COE 301 / ICS 233
Computer Organization

Exam 2 – Spring 2017

Saturday, April 29, 2017
6:30 PM – 8:30 PM

Computer Engineering Department
College of Computer Sciences & Engineering
King Fahd University of Petroleum & Minerals

Student Name: **SOLUTION**_____

Student ID: _____

Section: _____

Q1	/ 15	Q2	/ 15
Q3	/ 15	Q4	/ 25
Q5	/ 15	Q6	/ 20
Total	/ 105		

Important Reminder on Academic Honesty

Using unauthorized information or notes on an exam, peeking at others work, or altering graded exams to claim more credit are severe violations of academic honesty. Detected cases will receive a failing grade in the course.

Question 1: Writing a Recursive Function in MIPS

(15 pts) Write a MIPS assembly-language function **`sum`** that receives two arguments: **`list[]`** and **`length`**, passed in **`$a0`** and **`$a1`**, respectively, computes **recursively** and returns the sum of the array elements in **`$f0`**. **`list[]`** is the **address** of an array of single-precision floats. The result of the function is a single-precision float.

```
float sum (float list[], int length) {
  if (length == 0) return 0;
  else return (list[0] + sum(&list[1], length-1));
}
```

SOLUTION:

```
sum:
 bne      $a1, $zero, else    # if (length != 0) else:
 sub.s    $f0, $f0, $f0       # $f0 = $f0 - $f0 = 0
 jr       $ra                 # return to caller
else:
 addiu    $sp, $sp, -8        # allocate stack frame = 8 bytes
 lwc1     $f0, 0($a0)         # load $f0 = list[0]
 sw       $ra, 0($sp)         # save return address
 swc1     $f0, 4($sp)         # save $f0 = list[0]
 addiu    $a0, $a0, 4         # $a0 = &list[1]
 addiu    $a1, $a1, -1        # $a1 = length–1
 jal      sum                 # recursive call
 lw       $ra, 0($sp)         # restore return address
 lwc1     $f1, 4($sp)         # restore $f1 = list[0]
 add.s    $f0, $f1, $f0       # $f0 = list[0] + sum(,)
 addiu    $sp, $sp, 8         # free stack frame
 jr       $ra                 # return to caller
```

Question 2: Greatest Common Divisor

(15 pts) The greatest common divisor of two integers **a** and **b** can be computed as follows:

```
gcd(a, 0) = a
gcd(a, b) = gcd(b, a % b)     where % is the remainder operator
```

For example,

```
gcd(30, 18) = gcd(18, 30%18) =
gcd(18, 12) = gcd(12, 18%12) =
gcd(12, 6)  = gcd(6,  12%6)  = gcd(6, 0) = 6
```

Write a MIPS assembly-language function that receives two integer arguments in **$a0** and **$a1**, computes and returns the greatest common divisor in **$v0**. Hint: use integer division and remainder in your computation, and write a loop to repeatedly compute the **gcd**.

SOLUTION:

```
gcd:
   bne    $a1, $zero, loop     # if (b != 0) branch to loop
   move   $v0, $a0             # $v0 = a
   jr     $ra                  # return to caller
loop:
   div    $a0, $a1             # divide a/b; HI = remainder = a%b
   move   $a0, $a1             # $a0 = b
   mfhi   $a1                  # $a1 = a % b (remainder)
   bne    $a1, $zero, loop     # loop if (remainder != 0)
   move   $v0, $a0             # $v0 = a
   jr     $ra                  # return to caller
```

Question 3: Sequential Signed Integer Multiplication

(15 pts) Given that the **Multiplicand** = **10100101** and the **Multiplier** = **10101101** are signed 2's complement numbers, show the **signed** multiplication of the **Multiplicand** by the **Multiplier**. The result of the multiplication should be a **16-bit signed** number in **HI** and **LO** registers. Show the steps of your work for a full mark.

