COE 301 / ICS 233: Computer Organization

Midterm Exam – Term 211

Saturday, October 30, 2021

10 am – 12 noon

Computer Engineering Department

College of Computing & Mathematics

King Fahd University of Petroleum & Minerals

SOLUTION

Q1	/ 15	Q2	/ 16
Q3	/ 15	Q4	/ 10
Q5	/ 10	Q6	/ 10
Total	/ 76		

Select you Section:

☐ COE 301-01	Dr. Muhamed Mudawar	UTR 8 – 8:50 AM
☐ COE 301-02	Dr. Ayaz Khan	UTR 11 – 11:50 AM
☐ ICS 233-01	Dr. Abdel-Aziz Tabakh	UTR 9 – 9:50 AM
☐ ICS 233-02	Dr. Abdel-Aziz Tabakh	UTR 10 – 10:50 AM
☐ ICS 233-03	Dr. Ayaz Khan	UTR 11 – 11:50 AM
☐ ICS 233-04	Dr. Ayaz Khan	UTR 8 – 8:50 AM

Q1. Instruction Type, Data Definition, and Loaded Values [15 points]

a) **(6 points)** Write the instruction format (R-type, I-type, or J-type) for each of the following MIPS instructions.

MIPS Instruction	R-Type, I-Type, or J-Type ?
addiu	I-type
sll	R-type
jal	J-type
jr	R-type
beq	I-type
lw	I-type

b) **(4 points)** Complete the symbol table for the following data definitions showing the address of each label, given the address of **var1** is **0x10010000** in the data segment.

```
.DATA
    var1:   .BYTE    1, 2, -3, -4, 5, 6, 7
    var2:   .WORD    0x12345678
    str1:   .ASCIIZ  "Test String\n"
    .ALIGN   3
    var3:   .HALF    1000
    var4:   .DOUBLE  1.5e-10
```

Label	Address
var1	0x10010000
var2	0x10010008
str1	0x1001000C
var3	0x10010020
var4	0x10010028

c) (**5 points**) Given the data definition of **part b**, what value is loaded into register **$t1** (in hexadecimal), if **Little Endian** Byte ordering is used?

Instruction Sequence	Value loaded into $t1 (hexadecimal)
la $t0, var1 lb $t1, 0($t0)	$t1 = 0x00000001 (1)
la $t0, var1 lb $t1, 2($t0)	$t1 = 0xFFFFFFFD (-2)
la $t0, var1 lh $t1, 4($t0)	$t1 = 0x00000605
la $t0, var2 lb $t1, 0($t0)	$t1 = 0x00000078
la $t0, var2 lh $t1, 2($t0)	$t1 = 0x00001234

Q2. ALU Instructions [16 points]

a) **(5 points)** Given that `$t0 = 0x78901234` and `$t1 = 0xAB015678` are two **signed** integers, compute the following.

Instruction	Value computed (hexadecimal)
`add $t2, $t0, $t1`	`$t2 = 0x239168AC` Overflow (Yes / No)? NO
`sub $t3, $t0, $t1`	`$t3 = 0xCD8EBBBC` Overflow (Yes / No)? YES
`sra $t4, $t1, 8`	`$t4 = 0xFFAB0156`

Show the addition / subtraction in hexadecimal and indicate whether there is overflow.

```
1 1                                 1     1
  78901234          78901234      78901234
+ AB015678        - AB015678    + 54FEA987  (1's compl +1)
----------                      ----------
  239168AC                        CD8EBBBC
```

b) **(4 points)** Show the binary representation of the following instructions. Register `$t0` is register number `8`. The function code of `addu` is `0x21`, and the opcode of `addiu` is `0x9`.

Instruction	32-bit Binary Representation
`addu  $t2, $t0, $t1`	`Op     Rs    Rt    Rd    sa    func` `000000 01000 01001 01010 00000 100001`
`addiu $t3, $t1, 8`	`Op     Rs    Rt    Imm16` `001001 01001 01011 0000000000001000`

c) **(4 points)** Translate the following assignment statement into a minimal number of MIPS basic instructions. Assume that `f`, `g`, and `h` are 32-bit integers that are stored in `$t0`, `$t1`, and `$t2`, respectively. Use shift instructions to achieve multiplication.

```
f = g + h * 10
```

```
sll  $t3, $t2, 3          # $t3 = $t2*8
sll  $t4, $t2, 1          # $t4 = $t2*2
addu $t5, $t3, $t4        # $t5 = $t2*10 = h*10
addu $t0, $t1, $t5        # $t0 = f = g + h*10
```

d) **(3 points)** Write a minimum sequence of MIPS basic instructions to implement the following pseudo instructions. You can only modify the `$at` register as a side effect.

```
andi $t1,$t2,0xABCD0001      # AND with a 32-bit immediate
```

```
lui $at, 0xABCD              # $at = 0xABCD0000
ori $at, $at, 0x0001         # $at = 0xABCD0001
and $t1, $t2, $at
```

Q3. Control Instructions [15 points]

a) **(2 points)** Write a minimum sequence of MIPS basic instructions to implement the following pseudo-instruction:

```
sne $s1, $s2, $s3            # set if not equal
subu   $s1, $s2, $s3
sltu   $s1, $zero, $s1
```

b) **(2 points)** Write a minimum sequence of MIPS basic instructions to implement the following pseudo-instruction. You can only modify the **$at** register as a side effect.

```
bgeu $s1, $s2, next    # branch if greater or equal unsigned

sltu $at, $s1, $s2           # 1 point per instruction
beq  $at, $zero, next        # -0.5 for not using $at
```

c) **(6 points)** The following is a partial MIPS assembly language code:

