Name: ______________________________

Student #: ____________________________

King Fahd University of Petroleum and Minerals
College of Computing and Mathematics
Department of Computer Engineering

COE 301 – Computer Organization (T212)
ICS 233 – Computer Architecture & Assembly Language (T212)

Midterm Exam – SOLUTION

Date & Time: Friday March 18, 2022 (06:00 PM – 08:00 PM)

- **This is a CLOSED books, CLOSED notes exam.**
- **Answer ALL problems.**
- **Show all your work. NO partial credit will be given if work is not shown.**
- **Use of mobile phones, smart phones/watches, tablets is prohibited.**

Problem	Mark	Score
1	4.0	
2	6.0	
3	4.0	
4	6.0	
5	2.0	
6	2.0	
7	4.0	
8	3.0	
9	4.0	
10	2.0	
11	2.0	
~~12~~	~~2.0~~	
13	8.0	
Total	49.0	

Select your section number:

- ☐ **COE 301 – Section 1** (UTR 08:00 – Dr. Ayaz Khan)
- ☐ **COE 301 – Section 2** (UTR 11:00 – Dr. Marwan Abu-Amara)
- ☐ **ICS 233 – Section 1** (UTR 11:00 – Dr. Ayaz Khan)
- ☐ **ICS 233 – Section 2** (UTR 10:00 – Dr. Ayaz Khan)

Problem 1 (4 points): Write a MIPS code fragment that computes **$s1 = ($s0 × 45)** without the use of multiplication instructions while using a minimum number of instructions. HINT: **45 = (3 × 15)**

```
sll  $t0, $s0, 2
subu $t1, $t0, $s0
sll  $t2, $t1, 4
subu $s1, $t2, $t1
```

Problem 2 (6 points): Translate the following nested if-statements into minimal MIPS assembly code. All variables are signed integers. The values of **a**, **b**, and **c** are stored in **$t0**, **$t1**, and **$t2**, respectively.

```
if ((a >= 0) && (a <= 9)) {
  if (b >= c) a = b - c;
  a = a / 4;
}
```

```
     addi $t3, zero, 9    # $t3 = 9
     blt  $t0, zero, L1   # if (a < 0), skip outer if statement
     bgt  $t0, $t3, L1    # if (a > 9), skip outer if statement
     bgt  $t2, $t1, L2    # if (b < c), skip computing a = b - c
     subu $t0, $t1, $t2   # a = b - c
L2:  sra  $t0, $t0, 2     # compute a = a / 4 (use “sra” as a is signed)
L1:  . . .
```

Problem 3 (4 points): The following is a partial MIPS assembly language code:

Address	Label	Instruction
0x40B8100C	L1:	add $a0, $t1, $t2
		. . .
0x40B82000	L2:	and $a1, $t1, $t2
		. . .
0x40B82028		bgt $a0, $a1, L2
		. . .
0x40B9C000		j L1

i. Calculate the **hexadecimal** 16-bit immediate value (imm_{16}) in the **bgt** instruction:

imm_{16} = (0x40B82000 – 0x40B8202C)/4 = – 0x002C/4 = – 0x000B = 0xFFF5

ii. Calculate the **hexadecimal** 26-bit immediate value (imm_{26}) in the **j** instruction:

PC L1: 0100 [<u>0000 1011 1000 0001 0000 0000 11</u>]00
⇒ imm_{26} : <u>00 0010 1110 0000 0100 0000 0011</u> = 0x02E0403

Problem 4 (6 points):

(a) **(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: .HALF      -2, -3, 4, 5
str1: .ASCIIZ    "EXAM"
var2: .WORD      0x5678ABCD
      .ALIGN     4
var3: .HALF      1000
```

Label	Address
var1	0x10010000
str1	0x10010008
var2	0x10010010
var3	0x10010020

(b) **(2 points)** Given the data definition of part (a), show the value loaded into register **`$t1`** (in hexadecimal). Assume Little **Endian Byte** ordering is used.

Instruction	Sequence Value loaded into $t1 (hexadecimal)
la $t0, var1 lb $t1, 0($t0)	$t1 = 0xFFFFFFFE (-2)
la $t0, var2 lh $t1, 0($t0)	$t1 = 0xFFFFABCD

Problem 5 (2 points): Given that `$t0 = 0x07B95342` and `$t1 = 0x85305421` are two **signed** integers, consider the following instruction:

```
sub $t2, $t0, $t1
```

Perform the subtraction in hexadecimal and indicate whether there is overflow. Show the subtraction in hexadecimal below.

