UFSM Universidade Federal de Santa Maria
Centro de Tecnologia – DELC
ELC1011 – Organização de Computadores

Nome: Gabarito _____

Prova 1 (09/10/2023)

Responda a cada uma das questões de forma clara e organizada. Apresente todos os passos da solução.

Questão 01 (2,0 pontos, 20 min.) – Traduza o trecho de código da listagem 1, da linguagem C para assembly do MIPS.

```
/* variáveis globais -- traduzir */
int b[6];
int c[5];
int i, j;

/*...trecho de código -- traduzir */
        i = 3;
        j = 2;
    c[2] = 12345678;  /* 12345678 = 0x00BC614E */
    b[i] = c[j] + b[4];
```

Listagem 1: Trecho de código em C do problema 1.

Resposta:

```
.data
# /* variáveis globais -- traduzir */
# int b[6];
b:          .word 0, 0, 0, 0, 0, 0  # vetor b[6]
# int c[5];
c:          .word 0, 0, 0, 0, 0     # vetor c[5]
# int i, j;
i:          .word 0                 # variável i
j:          .word 0                 # variável j

.text
# mapa de registradores
# $t0 : registrador para guardar valores temporários
# $t1 : registrador para guardar valores temporários
```

```
# $s0 : variável i
# $s1 : variável j
# $s2 : endereço base do vetor c
# $s3 : endereço base do vetor b

# /*...trecho de código -- traduzir */
#         i = 3;
            addiu   $s0, $zero, 3   # $s0 <- 3
            la      $t0, i          # $t0 <- endereço da variável i
            sw      $s0, 0($t0)     # i = 3
#         j = 2;
            addiu   $s1, $zero, 2   # $s1 <- 2
            la      $t0, j          # $t0 <- endereço da variável j
            sw      $s1, 0($t0)     # j = 2
#     c[2] = 12345678;  /* 12345678 = 0x00BC614E */
            lui     $t0, 0x00BC     # $t0 <- 0x00BC_0000
            ori     $t0, $t0, 0x614E # $t0 <- 00BC_614E = 12345678
            la      $s2, c          # $s2 <- endereço base do vetor c
            sw      $t0, 8($s2)     # b[2] = 12345678
#     b[i] = c[j] + b[4];
            sll     $t0, $s1, 2     # $t0 <- 4 * j
            add     $t0, $s2, $t0   # $t0 <- endereço de c[j]
            lw      $t0, 0($t0)     # $t0 <- c[j]
            la      $s3, b          # $s3 <- endereço base do vetor b
            lw      $t1, 16($s3)    # $t1 <- b[4]
            add     $t0, $t0, $t1   # $t0 <- c[j] + b[4]
            sll     $t1, $s0, 2     # $t1 <- 4 * i
            add     $t1, $s3, $t1   # $t1 <- endereço de b[i]
            sw      $t0, 0($t1)     # b[i] = c[j] + b[4]
```

Listagem 2: Tradução para o assembly do trecho de código em C do problema 1.

Questão 02 (2,0 pontos, 20 min.) – Traduza o trecho de código da listagem 3, da linguagem C para assembly do MIPS.

```
/* variáveis globais -- traduzir */
int b[5];
int i, j;

/*...trecho de código -- traduzir */
l0:
        b[2] = i;
        if (b[2] < 10){
                i = i + b[1];
        }else{
                i = 0;
        }
        goto l0;
```

Listagem 3: Trecho de código em C do problema 2.

Resposta:

```
.data
# /* variáveis globais -- traduzir */
# int b[5];
b:          .word 0, 0, 0, 0, 0     # vetor b
# int i, j;
i:          .word 0                 # variável i
j:          .word 0                 # variável j

.text
# mapa de regsitradores
# $t0 : armazenamos valores temporários
# $s0 : endereço da variável i
# $s1 : i, b[2]
# $s2 : endereço base do vetor b

#
# /*...trecho de código -- traduzir */
# l0:
l0:
#         b[2] = i;
            la      $s0, i          # $s0 <- endereço da variável i
            lw      $s1, 0($s0)     # $s1 <- i
            la      $s2, b          # $s2 <- endereço base do vetor b
            sw      $s1, 8($s0)     # b[2] = i
#         if (b[2] < 10){
if_verifica_condicao:
```

```
            slti    $t0, $s1, 10    # $t0 = 1 se b[2]<10
            bne     $t0, $zero, if_condicao_verdadeira
#         }else{
#                 i = 0;
            addiu   $s1, $zero, 0   # $s1 <- 0
            sw      $s1, 0($s0)     # i = 0
#         }
            j       if_fim          # desvia da condição verdadeira
if_condicao_verdadeira:
#                 i = i + b[1];
            lw      $t0, 4($s2)     # $t0 <- b[1]
            add     $s1, $s1, $t0   # $s1 <- i + b[1]
            sw      $s1, 0($s0)     # i = i + b[1]
if_fim:
#         goto l0;
            j       l0              # desvio incondicional para l0
```