Iteration		Multiplicand	Sign	Product = HI, LO
0	Initialize	10100101		00000000, 10101101
1	LO[0]=1 ➔ ADD	10100101	1	10100101, 10101101
	Shift right arithmetic			11010010, 11010110
2	LO[0]=0 ➔ Do Nothing	10100101		11010010, 11010110
	Shift right arithmetic			11101001, 01101011
3	LO[0]=1 ➔ ADD	10100101	1	10001110, 01101011
	Shift right arithmetic			11000111, 00110101
4	LO[0]=1 ➔ ADD	10100101	1	01101100, 00110101
	Shift right arithmetic			10110110, 00011010
5	LO[0]=0 ➔ Do Nothing	10100101		10110110, 00011010
	Shift right arithmetic			11011011, 00001101
6	LO[0]=1 ➔ ADD	10100101	1	10000000, 00001101
	Shift right arithmetic			11000000, 00000110
7	LO[0]=0 ➔ Do Nothing	10100101		11000000, 00000110
	Shift right arithmetic			11100000, 00000011
8	LO[0]=1 ➔ SUBTRACT	01011011	0	00111011, 00000011
	Shift right arithmetic			00011101, 10000001

Check: Multiplicand = 10100101 = -91
Multiplier = 10101101 = -83
Product = -91 × -83 = 7,553 = 00011101 10000001

Question 4: Floating-Point Numbers and Arithmetic

a) (4 pts) Find the **decimal value** of the following single-precision float:

S	Exponent	Fraction
1	1000 1110	000 0100 1100 0000 0110 0000

Decimal Value = $-1.000\ 0100\ 1100\ 0000\ 0110\ 0000 \times 2^{15}$

Decimal Value = -1000 0100 1100 0000.0110 0000

Decimal Value = -1000 0100 1100 0000.0110 0000 = -33984.375

b) (4 pts) Find the **decimal value** of the following single-precision float:

S	Exponent	Fraction
0	0000 0000	010 1100 0001 0000 0000 0000

Decimal Value = $0.010\ 1100\ 0001\ 0000\ 0000\ 0000 \times 2^{-126}$

Decimal Value = $0.010\ 1100\ 0001\ 0000\ 0000\ 0000 \times 2^{-126}$

Decimal Value = $10\ 1100\ 0001.0000\ 0000\ 0000 \times 2^{-137}$

Decimal Value = $705 \times 2^{-137} = 4.0465015 \times 10^{-39}$

c) (4 pts) Find the IEEE 754 single-precision representation of **-126.2**, rounded to the nearest even.

126.2 = 1111110.0011 0011 0011 0011 0011 ...

Normalize: 126.2 = $1.111\ 1100\ 0110\ 0110\ 0110\ 0110\ 0110 \ldots \times 2^{+6}$

Round: 126.2 = $1.111\ 1100\ 0110\ 0110\ 0110\ 0110 \times 2^{+6}$ (R=0, S=1)

IEEE Representation:

S	Exponent	Fraction
1	1000 0101	111 1100 0110 0110 0110 0110

d) (4 bits) **Normalize and Round** the given single-precision number with given GRS (Guard, Round, and Sticky) bits using the following four rounding modes. Show the final **normalized** number and its exponent:

GRS

$-0.111\ 1111\ 1111\ 1111\ 1111\ 1111\ 110 \times 2^{-12}$

Round towards Zero: $-1.111\ 1111\ 1111\ 1111\ 1111\ 1111 \times 2^{-13}$ (Always Truncate)

Round towards +Infinity: $-1.111\ 1111\ 1111\ 1111\ 1111\ 1111 \times 2^{-13}$ (Negative ➔ Truncate)

Round towards -Infinity: $-1.000\ 0000\ 0000\ 0000\ 0000\ 0000 \times 2^{-12}$ (RS=10 ➔ Inc + Renormalize)

Round towards Nearest Even: $-1.000\ 0000\ 0000\ 0000\ 0000\ 0000 \times 2^{-12}$ (Inc + Renormalize)

e) (9 pts) Given that **A** and **B** are single-precision floats, compute the difference **A−B**. Use rounding to **nearest even**. Perform the operation using **guard**, **round** and **sticky** bits.

A = +1.010 1001 1111 1010 0000 1101 × 2^{+3}

B = +1.001 1111 1010 0000 1110 0100 × 2^{-1}

A−B =

1.010 1001 1111 1010 0000 1101 × 2^{+3}

− 1.001 1111 1010 0000 1110 0100 × 2^{-1}

1.010 1001 1111 1010 0000 1101 × 2^{+3}

− 0.000 1001 1111 1010 0000 1110 010 × 2^{+3} (Right-shift)

1.010 1001 1111 1010 0000 1101 × 2^{+3}

+ 1.111 0110 0000 0101 1111 0001 110 × 2^{+3} (2's complement)

