Listão de Exercícios
Arquitetura de Computadores
Lista de Exercícios

1

Explique como funciona cada um dos modos de endereçamento abaixo e indique qual deles seria mais indicado para o acesso a um vetor na memória.

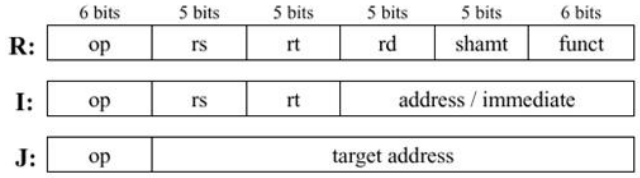
a) Registrador
b) Imediato
c) Deslocamento
d) Direto ou absoluto

2

O que é frame de pilha e para que finalidade ele é utilizado? Cite pelo menos dois propósitos do frame de pilha.

3

O MIPS possui três tipos de instruções: R, I e J, com a codificação mostrada abaixo:



Para cada uma das instruções abaixo, dado também seu código em hexadecimal, classifique quanto ao formato (R, I ou J), quanto à classe da instrução (Aritmética, lógica, transferência de dados, desvio condicional ou desvio incondicional), e quanto ao modo de endereçamento (registrador, imediato, deslocamento, relativo ao PC). Responda também qual é o opcode de cada instrução.

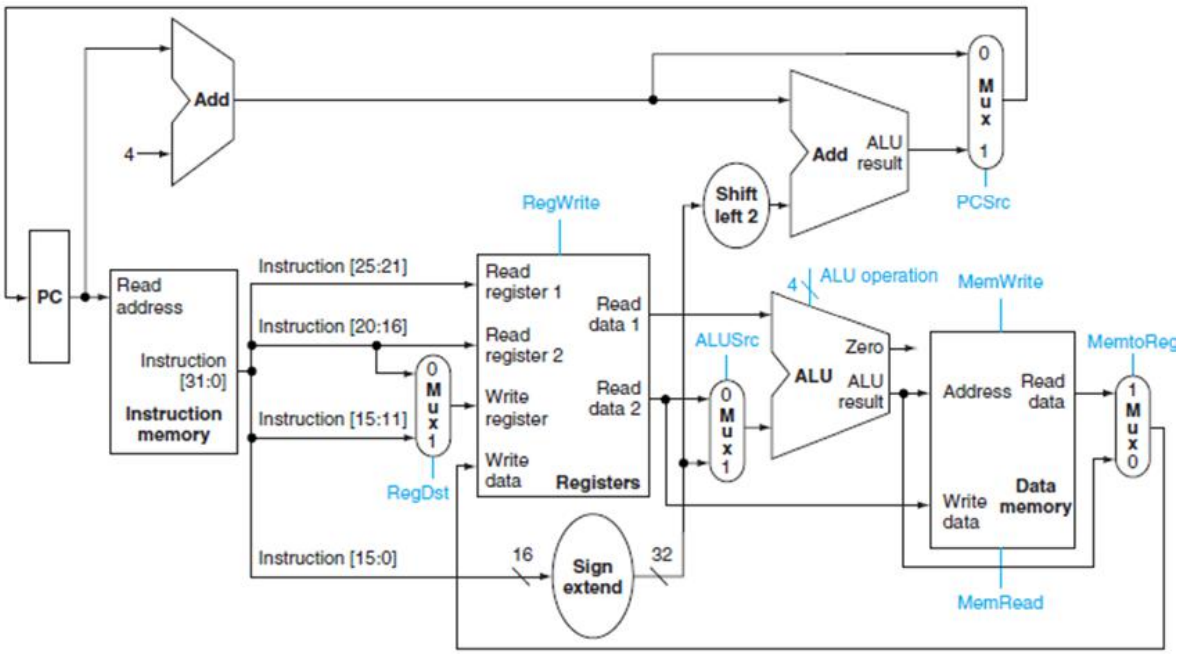
Instruções	Formato	Classe da Instrução	Modo de Endereçamento	Opcode
0x3c011001 lui $1,4097				
0x34240064 ori $4,$1,100				
0x0c100020 jal 0x00400080				
0x24020008 addiu $2,$0,8				
0x00854020 add $8,$4,$5				
0x240a000a addi $10,$0,10				
0x10880006 beq $4,$8,6				
0x8ca90000 lw $9,0($5)				
0xaca80000 sw $8,0($5)				
0x03e00008 jr $31				

4

Um sistema foi construído com dois processadores de arquiteturas diferentes compartilhando um mesmo chip de memória flash. Porém, quando o processador A salva na memória o número de 32 bits 0x12345678, o processador B lê a mesma posição de memória e obtém o número 0x78563412. Por que isso aconteceu?

