18/10/2017

UFSM
Universidade Federal de Santa Maria

Departamento de Eletrônica e Computação
ELC 1011 – Organização de Computadores

Nome:_____

2ª Prova – Gabarito

Nas questões a seguir, considere i, j e k inteiros de 32 bits e a um vetor com 20 elementos inteiros de 32 bits.

1 – (2,0 pontos, 15 min.) Traduza o seguinte código em C para assembly do MIPS.

```
for(i = 0; i < 10; i++){
    a[i] = a[i + 1] + i;
}
```

Solução:

.text

```
# $t0 <- endereço base do vetor a
# $t1 <- variável i, armazenada em um registrador.
# $t2 <- endereço de um elemento do vetor a
# $t3 <- conteúdo de um endereço
# $t4 <- resultado de comparacoes
            # instruções antes do laço for
            nop
            # carregamos o endereço do vetor em $t0
            la    $t0, vetor_a
            #inicialização do laço for
inicia_laco_for:
            addi  $t1, $zero, 0    # i = 0
            j     verifica_condicao_laco_for
instrucoes_laco_for:
            # calculamos o endereço de a[i]: end_base + 4*indice_i
            sll   $t2,$t1, 2    # end = 4 * indice_i
            add   $t2, $t0, $t2 # end = end_base + 4*indice_i = endereço de a[i]
            lw    $t3, 4($t2)   # $t3 <- a[i+1]
            add   $t3, $t3, $t1 # $t3 <- a[i+1] + i
            sw    $t3, 0($t2)   # a[i] <- a[i+1]+i
incremento_laco_for:
            addi  $t1, $t1, 1   # i++
verifica_condicao_laco_for:     # se i<10 continue o laco em instrucoes _laço
            slti  $t4, $t1, 10
            bne   $t4, $zero, instrucoes_laco_for
fim_laco_for:
            # instruções após o laço for
            nop

.data
# o vetor a foi incializado com valores para teste
vetor_a: .word 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
```

2 – (1,5 pontos, 15 min.) Traduza o seguinte código em C para assembly do MIPS.

```
if(i > j){
    k = i;
}else{
    a[i] = j;
}
```

Solução:

```
.text
# $t0 <- endereço base do vetor a
# $t1 <- i
# $t2 <- j
# $t3 <- k
# $t4 <- endereco de a[i]
# $t5 <- resultado da comparação de i>j
            #código antes da instrução if
            # carregamos alguns valores para i,j,k para teste
            li    $t1, 10
            li    $t2, 20
            li    $t3, 0
            # carregamos o endereço do vetor base a em $t0
            la    $t0, vetor_a
            # início da estrutura if
            slt   $t5, $t2, $t1 # $t5 = 1 se i>j (ou j < i)
            bne   $t5, $zero, condicao_if_verdadeira
condicao_if_falsa:
            sll   $t4, $t1, 2   # end(a[i]) = indice_i * 4
            add   $t4, $t0, $t4 # end(a[i]) = endereco_base + 4*indice_i
            sw    $t2, 0($t4)   # a[i] = j
            j     fim_instrucao_if
condicao_if_verdadeira:
            add   $t3, $zero, $t1
#fim da instrução if
fim_instrucao_if:
            # outras instruções
            nop
.data
# o vetor a foi incializado com valores, para teste
vetor_a: .word 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
```

3 – (2,0 pontos, 15 min.) Traduza o seguinte código em C para assembly do MIPS.

```
switch(i){
    case 0:
        a[i] = a[i] - 1;
        break;
    case 1:
        a[i] = a[i] + 1;
        break;
    case 2:
        a[i] = a[i] - 10;
        break;
};
```

Solução:

```
.text
# $t0 <- endereço base do vetor a
# $t1 <- i
# $t2 <- valor que será comparado com i
# $t3 <- endereço de a[i]
# $t4 <- valor de a[i]
            # instruções antes da instrução switch
            nop
            # carregamos um valor para i, para teste
            li    $t1, 3
            # carregamos o endereço base do vetor a
            la    $t0, vetor_a
            # início da instrução switch
            # carregamos o vetor a[i]: todas as instruções case usam este valor.
            # o código e o desempenho médio diminuem. Para melhorar o desempenho
            # usar este código em cada instrução case
            sll   $t3, $t1, 2   # end = 4 * indice_i
            add   $t3, $t0, $t3 # end(a[i]) = end_base(a[i]) + 4 * indice_i
            lw    $t4, 0($t3)   # $t4 <- a[i]
            # início das verificações de i com os valores 0, 1 e 2
            addi  $t2, $zero, 0
            beq   $t1, $t2, case_i_igual_0
            addi  $t2, $zero, 1
            beq   $t1, $t2, case_i_igual_1
            addi  $t2, $zero, 2
            beq   $t1, $t2, case_i_igual_2
            j     fim_instrucao_switch
case_i_igual_0:
            addi  $t4, $t4, -1 #        $t4 <- a[i] -1
          # normalmente a instrução j a seguir desviaria para fim_instrucao_switch.
          # No entanto, como todos os cases atualizam a[i], saltamos para atualiza_vetor_a
          # economizando 2 instruções
            j     atualiza_vetor_a
case_i_igual_1:
            addi  $t4, $t4, +1  #       $t4 <- a[i] +1
            j     atualiza_vetor_a
case_i_igual_2:
            addi  $t4, $t4, -10 #       $t4 <- a[i] -10
            j     atualiza_vetor_a
atualiza_vetor_a:
            # atualiza a[i].
            sw    $t4, 0($t3)
fim_instrucao_switch:
            # instruções após a instrução case
            nop
.data
# o vetor a foi incializado com valores, para teste
vetor_a: .word 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
```

