Exercícios Teórico-Práticos

1. Quais são os 3 blocos fundamentais de um sistema computacional?

CPU, Memória e unidades entrada/saída

2. Quais são os 3 blocos (principais) funcionais que integram um CPU?

Conjunto de registos, unidades funcionais (aritmética, lógicas, de vírgula flutuante) e unidade de controlo

3. Qual a função do registo Program counter?

Armazena o endereço de memória onde está alocado o código da próxima instrução a executar.

4. Quais os passos + importantes em que se decompõe a execução de 1 inst. no CPU?

Instruction Fetch → Instruction Decode → Operand fetch → Execute → Store result

5. Descreva de forma sucinta a função de um compilador.

O compilador traduz um programa escrito numa linguagem de alto nível (ex. C++) para linguagem de máquina (Assembly para o MIPS)

6. Descreva de forma sucinta a função de um assembler.

Permite converter as várias instruções do programa numa "versão" binária para que o CPU tenha toda a informação necessária para os executar.

7. Quantos registos internos de uso geral tem o MIPS?

32 registos

8. Qual a dimensão, em bits, que cada 1 dos registos internos do MIPS pode armazenar?

32 bits

9. Qual a sintaxe, em Assembly, de 1 instrução aritmética no MIPS?

A instrução é "identificada" pela sua mnemónica, seguida do registo onde armazena o resultado e os registos que contêm os operandos.

10. O que distingue a instrução SRL da inst. SRA do MIPS?

SRL → Shift right logical ; i.e, é deslocamento à direita lógico, não considera sinal, a variável é unsigned.

SRA → Shift right arithmetic ; i.e, é deslocamento à direita aritmético, tem em conta o sinal, a variável é signed

11. Se $5 = 0x81354AB3 = 1000 0001 0011 0101 0100 1010 1011 0011

a) srl $3, $5, 1 $3 = 0100 0000 1001 1010 1010 0101 0101 1001
= 0x409AA559

b) sra $4, $5, 1 $4 = 1100 0000 1001 1010 1010 0101 0101 1001
= 0xC09AA559

12. a) O que é uma system call? Função específica que invoca o sistema operacional para que este faça algo que seja demasiado abstrato para o usuário.

b) No MIPS, qual o registo usado para identificar a system call a executar?
$v0

c) Qual o registo ou registos usados para passar argumentos para os systems calls?
$a0 → passa inteiro, string, chars.
↳ binário, decimal ou hexadecimal

$a1 → usado em read_string, passa o seu comprimento.

$f12 → passa float e double

d) Qual o registo usado para obter o resultado devolvido por uma system call nos casos em que isso se aplica.
$f0 → devolve float e double

$v0 → restantes casos

13. Em AC, defina o conceito de endereço?

Endereço é a referência a uma parte específica da memória que armazena algo (dados, instruções, etc)

14. O que é o espaço de endereçamento de um processador?

Gama total de endereços que o CPU consegue referenciar.

15. Como se organiza internamente um processador? Quais são os blocos fundamentais da secção de dados? Para que serve a unidade de controlo?

O processador possui um conjunto de registos, unidades funcionais (ex. ALU) e uma unidade de controlo. Os blocos fundamentais são os multiplexers, a ALU e os registos internos. A unidade de controlo emite sinais que permitem coordenar os elementos do datapath.

16. Qual é o conceito fundamental por detrás do modelo de arquitetura "stored-program"? Coexistência de vários tipos de informação armazenados na memória: código de um programa; a compilação do mesmo, um compilador; dados, etc.

17. Como se codifica 1 instrução? Que informação fundamental deverá ter o seu código?
A codificação de uma instrução depende do seu formato: existem instruções do tipo R, tipo I e tipo J.

Codificação tipo R:

opCode	rs	rt	rd	shamount	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Codificação tipo I:

opcode	rs	rt	offset
6 bits	5 bits	5 bits	16 bits

Codificação tipo J:

opCode	Endereço-alvo
6 bits	26 bits

Todas as instruções terão 'opcode' e registos (alguns operandos, outros de destino); há algumas variações nos campos de codificação do tipo dom tipo das instruções.

