

**IT5002 Computer Systems and
Applications**

Tutorial 2

SUGGESTED SOLUTIONS

1. MIPS Bitwise Operations

Implement the following in MIPS assembly. Assume that integer variables **a**, **b** and **c** are mapped to registers \$s0, \$s1 and \$s2 respectively. Each part is independent of all the other parts. **For bitwise instructions (e.g. ori, andi, etc)**, any immediate values you use should be written in binary for this question. This is optional for non-bitwise instructions (e.g. addi, etc).

Note that bit 31 is the most significant bit (MSB) on the left, and bit 0 is the least significant bit (LSB) on the right, i.e.:

MSB					LSB
Bit 31	Bit 30	Bit 29	...	Bit 1	Bit 0

- a. Set bits 2, 8, 9, 14 and 16 of **b** to 1. Leave all other bits unchanged.

To set bits, we create a “mask” with 1’s in the bit positions we want to set. Since bit 16 is in the upper 16 bits of the register, we need to use lui to set it.

```
lui $t0, 1    # Sets bit 16 of $t0.
ori $t0, $t0, 0b0100001100000100 # Set bits 14, 9, 8 and 2.
or $s1, $s1, $t0
```

- b. Copy over bits 1, 3 and 7 of **b** into **a**, without changing any other bits of **a**.

```
# We use the property that x AND 1 = x to copy out the values of
# bits 7, 3 and 1 of b into $t0. Note that we zero all other bits
# so that they don't change anything in $s0 when we OR later on.
andi $t0, $s1, 0b0000000010001010
```

```
# We use the property of x OR 0 = x to copy in
# the bits into a, so we prepare a by zero-ing bits 7, 3 and 1.
# To do this we need the mask 1111111111111111 111111101110101
lui $t1, 0b1111111111111111
ori $t1, $t1, 0b1111111101110101
and $s0, $s0, $t1
```

```
# Now OR together a and $t0 to copy over the bits
or $s0, $s0, $t0
```

- c. Make bits 2, 4 and 8 of **c** the inverse of bits 1, 3 and 7 of **b** (i.e. if bit 1 of **b** is 0, then bit 2 of **c** should be 1; if bit 1 of **b** is 1, then bit 2 of **c** should be 0), without changing any other bits of **c**.

```
# We use the property that x XOR 1 = ~x to flip the values of bits 7, 3 and 1.
xori $t0, $s1, 0b10001010
```

Zero every bit except 7, 3 and 1.

```
andi $t0, $t0, 0b10001010
```

Shift left one position: bit 1 becomes 0, bit 1 becomes bit 2, bit 3 becomes bit 4, and bit 7 becomes bit 8.

```
sll $t0, $t0, 1
```

Now, to clear bits 8, 4 and 2 of c,

we need the mask 0b1111111111111111 1111111011101010

```
lui $t1, 0b1111111111111111
```

```
ori $t1, $t1, 0b1111111011101011
```

```
and $s2, $s2, $t1
```

and we OR the new c with \$t0

```
or $s2, $s2, $t0
```

2. The mysterious MIPS code below assumes that **\$s0 is a 31-bit binary sequence**, i.e. the MSB (most significant bit) of **\$s0** is assumed to be zero at the start of the code.

```
add $t0, $s0, $zero # make a copy of $s0 in $t0
lui $t1, 0x8000
lp: beq $t0, $zero, e
    andi $t2, $t0, 1
    beq $t2, $zero, s
    xor $s0, $s0, $t1
s:  srl $t0, $t0, 1
    j lp
e:
```

- a) For each of the following initial values in register **\$s0** at the beginning of the code, give the hexadecimal value of the content in register **\$s0** at the end of the code.
- Decimal value **31**.
 - Hexadecimal value **0x0AAAAAAA**.
- b) Explain the purpose of the code in one sentence.

Answers:

a. i. **\$s0 = 0x8000 001F**

ii. **\$s0 = 0x0AAA AAAA**

- b. The code sets bit 31 of \$s0 to 1 if there are odd number of '1' in \$s0 initially, or 0 if there are even number of '1'. (This is called the even parity bit.)

3. Given two integer arrays *A* and *B* with unknown number of elements, and their base addresses stored in registers *\$s0* and *\$s1* respectively, study the MIPS code below. Note that an integer takes up 32 bits of memory.

```
        addi $t0, $s0, 0
        addi $t1, $s1, 0
loop:   lw    $t3, 0($t0)
        lw    $t4, 0($t1)
        slt   $t5, $t4, $t3      # line A
        beq   $t5, $zero, skip   # line B
        sw    $t4, 0($t0)
        sw    $t3, 0($t1)
skip:   addi   $t0, $t0, 4
        addi   $t1, $t1, 4
        bne    $t3, $zero, loop
```

- a. What is the purpose of register *\$t1* in this code?

Answer: *\$t1* is the address of the *B[]* element to be read

- b. If array *A* = {7, 4, 1, 6, 0, 5, 9, 0} and
array *B* = {3, 4, 5, 2, 1, 0, 0, 9},
give the final content of these two arrays.

Answer:

A = {3, 4, 1, 2, 0, 5, 9, 0}

B = {7, 4, 5, 6, 1, 0, 0, 9}

- c. How many **store word** operations are performed given the contents of the arrays in part (b)?

Answer: 4

- d. What is the value (in decimal) of the immediate field in the machine code representation of the **bne** instruction?

Answer: -9

- e. The two lines indicated as “line A” and “line B” represent the translation of a MIPS pseudo-instruction. Give the corresponding pseudo-instruction.

Answer: *bge \$t4, \$t3, skip*

4. You accidentally spilled coffee on your best friend's MIPS assembly code printout. Fortunately, there are enough hints for you to reconstruct the code. Fill in the missing lines (shaded cells) below to save your friendship.

Answer:

Instruction Encoding	MIPS Code
	# \$s1 stores the result, \$t0 stores a non-negative number
0x20110000	addi \$s1, \$zero, 0 #Inst. address is 0x00400028
0x00084042	loop: srl \$t0, \$t0, 1
0x11000002	beq \$t0, \$zero, exit
0x22310001	addi \$s1, \$s1, 1
0x0810000B	j loop
	exit:

- b. Give a simple mathematic expression for the relationship between **\$s1** and **\$t0** as calculated in the code.

Answer: $\$s1 = \lfloor \log_2 (\$t0) \rfloor$, where $\lfloor x \rfloor$ denotes the floor(x) function.

Workings:

0x20110000 = 0010 0000 0001 0001 0000 = 001000 00000 10001 0000...
= addi \$17, \$0, 0 = addi \$s1, \$zero, 0

0x11000002 = 0001 0001 0000 0000 00...010 = 000100 01000 00000 00...010
= beq \$8 \$0 2 = beq \$t0, \$zero, exit

0x22310001 = 0010 0010 0011 0001 00...01 = 001000 10001 10001 00...01
= addi \$17 \$17 1 = addi \$s1, \$s1, 1

0x0810000b = 0000 1000 0001 0000 00...1011 = 000010 0000 0100 0000 ... 1011
= j {0000} 0000 0100 0000 ... 0010 11{00} = j 040002c

5.