4 – (3,0 pontos, 30 min.) Traduza o seguinte código em C para assembly do MIPS.

```c
int fib(int a, int b)
{
    return a + b;
}

void preenche(int* f, int tamanho)
{
    int i;
    for(i = 3; i < tamanho; i++){
        f[i] = fib(f[i-1], f[i-2]);
    }
}

int main(void)
{
    int f[10]; // f é uma variável local. Use a pilha.

    f[0] = 0;
    f[1] = 1;
    f[2] = 1;

    preenche(f, 10);

    return 0;
}
```

Solução:

.text

```
main:
# prólogo
            # ajustamos a pilha para receber 10 itens
            addiu $sp, $sp, -40 # int f[10];
# corpo do procedimento
            li    $t0, 0
            sw    $t0, 0($sp)   # f[0] = 0;
            li    $t0, 1
            sw    $t0, 4($sp)   # f[1] = 1;
            sw    $t0, 8($sp)   # f[2] = 1;
            addu  $a0, $zero, $sp # $a0 <- endereço de f = ($sp+0)
            li    $a1, 10
            jal   preenche
# epílogo
            addiu $sp, $sp, 40  # restauramos a pilha
# Este é o procedimento principal do programa. O programa deve ser encerrado.
            li    $v0, 17       # serviço exit2
            li    $a0, 0        # programa terminou sem erros
            syscall  # fazemos uma chamada ao sistema

preenche:
# prólogo
            # $t0 <- variável local i
            # $t1 <- condicao (i<tamanho)
            # $t2 <- endereço de a[i]

            # mapa da pilha
            # end($sp + 16) <- $ra. O procedimento não é um procedimento folha.
            # end($sp + 12) <- endereço de a[i]
```

```
            # end($sp + 8) <- $a1
            # end($sp + 4) <- $a0
            # end($sp + 0) <- int i;
            addiu $sp, $sp, -20 # ajustamos a pilha para receber 4 itens
            sw    $ra, 16($sp)  # armazenamos o endereço de retorno na pilha
            sw    $a1, 8($sp)   # armazenamos $a1 na pilha
            sw    $a0, 4($sp)   # armazenamos $a0 na pilha
# corpo do procedimento
inicializacao_for:
            li    $t0, 3 # i = 3
            j condicao_for
instrucoes_for:
            sll   $t2, $t0, 2   # end = 4 * indice_i
            addu  $t2, $a0, $t2 # end(f[i]) = end_base(f) + 4 * indice_i
            lw    $a0, -4($t2)  # $a0 <- f[i-1]
            lw    $a1, -8($t2)  # $a1 <- f[i-2]
            sw    $t2, 12($sp)  # armazenamos na pilha o endereço de f[i]
            sw    $t0, 0($sp)   # armazenamos na pilha i
            jal   fib
            lw    $t2, 12($sp)  # restauramos o endereço de f[i]
            lw    $a1, 8($sp)   # restauramos $a1
            lw    $a0, 4($sp)   # restauramos $a0
            sw    $v0, 0($t2)   # f[i] = fib(f[i-1], f[i-2])
            lw    $t0, 0($sp)   # restauramos i
incremento_for:
            addi  $t0, $t0, 1
condicao_for:
            slt   $t1, $t0, $a1 # $t1 = 1 se i<tamanho
            bne   $t1, $zero, instrucoes_for
fim_laco_for:
# epílogo
            lw    $ra, 16($sp)  # restauramos o endereço de retorno
            addiu $sp, $sp, 20  # retauramos a pilha
            jr    $ra # retorna ao procedimento chamador
fib:
# prólogo
          # não precisamos usar a pilha. Não existem variáveis locais. O procedimento
          # é um procedimento folha.
# corpo do procedimento
# epílogo
            add   $v0, $a0, $a1 # $v0 <- a + b
            jr    $ra           # retorna ao procedimento chamador

.data
```

5 – (1,5 pontos (0,5 ponto por instrução *assembly*), 20 min.) Escreva as correspondentes instruções em linguagem de montagem para as seguintes instruções em linguagem de máquina:

Endereço	Código	Instruções em linguagem de montagem (completar)
0x00400000	0x12000001	beq $s0, $zero, loop # se $s0 = $zero salte 1 instrução. Na execução da instrução o valor de PC é 0x0040004 (PC+4). Se $s0 = $zero salte para a instrução de endereço 0x00400008.
0x00400004	0x022b4020	add $t0, $s1, $t3 # $t0 ← $s1 + $t3
0x00400008	0x23e80014	loop: addi $t0, $ra, 20 # $t0 ← $ra + 20

Se houver instruções de desvio incondicional ou condicional, adicionar um comentário, indicando o endereço do desvio.

