end
case3:
    addi    s1, zero, 53
    j       end
default:
    addi    s1, zero, 71

end:
    sw      s1, 0(s0)
    ret

    .data
x:
    .word   1
```

9 Programa 18

```
# Definicoes dos valores dos cases
    .equ    CASE_10, 10
    .equ    CASE_25, 25

# temp deve ser armazenado em s0 e x em s1
    .text
    .globl _start
_start:

# Inicio do switch-case
    li      s1, 0               # Inicializa x com 0
    nop

# Verificacao do case 10
    li      t0, CASE_10         # Carrega o valor do case 10
    bne     s0, t0, check_25    # Verifica se temp e igual a 10
    li      s1, CASE_10         # Se sim, x = 10
    j       end_switch          # Pula para o final do switch-case

check_25:
# Verificacao do case 25
    li      t0, CASE_25         # Carrega o valor do case 25
    bne     s0, t0, default_case # Verifica se temp e igual a 25
    li      s1, CASE_25         # Se sim, x = 25
    j       end_switch          # Pula para o final do switch-case

default_case:
# Caso padrao
    li      s1, 0               # Se nao houver correspondencia, x = 0

end_switch:
    ret
```

10 Programa 19

```
# while(i == 8){
# x = i++;
# }
    .text
    .globl _start
_start:
    li      s0, 8
    li      s1, 0

    li      t0, 8

loop:
    bne     s0, t0, end
    mv      s1, s0
    addi    s0, s0, 1
    j       loop
end:
    ret
```

11 Programa 20

```
# int i;
# int A[10];
# for (i=0; i<10; i++) {
# A[i]=A[i]+1;
# }
    .text
    .globl _start
_start:
    la      s0, A

    li      s1, 0

    li      t2, 10
loop:
    beq     s1, t2, end

    slli    t0, s1, 2
    add     t0, t0, s0

    lw      t1, 0(t0)
    addi    t1, t1, 1
    sw      t1, 0(t0)

    addi    s1, s1, 1
    j       loop
end:
    ret

    .data
A:
    .word   0
```

12 Programa 21

```
# int i;
# int A[10];
# for (i=0; i<10; i++) {
# if (i%2==0)
# A[i]=A[i]+A[i+1];
# else
# A[i]=A[i]*2;
# }
    .text
    .globl _start
_start:
    la      s0, A
    li      s1, 0

    li      t5, 10

loop:
    beq     s1, t5, end

    andi    t0, s1, 1
    beqz    t0, even
odd:
    slli    t1, s1, 2
    add     t1, t1, s0
    lw      t2, 0(t1)
    slli    t2, t2, 1
    sw      t2, 0(t1)
    j       endif
even:
    slli    t1, s1, 2
    add     t1, t1, s0
    lw      t2, 0(t1)
    addi    t4, t1, 4
    lw      t3, 0(t4)
    add     t2, t2, t3
    sw      t2, 0(t1)
endif:
    addi    s1, s1, 1
    j       loop
end:
    nop

    .data
A:
    .word   1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

13 Programa 22

```
## 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

    .text
    .globl _start
_start:

    li      s0, 1
# Inicializa s0 com 1 (primeiro termo da sequencia)
    li      s1, 1
# Inicializa s1 com 1 (segundo termo da sequencia)

    li      s2, 0
# Inicializa s2 com 0 (contador para controlar a iteracao)

    li      t0, 7
# Carrega o valor 7 para t0 (representando o indice do ultimo termo
    desejado)

loop:
# Compara s2 (contador) com t0 (indice do ultimo termo desejado)
# Se forem iguais, o loop termina
    beq     s2, t0, end

# Calcula o proximo termo da sequencia de Fibonacci
# somando os dois ultimos termos (s0 e s1)
    add     s0, s0, s1

# Incrementa o contador s2
    addi    s2, s2, 1

# Verifica se o contador s2 e igual ao indice do ultimo
# termo desejado (t0) Se for, o loop termina
    beq     s2, t0, end

# Calcula o proximo termo da sequencia de Fibonacci
# somando o ultimo termo calculado (s0) com o termo anterior (s1)
    add     s1, s0, s1

# Incrementa o contador s2
    addi    s2, s2, 1

    j       loop        # Pula de volta para o inicio do loop

end:
    ret                 # Retorna (termina a execucao)
```