Beatriz Caroline Moggio

+27/1/8+

Teste 2 | Arquitetura e Organização de Computadores
Capítulo 2. A linguagem dos computadores
Prof. Dr. João Fabrício Filho

RA: 225604 Nome: Beatriz Caroline Moggio

Questões com o símbolo ♣ podem uma ou mais alternativas corretas. Outras questões possuem somente uma alternativa correta. Somente a folha de prova será corrigida, utilize as folhas extras como rascunhos.

Questão 1 ♣ [20%] Assinale a(s) alternativa(s) correta(s).

- Arquiteturas RISC possuem um grande número de instruções, com *hardware* mais complexo.
- Em arquiteturas CISC podemos operar dados que estão na memória, assim são necessários poucos registradores.
- No MIPS, há 32 registradores principais, dentre os quais 10 para valores temporários ($t0-$t9).
- Em uma arquitetura RISC, qualquer instrução pode referenciar a memória.
- É possível emular todas as instruções de uma arquitetura CISC em uma máquina RISC.
- No MIPS, qualquer instrução pode alterar diretamente o valor do registrador especial PC.
- A instrução add realiza operações entre registradores e constantes.
- A instrução sw armazena o valor de um registrador em um endereço de memória especificado.

8/8

Para as próximas questões, considere o trecho de código de alto nível abaixo:

```
int soma = 0;
soma += A[0];
soma += A[1];
A[1] = soma;
soma += A[2];
A[i] = soma;
```

O *assembly* correspondente para o respectivo código é apresentado abaixo, com algumas partes faltando, indicadas por alguma letra entre pares de "__" . Nesse *assembly* considera-se os pares de (registrador, variável) como correspondentes: ($s0, soma), ($s1, &A), ($s2, i). Além disso, considere cada valor no vetor A, bem como o tamanho das variáveis soma e i como de 32 bits.

```
1  add $s0, $zero, __A__
2  lw  $t0, __B__($s1)
3  add $s0, $s0, __C__
4  __D__ $t1, __E__($s1)
5  add __F__, $s0, $t1
6  sw  $s0, __G__($s1)
7  lw  $t2, __H__($s1)
8  add $s0, __I__, $t2
9  __J__ $t3, $s2, 2
10 add $t4, $t3, __K__
11 __L__ $s0, __M__($t4)
```

Questão 2 [10%] Na linha 1, qual expressão pode ser utilizada no lugar de __A__?

- 1
- $0
- addi
- $s2

1/1

Questão 3 [10%] Na linha 4, qual instrução deve ser utilizada no lugar de __D__?

- sw
- add
- lw
- $zero

1/1

Questão 4 [10%] Na linha 7, qual número deve ser utilizado no lugar de __H__?

- 12
- 0
- 4
- 8

0/1

Questão 5 [10%] Na linha 9, qual expressão deve ser utilizada no lugar de __J__?

- srl
- sll
- lw
- sw

1/1

Questão 6 [10%] Na linha 10, qual expressão deve ser utilizada no lugar de __K__?

- $s1
- $t0
- $t2
- $s0

0/1

Para as próximas questões, considere o trecho do código Assembly MIPS a seguir:

```
andi $t0, $t0, 0xFFFF
or   $t1, $t0, $t1
sll  $t2, $t2, 30
srl  $t2, $t2, 30
```

Considere também os valores inciais dos registradores como $t0=0x1, $t1=0xa, $t2=0xFF.

Questão 7 [10%] Ao final da execução, qual o novo valor de $t0?

- 0x0
- 0xFF
- 0xFFFF
- 0x1

1/1

Questão 8 [10%] Ao final da execução, qual o novo valor de $t1?

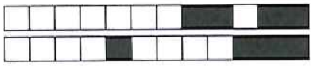
- 0x0
- 0x1
- 0xa
- 0xb

1/1

Questão 9 [10%] Ao final da execução, qual o novo valor de $t2?