```
                111  11
  07B95342      07B95342
- 85305421  ⇒  + 7ACFABDF
                8288FF21 = $t2
```

⇒ Overflow occurred since adding 2 positive numbers resulted in a negative number.

Problem 6 (2 points): Let `M[6][9]` be an array of **integers** with 6 rows and 9 columns that have been saved in the memory with `&M[0][0]` stored in `$t0`. Determine the displacement `XX` (in decimal) in the following instruction to properly access the integer stored at `M[2][8]`:

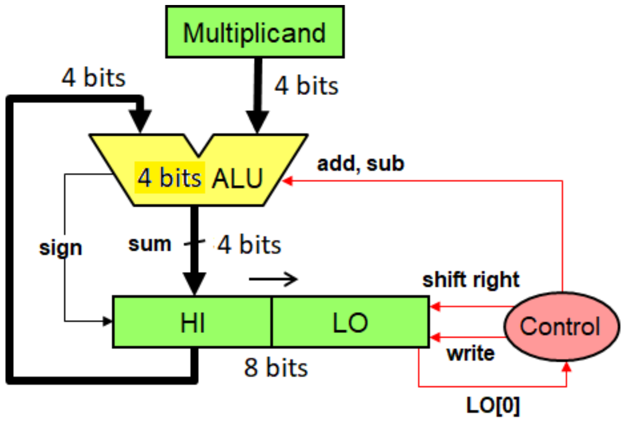
```
lw   $t1, XX($t0)
```

`XX = (2 × 9 + 8) × 4 = 104`

Problem 7 (4 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.

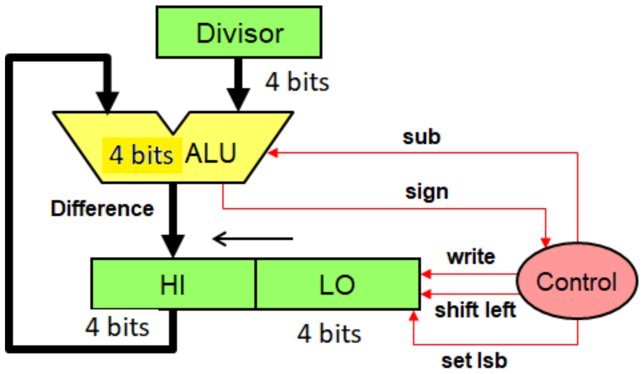
```
                                      1 1 1 1 0 1 1 0 1 0 0 0 1 0 1 1
                                    × 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0
                                  ----------------------------------
Solution:                         1
Carry bits          1 1 1 1 1 1 1 1 1 1 1       1
                                  1 1 1 1 0 1 1 0 1 0 0 0 1 0 1 1
                          1 1 1 1 0 1 1 0 1 0 0 0 1 0 1 1
            1 1 1 1 0 1 1 0 1 0 0 0 1 0 1 1
----------------------------------------------------------------------
      0 0 0 1 1 1 1 1 0 0 0 1 0 0 1 0 1 1 0 1 1 1 0 0 1 1 1 0 1 1 0 0
```

Problem 8 (3 points): Complete the table below to perform the multiplication of two 4-bits signed numbers **`1011 (-5)`** and **`1010 (-6)`** using the following MIPS hardware.



Iteration	Steps	Multiplicand	Sign	Product = HI, LO
0	Initialize	1011		0000 1010
1	No Add			
	Shift Right			0000 0101
2	Add	1011	1	1011 0101
	Shift Right			1101 1010
3	No Add			
	Shift Right			1110 1101
4	Sub (Add 2's Compl)	0101	0	0011 1101
	Shift Right			0001 1110

Problem 9 (4 points): Complete the table below to perform the division of two 4-bits signed numbers: **`1010 (-6) / 1011 (-5)`** using the following MIPS hardware **and** calculate the final answers for the quotient and the remainder.



Iteration	Steps	HI	LO	Divisor	Difference
0	Initialize	0000	0110	0101	
1	Shift Left, HI – Divisor	0000	1100	0101	< 0
	Do Nothing				
2	Shift Left, HI – Divisor	0001	1000	0101	< 0
