Universidade Federal
do Espírito Santo

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Arquitetura de Computadores I – Turmas 1 e 2 (EARTE) – 2021/2
Prof. Rodolfo da Silva Villaça – rodolfo.villaca@ufes.br
Primeira Prova – 16 de dezembro de 2021

NOME:	
MATRÍCULA:	2019108674

Importante: Para esta prova considere que o seu número de matrícula na UFES pode ser representado pelo formato 20*****ZYX_{10}, , sendo Z, Y e X inteiros decimais no intervalo [0..9].

1ª Questão – Foi realizado um *dump* dos segmentos de texto (*.text*) e dados (*.data*) da memória do simulador MARS, programado com a linguagem de montagem MIPS em 32 bits, conforme a referência do livro texto da disciplina. O *dump* de ambos os segmentos está apresentado a seguir e foi realizado antes da execução do programa, imediatamente após a sua carga em memória:

.text
Dump da memória em formato hexadecimal nos endereços de 0x00400000 à 0x00400040.

Endereço (Hex)	Valor (Hex)	Comentário
00400000	3c011001	# lui $1, 0x00001001 – la $a0, numbers*
00400004	34300008	# ori $16, $1, 0x00000008
00400008	3c011001	# lui $1, 0x00001001 – **lw $1, count**
0040000c	8c310000	# lw $17, 0x00000000($1)
00400010	00005020	# Linha 0
00400014	0151082a	# Linha 1
00400018	10200005	# Linha 2
0040001c	214a0001	# Linha 3
00400020	8e080000	# Linha 4
00400024	01284820	# Linha 5
00400028	22100004	# Linha 6
0040002c	1000fff9	# Linha 7
00400030	3c011001	# Linha 8
00400034	ac290004	# Linha 9
00400038	2402000a	# Linha 10
0040003c	0000000c	# syscall

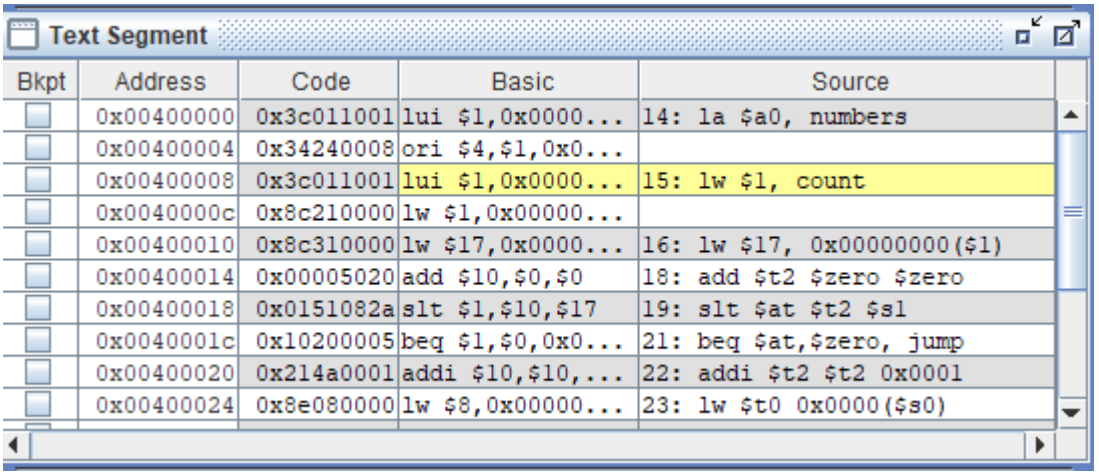
* O comentário representa uma instrução sintética (pseudoinstrução) equivalente

.data

```
count: .word XY      #Alterar para XY da matrícula
res: 0
numbers: .word       1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29,
                     31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63,
                     65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97,
                     99, 101, 103, 105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125,
                     127, 129, 131, 133, 135, 137, 139, 141, 143, 145, 147, 149, 151,
153,
                     155, 157, 159, 161, 163, 165, 167, 169, 171, 173, 175, 177, 179,
181,
                     183, 185, 187, 189, 191, 193, 195, 197, 199, 201, 203, 205, 207,
209,
                     211, 213, 215
```

Considerando que essas são as únicas informações que estão disponíveis, responda:

a) (1,0) Explique como ocorreu o processo de montagem da instrução ***lw $1, count***.
Foi formado por duas instruções nativas:
lui $1, 0x00001001 que carrega no registrador $1 a parte superior da constante e lw $17, 0x00000000($1) a parte inferior. No fim atribuindo a $1 o endereço 0x10010000