+ 1.001 1111 1111 1111 1111 1110 110 × 2^{+3} (positive result)

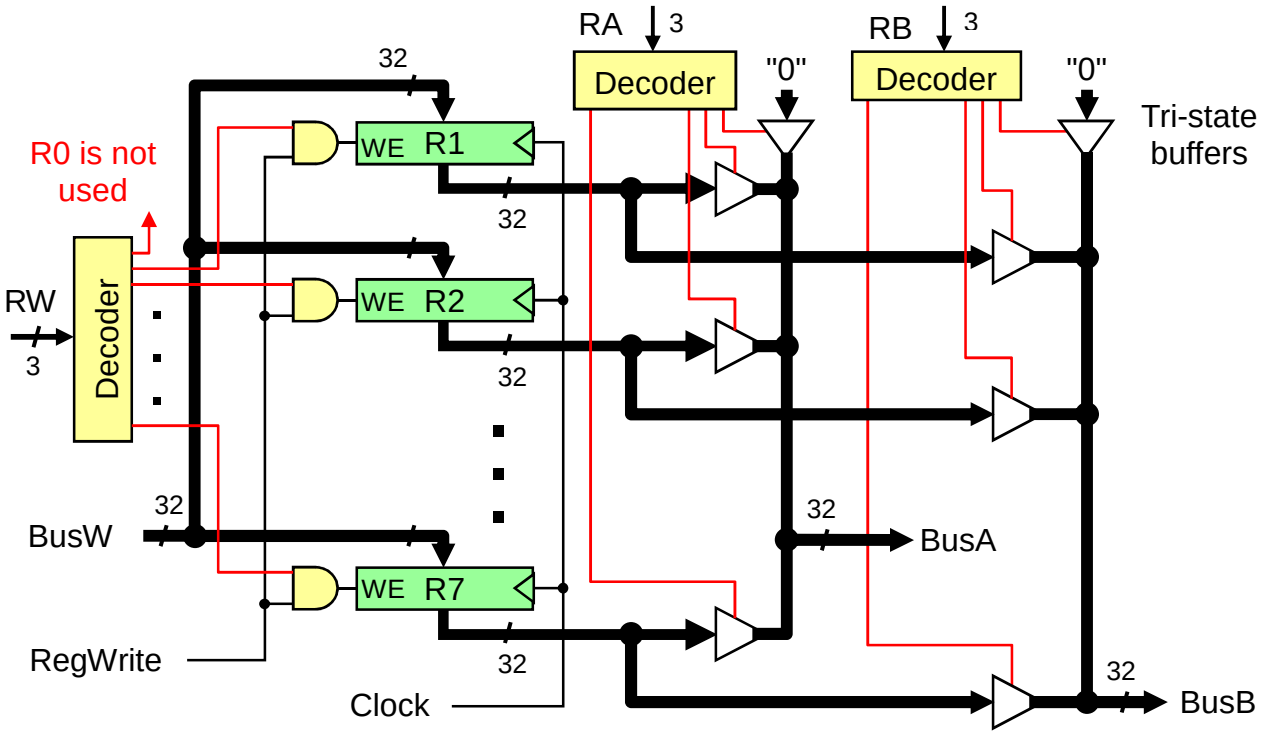
GRS

Result is already normalized, GRS bits = 110
Therefore, Increment Fraction

+ 1.001 1111 1111 1111 1111 1111 × 2^{+3} (Final result)

