UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
GRADRUAÇÃO EM ENGENHARIA MECATRÔNICA
DISCIPLINA ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES
PROFESSOR CLÁUDIO C. RODRIGUES

ATIVIDSDADE PRÁTICA II
PROGRAMAÇÃO EM MIPS

EDUARDO MARQUES DA SILVA (11721EMT018)
ENRICO SAMPAIO BONELA (11721EMT007)
LUIZ RENATO RODRIGUES CARNEIRO (11721EMT004)

Uberlândia
2019

P2- Qual é o valor armazenado no registrador $t2 após a execução da sequência de instruções dos itens abaixo? Considere que o valor armazenado no registrador $t0 é **0x55555555** e, no registrador $t1 é **0x12345678**.

A- sll $t2, $t0, 4
or $t2, $t2, $t1

```
Inicialmente:
     t0 = 01010101010101010101010101010101
     t1 = 00010010000110100010101100111000
     t2 = 00000000000000000000000000000000
Após sll:
     t0 = 01010101010101010101010101010101
     t1 = 00010010000110100010101100111000
     t2 = 01010101010101010101010101010000
Após or:
     t0 = 01010101010101010101010101010101
     t1 = 00010010000110100010101100111000
     t2 = 01010111011101010101011101111000
```

RESPOSTA: t2 = 01010111011101010101011101111000 = 0x57755778

B- sll $t2, $t0, 4
andi $t2, $t2, -1

```
Inicialmente:
     t0 = 01010101010101010101010101010101
     t1 = 00010010000110100010101100111000
     t2 = 00000000000000000000000000000000
     -1 = 11111111111111111111111111111111
Após sll:
     t0 = 01010101010101010101010101010101
     t1 = 00010010000110100010101100111000
     t2 = 01010101010101010101010101010000
     -1 = 11111111111111111111111111111111
Após andi:
     t0 = 01010101010101010101010101010101
     t1 = 00010010000110100010101100111000
     t2 = 01010101010101010101010101010000
```

RESPOSTA: t2 = 01010101010101010101010101010000 = 0x55555550

C- srl $t2, $t0, 3
andi $t2, $t2, 0xFFEF

```
Sendo 0xFFEF t3
Inicialmente:
      t0 = 01010101010101010101010101010101
      t1 = 00010010001101000101011001111000
      t2 = 00000000000000000000000000000000
      t3 = 00000000000000001111111111101111
Após srl:
      t0 = 01010101010101010101010101010101
      t1 = 00010010001101000101011001111000
      t2 = 00001010101010101010101010101010
      t3 = 00000000000000001111111111101111
Após andi:
      t0 = 01010101010101010101010101010101
      t1 = 00010010001101000101011001111000
      t2 = 00000000000000001010101010101010
      t3 = 00000000000000001111111111101111
```

RESPOSTA: t2 = 00000000000000001010101010101010 = 0xAAAA

P6- Elabore em linguagem de montagem (assembly) uma sequência mínima de instruções MIPS para realizar as ações das pseudoinstruções abaixo:

A- ble $t3, $t5, L ----> # if ($t3<=$t5) goto L

RESPOSTA:

```
      slt $at, $t5, $t3
      beq $at, $zero, L
```

B- bgt $t4, $t5, L ----> # if ($t4>$t5) goto L

RESPOSTA:

```
      slt $at, $t5, $t4
      bne $at, $zero, L
```

C- if ($t0 >= 0x12345678) {$t2=0;}

RESPOSTA:

```
val:  .word 0x12345678
      .globl main
main:
      la $a1, val
      lw $t1, 0($a1)
      sgt $t3, $t0, $t1
      bne $t3, $zero, if
if:   move $t2, $zero
```

P8- Modos de Endereçamento do MIPS: Temos vários modos de endereçamento para o acesso à memória (imediato não listados):

- endereçamento base deslocamento.
- endereçamento relativo ao PC.
- endereçamento pseudo-direto.
- endereçamento por registrador.

A- Uma determinada solução de programação em MIPS assembly necessita de uma instrução para executar um salto para um endereço (2^28) + 4 bytes distante da atual posição do PC. Como você faria para resolver? Considere que o endereço de destino será conhecido em tempo de compilação.