18. Descreva o conceito de ISA. → Instruction Set Architecture; descreve tudo o que o programador necessita de saber para programar corretamente um processador, em assembly. Descreve a funcionalidade do processador independentemente da sua implementação hardware.

19. Quantas e quais são as classes de instruções que agrupam os diferentes instruções de uma dada arquitetura?

Há 3 classes de instruções: processamento, transferência e controlo.

Processamento → operações lógicas e aritméticas sobre o conteúdo de registos

Transferência → Leitura e armazenamento entre conteúdo de registos internos ou entre registos internos e memória externa.

controlo (de fluxo) → Alteram o "caminho" do programa (ex: branchs)

20. O que caracteriza e distingue as arquiteturas do tipo "register-memory" e "load-store"? De que tipo é a arquitetura do MIPS?

Register-memory: os operandos podem estar armazenados nos registos internos do CPU e/ou memória.

Load-store: os operandos podem apenas residir nos registos internos do CPU de uso geral.

MIPS → Load-store.

21. O ciclo de execução de uma instrução é composto por uma sequência ordenada de operações. Quantas e quais são essas operações?

5 operações: Instruction fetch → Instruction decode → Operand fetch → Execute → store results

22. Como se designa o barramento que permite identificar, na memória, a origem/destino da informação transferida?

Address Bus

23. Qual a finalidade do barramento normalmente designado por Data Bus?

Transferência dos dados/informação.

24. Os processadores da arquitetura hipotética ZWYZ possuem 4 registos internos e todas as instruções são codificados em 24 bits. Num dos formatos de codificação existem 5 campos: OpCode → 5 bits ; 3 campos para identificar registos internos em operações aritméticos e lógicos e 1 campo para codificar valores constantes imediatos em complemento para 2. Qual a gama de representação destas constantes.

Opcode	rg1	rg2	rg3	Const.
5 bits	n bits	n bits	n bits	X bits

(Por analogia MIPS)

rs	rt	rd
5 bits	5 bits	5 bits

$2^5 = 32$ registos internos

$ZWYZ$

$2^n = 4$

$\Leftrightarrow n = 2$

$5 + 3 \times 2 + X = 24 \Leftrightarrow X = 13$ bits para constante

$\left[-2^{13-1} ; 2^{13-1} - 1 \right]$

25. A arquitetura hipotética ZPTZ tem 1 barramento de endereços de 32 bits e 1 barramento de dados de 16 bits. Se a memória desta arquitetura for bit-addressable:

a) Qual a dimensão do espaço de endereçamento desta arquitetura?

2^{32} → posições de memória

b) Qual a dimensão máxima da memória suportada por esta arquitetura (em bytes)?

$\frac{2^{32}}{8} = 2^{29} = 512\ MB$

26. Considerando a organização de memória do tipo word-addressable em que a dimensão da word é 32 bits. Sendo o espaço de endereçamento do processador de 24 bits, qual a dimensão máxima de memória que este sistema pode acomodar (em bytes)?

$\frac{2^{24} \times 32}{8} = \frac{2^{24} \times 2^5}{2^3} = 2^{26} = 64\ MB$

27. Relativamente ao MIPS

a) Com quantos bits são codificadas as instruções?
32 bits

b) O que diferencia o registo $0 dos restantes de uso geral?
$0 contém sempre o valor '0'. Este registo não pode ser alterado

c) Qual o endereço do registo interno a que corresponde $ra?
$31

28. No MIPS, um dos formatos de codificação de instruções é designado por R: a) quais os campos deste formato?

opcode rs rt rd shamt funct

b) qual o significado de cada campo?

opcode → identifica o tipo de operação.

rs → registo que contém o 1º operando.

rt → " " " " 2º operando.

rd → " " irá armazenar o resultado produzido.

shamt → usado em instruções de deslocamento.

funct → código da operação que a ALU irá executar.

c) qual o valor do campo opcode?

000000

d) o que faz a instrução: 0x00000000?

Nada; é a instrução NOP (no operation).

29. O símbolo ">>" significa deslocamento à direita e é traduzido por SRL ou SRA (no MIPS). Em que casos é que o compilador gera um SRL e quando é que gera um SRA?

