King Fahd University of Petroleum and Minerals
College of Computer Science and Engineering
Computer Engineering Department

COE 301 COMPUTER ORGANIZATION
ICS 233: COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE

Term 151 (Fall 2015-2016)
Major Exam 2
Saturday Nov. 21, 2015

Time: 120 minutes, Total Pages: 15

Name:_KEY____________________________ ID:____________________ Section: ________

Notes:

- Do not open the exam book until instructed
- Answer all questions
- All steps must be shown
- Any assumptions made must be clearly stated

Question	Max Points	Score
Q1	16	
Q2	20	
Q3	20	
Q4	30	
Total	86	

Dr. Aiman El-Maleh
Dr. Mayez Al-Muhammad

[16 Points]

(Q1)

(i) **[6 points]** A recursive procedure TH(N) returns 1+2TH(N-1) for N >1, 1 if N=1, and zero otherwise. This is called Tower of Hanoi. TH(N) is defined as follows:

```
int TH(int N) {
    if (N =< 0) return 0;
   else if (N=1) return 1;
   else return (1 + 2*TH(N-1));
}
```

Assume TH receives its argument N in register $a0 and return its results in $v0. The above procedure is called from some Main program, which <u>needs **not** to be</u> implemented here. Write a minimal MIPS program for the above procedure.

<u>**Solution:**</u>

```
TH:      slti   $t0, $a0, 1     # (n< 1)?
         beq    $t0,$0, next    # if false branch to next
         li     $v0,0           # $v0 = 0
         jr     $ra             # return to caller
next:    slti   $t0, $a0, 2     # (n < 2)?
         bne    $t0,$0,iterate  # if false branch to iterate
         li     $v0,1           # $v0 = 1
         jr     $ra             # return to caller

iterate: addiu  $sp,$sp,-4      # allocate 1 word on stack
         sw     $ra,0($sp)      # save return address
         addiu  $a0,$a0,-1      # argument = n-1
         jal    TH              # call TH(n-1)
         sll    $v0, $v0, 1     # $v0 = 2*TH(n-1)
         addi   $v0, $v0, 1     # return 1 +2*TH(n-1)
         lw     $ra,0($sp)      # restore return address
         addi   $sp, $sp, 4     # free stack frame
         jr     $ra             # return to caller
```

(ii) **[10 points]** Suppose we enter i integers q(1), q(2), …, q(i). The objective is to compute the result p(i) = q(1) +… + q(i) for each i, where p is an array of results. A better way to compute the results is p(i) = p(i-1)+ q(i) for i >=1 after setting p(0)=0. The above function is called prefix sum. For example, if we enter 4, 3, 5, 2, 3, 0 (termination) as follows:

Order of entries	1	2	3	4	5	6	
Value of entries q:	4	3	5	2	3	0	then the results will be:
Value of results p:	4	7	12	14	17		

Assume the following strings in the data segments:

```
prompt-1:     .asciiz   "Please enter at most 100 singed integers terminating
                         with 0: \n"
prompt-2:     .asciiz   "Prefix sum of the entered integers: \n"
```

Use $s0 to store the address of array of words p as a base address and $s1 to store the number of entered integers by the user.

Write a MIPS program with minimal instructions that carries out the following steps:

1. Print “prompt-1”,

2. Reads at most 100 signed integers q(i) terminated with a zero,

3. Compute the results p(i) and store them in memory,

4. Print “prompt-2”, and

5. Print all the results p(i).

Solution:

