Universidade de Brasília
Departamento de Ciência da Computação

Disciplina: CIC 116394 – Organização e Arquitetura de Computadores – Turma A 2018/1
Prof. Marcus Vinicius Lamar

d_0 d_1 / d_2 d_3 d_4 d_5 d_6 d_7 d_8

Nome: GABARITO Matrícula: 1 8 / 0 1 2 3 4 5 6

Prova 1

(4.0)1) Dado as funções em C ao lado.

(2.0) a) Respeitando a convenção do uso de registradores, compile eficientemente os procedimentos para Assembly RISC-V RV32IMF.

(2.0) b) Para o pior caso de um vetor de entrada de dimensão 100, qual a CPI média que as operações aritméticas/comparação em ponto flutuante devem ter para que seu programa `sort(v,100)` seja executado em 184μs em um processador com frequência de clock de 1GHz, onde as operações da ISA RV32I e de leitura/escrita na memória (lw,sw,flw,fsw) são executadas em um ciclo de clock?

```
void swap(float v[], int k) {
   float temp;
   temp=v[k];
   v[k]=v[k+1];
   v[k+1]=temp;
}

void sort(float v[], int n) {
   int i, t=1;
   while(t==1) {
      t=0;
      for(i=0;i<n-1;i++)
          if(v[i]>v[i+1]) {
               swap(v,i);
               t=1;
          }
   }
}
```

(4.0) 2) Para o programa em linguagem de máquina do processador RISC-V dado no mapa de memória ao lado, onde os endereços e os conteúdos estão em hexadecimal:

(2.0) a) Dessamble o programa escrevendo os mnemônicos e dados usados.

(0.5) b) Quantos bytes de código (instruções) e quantos bytes de dados estão sendo utilizados na sua execução?

(0.5) c) Qual é o valor impresso na tela?

(1.0) d) Se transferirmos todo programa do endereço 0x00400000 para o endereço 0x00520050, reescreva o mapa de memória com o código em linguagem de máquina (em hexadecimal) de modo que ele seja executado corretamente neste novo endereço.

```
00400000 : 00000297
00400004 : 02428293
00400008 : 0002a007
0040000C : 0042a087
00400010 : 18107553
00400014 : 101075d3
00400018 : 00200893
0040001C : 00000073
00400020 : 00a00893
00400024 : 00000073
00400028 : c0b00000
0040002C : 40100000
```

(1.5) 3) O programa montador permite que o programador use diversas pseudoinstruções que tornam um programa em Assembly mais fácil de ser lido e escrito. Escreva o código da ISA RV32IMF que implementa as seguintes pseudoinstruções:

(0.5) a) `jr t0` `# PC = t0`

(1.0) b) `fisqrt.s ft0, ft1` `# ft0 = 1/sqrt(ft1)`

(2.0) 4) Responda:

(1.0) a) A potência dissipada em circuitos digitais CMOS é devido a fatores estáticos e dinâmicos. Cite 2 fatores responsáveis pela potência estática e 2 fatores responsáveis pela potência dinâmica dissipada.

(1.0) b) "*Um processador capaz de executar 10.000.000 de instruções por segundo é mais rápido que outro capaz de executar 5.000.000 de instruções por segundo.*" A afirmação é verdadeira ou falsa? Por que?

Boa sorte!

OAC-A 2018/1

1ª PROVA

GABARITO

1) Na Folha em anexo

2) Pior caso p/ vetor N

Só tem 1 instrução float → FLE.S

t=1 → indica que houve troca (swap)

Logo no pior caso existirão N passadas while

cada for possui N-1 passos

O procedimento swap será executado:

1º N-1 x → 5, 4, 3, 2, 1, 0

2º N-2 x → ↻↻↻↻ 5 vezes

... ... → 4, 3, 2, 1, 0, 5

N-2: 1x ↻↻↻ → 4 vezes

N-1: 0x ⋮

Logo: SWAP: $\frac{N \times (N-1)}{2}$ vezes

Assim: Nº INSTRUÇÕES: FLE.S

$$I = 4 + N \times \left[1 + 2 + (N-1) \times \left[1 + 5 + F + 2 + \frac{4+7}{2} \right] + 1 \right] + 1 + 1$$

$$I = 6 + 4N + N(N-1) \times (F + 13,5) \quad \text{1,0}$$

F ciclos

Logo: $t_{exec} = I \times CPI \times T$

$$184\,\mu = [6 + 4 \times 100 + 100(100-1) \times (F + 13,5)] \times \frac{1}{16} \quad \text{1,0}$$

F = 5,04 ciclos

como só tem 1 instrução float

$CPI_{float} = 5,04$ //

2)

a)

0x00400000	AUIPC t0, 0	0,2
0x00400004	ADDI t0, t0, 0x24	0,2
0x00400008	FLW ft0, 0(t0)	0,2
0x0040000C	FLW ft1, 4(t1)	0,2
0x00400010	FDIV.S Fa0, FT0, FT1	0,2
0x00400014	EMUL.S Fa1, FT0, FT1	0,2
0x00400018	addi a7, ZERO, 2	0,2
0x0040001C	ECALL	0,2
0x00400020	addi a7, zero, 10	0,2
0x00400024	ECALL	0,2
0x00400028	0xC0B00000	
0x0040002C	0x40100000	

Problema: Rars coloca $\sqrt{A}$ ft0 = PC+4!

