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: ______________________________

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.

Prepared by Dr. Muhamed Mudawar and Dr. Marwan Abu-Amara

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));
}
```

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**.

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			
1				
2				
3				
4				
5				
6				
7				
8				

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

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

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

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 × 2^{-12}

Round towards Zero:

Round towards +Infinity:

Round towards -Infinity:

Round towards Nearest Even:

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 × 10^{+3}

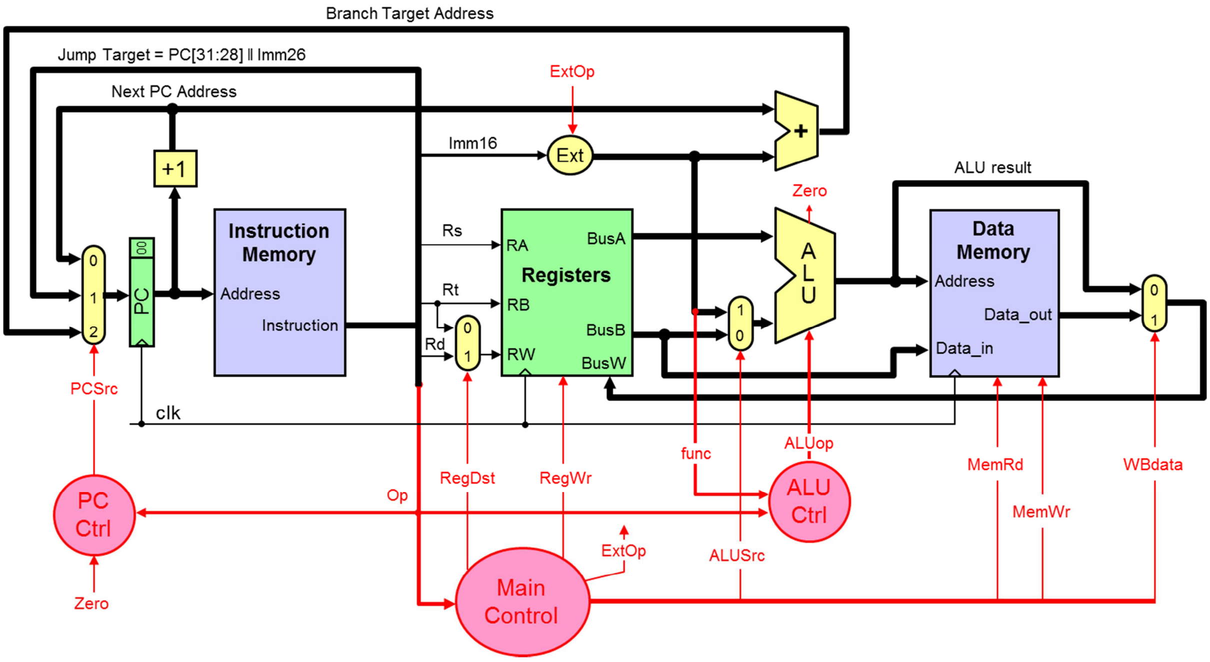
B = +1.001 1111 1010 0000 1110 0100 × 10^{-1}

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.

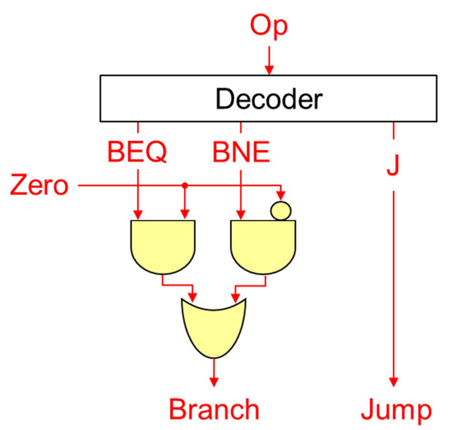
Question 6: Single-Cycle Datapath and Control

(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.

Additional Page if needed

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

Instruction	Meaning	Format					
add.s fd, fs, ft	(fd) = (fs) + (ft)	0x11	0	ft^5	fs^5	fd^5	0
add.d fd, fs, ft	(fd) = (fs) + (ft)	0x11	1	ft^5	fs^5	fd^5	0
sub.s fd, fs, ft	(fd) = (fs) – (ft)	0x11	0	ft^5	fs^5	fd^5	1
sub.d fd, fs, ft	(fd) = (fs) – (ft)	0x11	1	ft^5	fs^5	fd^5	1
mul.s fd, fs, ft	(fd) = (fs) × (ft)	0x11	0	ft^5	fs^5	fd^5	2
mul.d fd, fs, ft	(fd) = (fs) × (ft)	0x11	1	ft^5	fs^5	fd^5	2
div.s fd, fs, ft	(fd) = (fs) / (ft)	0x11	0	ft^5	fs^5	fd^5	3
div.d fd, fs, ft	(fd) = (fs) / (ft)	0x11	1	ft^5	fs^5	fd^5	3
sqrt.s fd, fs	(fd) = sqrt (fs)	0x11	0	0	fs^5	fd^5	4
