Universidade de Brasília
Departamento de Ciência da Computação

Disciplina: CIC 116394 – Organização e Arquitetura de Computadores – Turma A 2016/1
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) A arquitetura ARM possui 9 modos de endereçamento, o que a torna muito mais flexível que a ISA MIPS. Um desses modos de endereçamento permite o acesso a um dado em um endereço calculado pelo *offset* de um número imediato a partir do registador PC. Considere que você tenha disponíveis as instruções tipo I lwpc e swpc que sejam capazes de realizar este tipo de endereçamento na ISA MIPS:

```
lwpc rt,Imm     # R[rt]=M[PC+SigExtImm],  opcode=0x2C
swpc rt,Imm     # M[PC+SigExtImm]=R[rt],  opcode=0x2D
```

Dado o mapa de memória ao lado onde os dados e os endereços estão em hexadecimal:

```
00400000: 3C010040  3428000C  8D0B0004 34080008
00400010: 00010000  11000006  21100001  B00AFFFC
00400020: 014B5020  B40AFFF4 2108FFFF  08100005
00400030: 02172020  24020001  0000000C  2402000A
00400040: 0000000C
```

(3.0)a) Disassemble o código do programa, isto é, escreva o programa em Assembly MIPS.

(1.0)b) Calcule o tempo necessário à execução deste programa em um processador MIPS com CPI=1 e 50MHz de frequência de clock.

(1.0)c) Que valor é impresso na tela?

(2.0)2) A famosa tela azul do sistema operacional Windows é um exemplo clássico de saída da rotina de tratamento de erros. Crie uma rotina de tratamento de erros (label ERROR) que possa ser chamada (jal ERROR) sem argumentos, caso algum erro ou problema seja detectado durante a execução de um programa em assembly MIPS. A rotina deve apresentar a tela abaixo e terminar a execução passando o comando ao sistema operacional.

```
ERROR!!!
PC=<PC>
IR=<instrução>
```

Onde <PC> é o endereço e <Instrução> o código decimal da instrução anterior à chamada de erro.

(4.0)3) O desenvolvimento dos coprocessadores matemáticos foi um passo muito importante para a computação científica. Em 1980 foi lançado o processador 8087, o coprocessador matemático do processador 8086. Nos anos seguintes foram lançados os coprocessadores 80187, 287, 387 e 487 a medida que a família de processadores iniciada pelo 8086 ia avançando. O coprocessador matemático foi incorporado definitivamente ao processador com o lançamento do 486DX. A Intel escolheu uma arquitetura computacional baseada em pilha para a implementação do coprocessador matemático. A pilha é constituída por 8 registradores (ST0 a ST7) de 80 bits cada. Em seu formato mais simples, as instruções não possuem argumentos, sendo os operandos retirados pilha e o resultado da operação escrito novamente na pilha.

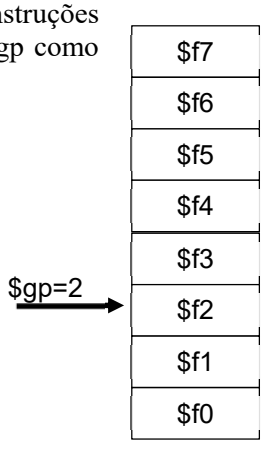
Você é desafiado a criar as pseudo-instruções abaixo, inspiradas pela arquitetura x87. Estas instruções utilizam os registradores $f0 a $f7 como registradores de 32 bits da pilha, e o registrador $gp como apontador de topo da pilha. Se $gp<=-1 a pilha estará vazia e se $gp>=7 a pilha estará cheia.

(1.0)a) push.p $f12 # Insere o valor do registrador $f12 na pilha

(1.0)b) pop.p $f12 # Retira um valor da pilha e coloca no registrador $f12

(1.0)c) add.p # Soma os dois elementos superiores da pilha e coloca o resultado na pilha

(1.0)d) sqrt.p # Substitui o elemento superior da pilha pela sua raiz quadrada



Dica1: Pode usar as instruções da aritmética em ponto flutuante original da ISA MIPS.
Dica2: Pode usar a chamada jal ERROR caso algum erro seja detectado.

Boa Sorte!

