16/05/2017

UFSM
Universidade Federal de Santa Maria

Departamento de Eletrônica e Computação
ELC 1011 – Organização de Computadores

Nome: Gabarito _____

1ª Prova

1 – (3,0 pontos) Considere duas máquinas, P1 e P2, que usam o mesmo conjunto de instruções mas possuem implementações diferentes. O conjunto de instruções possui 4 classes de instruções: A, B, C e D. A frequência de relógio e o CPI em cada uma das implementações é dada pela seguinte tabela:

	Frequência do relógio (GHz)	CPI das classes de Instruções			
		A	B	C	D
P1	2,0	1	2	4	4
P2	?	2	2	2	2

Executamos um programa X, com 10^6 instruções, dividida nas classes da seguinte forma: 10% na classe A, 20% na classe B, 40% na classe C e 30% na classe D. (a) Qual a frequência do relógio da máquina P2, para que o desempenho das máquinas sejam iguais?

Solução:

Se o desempenho é igual, o tempo de execução do programa X nas duas máquinas é igual:

$$t_{EXE\ P1} = t_{EXE\ P2}$$

$$\frac{N_{P1}}{f_{P1}} = \frac{N_{P2}}{f_{P2}}$$

e

$$f_{P2} = \frac{N_{P2}}{N_{P1}}\ f_{P1}$$

onde

N_{Pi} = Número de ciclos de relógio da máquina P_i
f_{Pi} = frequência do relógio da máquina P_i

O número de ciclos de relógio durante a execução do programa X é dado por:

$$N_{Pi} = \sum_{j=A,B,C,D} CPI_j\ NI_j$$

Onde

CPI_j = número de ciclos por instrução da classe $j = A, B.C$ ou D
NI_j = Número de instruções da classe $j = A, B.C$ ou D

Calculamos o número de ciclos de relógio para cada uma das máquinas. Para a máquina P1:

$$N_{P1} = (0,1 \cdot 10^6) \cdot 1 + (0,2 \cdot 10^6) \cdot 2 + (0,4 \cdot 10^6) \cdot 4 + (0,3 \cdot 10^6) \cdot 4$$
$$N_{P1} = 3,3 \times 10^6 \text{ ciclos de relógio}$$

Para a máquina P2:

$$N_{P1} = (0,1 \cdot 10^6) \cdot 2 + (0,2 \cdot 10^6) \cdot 2 + (0,4 \cdot 10^6) \cdot 2 + (0,3 \cdot 10^6) \cdot 2$$
$$N_{P1} = 2,0 \times 10^6 \text{ ciclos de relógio}$$

Calculamos a frequência da máquina P2:

$$f_{P2} = \frac{N_{P2}}{N_{P1}} \ f_{P1} = \frac{2,0 \cdot 10^6}{3,3 \cdot 10^6} \ 2 \cdot 10^9 = 1,2 \cdot 10^9 = 1,2 \text{ GHz}$$

2 – (3,0 pontos) Um programa é executado em t segundos: 10 % do tempo é usado na execução de operações em ponto flutuante e 90 % do tempo é usado na execução de operações com inteiros. (a) Quantas vezes mais rápidas devem ser executadas as instruções em ponto flutuante para que o programa seja executado na metade do tempo? Justifique a resposta. (b) Qual a aceleração do programa se as instruções com inteiros forem executadas 5 vezes mais rapidamente?

Solução:

(a) Não é possível executar o programa na metade do tempo otimizando somente as instruções em ponto flutuante. Mesmo que as instruções em ponto flutuante pudessem ser executadas em 0 s, o programa é executado em 0,9 t segundos. Não é possível reduzir este tempo somente otimizando as instruções em ponto flutuante.

(b) A aceleração do sistema é calculada usando a fórmula relacionada à lei de Amdahl:

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

onde

a_s = aceleração do sistema
p = fração otimizada
a = aceleração da fração otimizada

$$a_s = \frac{1}{(1-p) + \frac{p}{a}} = \frac{1}{(1-0,9) + \frac{0,9}{5}} = 3,57$$

3 – (2,0 pontos) Traduza o seguinte trecho de código, da linguagem C para a linguagem assembly para o MIPS:

a = b+ c; d = a + a;	.text ; seu código ← Completar .data varA: .space 4 varB: .word 5 varC: .word 89 varD: .word 56

Solução:

```
.text
# carregamos as variáveis b e c, da memória para registradores
# $s0 <- b
# $s1 <- c

            la    $t0, varB         # carregamos em $t0 o endereço base da variável b
            lw    $s0, 0($t0)       # carregamos em $s0, o valor de b, do endereço efetivo $t0+0
            la    $t0, varC         # carregamos em $t0, o endereço base da variável b
            lw    $s1, 0($t0)       # carregamos em $s1, o valor de c, do endereço efetivo $t0+0
# calculamos b+c e a+a
            add   $t1, $s0, $s1     # $t1 <- $s0 + $s1 = b + c
            add   $t2, $t1, $t1     # $t2 <- $t1 + $t1 = a + a
# atualizamos as variáveis a e d na memória
            la    $t0, varA         # carregamos em $t0 o endereço base da variável a
            sw    $t1, 0($t0)       # atualizamos o valor da variável a na memória
            la    $t0, varD         # carregamos em $t0 o endereço base da variável d
            sw    $t2, 0($t0)       # atualizamos o valor da variável d na memória
.data
varA:       .space 4
varB:       .word 5
varC:       .word 89
varD:       .word 56
```

4 – (2,0 pontos) Traduza o seguinte trecho de código, da linguagem C para a linguagem assembly para o MIPS:

a[2] = a[3] + a[k];	.text ; seu código ← Completar .data varK: .word 1 vetorA: .word 1,2,3,4,5

Obs.: Nos problemas 3 e 4, as variáveis e elementos do vetor são inteiros (32 bits, 4 bytes).

Solução:

```
.text
# carregamos k, a[3] e a[k], da memória para os registradores:
# $s0 <- k
# $s1 <- a[3]
# $s2 <- a[k]

            la    $t0, varK         # carregamos o endereço base da variável k para $t0
            lw    $s0, 0($t0)   # carregamos o valor da variável k da memória para o registrador $s0
            la    $t0, vetorA       # carrgamos o endereço base do vetor A para $t0
            lw    $s1, 12($t0)      # carregamos o valor de a[3] do endereço efetivo $t0+12
            sll   $t1, $s0, 2       # multiplicamos $s0 = k por 4
            add   $t2, $t0, $t1     # $t2 = endereço efetivo do elemento a[k]: end base + 4 * indice
            lw    $s2, 0($t2)       # carregamos o valor do elemento a[3] em $s2
            add   $t3, $s1, $s2     # $t3 <- a[3] + a[k]
# atualizamos o valor de a[2]
            sw    $t3, 8($t0)       # guardamos o valor atualizado de a[2]
.data
varK:       .word 1
vetorA:     .word 1,2,3,4,5
```

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
Shift left logical	sll $1, $2, IMMED	$1 = $2 << IMMED	Deslocamento lógico para a esqueda de IMMED bits

$$Tempo = \frac{Instruções}{Programa} \cdot \frac{Ciclos\ de\ Relógio}{Instrução} \cdot \frac{Segundos}{ciclos\ de\ relógio}$$

Equação 1 – Tempo de execução de um programa.

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