sqrt.d fd, fs	(fd) = sqrt (fs)	0x11	1	0	fs^5	fd^5	4
abs.s fd, fs	(fd) = abs (fs)	0x11	0	0	fs^5	fd^5	5
abs.d fd, fs	(fd) = abs (fs)	0x11	1	0	fs^5	fd^5	5
neg.s fd, fs	(fd) = – (fs)	0x11	0	0	fs^5	fd^5	7
neg.d fd, fs	(fd) = – (fs)	0x11	1	0	fs^5	fd^5	7

Instruction	Meaning	Format			
lwc1 $f2, 40($t0)	($f2) = Mem[($t0)+40]	0x31	$t0	$f2	im^{16} = 40
ldc1 $f2, 40($t0)	($f3$f2) = Mem[($t0)+40]	0x35	$t0	$f2	im^{16} = 40
swc1 $f2, 40($t0)	Mem[($t0)+40] = ($f2)	0x39	$t0	$f2	im^{16} = 40
sdc1 $f2, 40($t0)	Mem[($t0)+40] = ($f3$f2)	0x3d	$t0	$f2	im^{16} = 40

Instruction	Meaning	Format					
mfc1 $t0, $f2	($t0) = ($f2)	0x11	0	$t0	$f2	0	0
mtc1 $t0, $f2	($f2) = ($t0)	0x11	4	$t0	$f2	0	0
mov.s $f4, $f2	($f4) = ($f2)	0x11	0	0	$f2	$f4	6
mov.d $f4, $f2	($f5$f4)=($f3$f2)	0x11	1	0	$f2	$f4	6

Instruction	Meaning	Format					
cvt.s.w fd, fs	to single from integer	0x11	0	0	fs^5	fd^5	0x20
cvt.s.d fd, fs	to single from double	0x11	1	0	fs^5	fd^5	0x20
cvt.d.w fd, fs	to double from integer	0x11	0	0	fs^5	fd^5	0x21
cvt.d.s fd, fs	to double from single	0x11	1	0	fs^5	fd^5	0x21
cvt.w.s fd, fs	to integer from single	0x11	0	0	fs^5	fd^5	0x24
cvt.w.d fd, fs	to integer from double	0x11	1	0	fs^5	fd^5	0x24

Instruction	Meaning	Format					
c.eq.s fs, ft	cflag = ((fs) == (ft))	0x11	0	ft^5	fs^5	0	0x32
c.eq.d fs, ft	cflag = ((fs) == (ft))	0x11	1	ft^5	fs^5	0	0x32
c.lt.s fs, ft	cflag = ((fs) < (ft))	0x11	0	ft^5	fs^5	0	0x3c
c.lt.d fs, ft	cflag = ((fs) < (ft))	0x11	1	ft^5	fs^5	0	0x3c
c.le.s fs, ft	cflag = ((fs) <= (ft))	0x11	0	ft^5	fs^5	0	0x3e
c.le.d fs, ft	cflag = ((fs) <= (ft))	0x11	1	ft^5	fs^5	0	0x3e
bc1f Label	branch if (cflag == 0)	0x11	8	0	im^{16}		
bc1t Label	branch if (cflag == 1)	0x11	8	1	im^{16}		

Prepared by Dr. Muhamed Mudawar and Dr. Marwan Abu-Amara