↳ então ft0 = -5,5 → 0xC0B00000

ft1 = 2,25 → 0x40100000

c) -5,5/2,25 = -2,44444 → impresso na tela 0,5

b) Bytes código = 10x4 = 40 bytes 0,25

Bytes dados = 2x4 = 8 bytes 0,25

↳ considerando ft0 = PC

então ft0 = 0x0000 0073 = $1,61 \times 10^{-43}$

↳ número desnormalizado

Causa underflow → ft0 = 0 //

ft1 = -5,5

Logo

c) 0/-5,5 = 0,0000 → impresso na tela 0,5

d) Como o Programa não usa Endereços Absolutos o código em Linguagem de máquina gerado para o endereço 0x00520050 permanece exatamente o mesmo // 1,0

3) a) jr $t0 → jalr zero, $t0, 0 0,5

b) Não pode alterar nenhum Registrador além de $t0!

```
addi $sp, $sp, -8          fsgnjs $f0, $f1
sw $t0, 0($sp)
fsw $f1, 4($sp)
addi $t0, zero, 1
fcvt.s.w $f0, $t0                    1,0
fsgnj.s $f1, $f1
fdiv.s $f0, $f0, $f1
lw $t0, 0($sp)
flw $f1, 4($sp)
addi $sp, $sp, 8
```

4)

a) Potência ESTÁTICA:

- Corrente de fuga dos transistores 0,25
- Corrente reversa dos diodos 0,25

Potência DINÂMICA: 0,25

- Corrente de carga do gate dos transistores
- Curto circuito de chaveamento 0,25

5) VERDADEIRA!

Não está se falando em desempenho e sim em apenas velocidade na execução de instruções! 1,0

```
.data
# Pior caso, vetor reversamente ordenado
V:  .float 100, 99, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86,
85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71,
70, 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59, 58, 57, 56,
55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42, 41,
40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26,
25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11,
10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0

newl:   .string "\n"
tab:    .string "\t"

.text

#################### Apenas para Verificação ######################
main:   la      a0, V   # argumento0 V
        li      a1, 100 # argumento1 100
        jal     show

        la      a0, V   # argumento0 V
        li      a1, 100 # argumento1 100
        jal     sort    # chama sort

        la      a0, V   # argumento0 V
        li      a1, 100 # argumento1 100
        jal     show

        li      a7, 10  # serviço exit
        ecall
###################################################################


sort:   mv      t0, a0          # salva ponteiro v em t0
        addi    t1, a1, -1      # salva n-1 em t1
        mv      t5, ra          # salva ra em t5 (procedimento não folha)
        li      t2, 1           # t=1

while:  beq     t2, zero, endwhile      # while(t==1)
        mv      t2, zero                # t=0
        mv      t3, zero                # i=0
for:    bge     t3, t1, while           # i>=n-1 ? fim do for, volta para o While
        slli    t4, t3, 2               # i*4
        add     t4, t0, t4              # v+4*i
        flw     f0, 0(t4)               # le v[i]
        flw     f1, 4(t4)               # le v[i+1]
        #addi s0,s0,1   ######### Apenas para confirmar o número de vezes que passa pelo fle.s
        fle.s   t4, f0, f1              # v[i]<=v[i+1] ?
        bne     t4, zero, nextfor       # sim -> nextfor
        mv      a0, t0                  # a0=v
        mv      a1, t3                  # a1=i
        jal     swap                    # call swap
        li      t2, 1                   # t=1
nextfor: addi   t3, t3, 1               # i=i+1
        j       for                     # j for
endwhile: jr    t5                      # retorna para o ra salvo
```

```
swap:   slli    a1, a1, 2       # multiplica k*4
        add     a0, a0, a1      # soma endereço base
        flw     ft0, 0(a0)      # le v[k]
        flw     ft1, 4(a0)      # le v[k+1]
        fsw     ft0, 4(a0)      # grava v[k]
        fsw     ft1, 0(a0)      # grava v[k+1]
        ret



###################### Apenas para Verificação ############################
show:   mv t0,a0
        mv t1,a1
        mv t2,zero

loop1:  beq t2,t1,fim1
        li a7,2
        flw fa0,0(t0)
        ecall
        li a7,4
        la a0,tab
        ecall
        addi t0,t0,4
        addi t2,t2,1
        j loop1

fim1:   li a7,4
        la a0,new1
        ecall
        ret
###########################################################################
```

0,5