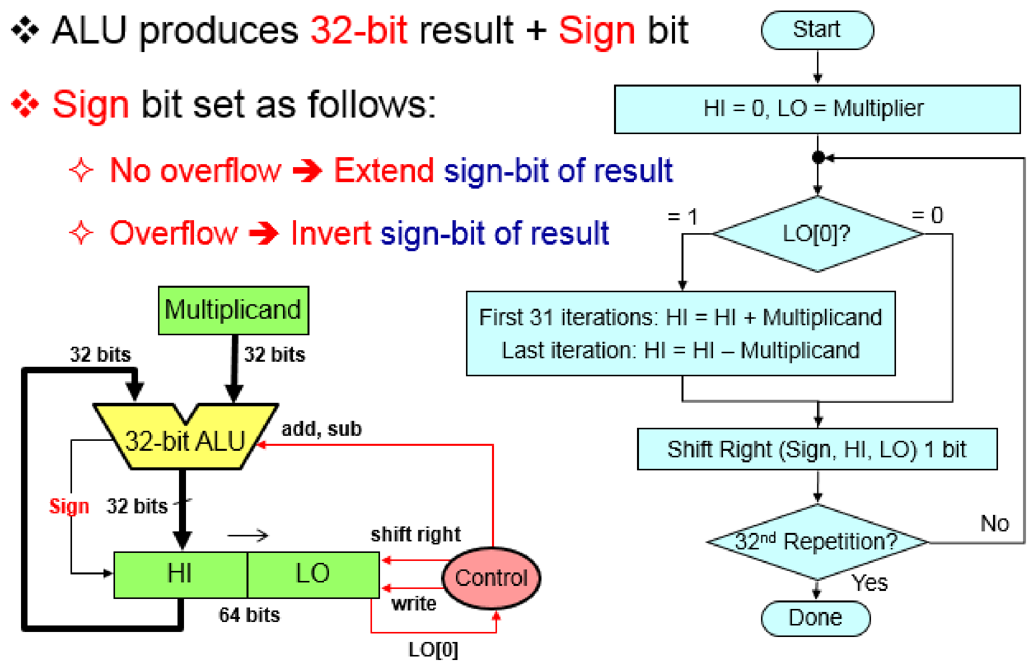
```
          move    $s2, $s0          # $s2 is pointer to array p
          li    $s1,0             # Current count $s1 = 0
          li    $s3, 100          # initialize $s3 to 100

          la   $a0, prompt-1      # display prompt-1 as a null-terminated string
          li    $v0,4             # print null-terminated string
          syscall                 # system call

loop1:    $v0, 5                  # Loop reading at most 100 integers
          syscall
          sw    $v0, 0($s2)       # store q(i)
          beq   $v0, $zero, comp  # check if 0 integers are entered
          addi  $s2, $s2, 4       # update pointer q(i)
          addi  $s1, $s1, 1       # increment count
          blt   $s1, $s3, loop1   # branch if count < 100

comp:     move     $s2, $s0       # restore base address of p in $s2
```

```
                li      $t0, 0                  # initialize p(0)=0
loop2:          lw     $t1, 0($s2)          # load q(i)
                beq    $t1, $zero, print   # branch if this a terminating 0
                add    $t0, $t1, $t0        # compute p(i) = p(i-1) + q(i)
                sw     $t0, 0($s2)          # store p(i) at location q(i)
                addi   $s2, $s2, 4         # update pointer
                J        loop2

print:        la    $a0, prompt-2      # display prompt-2 as a null-terminated string
                li     $v0,4                    # print null-terminated string
                syscall                         # system call

                move       $s2, $s0              # restore base address of p in $s2
loop3:          li       $v0, 1                 # print integer
                lw     $a0, 0($s2)         # load p(i)
                syscall
                addi   $s2, $s2, 4          # update pointer q(i)
                bne   $a0, $zero, loop3      # continue if this is not 0 terminating integer
```

[20 Points]

(Q2)

(i) **[10 points]** You are required to design a circuit that can be used to perform **signed** multiplication of two 32-bit operands A and B. Show the block diagram of all used components and their sizes. Explain how the circuit will be used to perform signed multiplication by showing a flow chart or pseudo code.



❖ ALU produces 32-bit result + Sign bit

❖ Sign bit set as follows:

- ✧ No overflow ➔ Extend sign-bit of result
- ✧ Overflow ➔ Invert sign-bit of result

(ii) **[4 points]** Given that **Multiplicand=1001** and **Multiplier=1011,** using the **signed multiplication** hardware, show the **signed** multiplication of **Multiplicand** by **Multiplier**. The result of the multiplication should be an 8 bit **signed** number in HI and LO registers. Show the steps of your work.