Listagem 4: Tradução do C para o assembly do MIPS do trecho de programa da listagem 3.

Questão 03 (2,0 pontos, 20 min.) – Traduza o trecho de código da listagem 5, da linguagem C para assembly do MIPS.

```
/* variáveis globais -- traduzir */
int i, j;

/*...trecho de código -- traduzir */
    j = 0;
    for(i = 0; i < 10; i++){
        j = j + i;
    }
```

Listagem 5: Trecho de código em C do problema 3.

Resposta:

```
.data
# /* variáveis globais -- traduzir */
# int i, j;
i:          .word   0                   # variável i
j:          .word   0                   # variável j

.text
# mapa de registradores
# $t0 : armazena valores temporários
# $s0 : endereço de i
# $s1 : i
# $s2 : endereço de j
# $s3 : j
# /*...trecho de código -- traduzir */
#     j = 0;
            addiu   $s3, $zero, 0       # $s3 <- 0
            la      $s2, j              # $s2 <- endereço da variável j
            sw      $s3, 0($s2)         # j = 0
#     for(i = 0; i < 10; i++){
for_inicializa:
            la      $s0, i              # $s0 <- endereço da variável i
            addiu   $s1, $zero, 0       # $s1 <- 0
            sw      $s1, 0($s0)         # i = 0
            j       for_testa_condicao  # desviamos para verificar se a condição é verda
for_codigo:
#         j = j + i;
            add     $s3, $s3, $s1       # $s3 <- j + i
            sw      $s3, 0($s2)         # j = j + i
for_incrementa:
#     for(i = 0; i < 10; i++){
            addi    $s1, $s1, 1         # $s1 <- i + 1
```

```
            sw      $s1, 0($s0)         # i = i + 1
for_testa_condicao:
#     for(i = 0; i < 10; i++){
            slti    $t0, $s1, 10        # $t0 = 1 se i < 10
            bne     $t0, $zero, for_codigo # se condição verdadeira, execute o código d
#     }
for_fim:
```

Listagem 6: Trecho de código em C do problema 3.

Questão 04 (2,0 pontos, 15 min.) – Represente os números decimais $A = -2{,}7$ e $B = 5{,}3$ em ponto fixo com sinal, em complemento de 2, usando 4 bits para a parte inteira e 4 bits para a parte fracionária. Faça a subtração $D = A – B$ em binário. Apresente os operandos, o resultado e o vem-um. Houve *overflow* na operação de subtração? Por quê?

Resposta:

$A = -2{,}7 =$	1101,0101
$B = 5{,}3 =$	0101,0100

Vai um		1	1	1	1	1	1	1	1	1
A			1	1	0	1	0	1	0	1
$+\overline{B}$			1	0	1	0	1	0	1	1
$= D$			1	0	0	0	0	0	0	1

Os números A e B estão representados em complemento de 2. Não ocorreu estouro na operação de subração $D = A - B$, porque os bits vem-um e vai-um na coluna dos bits mais significativos são iguais.

Questão 05 (2,0 pontos, 15 min.) – Traduza as duas seguintes instruções da linguagem *assembly* para instruções em linguagem de máquina do processador MIPS.

```
0x0040 0000   bne $s0, $s2, loop
0x0040 0004   ...
0x0040 0008   j loop
0x0040 000C   ...
0x0040 0010   loop: ...
```

Resposta:

1ª instrução: `0x0040 0000 bne $s0, $s2, loop`

Nesta primeira instrução temos PC = 0x0040 0000 e loop = 0x0040 0010. Esta é uma instrução tipo I com o opcode 0x05 = 000101 e com o seguinte formato `bne rs, rt, loop`. Nesta instrução rs=$s0=$16=10000 e rt=$s2=$18=10010. Para encontrarmos o valor imediato da instrução, calculamos loop-(PC+4), deslocamos 2 bits para a direita e tomamos os 16 bits menos significativos:

(a) loop - (PC+4) = 0x0040 0010 - 0x0040 0004 = 0x0000 000C.