OAC-A 1ª PROVA 2016/1

Gabarito

4) a) 0,2 / instrução

```
0x00400000   3C010040   Lui $at, 0x0040
      04     34280000   ORi $t0, $at, 0x000C
      08     8D0B0004   LW $t3, 4($t0)
      0C     34080008   ORi $t0, $ZERO, 0x0008
      10     00010000   SLL $ZERO, $at, 0
      14     11000006   beq $t0, $ZERO, 6 <L1
      18     21100001   addi $s0, $t0, 1        <-
      1C     BD0AFFFC   LWPC $t2, -4
      20     014B5020   add $t2, $t2, $t3
      24     B40AFFF4   SWPC $t2, -12
      28     2108FFFF   addi $t0, $t0, -1
      2C     0810 0005  j 0x00400014
L1:   30     02172020   add $a0, $s0, $s7
      34     24020001   addiu $v0, $ZERO, 1
      38     0000000C   syscall
      3C     24020000A  addiu $v0, $ZERO, 10
      40     0000000C   syscall
```

$t3 = 0x00010000

$t0 = 0x0040000C + 4

$t0 = 0x00000008 * 4

$s0 = 9

$t2 = 0x21100001 + 0x00010000

= 0x21110001 → addi $s1, $t0, 1

21120001 → addi $s2, $t0, 1

21130001 → addi $s3, $t0, 1

4 → $s4

5 → $s5

6 → $s6

7 → $s7

8

1,0 b) $I = 5 + 10 \times 7 + 1 + 5$

↳ $t0 = 8, 7, 6, 5, 4, 3, 2, 1$

$I = 67$

$t_{exec} = 67 \times 1 \times \frac{1}{50M} = 1,34\,\mu s$

1,0 c) O VALOR 11

2)

.data

0,5 L1: .asciiz "ERROR!!! \nPC="

L2: .asciiz "\nIR="

.text

ERROR: addi $t0, $ra, -8 ← 0,5

la $a0, L1

li $v0, 4

syscall

move $a0, $t0 | 0,5

li $v0, 1

syscall

la $a0, L2

li $v0, 4

syscall

lw $a0, 0($t0)

li $v0, 1

syscall

Fim: li $v0, 10 | 0,5

syscall

3) Três Propostas de solução Push e Pop

- Solução trivial: Salva os Regs $f0 ~ $f7 Na Pilha (SSP), manipula e lê de volta Para os Regs.

- Solução direta: verifica qual o Registrador a ser manipulado com sequência de IFS

- Solução mais eficiente? Algoritmo similar à questão 2 com código automodificável.
→ Usar fail ERROR caso estouro de Pilha!