	Do Nothing				
3	Shift Left, HI – Divisor	0011	0000	0101	< 0
	Do Nothing				
4	Shift Left, HI – Divisor	0110	0000	0101	0001
	HI, LO update	0001	0001		

Quotient = ______________________________ **Remainder =** ______________________________

`Both Dividend and Divisor are negative.`
`Quotient = HI = 0001 (+1), Remainder = 2's compl of LO = 1111 (-1)`

Problem 10 (2 points): Find the decimal value of the following single-precision floating-point number:

1100 0010 1011 1110 0100 0000 0000 0000

Sign = negative

Exponent value = $(1000\ 0101)_2$ – 127 = (128+4+1) - 127 = 133 – 127 = 6

$-(1.011\ 1110\ 0100\ 0000\ 0000\ 0000)_2 \times 2^6 = -(1011111.00100\ 0000\ 0000\ 0000)_2 =$

-(64+16+8+4+2+1+0.125) = -95.125

Problem 11 (2 points): Show the normalized single-precision floating-point in IEEE754 format binary representation for: **-116.325** (solve up to 4 fractional bits)

$-(116.325)_{10} = -(1110100.0101)_2 = -(1.1101000101)_2 \times 2^6$

E = Exponent + 127 = 6 + 127 = 133 = $(10000101)_2$

F = $(110\ 1000\ 1010\ 0000\ 0000\ 0000)_2$

IEEE representation = 1 10000101 110 1000 1010 0000 0000 0000

Problem 12 (2 points): Convert the following IEEE 754 double-precision floating-point number into IEEE 754 single-precision. Use rounding to zero (i.e., truncate the lesser significant bits) if needed.

0011 0101 0101 1101 1101 1001 0010 1001 1100 0101 0101 1010 1001 0011 0100 1011

Sign = 0

E_{DP} = $(011\ 0101\ 0101)_2$ = 853 ➔ Exponent = 853-1023 = -170

It will be an underflow for single precision floating point number as the exponent value is out of range.

Problem 13 (8 points): Write a MIPS function ***sum_digits*** that computes and returns the sum of decimal digits in an **unsigned integer**. For example, the sum of decimal digits for **1536** is **1+5+3+6 = 15**. The function ***sum_digits*** receives the unsigned integer argument in binary in register **$a0**. For example, **1536** = **(0000 … 0110 0000 0000)₂**. It should extract the decimal digits, compute, and return their sum, also in binary, in register **$v0**. Hint: divide the unsigned integer by **10** to extract the decimal digits.

Non-recursive Solution