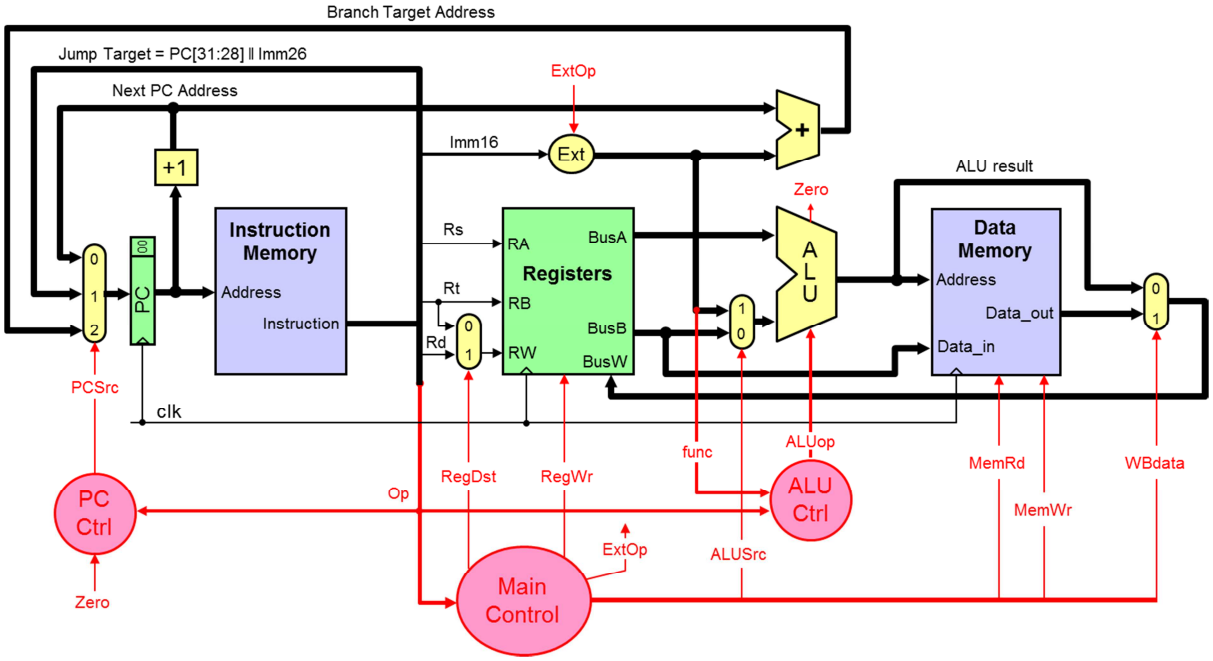
Question 5: Register File

(15 pts) Draw a register file having 7 registers only (R1 to R7) with two register read ports (Ra and Rb) and one register write port (Rw). R0 should be hardwired to zero and cannot be written. The register file should have two output data busses (BusA and BusB) and one input data bus (BusW). A control signal (RegWrite) should be used to enable the writing of the register file at the edge of the Clock signal.



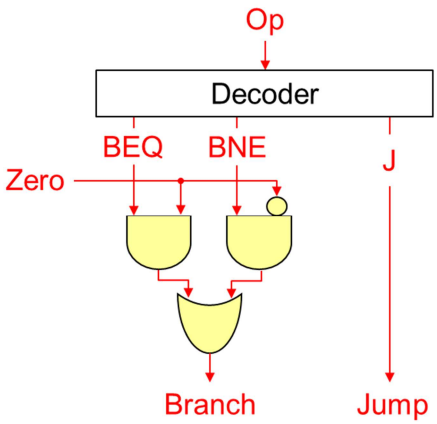
Question 6: Single-Cycle Datapath and Control

(20 pts) Consider the single-cycle datapath and control given below that implements a subset of the MIPS instruction set:



The PC control logic can be described as follows:

```
if (Op == J) PCSrc = 1;
else if ((Op == BEQ && Zero) || (Op == BNE && ~Zero)) PCSrc = 2;
else PCSrc = 0;
```



Branch, Jump = 00 ➔ PCSrc = 0
Branch, Jump = 01 ➔ PCSrc = 1
Branch, Jump = 10 ➔ PCSrc = 2

We wish to add the following instructions to the MIPS single-cycle datapath:

Instruction	Meaning	Format					
`jalr Rd, Rs`	`Rd = PC+4; PC = Rs`	Op = 0	Rs	0	Rd	0	f = 9
`movz Rd, Rs, Rt`	`if (Rt==0) Rd = Rs`	Op = 0	Rs	Rt	Rd	0	f = 10
`lwr Rd, Rs, Rt`	`Rd = MEM[Rs+Rt]`	Op = 0	Rs	Rt	Rd	0	f = 48

a) (10 pts) **Redraw** the single-cycle datapath. Show and describe any necessary modifications to the datapath and control signals needed for the implementation of the above three instructions.

