Exercícios Teórico-Práticos

1) CPU, Memória, I/O

2) ALU, Unidade de Controlo, Registos

3) Armazenar o endereço da próxima instrução

4) Fetch Instruction, Instruction Decode, Operand Fetch, Execute, Store Result

5) Traduzir de uma linguagem high-level para assembly

6) Traduzir de assembly para código máquina

7) 32 registos

8) 32 bits

9) Em formato R: func Rd, Rs, Rt

10) SRL: coloca um zero no bit mais significativo
SRA: clona o bit mais significativo (mantém o sinal)

11) $5 = 1000 0001 0011 0101 0100 1010 1011 0011
(8 1 3 5 4 A B 3)

a) $3 = 0100 0000 1001 1010 1010 0101 0101 1001
= 0X409AA559

b) $4 = 1100 0000 1001 1010 1010 0101 0101 1001
= 0xC09AA559

12)

a) O método pelo qual um program pede um serviço ao kernel do sistema operativo

b) syscal

c) 5a
d) 2V

13) Um número único que identifica cada registo na memória

14) A gama total de endereços que o CPU consegue referenciar

15) • Registos internos, ALU, Unidade de Controlo
• Multiplexers
• Coordena os elementos da datapath, gerando sinais de controlo

16) Implica que na memória possa residir, ao mesmo tempo, informação de natureza variada.

17) É codificado sob a forma de um número em binário, que contem informacao sobre:

18) ISA - arquitetura de todas as operacoes/instrucoes que o processador pode executar

19) Processamento, Transferencia de informação, Controlo de fluxo de execução

20) Register-Memory: Operandos residem em registos internos do CPU
Load-Store: Operandos das instruções residem em registos internos do CPU de uso geral
↑
MIPS

21) Instruction fetch
" decode
Operand fetch
Execute
Store result

22) Address bus / Barramento de endereços

23) Transferência de informação

24)

OpCode	reg	reg	reg	const
5	2	2	2	13

4 registos internos
=
00
01
10
11
} só são necessário 2 bits para referênciar

logo reg = 2 bits

Instruções → 24 bits

24
- 5 → OpCode
- 6 → 3 registos
18 → 13 bits restantes para const

$2^{13} = 8,192$ → Gama representação de constantes

25)

a) 2^{32} endereços passiveis → 4 GB (bits) de memória

b) 4 Gigabits = 0,5 gigabytes ou 500 Megabytes

26) 2^{24} → Gama total de endereços

32 → Apenas são referenciados "de 32 em 32" bits

=

524,288 → se dividirmos por 8 → 65 536

↓

aprox. 65 Megabytes

27)

a) 32 bits

b) Tem sempre o valor 0x00000000 e apenas pode ser lido

c) $ra - $31

28

a.b.c)

op - 6 bits - sempre 0 no formato R
rs - 5 " - 1º operando
rt - 5 " - 2º "
rd - 5 " - registo destino
shamt - 5 " - shift amout (não utilizado em R)
funct - 6 " - operação a realizar

d)

NOP

29)

SRL - se variavel unsigned

SRA - " signed

30)

OR $15, $0, $4

31)

a) XOR $5, $13, $24

00000	01101	11000	00101	00000	100110
Opcode	$13	$24	$5	shamt	funct (XOR)

0x01B82826

b) sub $25, $14, $8

000000 01110 01000 11001 00000 100010
$14 $8 $25 sub

0x01C8C822

c) sll $3, $9, 7

000000 00000 01001 00011 00111 000000
$9 $3 7 sll

0x000919C0

d) sra $18, $9, 8

000000 00000 01001 10010 01000 000011
$9 $18 8 sra

0x00099203

32)

n - $t2 y = -3 x n + 5
y - $t5

n + 5 — addi $t2, $0, 5
$t1 = -3 — addi $t1, $0, -3
— mult $t1, $t2
mflo $t5

33)

sll $a0, $a0, 2 -> n = n >> 2
add $a2, $a0, $a1 -> z = n + y

sll $t0, $t0, 5 -> a = a >> 5
sll $t1, $t1, 1 -> b = 2 x b (=>> 1)
sub $t2, $t0, $t1 -> c = a - b

34)

a) `h = i + g ;`

b) `j++ ;`
`h += j ;` ou `h += ++j ;`

35)

a) h = 4

b) h = 7

ab) h = 10

36)

slt = "set less than"

Compara dois valores de dois registos, os resultados possiveis é 0 ou 1

Ex:
$t0 = 0 $t1 = 6 $t2 = 4

slt $t0, $t1, $t2

Isto comparar se o valor de $t1 < $t2, se sim $t0 = 1

37)

a) $1 = 1

b) $1 = 0

38)

$0

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

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

c)
```
ori $1, $0, 0xA43
slt $1, $1, $5
bne $1, $0, exit
```

d)
```
ori $1, 0x57 exit
slt $1, $10, $1
beq $1, $0, exit
```

e)
```
ori $1, $0, 0x39
slt $1, $19, $1
bne $1, $0, emit
```

f)
```
ori $1, $0, 0x16
slt $1, $1, $23
beq $1, $0, emit
```

40) A colocação da verificação/comparação irá ser deslocada do início para o fim do ciclo.

No início, a verificação terá de ser no negativo, de forma a que se a condição não se verificar o branch "salta por cima" do loop.

No fim, a verificação tem de ser na afirmativo de forma que se esta se cumprir o branch "salta" para o início do loop.