(b) deslocamos 2 bits para a direita: 0x0000 000C>>2 = 0x0000 0003

(3) tomamos os 16 bits menos significativos: imm = 0x0003.

Juntando todos os campos encontramos:

opcode	rs	rt	imm
000101	10000	10010	0000000000000011

```
= 0001 0110 0001 0010 0000 0000 0000 0011
= 0x1612 0003
```

A instrução assembly `bne $s0, $s2, loop` é traduzida para a instrução em linguagem de máquina `0x1612 0003`.

2ª instrução: `0x0040 0008 j loop`

Nesta instrução tipo j, o endereço de PC = 0x0040 0008 e loop = 0x0040 0010. O opcode da instrução j é 0x02 = 000010. O valor imediato da instrução j possui 26 bits: tomamos os 28 bits menos significativos de loop e descartamos os dois bits menos significativos, que necessariamente devem ser 00. Para que a instrução seja corretamente traduzida, é necessário que os 4 bits mais significativos de PC+4 (0x000 000C) devem ser iguais aos 4 bits mais significativos de loop (0x0040 0010). Neste caso a condição é satisfeita. Vamos calcular o valor do campo imediato com 26 bits:

(a) Tomamos os 28 bits menos significativos de loop = 0x0040 0010: imm' = 0x040 0010.

(b) Verificamos se os 2 bits menos significativos de imm' são iguais a 0. O valor imm' = 0x040 0010 e os dois bits menos significativos são iguais a zero.

(c) Descartamos os dois bits menos significativos de imm' (equivalente a deslocarmos imm' dois bits para a direita): imm=0x0100004.

Juntando todos os campos encontramos:

opcode	imm26
000010	00000100000000000000000100

= 0000 1000 0001 0000 0000 0000 0000 0100
= 0x0810 0004

A instrução assembly `j loop` é traduzida para a instrução em linguagem de máquina `0x0810 0004`.

Fórmulas Desempenho

$$T_{exe} = N_{clk} \cdot T_{clk}$$

$$T_{exe} = N_{inst} \cdot \overline{CPI} \cdot T_{clk}$$

$$T_{exe} = \sum_{i=1}^{N} \left(N_{inst}^{i} \cdot CPI^{i} \right) \cdot T_{clk}$$

$$a_s = \frac{1}{(1-p) + \frac{p}{a}}$$

Registradores

Nome	Número	Nome	Número
$zero	$0	$t8 a $t9	$24 a $25
$at	$1	$k0 a $k1	$26 a $27
$v0 a $v1	$2 a $3	$gp	$28
$a0 a $a3	$4 a $7	$sp	$29
$t0 a $t7	$8 a $15	$fp	$30
$s0 a $s7	$16 a $23	$ra	$31