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x0000...	14: la $a0, numbers
	0x00400004	0x34240008	ori $4,$1,0x0...	
	0x00400008	0x3c011001	lui $1,0x0000...	15: lw $1, count
	0x0040000c	0x8c210000	lw $1,0x00000...	
	0x00400010	0x8c310000	lw $17,0x0000...	16: lw $17, 0x00000000($1)
	0x00400014	0x00005020	add $10,$0,$0	18: add $t2 $zero $zero
	0x00400018	0x0151082a	slt $1,$10,$17	19: slt $at $t2 $s1
	0x0040001c	0x10200005	beq $1,$0,0x0...	21: beq $at,$zero, jump
	0x00400020	0x214a0001	addi $10,$10,...	22: addi $t2 $t2 0x0001
	0x00400024	0x8e080000	lw $8,0x00000...	23: lw $t0 0x0000($s0)

Como podemos observar na tabela, o código da instrução é:
0x3c011001

Em binário: `0011 1100 0000 0001 0001 0000 0000 0001`
`Ajustando o formato para interpretar a instrução:`

`001111 00000 00001 0001000000000001`

`op:15 = lui`
`rt: $at`
`immediate:0001000000000001`
`Logo chegamos a: LUI $at 0x1001`

b) (1,0) Decodifique as instruções presentes nas linhas “Linha X” e “Linha Y+1” presentes no segmento de texto (*.text*) fornecido nesta questão.
– Restrição: Se X for igual a Y+1 (por exemplo: X=5 e Y=4) então decodifique as linhas “Linha X” e “Linha Y+2”.

x = 6 = addi $s0 $s0 0x0004
y + 1 = 8 -> lui $at 0x1001

c) (1,0) Execute o programa armazenado até o final no simulador MARS. Após essa execução apresente o valor contido no endereço da variável res e identifique esse endereço. O que faz esse programa?

em resumo, o programa soma tudo dos number até o valor de count. Agora escreve bonitinho

```
.data
	count: .word 35
	res: 0
	numbers: .word	1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29,
		  31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63,
		65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97,
		99, 101, 103, 105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125,
		127, 129, 131, 133, 135, 137, 139, 141, 143, 145, 147, 149, 151, 153,
		155, 157, 159, 161, 163, 165, 167, 169, 171, 173, 175, 177, 179, 181,
		183, 185, 187, 189, 191, 193, 195, 197, 199, 201, 203, 205, 207, 209,
		211, 213, 215
.text

 la $s0, numbers # Falva o endereço de numbers no registrador a0.
 lw $s1, count # Salva o conteudo do endereço de count no registrador s1.

 add $t2 $zero $zero # Coloca o valor 0 no registrador t2.

 jump2: slt $at $t2 $s1 # Coloca o registrador at em 1 se o registrador t2 for menor que s1; caso contrário,
coloca-o em 0.

 beq $at,$zero, jump1 # Caso o registrador at seja 0 da um jump para o label: jump
 addi $t2, $t2, 0x0001 # $t2++
 lw $t0, 0x0000($s0) # Salva o conteudo de 0x0000($s0) em t0
 add $t1 $t1 $t0 # Faz t1 = t1 + t0
 addi $s0 $s0 0x0004  # Faz s0 = s0 + 0x0004
 beq $zero $zero jump2 # Da um jump para o final do arquivo label: exit
 jump1:

 sw $t1, res # Salva o valor do registrador t1 em res

 li $v0 10 # Finalisa o programa
 syscall
```

O valor contido no endereço da variável **res** corresponde a 1225. O programa simula uma execução de um for 35 vezes de acordo com o valor do count dentro do loop, acrescendo o valor de t2. Dentro do loop temos um registrador t1 que somará com t0, sendo t0 um dos valores do vetor **numbers**, onde por fim o valor final de t1 será armazenado em **res.**

d) (1,0) Considere que a instrução contida no endereço 0x0040002c foi alterada para 0x08100005 Execute o programa armazenado até o final no simulador MARS. Após essa execução apresente o valor contido no endereço da variável res. Houve alteração?

Explique a diferença entre as duas instruções (a original no item c e a modificada nesta questão).

A instrução no endereço 0x0040002c é: beq $zero $zero exit

Se esse endereço fosse alterado para 0x08100005 a nova instrução seria: *j 0x0100005*

A instrução "beq $zero $zero exit" compara os valores em $zero e $zero, e caso seus valores sejam iguais, o programa recebe o comando de "exit"

A instrução "j 0x0100005" ordena que seja feito um salto para o endereço 0x0100005.

A instrução 0x08100005 equivale a um j loop. O valor contido no endereço da variável "res" permanece inalterado. Por mais que haja diferença entre os códigos, o resultado do programa será o mesmo pois beq $zero $zero loop sempre realizará o jump, uma vez que o registrador $zero está sendo comparado com ele mesmo.
A diferença entre as duas instruções é que j loop não precisa realizar comparações entre dois registradores, sendo portanto, mais eficiente.