- 0x1
- 0xFF
- 0x0
- 0x3

1/1



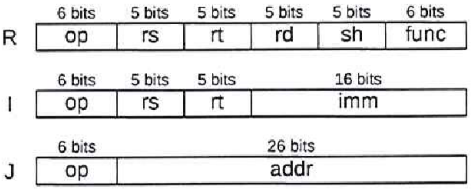
+27/2/7+

MIPS reference card

add	rd, rs, rt	Add	rd = rs + rt	R 0 / 20
sub	rd, rs, rt	Subtract	rd = rs - rt	R 0 / 22
addi	rt, rs, imm	Add Imm.	rt = rs + $imm_{\pm}$	I 8
addu	rd, rs, rt	Add Unsigned	rd = rs + rt	R 0 / 21
subu	rd, rs, rt	Subtract Unsigned	rd = rs - rt	R 0 / 23
addiu	rt, rs, imm	Add Imm. Unsigned	rt = rs + $imm_{\pm}$	I 9
mult	rs, rt	Multiply	{hi, lo} = rs * rt	R 0 / 18
div	rs, rt	Divide	lo = rs / rt; hi = rs % rt	R 0 / 1a
multu	rs, rt	Multiply Unsigned	{hi, lo} = rs * rt	R 0 / 19
divu	rs, rt	Divide Unsigned	lo = rs / rt; hi = rs % rt	R 0 / 1b
mfhi	rd	Move From Hi	rd = hi	R 0 / 10
mflo	rd	Move From Lo	rd = lo	R 0 / 12
and	rd, rs, rt	And	rd = rs & rt	R 0 / 24
or	rd, rs, rt	Or	rd = rs | rt	R 0 / 25
nor	rd, rs, rt	Nor	rd = ~(rs | rt)	R 0 / 27
xor	rd, rs, rt	eXclusive Or	rd = rs ^ rt	R 0 / 26
andi	rt, rs, imm	And Imm.	rt = rs & imm_0	I c
ori	rt, rs, imm	Or Imm.	rt = rs | imm_0	I d
xori	rt, rs, imm	eXclusive Or Imm.	rt = rs ^ imm_0	I e
sll	rd, rt, sh	Shift Left Logical	rd = rt << sh	R 0 / 0
srl	rd, rt, sh	Shift Right Logical	rd = rt >>> sh	R 0 / 2
sra	rd, rt, sh	Shift Right Arithmetic	rd = rt >> sh	R 0 / 3
sllv	rd, rt, rs	Shift Left Logical Variable	rd = rt << rs	R 0 / 4
srlv	rd, rt, rs	Shift Right Logical Variable	rd = rt >>> rs	R 0 / 6
srav	rd, rt, rs	Shift Right Arithmetic Variable	rd = rt >> rs	R 0 / 7
slt	rd, rs, rt	Set if Less Than	rd = rs < rt ? 1 : 0	R 0 / 2a
sltu	rd, rs, rt	Set if Less Than Unsigned	rd = rs < rt ? 1 : 0	R 0 / 2b
slti	rt, rs, imm	Set if Less Than Imm.	rt = rs < $imm_{\pm}$? 1 : 0	I a
sltiu	rt, rs, imm	Set if Less Than Imm. Unsigned	rt = rs < $imm_{\pm}$? 1 : 0	I b
j	addr	Jump	PC = PC&0xF0000000 | ($addr_0$ << 2)	J 2
jal	addr	Jump And Link	$ra = PC + 8; PC = PC&0xF0000000 \| ($addr_0$ << 2)	J 3
jr	rs	Jump Register	PC = rs	R 0 / 8
jalr	rs	Jump And Link Register	$ra = PC + 8; PC = rs	R 0 / 9
beq	rt, rs, imm	Branch if Equal	if (rs == rt) PC += 4 + ($imm_{\pm}$ << 2)	I 4
bne	rt, rs, imm	Branch if Not Equal	if (rs != rt) PC += 4 + ($imm_{\pm}$ << 2)	I 5
syscall		System Call	c0_cause = 8 << 2; c0_epc = PC; PC = 0x80000080	R 0 / c
lui	rt, imm	Load Upper Imm.	rt = imm << 16	I f
lb	rt, imm(rs)	Load Byte	rt = SignExt(M_1[rs + $imm_{\pm}$])	I 20
lbu	rt, imm(rs)	Load Byte Unsigned	rt = M_1[rs + $imm_{\pm}$] & 0xFF	I 24
lh	rt, imm(rs)	Load Half	rt = SignExt(M_2[rs + $imm_{\pm}$])	I 21
lhu	rt, imm(rs)	Load Half Unsigned	rt = M_2[rs + $imm_{\pm}$] & 0xFFFF	I 25
lw	rt, imm(rs)	Load Word	rt = M_4[rs + $imm_{\pm}$]	I 23
sb	rt, imm(rs)	Store Byte	M_1[rs + $imm_{\pm}$] = rt	I 28
sh	rt, imm(rs)	Store Half	M_2[rs + $imm_{\pm}$] = rt	I 29
sw	rt, imm(rs)	Store Word	M_4[rs + $imm_{\pm}$] = rt	I 2b
ll	rt, imm(rs)	Load Linked	rt = M_4[rs + $imm_{\pm}$]	I 30
sc	rt, imm(rs)	Store Conditional	M_4[rs + $imm_{\pm}$] = rt; rt = atomic ? 1 : 0	I 38

pseudo-instructions

bge	rx, ry, imm	Branch if Greater or Equal
bgt	rx, ry, imm	Branch if Greater Than
ble	rx, ry, imm	Branch if Less or Equal
blt	rx, ry, imm	Branch if Less Than
la	rx, label	Load Address
li	rx, imm	Load Immediate
move	rx, ry	Move register
nop		No Operation



registers

$0	$zero
$1	$at
$2–$3	$v0–$v1
$4–$7	$a0–$a3
$8–$15	$t0–$t7
$16–$23	$s0–$s7
$24–$25	$t8–$t9
$26–$27	$k0–$k1
$28	$gp
$29	$sp
$30	$fp
$31	$ra
hi	—
lo	—
PC	—
co $13	c0_cause
co $14	c0_epc

syscall codes for MARS/SPIM

1 print integer
2 print float
3 print double
4 print string
5 read integer
6 read float
7 read double
8 read string
9 sbrk/alloc. mem.
10 exit
11 print character
12 read character
13 open file
14 read file
15 write to file
16 close file

exception causes

0 interrupt
1 TLB protection
2 TLB miss L/F
3 TLB miss S
4 bad address L/F
5 bad address S
6 bus error F
7 bus error L/S
8 syscall
9 break
a reserved instr.
b coproc. unusable
c arith. overflow

F: fetch instr.
L: load data
S: store data