SRL → quando a variável é do tipo unsigned

SRA → quando a variável é do tipo signed

30. Qual a instrução nativa do MIPS em que é traduzida a instrução virtual "move $4, $15"?

or $4, $0, $15

31. Determine o cód. máquina das seguintes instruções

a) xor $5, $13, $24

000000 01101 11000 00101 00000 100110

b) sub $25, $14, $8

000 000 01110 01000 11110 00000 100010

c) sll $3, $9, 7

000000 00000 01001 00011 00111 000000

d) sra $18, $9, 8

000000 00000 01001 10010 01000 000011

32. x: $t2
y: $t5

$Y = -3 * x + 5$

```
add $t3, $t2, $t2  # $t3 = 2x
add $t3, $t3, $t2  # $t3 = 3x
sub $t5, 5, $t3    # $t5 = 5 - 3*x
```

33. Traduzir do C para assembly

```
int a, s, c;                    // a: $t0, s: $t1, c: $t2
unsigned int x, y, z;           // x: $a0, y: $a1, z: $a2

z = x >> 2 + y;
c = a >> s - 2 * s
```

```
srl $a2, $a0, 2
add $a2, $a2, $a1
sra $t0, $t0, s  # $t0 = a >> s
add $t1, $t1, $t1  # $t1 = 2*s
sub $t2, $t0, $t1
```

34. Assembly para C

a) add h, i, j # h = i + j

b) addi j, j, 1 # j++
add h, g, j # h = g + j

35. g=1, k=2, i=3, j=4

a) k = 3+4 = 7

b) j++ = 5
k = 1+5 = 6

36. slt - set if less than

Se o valor do 1º registo for menor que o do 2º registo, coloca o valor lógico '1' noutro registo. Se a condição não se verificar, coloca '0'.

37. a) slt $1, $3, $7 $3 = 5 $7 = 23
$1 = 1

b) $3 = 0xFE $7 = 0x913D45FC
$1 = 0

38. Registo $0

39. Decompor em nativos

a) slt $15, $3, exit
slt $1, $15, $3
bne $1, $0, exit

b) sle $6, $9, exit
slt $1, $9, $6
beq $1, $0, exit

c) sgt $5, 0xA3, exit
ori $1, $0, 0xA3
slt $2, $1, $5 # 0xA3 < $5
bne $2, $0, exit

d) sge $10, 0x57, exit
slti $1, $10, 0x57
beq $1, $0, exit

e) slt $19, 0x39, exit
slti $1, $19, 0x39
bne $1, $0, exit

f) sle $23, 0x16, exit
ori $1, $0, 0x16
slt $2, $1, $23
beq $2, $0, exit

40. Diferenças entre "while(...){...}" e "do{...} while(...);"

do{...} while(...) → teste condicional é efetuado no fim do ciclo, o que implica a execução do corpo do código pelo menos 1 vez. No ciclo while(...) o teste condicional é feito à "cabeça", no início do ciclo.

41. C para assembly

a: $4 b: $7 c: $13

a)
```
if(a>b && b!=0)
    c = b<<2;
else
    c=(a&b)^(a|b)
```

```
ble $4, $7, else
beq $7, $0, else
sll $13, $7, 2
j endif
else: xor $13, $4, $7
endif: (...)
```

b)
```
if(a>3 || b<=c)
    c=c-(a+b)
else
    c=c+(a-b)
```

```
bgt $4, 3, L1
bgt $7, $13, else
L1: add $4, $4, $7
    sub $13, $13, $4
    j endif
else: sub $4, $4, 5
      add $13, $13, $4
```

42. Endereçamento indireto por registo

43. $3 → Registo que armazenará a word de 32 bits lida da memória.

$5 → Registo de endereçamento indireto, contém parte do endereço para aceder à memória (endereço base)

0x24 → deslocamento que será somado com o endereço base. Esta constante terá que ser estendida para 32 bits com sinal.

44. Formato I

Opcode → código da operação (6 bits)

rs → contém o endereço base (5 bits)

rt → registo onde se armazena word lida ou que contém a word a escrever na memória (LW, SW respetivamente) (5 bits)

offset → valor de 16 bits que será estendido para 32. Soma-se com o conteúdo de rs.