Houve alteração?->Não sei, não consigo rodar com a instrução substituta.

e) (1,0) Ignore o XY da variável count nesta questão. Qual seria o maior valor de count que não geraria overflow na execução deste programa? Justifique sua resposta.
O maior valor
Nenhum valor geraria overflow pois a soma total de todos os valores de numbers não gera overflow

2ª Questão – Considere o programa "questao2.c", em linguagem C, conforme código a seguir:

```
// Início da declaração de variáveis (seção .data)
int dim=2;//declara uma variavel chamada dim, inteiro 32 bits, inicializada com valor 2
int a[] = {X, -Y};      // declara um vetor de inteiros, chamado a, com 2 posições.
                        // O vetor a é inicializado com 2 números inteiros X e -Y
int b[] = {-Z, X};      // declara uum vetor de inteiros, chamado b, com 2 posições.
                        // O vetor b é inicializado com 2 números reais -Z e X
int r1[] = {0, 0};      // declara um vetor de inteiros, chamado r1, com 2 posições.
                        // O vetor r1 é inicializado com valores 0 nas 2 posições
double r2[] = {0, 0}  // declara um vetor de float (PF duplo) chamado r2, com 2 posições.
                        // O vetor r2 é inicializado com valores 0 nas 2 posições
void main(void) {               // Início do programa (seção .text)
```

```
    for (int i=0; i<dim; i++) {       // Declara um loop que começa com valor i=0 (i é inteiro, 32 bits) e
                                      // repete enquanto i<dim. i é incrementado a cada iteração
        r1[i] = a[i] % b[i];          // Calcula o resto da divisão dos valores nas posições i (i=0 e i=1)
            // dos vetores a e b (ou seja a[i]%b[i]) e armazena o resultado na                                  //       mesma
posição i do vetor r1 (ou seja, r1[i] = a[i] % b[i])
        r2[i] = (double) a[i] / (double) b[i];    // Calcula a divisão real dos valores nas posições i
                                      // (i=0 e i=1) dos vetores a e b (ou seja a[i]%b[i]) e
            // armazena o resultado na mesma posição i do                                                   // vetor r2 (ou
seja, r2[i] = a[i] / b[i])
            }
}
```

```
void main(void){
        for (int i=0; i<dim; i++) {
                r1[i] = a[i] % b[i];
                r2[i] = (double) a[i] / (double) b[i];
        }
}
```

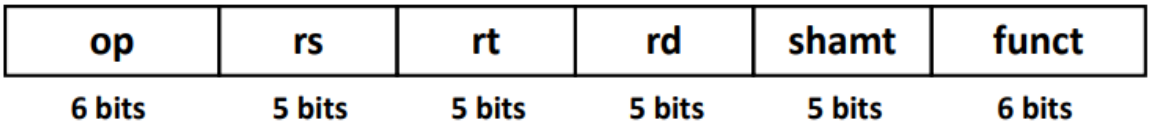
```
XYZ = 123
                .data
                        dim: .word 2  #Y
                        a: .word 1, -2  #X,-Y
                        b: .word  -3, 1 #-z, X
                        r1: .word 0, 0
                        r2: .double 0,0
                        i: .word 0
                .text
                        la $s1, a  # $s1 = a
                        la $s2, b  # $s2 = b
                        la $s3, r1  # $s3 = r1


                        lw $t2, dim  # $t2 = dim
                        lw $t1, i    # $t1 = 0
                        l.d $f2, r2($zero)     # $f2 = r2
                        loop:         # label para o loop

                                lw $t4, 0($s1) # t4 = a[i]
                                lw $t5, 0($s2) # t5 = b[i]
                                lw $t6, 0($s3) # t6 = r1[i]

                                div $t4, $t5  # a[i] / b[i]
                                mfhi $s3  #  r1[i] <— hi = resto da div

                                addi    $s1, $s1, 4  # move a para próxima posição
                                addi    $s2, $s2, 4  # move b para próxima posição
                                addi    $s3, $s3, 4  # move r1 para próxima posição
```

```
                    #mtc1 $t0, $f1
                    #cvt.d.w $f1, $f1 #converte o valor de int para double
                    add $t1, $t1, 1  # incrementa 1 para o loop
                    slt $t0, $t1, $t2  # compra se t1 é menor que t2 e coloca 1 em
$t0 se for, se não 0
                    beq $t0, 1, loop # verifica se $t0 é igual a 1, se for retorna para
o label loop
                    j endloop
                endloop:
```

Altere os valores de X, Y e Z nas variáveis a e b do programa “questao2.asm” de acordo com os valores correspondentes do seu número de matrícula.

a) (1,5) Utilizando a linguagem de montagem do MIPS, tendo como alvo o simulador MARS de 32 bits, apresente um código em linguagem de montagem (“questao2.asm”) que represente o programa “questao2.c”.
Dica 1: nesta questão você deve apenas traduzir o programa de C para ASM, não é preciso gerar código de máquina (binário ou hexa).
Dica 2: se o programa em C original não realiza operações de entrada de dados (*scanf*) e saída de dados (*printf*), seu programa na linguagem de montagem MIPS também não precisará fazer isso! Não faça o que não foi solicitado!
b) (1,5) Execute o seu programa “questao2.asm” no simulador MARS. Apresente um *dump* do conteúdo dos endereços de memória, no segmento de dados (.*data*) que representam as variáveis r1 e r2. Quais são esses endereços? Explique os valores encontrados nesses endereços e como interpretar esses valores no formato IEEE 754.