```
sum_digits: li    $v0, 0              # $v0 = sum = 0
            li    $t0, 10             # divisor = 10
loop:       divu  $a0, $t0            # divide by 10
            mfhi  $t1                 # $t1 = remainder
            mflo  $a0                 # $a0 = quotient
            add   $v0, $v0, $t1       # add decimal digit
            bne   $a0, $zero, loop    # loop if more digits in $a0
            jr    $ra                 # return to caller
```

Recursive Solution

```
sum_digits: li    $t0, 10             # divisor = 10
            bne $a0, $zero, next      # do recursive call if quotient ≠ 0
            li    $v0, 0              # otherwise return $v0 = 0
            jr    $ra                 # return to caller
next:       addiu $sp,$sp, -8         # allocate stack frame (2 words)
            sw    $ra, 0($sp)         # save return address in 1st word
            divu  $a0, $t0            # divide by 10
            mflo  $a0                 # $a0 = quotient
            mfhi  $t1                 # $t1 = remainder
            sw    $t1, 4($sp)         # save $t1 across recursive calls
            jal   sum_digits          # recursive call
            lw    $t1, 4($sp)         # restore last $t1
            addu  $v0, $v0, $t1       # add restored $t1 to current $v0
            lw    $ra, 0($sp)         # restore return address
            addiu $sp, $sp, 8         # release current stack frame
            jr    $ra                 # return to caller
```

Name	Register	Usage	
$zero	$0	Always 0	(forced by hardware)
$at	$1	Reserved for assembler use	
$v0 – $v1	$2 – $3	Result values of a function	
$a0 – $a3	$4 – $7	Arguments of a function	
$t0 – $t7	$8 – $15	Temporary Values	
$s0 – $s7	$16 – $23	Saved registers	(preserved across call)
$t8 – $t9	$24 – $25	More temporaries	
$k0 – $k1	$26 – $27	Reserved for OS kernel	
$gp	$28	Global pointer	(points to global data)
$sp	$29	Stack pointer	(points to top of stack)
$fp	$30	Frame pointer	(points to stack frame)
$ra	$31	Return address	(used for function call)

Instruction		Meaning	R-Type Format					
add	$s1, $s2, $s3	$s1 = $s2 + $s3	op = 0	rs = $s2	rt = $s3	rd = $s1	sa = 0	f = 0x20
addu	$s1, $s2, $s3	$s1 = $s2 + $s3	op = 0	rs = $s2	rt = $s3	rd = $s1	sa = 0	f = 0x21
sub	$s1, $s2, $s3	$s1 = $s2 – $s3	op = 0	rs = $s2	rt = $s3	rd = $s1	sa = 0	f = 0x22
subu	$s1, $s2, $s3	$s1 = $s2 – $s3	op = 0	rs = $s2	rt = $s3	rd = $s1	sa = 0	f = 0x23

Instruction		Meaning	R-Type Format					
and	$s1, $s2, $s3	$s1 = $s2 & $s3	op = 0	rs = $s2	rt = $s3	rd = $s1	sa = 0	f = 0x24
or	$s1, $s2, $s3	$s1 = $s2 | $s3	op = 0	rs = $s2	rt = $s3	rd = $s1	sa = 0	f = 0x25
xor	$s1, $s2, $s3	$s1 = $s2 ^ $s3	op = 0	rs = $s2	rt = $s3	rd = $s1	sa = 0	f = 0x26
nor	$s1, $s2, $s3	$s1 = ~($s2|$s3)	op = 0	rs = $s2	rt = $s3	rd = $s1	sa = 0	f = 0x27

Instruction		Meaning	R-Type Format					
sll	$s1,$s2,10	$s1 = $s2 << 10	op = 0	rs = 0	rt = $s2	rd = $s1	sa = 10	f = 0
