Universidade de Brasília
Departamento de Ciência da Computação

Disciplina: CIC 116394 – Organização e Arquitetura de Computadores – Turma A 2011/1
Prof. Marcus Vinicius Lamar

d_0 d_1 / d_2 d_3 d_4 d_5 d_6 d_7 d_8

Nome:_____ Matrícula: □□/□□□□□□□

Prova 1

(5.0) 1) Dada a série de McLamar definida recursivamente para n≥0 por:

McLamar(n) = McLamar(n-1) - McLamar(n-2) / 4

onde McLamar(0)=0 e McLamar(1)=1

(2.5)a) Implemente um procedimento que calcule recursivamente o enésimo termo da Série de McLamar em ponto flutuante precisão simples IEEE754: float McLamar(int n)

(0.5)b) Qual o maior valor n calculável pelo procedimento implementado em a)?

(2.0)c) Usando aritmética fracionária de ponto fixo nos registradores principais da CPU MIPS (sem usar o Co-processador 1), implemente o procedimento int McLamar (int n) de forma não recursiva, sabendo que não existe uma instrução de divisão em ponto fixo na ISA MIPS.

(4.0) 2) A aritmética computacional deu um grande passo com o desenvolvimento da representação em ponto flutuante. No entanto, para realizar a divisão de dois números em ponto flutuante é necessária a realização de uma operação de subtração e uma operação de divisão em ponto fixo.

(0.5)a) Realize a operação 0x41$d_7$10000 / 0x411$d_8$0000 e apresente o resultado em hexadecimal e decimal.

(1.0)b) Realize manualmente (mostrando todos os passos) a operação utilizando aritmética binária em 32 bits ponto fixo Q15 sinal e magnitude: (d_6d_7+2.25) / (d_8+1.5) até a 4 casa binária

Dica: Lembre-se do algoritmo aprendido no jardim da infância para dividir números com casas decimais.

(2.5)c) A mantissa positiva de um número em notação científica pode ser armazenada em ponto fixo Q31 e um registrador da CPU MIPS. Escreva o código do seguinte procedimento em Assembly MIPS

int divq(int a, int b) ;

que realiza a divisão em ponto fixo da mantissa positiva em a ($a0) pela mantissa positiva em b ($a1) e retorne o resultando no registrador $v0.

(2.0) 3) A arquitetura x86 possui instruções bastante poderosas inexistente na arquitetura MIPS.

cbwr $t0, $s0, $s1 #Copy Block Word Register– Instrução MIPS Tipo-R

Esta instrução copia um bloco de dados de tamanho $s1 words, do endereço inicial da memória apontado pelo registrador $s0, para o endereço inicial da memória apontada pelo registrador $t0. Você como programador de um montador MIPS, dada esta pseudo-instrução, escreva o código real a ser usado em seu lugar no programa.

(1.0) 4) Duas máquinas, A e B, rodam um programa P1 em t_A e t_B segundos respectivamente. Duplicando o número de linhas de P1, a máquina A necessita 20% a mais de tempo para executá-lo. A máquina A possui um CPI médio de 2,5 e a máquina B é um processador uniciclo. Sabendo a frequência da máquina B é o dobro da máquina A, qual será o incremento do tempo de execução do programa P1 na máquina B se o seu número de linhas for triplicado?

BOA SORTE!

OAC - A 1ª Prova 2011/1

Gabarito

1) a) Listagem em Anexo

b) Como $\lim_{n \to \infty} Mclaurin(n) = 0.0$ converge

idealmente: $n = \infty$ Porém n é int. 32 bits

Logo maior $n = 2^{31} - 1 = 2147483647$ ↳ com sinal!

c) Listagem em Anexo

Ponto Fixo: $Mclaurin(n) \leq 1$ Logo

Q31 representa: -1 a $0,9999\ldots$

não representa o $1,00$

Logo: Usa Q30 que representa:

-2^1 a $2^1 - 2^{-30}$ -2 a $1,999999\ldots$

$1,0 \Rightarrow 0x40000000 = 01.00\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$

Divisão em ponto Fixo por potência de 2 SRA

Shift Right Aritmetic

2)

a) $\frac{0x41510000}{0x41160000} = \frac{13,0625}{9,375} = 1,393333333$

$= 0x3FB258BF$

b) Q15:

A = (45+2,25) = 47,25 = 101111.01

B = (6+1,5) = 7,5 = 111.1

101111.01 | 111.1 = 6,3

```
 1011110,1 | 1111
- 1111       110,01001 = 6,25
  10001
   1111
  0010010
     1111
  00011000
      1111
```

c) Listagem em Anexo

mantissa positiva: ± 1,01 11 ... x 2^Y

Q31

```
1,5  = 1,100  =  1100 | 1010
1,25   1,010     1010   1001...
                 0010000 → nº n+1 bits!
                   1010   ↳ cuidado overflow!
```

P/ não ocorrer → aumentar a faixa dinâmica!

```
        addi $SP, $P, -12
        sw $t0, 0($SP)      # Salva valores
        sw $s0, 4($SP.)       originais
        sw $s1, 8 ($SP)
        addi $s1, $s1, -1
LOOP:   SLT $at, $s1, $ZERO
        bne $at, $ZERO, FIM
        Lw $at, 0 ($s0)
        sw $at, 0 ($t0)
        addi $s0, $s0, 4
        addi $t0, $t0, 4
        addi $s1, $s1, -1
        j LOOP
FIm:    Lw $t0, 0($SP)
        Lw $s0, 4 ($SP)
        Lw $s1, 8($SP)
        addi $SP, $SP, 12
```