Tabela 1. Algumas instruções e pseudoinstruções do MIPS.

Nome	Sintaxe	Significado	Nota
Add	add $1,$2,$3	$1 = $2 + $3	Adiciona dois registradores
Subtract	sub $1,$2,$3	$1 = $2 – $3	Subtrai dois registradores
Add immediate	addi $1,$2,CONST	$1 = $2 + CONST (signed)	Adiciona um registrador ($2) com uma constante de 16 bits (com extensão de sinal)
Load word	lw $1,CONST($2)	$1 = Memory[$2 + CONST]	Carrega 4 bytes da memória a partir do endereço de memória ($2 + CONST)
Load byte	lb $1,CONST($2)	$1 = Memory[$2 + CONST] (signed)	Carrega o byte armazenado do endereço de memória ($2 + CONST).
Store word	sw $1,CONST($2)	Memory[$2 + CONST] = $1	Armazena uma palavra a partir do endereço de memória ($2 + CONST)
Store byte	sb $1,CONST($2)	Memory[$2 + CONST] = $1	Armazena o byte menos significativo de $1 no endereço de memória ($2 + CONST)
And	and $1,$2,$3	$1 = $2 & $3	Operação AND bit a bit
And immediate	andi $1,$2,CONST	$1 = $2 & CONST	Operação AND entre $2 e CONST
Or	or $1,$2,$3	$1 = $2 | $3	Operação OR bit a bit
Or immediate	ori $1,$2,CONST	$1 = $2 | CONST	Operação OR entre $2 e CONST
Branch on equal	beq $1,$2,CONST	if ($1 == $2) go to PC+4+ (CONST<<2)	Desvie para o endereço PC + 4 + (const<<2) se $1 é igual a $2
Branch on not equal	bne $1,$2,CONST	if ($1 != $2) go to PC+4+(CONST<<2)	Desvie para o endereço PC + 4 + (const<<2) se $1 e $2 são diferentes
Jump	j CONST	goto address CONST	Salta incondicionalmente para o endereço
Jump register	jr $1	goto address $1	Desvia para o endereço armazenado em $1
jump and link	jal CONST	$31 = PC + 4; goto CONST	Usado para chamar um procedimento: Desvia para o endereço do procedimento e guarda o endereço de retorno em $ra
Set on less than	slt $1,$2,$3	$1 = ($2 < $3)	$1 é igual a 1 se ($2<$3) senão 0
Set on less than immediate	slti $1,$2,CONST	$1 = ($2 < CONST)	$1 é igual a 1 se ($2<CONST) senão 0
Load Address	la $1, LabelAddr		Carrega em $1 o endereço LabelAddr
Load Immediate	li $1, IMMED		Carrega no registrador $1 o valor IMMED
Load upper immediate	Lui $1, IMMED	$1 = IMMED<<16	Carrega nos bits mais significativos IMMED

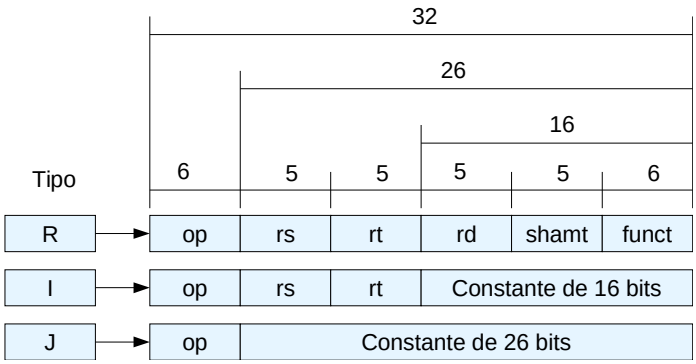

Figura 1 – Ilustração dos campos das instruções R, I e J de um processador MIPS.

Tabela 2 - Exemplos de instruções tipo R (opcode = 000000)

Instrução	Campo funct
add rd, rs, rt	0x20
addu rd, rs, rt	0x21
and rd, rs, rt	0x24
nor rd, rs, rt	0x27
sll rd, rt, shamt	0x00
sllv rd, rt, rs	0x04
sra rd, rt, shamt	0x03
sub rd, rs, rt	0x22
subu rd, rs, rt	0x23
slt rd, rs, rt	0x2A
sltu rd, rs, rt	0x2B

Tabela 3 – Exemplo de instruções tipo I

Instrução	Opcode
addi rt, rs, imediato	0x08
xori rt, rs, imediato	0x0E
slti rt, rs, imediato	0x0A
beq rs, rt, label	0x04
lb rt, imediato(rs)	0x20
lw rt, imediato(rs)	0x23
sb rt, imediato(rs)	0x28
sw rt, imediato(rs)	0x2b

Tabela 4 – Exemplo de instruções tipo J

Instrução	Opcode
j endereço-alvo	0x02
jal endereço-alvo	0x03

Tabela 5 – Nome e número dos registradores do MIPS.

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