5

No caminho de dados abaixo, explique o que deve ser feito para se adicionar a instruções ADDI. Quais sinais de controle mudam em relação à instrução ADD? Algum sinal novo deve ser adicionado?



6

Considerando o caminho de dados da questão 1, responda: (a) o que aconteceria se o sinal *MemtoReg* ficasse “travado” em “0” durante uma instrução **LW $s0,1200($zero)**? (b) No caso de **ADD $s0,$s1,$s1**, se o sinal *ALUSrc* ficar “travado” em “1”, o que acontecerá? (c) Que valor numérico em hexa será salvo em $s0 no item (b) considerando que $s1 tem zero?

7

O que é e quando é usado o flag “Zero” gerado pela ALU?

8

Compare as implementações monociclo (caminho de dados da questão 1) e multiciclo, citando as diferenças e as vantagens.

9

Abaixo está apresentado um trecho de código que implementa a soma de três números (D=A+B+C) em uma arquitetura tipo registrador-registrador (load-store):

```
LOAD R1,A
LOAD R2,B
ADD R1,R1,R2
LOAD R2,C
ADD R1,R1,R2
STORE R1,D
```

Implemente essa mesma lógica (que efetue D=A+B+C) para:

(a) Um processador com conjunto de instruções tipo pilha
(b) Um processador com conjunto de instruções tipo acumulador
(c) Um processador com conjunto de instruções tipo registrador-memória

10

Abaixo está a implementação em C e em assembly do MIPS para a função **tak**, seguindo-se as convenções de chamada de procedimento. Responda as questões abaixo:

a) Para que servem as instruções das linhas 2 e 3?
b) Qual instrução determina o frame de chamada de procedimento?
c) Há um erro nesse programa que impede o correto funcionamento da pilha. Aponte esse erro.
d) Por que os registradores $s0, $s1, $s2 e $s3 precisam ser salvos na pilha?
e) Explique o que faz a instrução da linha 38
f) Escreva como ficaria em assembly a chamada dessa função para x=15,y=20 e z=25. Após a chamada, salve o resultado em $s5.

```
int tak (int x, int y, int z)
{
    if (y < x)
        return 1 + tak (tak (x - 1, y, z),
                        tak (y - 1, z, x),
                        tak (z - 1, x, y));
    else
        return z;
}
```

```
1   tak:
2               subu $sp, $sp, 40
3               sw $ra, 32($sp)
4               sw $s0, 16($sp)     # x
5               sw $s1, 20($sp)     # y
6               sw $s2, 24($sp)     # z
7               sw $s3, 28($sp)     # temporario
8               move $s0, $a0
9               move $s1, $a1
10              move $s2, $a2
11              move $v0, $s2
12              bge $s1, $s0, L1    # if (y < x)
13              addiu $a0, $s0, -1
14              move $a1, $s1
15              move $a2, $s2
16              jal tak             # tak (x - 1, y, z)
17              move $s3, $v0
18              addiu $a0, $s1, -1
19              move $a1, $s2
20              move $a2, $s0
21              jal tak             # tak (y - 1, z, x)
22              addiu $a0, $s2, -1
23              move $a1, $s0
24              move $a2, $s1
25              move $s0, $v0
26              jal tak             # tak (z - 1, x, y)
27              move $a0, $s3
28              move $a1, $s0
29              move $a2, $v0
30              jal tak             # tak (tak(...), tak(...), tak(...))
31              addiu $v0, $v0, 1
32  L1:         lw $ra, 32($sp)
33              lw $s0, 16($sp)
34              lw $s1, 20($sp)
35              lw $s2, 24($sp)
36              lw $s3, 28($sp)
37              addiu $sp, $sp, 32
38              jr $ra
```

11

Escreva um programa em linguagem de alto nível (Ex.: C) que compilado geraria o programa abaixo:

```
1    .data
2 A:      .word    1234
3 B:      .word    5678
4    .text
5    la $a0,A
6    la $a1,B
7    jal swp
8          li $v0,10
9          syscall
10   swp:     lw $t0,0($a0)
11         lw $t1,0($a1)
12         sw $t0,0($a1)
13         sw $t1,0($a0)
14         jr $ra
```

12

No caminho de dados **mostrado na Questão 5**, construa uma tabela para os sinais binários de controle indicados, para as instruções LW, ADD, ADDI e BEQ (considerando desvio tomado). Para o sinal “ALU operation”, indique apenas que operação a ALU deve realizar. Para os demais sinais, os valores possíveis são: “0”, “1”, ou “X” (don’t care).