RESPOSTA:

```
li $t1, 1
sll $t1, $t1, 28
addi $a0,$t1, 4       #$a0 é o endereço de referência
```

B- Uma determinada solução de programação em MIPS assembly necessita de uma instrução para executar um desvio para um endereço (2^17) + 4 bytes distante da atual posição do PC, quando $t0 é igual a 0. Considere que não saltaremos para um endereço superior a 228 bytes. Como você faria para resolver?

RESPOSTA:

```
bne $t0, $zero, endif
li $t1, 1
sll $t1, $t1, 17
li $t2, 1
sll $t2, $t2, 28
bgt $a1, $t2, endif
addi $a1,$t1, 4

endif:
```

P9- Considere as seguintes definições de dados:

```
.data
      var1: .byte 3, -2, 'A'
      var2: .half 1, 256, 0xffff
      var3: .word 0x3de1c74, 0xff
.align 3
      str1: .asciiz "COE308"
```

A- Mostre em código hexadecimal como a memória alocada seria configurada com as declarações acima. Considere que a ordenação de bytes "Little Endian" é aplicada para os bytes das palavras (.word) e meias palavras (.half). Os caracteres 'A'e 'O' estão codificados em ASCII.

RESPOSTA:

DETALHES PARA RESOLUÇÃO DO PROBLEMA:

Na tabela ASCII:A=0x41, C=0x43, O=0x4f, E=Ox45, 3=0x33, 0=0x30, 8=0x38

-var1: .byte 0x03, 0xfe, 0x41

-var2: .half 0x0001, 0x0100, 0xffff

-var3: .word 0x03de1c74, Ox000000ff

-.align 3 |(3 significa que a próxima variável ou string terá o tamanho de um double) 0=byte 1=half 2=word 3=double

-str1: .asciiz "0x0000434f45333038"

-.half = variável tem 2 bits | .word = variável tem 4 bits | .double = variável tem 8 bits

-*little endian* significa o jeito de alocação na memória, exemplo 0x5F229B7C é salvo na memória 7C depois 9B depois 22 depois 5F já no *big endian* é 5F depois 22 depois 9B depois 7C

Considerando isso, temos a memória alocada da seguinte maneira:

```
var1: .byte 0x03, 0xfe, 0x41
var2: .half 0x0001, 0x0100, 0xffff
var3: .word 0x03de1c74, 0x000000ff
.align 3
str1: .asciiz "0x0000434f45333038"
```

P10- Considere as pseudo-instruções abaixo, produza uma sequência mínima de instruções MIPs reais, para que se obtenha o mesmo efeito. Utilize o registrador $at como um registrador temporário.

A- addiu $s1, $s2, imm32 # imm32 is a 32-bit immediate

RESPOSTA:

```
lui $t0, imm32
ori $t0, $t0, imm32
add $si, $s2, $t0
```

B- bge $s1, imm32, Label # imm32 is a 32-bit immediate

RESPOSTA:

```
main:
    la $a0, imm32
    lw $t1, 0($a0)
    sge $t0, $s1, $t1
    bne $t0, $zero, Label
Label:
```

C- rol $s1, $s2, 5 # rol = rotate left $s2 by 5 bits

RESPOSTA:

```
srl $t0, $s2, 27
sll $t1, $s2, 5
or  $s1, $t1, $t0
```

P12- Comente o código MIPS a seguir e descreva em uma sentença que ele computa. Assuma que $a0 é utilizado para a entrada e inicialmente contém n, um inteiro positivo. Assuma que $v0 é utilizado para a saída.

```
begin:   addi $t0, $zero, 0
         addi $t1, $zero, 1
loop:     slt $t2, $a0, $t1
          bne $t2, $zero, finish
          add $t0, $t0, $t1
          addi $t1, $t1, 2
          j loop
finish:   add $v0, $t0, $zero
```

RESPOSTA:

```
begin:   addi $t0, $zero, 0      # armazena valor 0 em $t0
         addi $t1, $zero, 1      # armazena valor 1 em $t1
loop:    slt $t2, $a0, $t1       # se $a0>$t1 $t2=0;se $a0<$t1 $t2=1
         bne $t2, $zero, finish  # se t2 ≠ 0, finaliza (goto finish)
         add $t0, $t0, $t1       # $t0 = $t0 + $t1
         addi $t1, $t1, 2        # $t1 = $t1 + 2
         j loop                  # reinicia o loop
                                 # continua até $t2 = 1 ($a0<$t1)
finish:  add $v0, $t0, $zero     # $v0 = $t0 + 0
```