c) add.P → POP, POP, +, PUSH 1,0

d) sqrt.P → POP, √, PUSH 1,0

```
.data
TESTE: .float 3.14159265659
L1: .asciiz "ERROR!!!!\nPC="
L2: .asciiz "\nIR="

.text
    li $gp,-1  #pilha inicialmente vazia
    l.s $f12,TESTE # carrega valor de $f12
    jal PUSHP # push.p $f12
    jal PUSHP # push.p $f12
    jal ADDP # add.p
    jal SQRTP # srt.p
    jal POPP # push.d $f12

    li $v0,10
    syscall

# push.p $f12  Solução trivial
PUSHPT: ori $at,$zero,7
    slt $at,$gp,$at
    bne $at,$zero,PULAAT
    jal ERROR
PULAAT: addi $gp,$gp,1
    addi $sp,$sp,-32
    swc1 $f0,0($sp)
    swc1 $f1,4($sp)
    swc1 $f2,8($sp)
    swc1 $f3,12($sp)
    swc1 $f4,16($sp)
    swc1 $f5,20($sp)
    swc1 $f6,24($sp)
    swc1 $f7,28($sp)
    sll $at,$gp,2
    add $at,$sp,$at
    swc1 $f12,0($at)
    lwc1 $f0,0($sp)
    lwc1 $f1,4($sp)
    lwc1 $f2,8($sp)
    lwc1 $f3,12($sp)
    lwc1 $f4,16($sp)
    lwc1 $f5,20($sp)
    lwc1 $f6,24($sp)
    lwc1 $f7,28($sp)
    addi $sp,$sp,32

    jr $ra

# pop.p $f12  Solução trivial
POPPT:  slt $at,$gp,$zero
    beq $at,$zero,PULABT
    jal ERROR
PULABT: addi $sp,$sp,-32
    swc1 $f0,0($sp)
    swc1 $f1,4($sp)
    swc1 $f2,8($sp)
    swc1 $f3,12($sp)
    swc1 $f4,16($sp)
```

```
    swc1 $f5,20($sp)
    swc1 $f6,24($sp)
    swc1 $f7,28($sp)
    sll $at,$gp,2
    add $at,$sp,$at
    lwc1 $f12,0($at)
    addi $gp,$gp,-1
    lwc1 $f0,0($sp)
    lwc1 $f1,4($sp)
    lwc1 $f2,8($sp)
    lwc1 $f3,12($sp)
    lwc1 $f4,16($sp)
    lwc1 $f5,20($sp)
    lwc1 $f6,24($sp)
    lwc1 $f7,28($sp)
    addi $sp,$sp,32

    jr $ra


# push.p  $f12 Solução direta
PUSHPD: ori $at,$zero,7
    slt $at,$gp,$at
    bne $at,$zero,PULAAD
    jal ERROR
PULAAD: addi $gp,$gp,1
    addi $at,$zero,0
    beq $gp,$at,Fzero
    addi $at,$at,1
    beq $gp,$at,Fum
    addi $at,$at,1
    beq $gp,$at,Fdois
    addi $at,$at,1
    beq $gp,$at,Ftres
    addi $at,$at,1
    beq $gp,$at,Fquatro
    addi $at,$at,1
    beq $gp,$at,Fcinco
    addi $at,$at,1
    beq $gp,$at,Fseis
Fsete:  mov.s $f7,$f12
    j FIM
Fseis:  mov.s $f6,$f12
    j FIM
Fcinco: mov.s $f5,$f12
    j FIM
Fquatro: mov.s $f4,$f12
    j FIM
Ftres:  mov.s $f3,$f12
    j FIM
Fdois:  mov.s $f2,$f12
    j FIM
Fum:    mov.s $f1,$f12
    j FIM
Fzero:  mov.s $f0,$f12

FIM:    jr $ra
```

```
# pop.p $f12 Solução direta
POPPD:  slt $at,$gp,$zero
    beq $at,$zero,PULABD
    jal ERROR
PULABD: addi $at,$zero,0
    beq $gp,$at,FzeroD
    addi $at,$at,1
    beq $gp,$at,FumD
    addi $at,$at,1
    beq $gp,$at,FdoisD
    addi $at,$at,1
    beq $gp,$at,FtresD
    addi $at,$at,1
    beq $gp,$at,FquatroD
    addi $at,$at,1
    beq $gp,$at,FcincoD
    addi $at,$at,1
    beq $gp,$at,FseisD
FseteD: mov.s $f12,$f7
    j FIMD
FseisD: mov.s $f12,$f6
    j FIMD
FcincoD: mov.s $f12,$f5
    j FIMD
FquatroD: mov.s $f12,$f4
    j FIMD
FtresD: mov.s $f12,$f3
    j FIMD
FdoisD: mov.s $f12,$f2
    j FIMD
FumD:   mov.s $f12,$f1
    j FIMD
FzeroD: mov.s $f12,$f0
FIMD:   addi $gp,$gp,-1
    jr $ra



# push.p $f12  Solução mais eficiente?
PUSHP: ori $at,$zero,7
    slt $at,$gp,$at
    bne $at,$zero,PULAA
    jal ERROR
PULAA:  addi $gp,$gp,1  # atualiza o ponteiro
    la $at,0x43006006
    addi $sp,$sp,-4  # preciso de 2 registradores:(
    sw $t0,0($sp)
    la $t0,XA
    lw $at,0($t0)
    addi $at,$at,32 # incrementa o registrador na instrução
    sw $at,0($t0)
XA: mov.s $f0,$f12  # esta instrução vai ser modificada
    lw $t0,0($sp)
    addi $sp,$sp,4
    jr $ra
```

```
# pop.p $f12    Solução mais eficiente?
POPP:   slt $at,$gp,$zero
    beq $at,$zero,PULAB
    jal ERROR
PULAB:  addi $gp,$gp,-1 #atualiza ponteiro
    addi $sp,$sp,-4
    sw $t0,0($sp)
    la $t0,XB
    lw $at,0($t0)
    addi $at,$at,-32 # decrementa o registrador na instrução
    sw $at,0($t0)
XB: mov.s $f12,$f0  # esta instrução vai ser modificada
    lw $t0,0($sp)
    addi $sp,$sp,4

    jr $ra


# add.p
ADDP:   addi $sp,$sp,-4
    sw $ra,0($sp)
    #pop.d $f11
    jal POPPD
    mov.s $f11,$f12
    #pop.d $f12
    jal POPPD
    add.s $f12,$f12,$f11
    #push.d $f12
    jal PUSHPD

    lw $ra,0($sp)
    addi $sp,$sp,4
    jr $ra

# sqrt.p
SQRTP:  addi $sp,$sp,-4
    sw $ra,0($sp)
    #pop.d $f12
    jal POPPT
    sqrt.s $f12,$f12
    #push.d $f12
    jal PUSHPT

    lw $ra,0($sp)
    addi $sp,$sp,4
    jr $ra


ERROR:  addi $t0,$ra,-8 # O erro aconteceu na instrução anterior ao jal
    la $a0,L1
    li $v0,4
    syscall

    move $a0,$t0
    li $v0,1
    syscall
```

```
la $a0,L2
li $v0,4
syscall

lw $a0,0($t0)
li $v0,1
syscall

li $v0,10
syscall
```