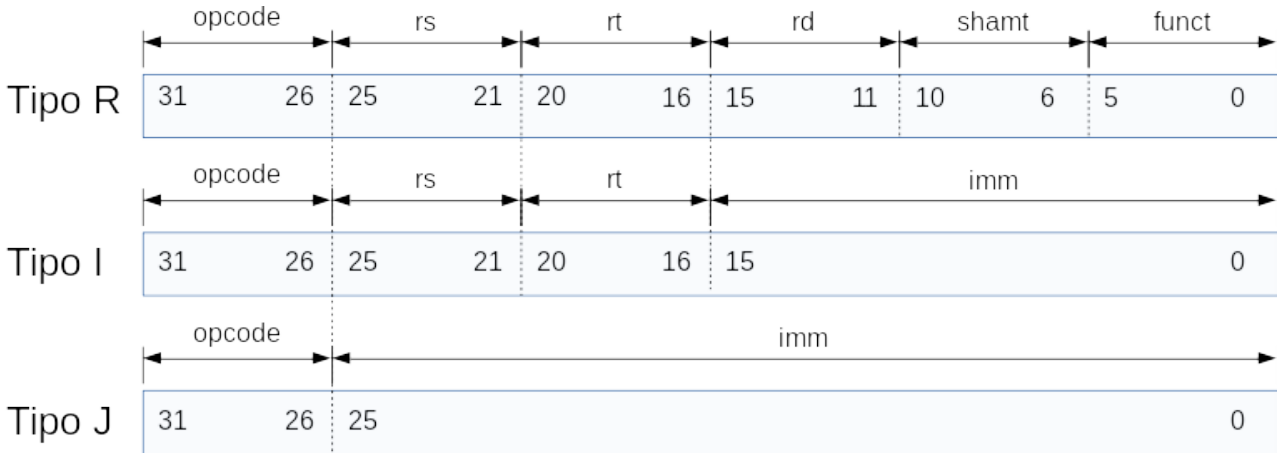

Figura 1: Os campos das instruções R, I e J.

opcode	Instrução
0x00	Tipo R
0x02	j
0x04	beq
0x05	bne
0x0A	slt
0x20	lb
0x23	lw
0x2B	sw

Campo funct	Instrução
0	sll
8	jr
12	syscall
32	add
33	addu
36	and
42	slt
43	sltu

Algumas Instruções

Sintaxe	Significado	Nota
`add $1, $2, $3`	$1 = $2 + $3	Adiciona os registradores $2 e $3 e armazena o resultado em $1.
`sub $1, $2, $3`	$1 = $2 - $3	Realiza a subtração $2-$3 e armazena o resultado em $1.
`addi $1, $2, imm`	$1 = $2 + imm	Adiciona um registrador ($2) com um valor imediato imm de 16 bits (com sinal), após a extensão de sinal.
`lw $1, imm($2)`	$1 = memória[$2+imm]	Carrega 4 bytes da memória, iniciando no endereço $2+imm, no registrador $1.
`lb $1, imm($2)`	$1 = memória[$2+imm]	Carrega o byte (com a extensão do sinal para 32 bits) da memória do endereço $2+imm, no registrador $1.
`sw $1, imm($2)`	memória[$2+imm] = $s1	Armazena os 4 bytes do registrador $1 na memória, iniciando no endereço $2+imm.
`sb $1, imm($2)`	memória[$2+imm] = $s1	Armazena o byte menos significativo do registrador $1 na memória, no endereço $2+imm.
`and $1, $2, $3`	$1 = $2 & $3	Operação AND bit a bit entre o registrador $2 e $3. O resultado é armazenado em $1.
`addi $1, $2, imm`	$1 = $2 & imm	Operação AND bit a bit entre $2 e um valor imediato, após a extensão do sinal (imm sem sinal).
`or $1, $2, $3`	$1 = $2 | $3	Operação OR bit a bit entre o registrador $2 e $3. O resultado é armazenado em $1.
`ori $1, $2, imm`	$1 = $2 + imm	Operação OR bit a bit entre $2 e um valor imediato, após a extensão do sinal (imm sem sinal).
`beq $1, $2, label`	if($1==$2) desvie para label	Desvia para o rótulo label se o valor dos registradores $1 e $2 são iguais.
`bne $1, $2, label`	if($1!=$2) desvie para label	Desvia para o rótulo label se o valor dos registradores $1 e $2 são diferentes.
`j label`	Desvie para label	Desvia para o rótulo label incondicionalmente.
`jr $1`	Desvie o endereço em $1	Desvia para o endereço dado pelo registrador $1.
`jal label`		Desvia para o rótulo label e guarda o endereço de retorno (PC+4) em $ra.
`slt $1, $2, $3`	$1 = 1 se ($2<$3)	Faz o registrador $1 igual a 1 se o registrador $2 é menor ou igual a $3, senão, faz o registrador $1 igual a 0.
`slti, $1, $2, imm`	$1 = 1 se ($2<imm)	Faz o registrador $1 igual a 1 se o registrador $2 é menor ou igual a imm, após a extensão do sinal (complemento de 2), senão 0.
`la $1, label`	$1 = label	Carrega em $1 o endereço dado por label.
`li $1, imm`	$1 = imm	Carrega em $1 o valor imediato imm.
`lui $1, imm`		Carrega o valor imediato imm de 16 bits nos bytes mais significativos de $1. Os bytes menos significativos são feitos iguais a zero.