O programa acima inicia com uma variável t1 de valor 1, t0 de valor 0, e a0 de valor n (inteiro e positivo). A cada ação do loop ele compara a variável t1 com a0, se t1 for maior que a0 o programa se encerra, mas caso t1 seja menor que a0 ele continua a operação do loop que soma t1+t0 e armazena em t0, depois soma t1+2 e armazena na própria t1, o loop só acaba quando essa variável t1 for maior ou igual ao a0, e assim o programa retorna o valor t0, onde t0 = 1+3+5+...+((2*N)-1), em que N é um valor imediatamente anterior ao valor de a0.

P14- Traduza o fragmento de código C abaixo para MIPS assembly. Considere que os arrays inteiros **a** e **b** tem seus endereços base nos registradores **$a0** e **$a1**, respectivamente. O valor de n está no registrador **$a2.**

```
for (i=0 ; i<n ; i++){
      if (i>2){
            a[i] = a[i-2] + a[i-1] + b[i];
      }
      else{
            a[i] = b[i];
      }
}
```

RESPOSTA:

```
      .data
n:    .word 5
a:    .word 1,2,3,4,5
b:    .word 11,12,13,14,15
      .text
      .globl main

main:
      la $a0, a       #armazena o ponteiro de a em $a0
      la $a1, b       #armazena o ponteiro de b em $a1
      la $a2, n
      lw $a2, 0($a2)  #armazena o valor de n em $a2
      move $t1, $zero #contador i=0
for:
      bge $t1, $a2, endfor  #teste do for
      sll $t8,$t1,2         #$t8=4*i
      add $t2, $a0, $t8     #$t2= ponteiro a[i]
      add $t3, $a1, $t8     #$t3= ponteiro b[i]
      lw $t4, 0($t2)        #$t4=a[i]
      lw $t5, -8($t2)       #$t5=a[i-2]
      lw $t6, -4($t2)       #$t6=a[i-1]
      lw $t7, 0($t3)        #$t7=b[i]

      ble $t1, 2, else      #teste do if else
      add $t4, $t5, $t6
      add $t4, $t4, $t7
      sw $t4, 0($t2)        #atualiza o valor de a[i] na memoria
      j endelse
else:
      move $t4, $t7
      sw $t4, 0($t2)        #atualiza o valor de a[i] na memoria
endelse:
```

```
      addi $t1, $t1, 1
endfor:    j for
```

P16- O programa a seguir tenta copiar palavras de um endereço no registrador **$a1**, contando o número de palavras copiadas no registrador **$v0**. O programa para de copiar quando encontra uma palavra igual a **0**. Você não tem que preservar o conteúdo dos registradores **$v1**, **$a0** e **$a1**. Esta palavra de terminação deve ser copiada, mas não contada.

```
Loop:       lw    $v1, 0($a0)           # read next word from source
            addi  $v0, $v0, 1           # increment count words copied
            sw    $v1, 0($a1)           # write to destination
            addi  $a0, $a0, 1           # advance pointer to next source
            addi  $a1, $a1, 1           # advance pointer to next dest
            bne   $v1, $zero, loop      # loop if word copied != zero
```

RESPOSTA:

```
.data
      a1: .word 0, 0, 0, 0, 0, 0
      a0: .word 3, 2, 8, 7, 5, 0
.text
.globl main

main:
      la    $a1,a1
      la    $a0,a0
      move  $v0, $zero
loop:
      lw    $v1, 0($a0)
      addiu  $v0, $v0, 1
      sw    $v1, 0($a1)
      addiu  $a0, $a0, 4
      addiu  $a1, $a1, 4
      bne   $v1, $zero, loop
```

P20- Converta o seguinte fragmento de código, escrito em linguagem C, para a linguagem MIPS assembly:

```
int sumton(unasigned int n){
      if(n==0) return 0;
      else return n + sumton(n-1);
}
```