Address	Label	Instruction
0x40601C00	L1:	bgtz $a1, L2
		. . .
0x40602000	L2:	and $t0, $t1, $t2
		. . .
0x4060201C		beq $a0, $a2, L2
		. . .
0x4062A000		J L1

Calculate the 16-bit immediate value (in hexadecimal) in **bgtz** instruction:

imm_{16} = (0x40602000 – 0x40601C04)/4 = 0x000003FC/4 = 0x00FF

Calculate the 16-bit immediate value (in hexadecimal) in **beq** instruction:

imm_{16} = (0x40602000 – 0x40602020)/4 = - 0x0020/4 = - 0x0008 = 0xFFF8

Calculate the 26-bit immediate value (in hexadecimal) in **J** instruction:

PC_{L1} : 0100 [<u>0000 0110 0000 0001 1100 0000 00</u>]00

➔ imm_{26} : 00 0001 1000 0000 0111 0000 0000 = 0x0180700

d) **(5 points)** Translate the following high-level if-statement into MIPS assembly code. Assume that **a**, **b**, **c** and **d** are signed integers loaded into registers **$t0**, **$t1**, **$t2**, and **$t3** respectively. You can use pseudo-branch instructions if needed.

```
if (((a > c) || (b <= d)) && (a == b)) {
   if (c != 0) { a = c; }
   d = a + b;
}
```

```
     bgt   $t0, $t2, L1         # if (a > c), skip OR
     bgt   $t1, $t3, next       # if (b > d), skip if statement
L1:  bne   $t0, $t1, next       # if (a != b), skip IF statement
     movn  $t0, $t2, $t2        # interior if statement
     addu  $t3, $t0, $t1
next: . . .
```

Q4. Integer Multiplication [10 points]

a) (7 points) Show the binary multiplication of the following two 16-bit unsigned integers. The product should be a 32-bit unsigned integer. Do NOT show partial products (rows) that contain only zeros.

```
                                    1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1
                                  × 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0
                                  ---------------------------------
Carry bits           1 1 1 1   1 1                 1
                                1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1
                          1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1
                1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1
      1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1
-----------------------------------------------------------------------
  0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 1 0 1 0 1 1 0 1 0 1 0 0 0 1 0 0

  0101 1011 1110 0101 1010 1101 0100 0100
```

b) (3 points) To implement a 32-bit by 32-bit tree multiplier in hardware, how many AND gates are used? How many adders are needed and what is their size? Explain your answer.

Number of AND gates = 32 × 32 = 1024

There are 32 partial product results

Total number of adders = 31 adders to add the 32 partial products

Each adder is 32-bit.

Q5. Tracing the Execution of Assembly Language Code [10 Points]

a) **(4 points)** Given that **`Array`** is defined below and starts at address **`0x10010000`**, explain what the code does and the value of **`$v0` and `$v1`** after executing the following code.

```
Array: .word 15, -19, 17, -20, 10, -18, 103, -6, -73, 2

                la   $a0, Array    # $a0 = 0x10010000
                addi $a1, $a0, 40
                move $v0, $a0
                lw   $v1, 0($v0)
                move $t0, $a0
        loop:   addi $t0, $t0, 4
                lw   $t1, 0($t0)
                bge  $t1, $v1, skip
                move $v0, $t0
                move $v1, $t1
        skip:   bne  $t0, $a1, loop
```

The above code locates the minimum element
`$v0 = 0x10010020 (address of -73)`
`$v1 = -73 (minimum value)`

b) **(6 points)** Given that **`Array`** is defined below, explain what the code does and determine the content of **`Array`** after executing the following code.

```
Array: .word 14, 28, -31, 47, 53, 80, -1, 13, 19, 4, 17, 12

            la   $a0, Array
            li   $a1, 6
            move $t0, $a0
            addi $t1, $a0, 24
      loop: lw   $t3, ($t0)
            lw   $t4, ($t1)
            sw   $t3, ($t1)
            sw   $t4, ($t0)
            addi $t0, $t0, 4
            addi $t1, $t1, 4
            addi $a1, $a1, -1
            bne  $a1, $zero, loop
```

The above loop swaps the first six elements of Array with the last six elements. The New Array Content:
`-1, 13, 19, 4, 17, 12, 14, 28, -31, 47, 53, 80`

Q6. Write a MIPS Function [10 points]

Write a function **`gcd(a,b)`** to compute the greatest common divisor of two unsigned integers:

```
gcd(a,0) = a          // if (b == 0)
gcd(a,b) = gcd(b,a%b) // a%b is the remainder of division
```

For example: **`gcd(8,12) = gcd(12,8) = gcd(8,4) = gcd(4,0) = 4`**.

Solution: function can be written as a simple loop. No need to allocate a stack frame and save registers.

Example of Loop Version:

```
gcd:    bne  $a1, $0, else   # branch if (b != 0) else
        move $v0, $a0        # $v0 = a
        jr   $ra             # return to caller
else:   divu $a0, $a1        # divide a by b (unsigned)
        move $a0, $a1        # $a0 = b
        mfhi $a1             # $a1 = remainder a%b
        j    gcd             # jump to gcd
```

example of recursive version:

```
gcd:
        bne  $a1, $0, else   # branch if (b != 0) else
        move $v0, $a0        # $v0 = a
        jr   $ra             # return to caller
else:   addiu $sp,$sp, -4    # allocate 4 bytes in the Stack
        sw   $ra, 0($sp)     # store return address ($ra)
        divu $a0, $a1        # divide a by b (unsigned)
        move $a0, $a1        # $a0 = b
        mfhi $a1             # $a1 = remainder a%b
        jal gcd              # call gcd recursively
        lw   $ra, 0($sp)     # restore return address
        addiu $sp,$sp, 4     # free the Stack
        jr $ra               # return to gcd
```