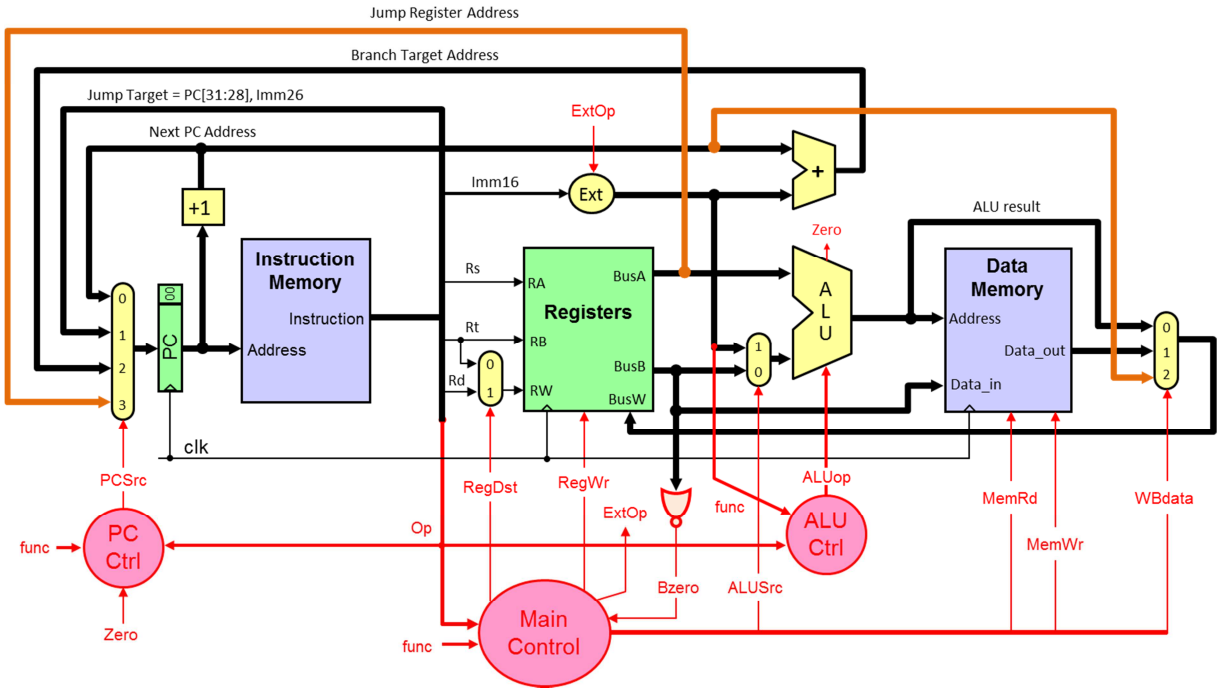
b) (10 pts) Draw a table showing the values of **ALL control signals** needed for the implementation of the above three instructions. Describe any changes in the main control and PC control needed for the implementation of the above three instructions.

Solution a: The modified single-cycle datapath is shown below.

For JALR, a larger multiplexer is added at the PC input. The Jump Register Address is taken from BusA and fed back to input 3 of the PCSrc multiplexer. The PC Ctrl also depends on the function. Similarly, a larger multiplexer is needed for Write Back (WBdata). The Next PC address is connected to input 2 of the WBdata multiplexer.

For MOVZ, the value of register Rt (on BusB) is detected whether it is zero (Bzero). A 32-input NOR gate is used to generate the Bzero signal (whether BusB is zero). The Zero flag of the ALU cannot be used because it indicates whether the ALU result is zero. The RegWr signal now depends also on the function code and the Bzero signal.

For LWR, no change in the datapath is required. Only changes in the control signals.



Solution b: Control Signals

Op	f	PCSrc	RegDst	RegWr	ExtOp	ALUSrc	ALUop	MemRd	MemWr	WBdata
0	JALR=9	BusA=3	Rd=1	1	X	X	X	0	0	RA=2
0	MOVZ=10	NextPC=0	Rd=1	Bzero	X	X	MoveA	0	0	ALU=0
0	LWR=48	NextPC=0	Rd=1	1	X	BusB=0	ADD	1	0	DM=1

The Main control now depends on the Opcode, function code, and the Bzero signal. The PC control also depends on the function code. For JALR, PCSrc = 3 (BusA) and WBdata = 2 (return address). For MOVZ, RegWr = Bzero (enabled if Bzero is 1), ALUop = MoveA (ALU result = BusA), and WBdata = 0 (ALU result). Finally, LWR is similar to LW except that the memory address is computed as Reg(Rs) + Reg(Rt). This is why ALUSrc = BusB, ALUop is ADD, MemRd = 1, and WBdata = 1 (Data memory).