RESPOSTA:

```
.data
n:    .word 5
      .text
      .globl main
main:
      lw $a0, n
      jal sumton
      move $a0, $v0
      li $v0, 1
      syscall
      li $v0, 10
      syscall

sumton:
      subi $sp, $sp, 8
      sw $ra, 4($sp)
      sw $a0, 0($sp)
      li $v0, 0
      beq $a0,$zero, return
      subi $a0, $a0, 1
      jal sumton
      lw $a0, 0($sp)
      add $v0, $v0, $a0

      return:
      lw $ra 4($sp)
      addi $sp, $sp, 8
      jr $ra
```

P23- Converta o fragmento de código abaixo, escrito em linguagem C, para a linguagem do MIPS assembly:

```
/**Returns the number of bytes in S, but not counting the null terminator.*/
size_t string_length(char *s){
      char *s2 = s;
      while(*s2++);
      return s2-s-1;
}
```

RESPOSTA:

```
.data
      string:.asciiz "12345"
      reposta:.asciiz "O numero de bytes é:"
.text
```

```
.globl main

main:
      la $a0, string
      jal string_lenght
      move $a1, $v0
      li $v0, 4
      la $a0, reposta
      syscall
      li $v0, 1
      move $a0, $a1
      syscall
      j fim

string_lenght:
      move $a2, $a0         #char *s2 = s
      lbu $t2, 0($a2)
while:
      beq $t2, $zero, return     #while(*s2++);
      addi $a2, $a2, 1
      lbu $t2, 0($a2)
      j while

return:
      sub $v0, $a2, $a0     #return s2 - s
      jr $ra
fim:                  #finaliza o programa
      li $v0, 10
      syscall
```

P26- Escreva em linguagem MIPS assembly um programa denominado *contadígitos* que leia do dispositivo padrão de entrada (teclado) um valor inteiro **n**. O programa deve imprimir na tela de saída o valor de **n** e o número de algarismos que possui.

RESPOSTA:

```
.data
     mensagem1: .asciiz " Seu número é: "
     mensagem2: .asciiz "\n O número de digitos desse número é: "
.text
     .globl inicio

     inicio:
          jal le_inteiro_do_teclado  # chama função para ler
          la  $t7, 0($v0)            # carrega o inteiro lido em $t7
          li $v0, 4
          la $a0, mensagem1
          syscall
          jal imprime_inteiro        # manda imprimir o número lido

          jal conta_digito
          li $v0, 4
          la $a0, mensagem2
          syscall
          jal imprime_digito

          j   fim    # encerra o programa

     le_inteiro_do_teclado:
          li $v0, 5  # código para ler um inteiro
          syscall    # executa a chamada do SO para ler
          jr $ra     # volta para o lugar de onde foi chamado (no
                     #caso, jal le_inteiro_do_teclado)

     imprime_inteiro:
          li $v0, 1       # código para imprimir um inteiro
          la $a0, ($t7)   # $a0 é o registrador que irá conter o valor
                          #a ser impresso

          syscall         # executa a chamado do SO para imprimir

          jr $ra          # volta para o lugar de onde foi chamado
                          #(no caso, jal imprime_inteiro)
```

```
conta_digito:
     la   $a0, ($t7) # $a1 é o registrador que irá conter o valor
                     #a ser analizado
     add  $s1, $zero, $a0    # passa o valor de a0 para s1
     addi $t0, $zero, 0      # variável de contagem
     addi $t1, $zero, 1      # variável de controle
     addi $t3, $zero, 0      # variável resposta
     li   $s2,10             # variável multiplicadora

     while:
          bgt $t1, $s1, exit    # enquanto t1 for menor que o
                                #valor digitado
          mult $t1, $s2
          mflo $t1
          addi $t0, $t0, 1     # t0 = t0 + 1
          j while
     exit:
          add $t3, $zero, $t0

          jr $ra     # volta para o lugar de onde foi chamado
                     #(no caso, jal conta_digito)

imprime_digito:
     li $v0,1        # codigo para imprimir um inteiro
     la $a0 , ($t3)  # a0 é o registrador que irá conter o valor
                     #a ser impresso
     syscall
     jr $ra          # volta para o lugar de onde foi chamado
                     #(no caso, jal imprime_digito)

fim:
     li $v0, 10      # código para encerrar o programa
     syscall         # executa a chamada do SO para encerrar
```