→ não tem ordem! ✓

4) O nº de linhas que um Programa possui não interfere no tempo de execução:

$$t = I \times CPI \times T$$

- T → período de clock
- CPI → ciclos por instrução
- I → instruções executadas

Não é possível definir quantas instruções são executadas dado o tamanho em linhas do Programa

```
1: .data
2: ZERO:    .float 0
3: UM:  .float 1
4: QUATRO:  .float 4
5:
6: .text
7:
8: main:
9:  li $a0,5
10:     jal McLamar
11:
12:     li $v0,10
13:     syscall
14:
15:
16: McLamar:            # $f0 = McLamar($a0)
17:
18:     addi $sp,$sp,-12    # libera espaço na pilha
19:     sw $ra,0($sp)       # empilha $ra
20:     sw $s0,4($sp)       # empilha registrador salvo $s0
21:     swc1 $f1,8($sp)     # empilha registrador salvo $f1
22:
23:     add $s0,$a0,$zero   # s0=n
24:
25:     addi $t1,$zero,1
26:     beq $s0,$zero,FIM0  # Mclamar(0)=0.0
27:     beq $s0,$t1,FIM1    # McLamar(1)=1.0
28:
29:     addi $a0,$s0,-1     # $a0=n-1
30:
31:     jal McLamar     # $f0=McLamar(n-1)
32:
33:     mov.s $f1,$f0       # $f0 é o registrador de retorno (como se fosse $v0)
34:
35:     addi $a0,$s0,-2         # a0=n-2
36:
37:     jal McLamar             #v0=McLamar(n-2)
38:
39:     l.s $f12,QUATRO
40:     div.s $f0,$f0,$f12  # McLamar(n-2)/4
41:     sub.s $f0,$f1,$f0       # f0 = McLamar(n-1) - McLamar(n-2)/4
42:
43: SAIDA:  lw $ra,0($sp)       # desempilha
44:     lw $s0,4($sp)
45:     lwc1 $f1,8($sp)
46:     addi $sp,$sp,12
47:     jr $ra          # retorna
48:
49: FIM1:
50:     l.s $f0,UM
51:     j SAIDA
```



```
52: FIM0 :
53:     l.s $f0,ZERO
54:     j SAIDA
```

```
1: .data
2:
3: .text
4:
5: main:
6:  li $a0,36
7:  jal McLamar
8:  move $at,$v0  # para visualização de $v0
9:
10:     li $v0,10
11:     syscall
12:
13:
14: McLamar:            # $v0 = McLamar($a0)
15:     li $t0,1
16:     beq $a0,$zero,FIM0  # McLamar(0)=0
17:     beq $a0,$t0,FIM1    # McLamar(1)=1
18:
19:     li $v1,0x00000000
20:     li $v0,0x40000000
21:
22:     li $t1,2            # contador de n
23:
24: LOOP:   sra $t0,$v1,2       # Mclamar(n-2)/4
25:     sub $t0,$v0,$t0     # Mc Lamar(n)=McLamar(n-1)-MacLamar(n-2)/4
26:
27:     move $v1,$v0
28:     move $v0,$t0
29:
30:     slt $t0,$t1,$a0
31:     beq $t0,$zero,SAIDA
32:
33:     addi $t1,$t1,1      #incrementa contador
34:     j LOOP
35:
36:
37: FIM0:   li $v0,0x00000000   # McLamar(0)=0.00
38:     jr $ra
39:
40: FIM1:   li $v0,0x40000000   # McLamar(1)=1.00
41: SAIDA:  jr $ra
```

```
1: .data
2:
3: .text
4:
5: main:
6:  li $a0,0xC000000   # teste 1,5
7:  li $a1,0xA000000    #teste 1,25
8:  jal divq
9:  move $at,$v0  # para visualização de $v0
10:
11:     li $v0,10
12:     syscall
13:
14: divq:   move $t1,$zero      #contador recebe zero : repetir 32 vezes
15:     li $t0,32
16:     move $v0,$zero     #saida recebe zero
17:     srl $a0,$a0,1      # transforma em Q30 perdendo precisão e ganhando em faixa dinâm
ica
18:     srl $a1,$a1,1
19:
20: LOOP:   beq $t1,$t0,FIM
21:
22:     sltu $t2,$a0,$a1    #se $a0>=$a1 $t2=1 else $t2=0
23:     xori $t2,$t2,1      #inverte bit menos significativo de $t2
24:
25:
26: PULA:   sll $v0,$v0,1   # desloca resultado
27:     or $v0,$v0,$t2  # seta o bit menos significativo
28:
29:     beq $t2,$zero,DESL  # se $t2 for 1
30:     subu $a0,$a0,$a1     # subtrai e desloca
31: DESL:   sll $a0,$a0,1   # só desloca $a0 para esquerda
32:
33:     addi $t1,$t1,1
34:     j LOOP
35:
36: FIM:    jr $ra
```