13

Considerando o caminho de dados **mostrado na Questão 5**, descreva o que deve ser modificado no desenho para implementarmos as instruções JAL e JR.

14

(a) Ligue cada a instrução assembly abaixo com seu modo de endereçamento correto.
(b) Explique o que faz cada uma delas. Exemplo: add $t0,$t1,$t2 faz: $t0 ← $t1 + $t2

add $t0,$t1,$t2

addi $t1,$t2,100

sw $t0,100($t2)

lw $t0,(1000)

add $t0,($t1+$t2)

- **Indexado**
- **Direto ou absoluto**
- **Imediato**
- **Base ou deslocamento**
- **Registrador**

15

Faça um esboço de alto nível para o caminho de dados Multiciclo, atendendo aos seguintes requisitos:

1. Uma única unidade de memória para instruções e dados
2. Uma única ALU em vez de uma ALU e dois somadores
3. Registradores intermediários entre as etapas

16

Dada a seguinte tabela de bits representando uma instrução de máquina:

op	rs	rt	rd	shamt	funct
0	8	9	10	0	34

Responda:

a) Qual é a instrução *assembly* do MIPS que resultaria no código de máquina dado acima? (Dica: 34_{10} = 0x22)

b) A instrução acima é Tipo R, Tipo I, ou Tipo J? Justifique.

c) Em geral, qual é a principal diferença de propósito entre as instruções tipo R, I e J? (Para que cada uma é usada tipicamente?)

d) Dadas as duas representações binárias de instruções de máquina abaixo:

1010-1110-0000-1011-0000-0000-0000-0100

1000-1101-0000-1000-0000-0000-0100-0000

Qual instrução *assembly* cada uma representa? (ex.: lw, add, sw). Represente a instrução completa, com seus operandos. A qual tipo elas pertencem? (R, I, ou J). Justifique.

17

Responda os itens (a) e (b) a seguir.

A função **min2** abaixo escrita em *assembly* do MIPS retorna o menor valor entre os dois parâmetros passados pelos registradores **$a0** e **$a1**:

```
min2:   slt $t0,$a0,$a1
        move $v0, $a1
        beqz $t0,saida
        move $v0, $a0
saida:  jr $ra
```

(a) Escreva uma função **min3**, com três parâmetros, que retorne o menor dos três parâmetros passados. A função **min3** deve chamar a função **min2** para obter o seu resultado. Você deverá fazer algo do tipo: **min2(x, min2(y,z))** para obter o menor entre **x**, **y** e **z**. Note que isto não é recursividade.

(b) Faça o programa principal chamando **min3(x,y,z)** sendo **x**, **y** e **z** três valores constantes quaisquer.

(c) Por que a função **min2** não precisa usar a pilha?

18

Na implementação do MIPS, responda:
(a) o que é e quando é usado o flag “Zero” gerado pela ALU?
(b) Qual a diferença de sinais de controle entre as operações ADD e SUB?

19

Você está viajando por uma galáxia, muito, muito distante, em busca do maior computador já desenvolvido, o *Deep Thought*, para obter a resposta para a pergunta fundamental da vida, do universo e tudo mais. Por sorte, o *Deep Thought* implementa o mesmo conjunto de instruções do MIPS de 32 bits. A resposta está escrita no endereço apontado por **$sp**, deslocado do *label* **resposta** mais 42. Você precisa ler o valor contido naquela posição de memória e armazenar em **$v0**. Porém, para seu desespero, você não dispõe de um *assembler*. Você olha na sua mochila e encontra apenas um livro escrito na capa “Don´t Panic”. Então você terá que escrever em linguagem de máquina as instruções para obter a resposta para a grande pergunta fundamental da vida, do universo e tudo mais, processada pelo “Deep Thought”. Finalmente, após 7,5 milhões de anos de processamento, a resposta está lá, no endereço **resposta+42**. O *label* resposta aponta para o endereço **0x0F000004**. Apresente as instruções em hexadecimal.