3ª Questão – Identifique a instrução em linguagem de montagem do MIPS e apresente as representações binárias e hexadecimal das instruções descritas pelos seguintes campos:
a) (1,0) op=0x0, rs=0x3, rt=0x2, rd=0x3, shamt=0x0, funct=0x34

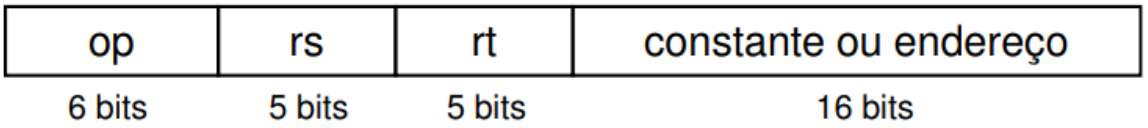


op = 0 (dec) = 000000 (bin)
rs = 3 = 00011 = $v1
rt = 2 = 00010 = $v0
rd = 3 = 00011 = $v1
shamt = 0 = 00000
funct = 52 = 110100 = teq
00000000011000100001100000110100 esse aqui é o certo

Binário: 0000 0000 0110 0010 0001 1110 100 = 00621834(hex)

Instrução: teq $V1, $V0

b) (1,0) op=0x23, rs=0x1, rt=0x2, const=0x4

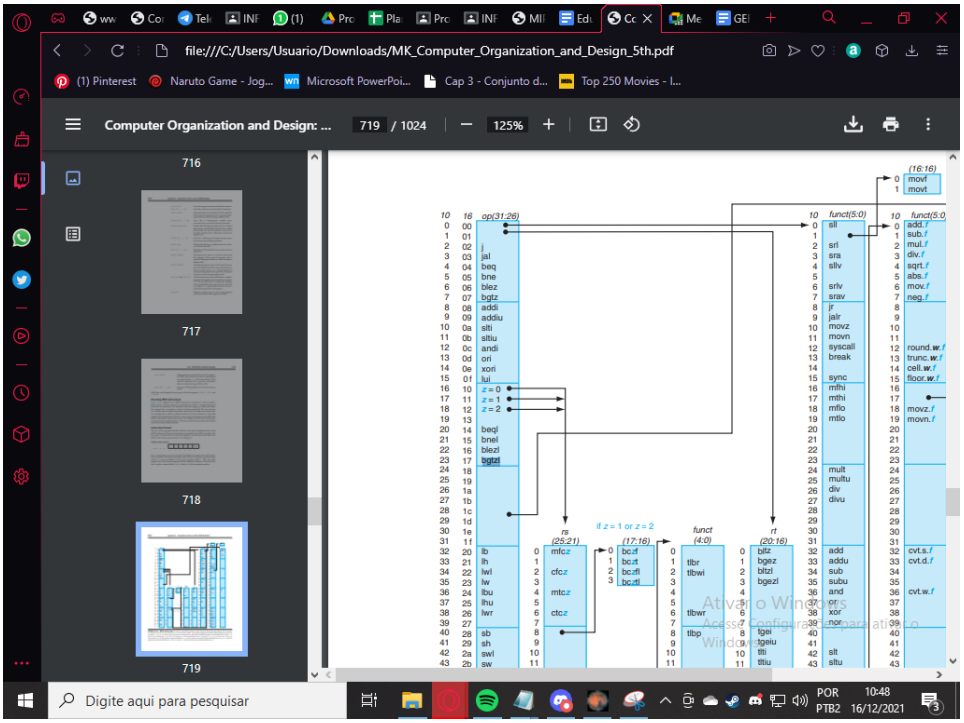


op = 35 (dec) = 100011 (bin) = lw
rs = 1 = 00001 = $at
rt = 2 = 00010 = $v0
const = 4 = 0000000000000100

Binário: 1000 1100 0010 0010 0000 0000 0000 0100 = 8c220004 (hex)
Instrução: lw $v0, 4($at)

Universidade Federal
do Espírito Santo

CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA



https://image.slidesharecdn.com/mipsopcodes-131215221657-phpapp01/95/mips-opcodes-2-638.jpg?cb=1387145844

Boa Prova!