41)

a)
```
    ble $7, $5, else
    beqz $7, else
    slt $13, $7, 2
else:
    and $13, $4, $7
    or  $4, $4, $7
    xor $13, $13, $4
```

b)
```
    bgt $4, 3, if
    ble $7, $13, if
    subi $4, $4, 5
    add $13, $13, $4
    j end
if: add $4, $4, $7
    sub $13, $13, $4
end: ...
```

42) Indireto por registo

43) $3 - registo destino
$5 - registo origem
0x24 - offset

44) Formato I: opcode (6 bits) - operação rs (5 bits) - origem rd (5 bits) - destino
offset (16 bits) - offset

45) sw - guarda na memória 1 word
sb " " " 1 byte

46) lb - carrega um byte, replicando o 8º bit para manter o sinal
lbu - os 24 bits ms são colocados a 0

47) Gera uma exceção e termina a execução

48) $t1 = i
$t2 = K

a)
```
    li $t1, 5
    li $t2, 0
for: bge $t1, 20, end
    addi $t1, $t1, 1
    addi $t2, $t2, 5
    j for
end: jr $ra
```

b)
```
    li $t1, 100
    li $t2, 0
for: bltz $t1, end
    addi $t1, $t1, -1
    addi $t2, $t2, -2
    j for
end: jr $ra
```

c)
```
    addiu $t2, $t2, 0
for: addi $t2, $t2, 10
    j for
    jr $ra
```

d)
```
    li $t2, 0
    li $t1, 100
do: addi $t2, $t2, 5
    addi $t1, $t1, -1
    bgez $t1, do
    jr $ra
```

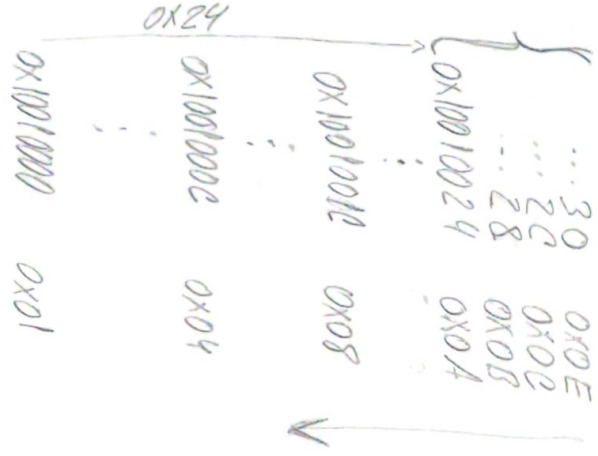
49)

lw $3, 0x24($5)

100011 00101 00011 0000 0000 0001 1000

50)

$3 = 0xDEDCOBOA



51)

a) lbu $3, 0xA3($5)

b) lb

52)

a) 10 bytes

b) 3 bytes

c) 12 bytes

d) 5 bytes

53

Endereço	Valor	
0x1001 0000	A	"AULAS5&6T\0"
	U	
	L	
	A	
	S	
	5	
	&	
	6	
	T	
0x0001 0009	\0	
0A	0x05	→ 5 = 0x05
0B	0x08	→ 8 = 0x08
0C	0x17	→ 23 = 0x17
0D	?	
0E	?	
0F	?	
10	0x05	5 = 0x05
	0x00	
	0x00	
	0x00	
14	0x08	8 = 0x08
	0x00	
	0x00	
	0x00	
24	0x17	23 = 0x17
	0x00	
	0x00	
	0x00	
34	??	space
	??	
	??	
	??	

54)

L1: 0x10010000

L2: 0x1001000A

L3: 0x10010010

L4: 0x10010034

55)

int b[25]

a) b[0]

b) lw $t1, 0x18 ($t0)

↑

6x4 = 24 = 0x18

56)

O label, que se somado ao PC atual, resulta no endereço alvo

57) O offset é multiplicado por 4 (visto que a memória é byte-addressable) e somado ao PC

58)

beq / bne - formato I

j - formato J

jr - formato J

59)

Campo offset e "shiftado" duas casas á esquerda e concatenado aos 4 msbs do PC

60)

- a = b[5]
- a = *(p + 5)

61)

f : $t0 A : $s0

g : $t1 B : $s1

h : $t2

i : $t3

j : $t4

a) f = g + h + B[2]

```
lw $t0, 8($s1)      f = B[2]
add $t0, $t0, $t1   f = f + g
add $t0, $t0, $t2   f = f + h
```

j = g - A[B[2]]

```
lw $t4, 8($s1)          lw $t4, 0($t4)
sll $t4, $t4, 2    ->   subu $t4, $t1, $t4
addu $t4, $t4, $s0
```

b)

- 3 e 5

c)

A[0] = 0x00000002 B[0] = 0xFFFFFFFE

A[1] = 0x22ED3400 B[1] = x00005002

A[2] = 0x00000001 B[2] = x00000002

62)

a = 2

b = 5

```
void troca (int *x, int *y) {
    int aux
    aux = *x
    x = *y
    y = &aux
}
```

63)

No $ra armazenado o endereço alvo, podendo assim realizar um salto para um endereço de 32 bits.

É um salto por endereçamento indireto por registo

64)

0x5A18F34C

0101

0x50000000

↕

0x5FFFFFFF

65)

0x5A180000

↕

0x5A18FFFF

66)

0xFFFFFFFF

↕

0x00000000