20

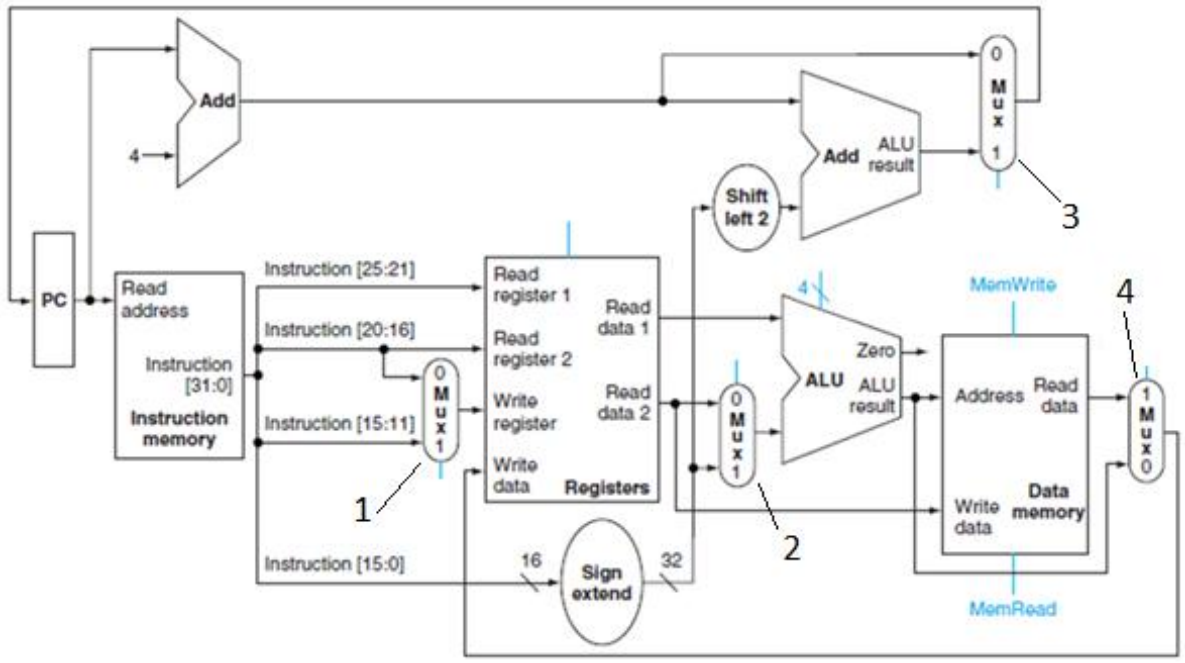
Escreva uma função em *assembly* do MIPS para calcular o n-ésimo termo da sequência de Fibonacci, dada por: F(1) = 1, F(2) = 1, F(n) = F(n-1) + F(n-2). Como ficaria a chamada da função para n=42, armazenando o resultado no endereço apontado por $s0 + 168?

21

Como é feita a passagem de parâmetros para uma função em *assembly* segundo as convenções para chamada de procedimento? E o retorno de um resultado? Crie um exemplo de função qualquer com pelo menos dois argumentos e mostre como seria a chamada dessa função no programa principal, atribuindo o resultado para o registrador $s0.

22

No caminho de dados abaixo, explique qual é a finalidade de cada um dos quatro MUXES indicados. Para cada um, utilize na sua explicação um exemplo de instrução que requer que o MUX seja controlado com 0 e outro exemplo controlado com 1.



23

Explique tudo que é necessário alterar no projeto do processador tal como mostrado na figura da acima para se implementar a instrução *Jump and Link* (JAL).

24

Na programação estruturada em assembly, qual é o problema resolvido pela instrução JAL? E o que devemos fazer para permitir funções com chamadas aninhadas (uma função chamada dentro de outra função)?

25

Uma determinada função (de nome calcula) escrita em assembly requer três parâmetros do tipo inteiro e retorna um valor também inteiro. Como fica a chamada dessa função para três valores constantes, por exemplo 18, 12 e 6, atribuindo o resultado no registrador $s0? O que se pede é a compilação da chamada da função, que em C ficaria x=calcula(18,12,6), considerando que x é o registrador $s0.

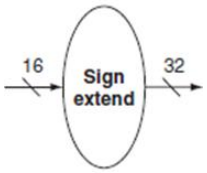
26

A que instruções MIPS os seguintes códigos hexadecimais correspondem? Consulte a tabela de *opcodes* no verso. Caso não conste na tabela, responda com “instrução ilegal”. Nos outros casos escreva a instrução completa, com seus operandos.

a) 0x20202020
b) 0x00E70018
c) 0x13D300C8

27

O que é e para que serve o elemento **sign extend** presente no caminho de dados do MIPS? Explique dando um exemplo de uma instrução que o utiliza.