srl	$s1,$s2,10	$s1 = $s2>>>10	op = 0	rs = 0	rt = $s2	rd = $s1	sa = 10	f = 2
sra	$s1, $s2, 10	$s1 = $s2 >> 10	op = 0	rs = 0	rt = $s2	rd = $s1	sa = 10	f = 3
sllv	$s1,$s2,$s3	$s1 = $s2 << $s3	op = 0	rs = $s3	rt = $s2	rd = $s1	sa = 0	f = 4
srlv	$s1,$s2,$s3	$s1 = $s2>>>$s3	op = 0	rs = $s3	rt = $s2	rd = $s1	sa = 0	f = 6
srav	$s1,$s2,$s3	$s1 = $s2 >> $s3	op = 0	rs = $s3	rt = $s2	rd = $s1	sa = 0	f = 7

Instruction		Meaning	I-Type Format			
addi	$s1, $s2, 10	$s1 = $s2 + 10	op = 0x8	rs = $s2	rt = $s1	imm^{16} = 10
addiu	$s1, $s2, 10	$s1 = $s2 + 10	op = 0x9	rs = $s2	rt = $s1	imm^{16} = 10
andi	$s1, $s2, 10	$s1 = $s2 & 10	op = 0xc	rs = $s2	rt = $s1	imm^{16} = 10
ori	$s1, $s2, 10	$s1 = $s2 | 10	op = 0xd	rs = $s2	rt = $s1	imm^{16} = 10
xori	$s1, $s2, 10	$s1 = $s2 ^ 10	op = 0xe	rs = $s2	rt = $s1	imm^{16} = 10
lui	$s1, 10	$s1 = 10 << 16	op = 0xf	0	rt = $s1	imm^{16} = 10

Instruction		Meaning	Format			
j	label	jump to label	op^{6} = 2	imm^{26}		
beq	rs, rt, label	branch if (rs == rt)	op^{6} = 4	rs^{5}	rt^{5}	imm^{16}
bne	rs, rt, label	branch if (rs != rt)	op^{6} = 5	rs^{5}	rt^{5}	imm^{16}
blez	rs, label	branch if (rs<=0)	op^{6} = 6	rs^{5}	0	imm^{16}
bgtz	rs, label	branch if (rs > 0)	op^{6} = 7	rs^{5}	0	imm^{16}
bltz	rs, label	branch if (rs < 0)	op^{6} = 1	rs^{5}	0	imm^{16}
bgez	rs, label	branch if (rs>=0)	op^{6} = 1	rs^{5}	1	imm^{16}

Instruction		Meaning	Format					
slt	rd, rs, rt	rd=(rs<rt?1:0)	op^{6} = 0	rs^{5}	rt^{5}	rd^{5}	0	0x2a
sltu	rd, rs, rt	rd=(rs<rt?1:0)	op^{6} = 0	rs^{5}	rt^{5}	rd^{5}	0	0x2b
slti	rt, rs, imm^{16}	rt=(rs<imm?1:0)	0xa	rs^{5}	rt^{5}	imm^{16}		
sltiu	rt, rs, imm^{16}	rt=(rs<imm?1:0)	0xb	rs^{5}	rt^{5}	imm^{16}		

Instruction		Meaning	I-Type Format			
lb	rt, imm^{16}(rs)	rt = MEM[rs+imm^{16}]	0x20	rs^{5}	rt^{5}	imm^{16}
lh	rt, imm^{16}(rs)	rt = MEM[rs+imm^{16}]	0x21	rs^{5}	rt^{5}	imm^{16}
lw	rt, imm^{16}(rs)	rt = MEM[rs+imm^{16}]	0x23	rs^{5}	rt^{5}	imm^{16}
lbu	rt, imm^{16}(rs)	rt = MEM[rs+imm^{16}]	0x24	rs^{5}	rt^{5}	imm^{16}
lhu	rt, imm^{16}(rs)	rt = MEM[rs+imm^{16}]	0x25	rs^{5}	rt^{5}	imm^{16}
sb	rt, imm^{16}(rs)	MEM[rs+imm^{16}] = rt	0x28	rs^{5}	rt^{5}	imm^{16}
sh	rt, imm^{16}(rs)	MEM[rs+imm^{16}] = rt	0x29	rs^{5}	rt^{5}	imm^{16}
sw	rt, imm^{16}(rs)	MEM[rs+imm^{16}] = rt	0x2b	rs^{5}	rt^{5}	imm^{16}

Instruction		Meaning	Format					
jal	label	$31=PC+4, jump	op^{6} = 3	imm^{26}				
jr	Rs	PC = Rs	op^{6} = 0	rs^{5}	0	0	0	8
jalr	Rd, Rs	Rd=PC+4, PC=Rs	op^{6} = 0	rs^{5}	0	rd^{5}	0	9

Instruction		Meaning	Format					
mult	Rs, Rt	Hi, Lo = Rs × Rt	op^{6} = 0	Rs^{5}	Rt^{5}	0	0	0x18
multu	Rs, Rt	Hi, Lo = Rs × Rt	op^{6} = 0	Rs^{5}	Rt^{5}	0	0	0x19
mul	Rd, Rs, Rt	Rd = Rs × Rt	0x1c	Rs^{5}	Rt^{5}	Rd^{5}	0	0x02
div	Rs, Rt	Hi, Lo = Rs / Rt	op^{6} = 0	Rs^{5}	Rt^{5}	0	0	0x1a
divu	Rs, Rt	Hi, Lo = Rs / Rt	op^{6} = 0	Rs^{5}	Rt^{5}	0	0	0x1b
mfhi	Rd	Rd = Hi	op^{6} = 0	0	0	Rd^{5}	0	0x10
mflo	Rd	Rd = Lo	op^{6} = 0	0	0	Rd^{5}	0	0x12