28

Implemente em assembly do MIPS uma funçao recursiva que calcula o fatorial de um número, utilizando as instruções MULT e MFLO. Faça a chamada da função no programa principal para o fatorial de 9 e armazene o resultado na posição dada por $t0, deslocado de 4 bytes.

29

A pseudo-instrução:

Tem o efeito de gravar no registardor $s0 o número hexadecimal de 32 bits 0x1234aaaa.
Pede-se:
(a) Qual é o modo de endereçamento dessa instrução? Explique.
(b) Gere o código necessário, em linguagem de máquina, para gerar essa pseudo-instrução no MIPS. Apresente o código em hexadecimal.

30

Por que a arquitetura multiciclo é mais rápida que a monociclo, embora o CPI médio da multiciclo seja bem maior, em média maior que 3, enquanto na monociclo CPI=1?

31

(a) Escreva uma sub-rotina em assembly do MIPS que execute o equivalente ao código de alto nível abaixo. Você deve saber como enviar argumentos para a sub-rotina e onde o resultado deve estar para enviá-lo de volta. Dica: não é necessário utilizar a instrução MULT.

```
int elvis(int a, int b)

{

    return a * 2 + b;

}
```

(b) Escreva as instruções em assembly MIPS para chamar sua sub-rotina, com o seguinte código C equivalente:

```
s0 = elvis (31, 29);
```

(c) Se a instrução jal que você usou acima estiver no local de memória 0x40001048, qual registrador será alterado e qual valor será colocado nele quando essa instrução jal for executada?

32

Implemente em MIPS o código para realizar a seguinte operação representada em linguagem C:

b = 25 | a;

Assuma que a variável a corresponde ao registrador $t0 e a variável b corresponde ao registrador $t1. Apresente o resultado em código de máquina, em hexadecimal.

Dados:

Instruções:

Mnemônico	Significado	Opcode	Funct
sll	Logical Shift Left	0x00	0x00
srl	Logical Shift Right (0-extended)	0x00	0x02
sra	Arithmetic Shift Right (sign-extended)	0x00	0x03
jr	Jump to Address in Register	0x00	0x08
add	Add	0x00	0x20
addu	Add Unsigned	0x00	0x21
sub	Subtract	0x00	0x22
subu	Unsigned Subtract	0x00	0x23
and	Bitwise AND	0x00	0x24
or	Bitwise OR	0x00	0x25
xor	Bitwise XOR (Exclusive-OR)	0x00	0x26
nor	Bitwise NOR (NOT-OR)	0x00	0x27
slt	Set to 1 if Less Than	0x00	0x2A
sltu	Set to 1 if Less Than Unsigned	0x00	0x2B
j	Jump to Address	0x02	NA
jal	Jump and Link	0x03	NA
beq	Branch if Equal	0x04	NA
bne	Branch if Not Equal	0x05	NA
addi	Add Immediate	0x08	NA
addiu	Add Unsigned Immediate	0x09	NA
slti	Set to 1 if Less Than Immediate	0x0A	NA
sltiu	Set to 1 if Less Than Unsigned Immedia	0x0B	NA
andi	Bitwise AND Immediate	0x0C	NA
ori	Bitwise OR Immediate	0x0D	NA
lui	Load Upper Immediate	0x0F	NA
lw	Load Word	0x23	NA
lbu	Load Byte Unsigned	0x24	NA
lhu	Load Halfword Unsigned	0x25	NA
sb	Store Byte	0x28	NA
sh	Store Halfword	0x29	NA
sw	Store Word	0x2B	NA

Registradores:

Número	Nome	Comentários
$0	$zero, $r0	Vale sempre zero
$1	$at	Reservedo para o assembler
$2, $3	$v0, $v1	Primeiro e segundo valores de retorno
$4, ..., $7	$a0, ..., $a3	Primeiros quatro argumentos para funções
$8, ..., $15	$t0, ..., $t7	Registradores temporários
$16, ..., $23	$s0, ..., $s7	Registradores salvos pelas funções
$24, $25	$t8, $t9	Mais registradores temporários
$26, $27	$k0, $k1	Reservado para o kernel (SO)
$28	$gp	Global pointer
$29	$sp	Stack pointer
$30	$fp	Frame pointer
$31	$ra	Endereço de retorno