Iteration		**Multiplicand**	**Sign**	**Product = HI,LO**
0	Initialize (LO = Multiplier)	1001		0000 1011
1	LO[0] = 1 => ADD		1	1001 1011
	Shift Product = (HI, LO) right 1 bit	1001		1100 1101
2	LO[0] = 1 => ADD		1	0101 1101
	Shift Product = (HI, LO) right 1 bit	1001		1010 1110
3	LO[0] = 0 => Do nothing		1	1010 1110
	Shift Product = (HI, LO) right 1 bit	1001		1101 0111
4	LO[0] = 1 => SUB (ADD 2's compl)	0111	0	0100 0111
	Shift Product = (HI, LO) right 1 bit			**0010 0011**

(iii) **[6 points]** Given that **Dividend=1001** and **Divisor=0011** represent two **4-bit signed numbers in 2's complement representation**, using the **unsigned division** hardware, show the **signed** division of **Dividend** by **Divisor**. The result of division should be stored in the Remainder and Quotient registers. Show the steps of your work.

Since the Dividend is negative, we take its 2's complement => Dividend = 0111
Sign of Quotient = negative, Sign of Remainder = negative

Iteration		**Remainder (HI)**	**Quotient (LO)**	**Divisor**	**Difference**
0	Initialize	0000	0111	0011	
1	1: SLL, Difference	0000	1110	0011	1101
	2: Diff < 0 => Do Nothing	0000	1110	0011	
2	1: SLL, Difference	0001	1100	0011	1110
	2: Diff < 0 => Do Nothing	0001	1100	0011	
3	1: SLL, Difference	0011	1000	0011	0000
	2: Rem = Diff, set lsb Quotient	**0000**	1001	0011	
4	1: SLL, Difference	0001	0010	0011	1110
	2: Diff < 0 => Do Nothing	**0001**	**0010**	0011	

Thus, the Quotient = 1110 and Remainder = 1111.

[20 Points]

(Q3)

1. **[2 Points]** What is the **decimal value** of following single precision float:

   [1, 1000 0101, 0101 1101 0000 0000 0000 000]

   = - $(1.0101110100000000...0)_2 * 2^{(133-127)}$ = - $(1.0101110100000000...0)_2 * 2^6$
   = -1010111.01 = -87.25

2. **[2 Points]** What is the **decimal value** of following single precision float:

   [0, 0000 0000, 0100 0000 0000 0000 0000 000]

   = + $(0.010000000000...0)_2 * 2^{-126}$ = $+2^{-128}$

3. **[3 Points]** Find the normalized single precision float representation of +59.25.

   59.25 = 111011.01 = $1.1101101 * 2^5$

   Exponent = 5 + 127 = 132

   [0, 1000 0100, 1101 1010 0000 0000 0000 000]

4. **[4 Points]** Round the given single precision float with the given GRS bits using the following rounding modes showing the resulting normalized number:

```
                                     GRS
-1.111 1111 1111 1111 1111 1111 100 x 2^23
```

Zero:	[**-1.111 1111 1111 1111 1111 1111 x** 2^{23}]
+infinity:	[**-1.111 1111 1111 1111 1111 1111 x** 2^{23}]
-infinity:	[**-1.000 0000 0000 0000 0000 0000 x** 2^{24}]
Nearest Even:	[**-1.000 0000 0000 0000 0000 0000 x** 2^{24}]

5. **[5 Points]** Find the normalized difference between A and B by using rounding to nearest even. Perform the operation using **guard**, **round** and **sticky** bits:

A= + 1.00000000000000000000000 × 2^4
B = +1.11110000000000000000001 × 2^3

```
        1.000 0000 0000 0000 0000 0000 000 x 2^4
  -     1.111 1000 0000 0000 0000 0001 000 x 2^3
-----------------------------------------------------------
       01.000 0000 0000 0000 0000 0000 000 x 2^4
  -    00.111 1100 0000 0000 0000 0000 100 x 2^4  (align)
-----------------------------------------------------------
       01.000 0000 0000 0000 0000 0000 000 x 2^4
  +    11.000 0011 1111 1111 1111 1111 100 x 2^4 (2's complement)
-----------------------------------------------------------
       00.000 0011 1111 1111 1111 1111 100 x 2^4
  = +   0.000 0011 1111 1111 1111 1111 100 x 2^4
  = +   1.111 1111 1111 1111 1110 0000 000 x 2^-2 (normalize)
  = +  1.111 1111 1111 1111 1110 0000      x 2^-2 (round)
```

6. **[4 Points]** Find the normalized result of the operation A+B+C, by performing A+B first followed by adding C, using rounding to nearest even. Perform the operation using **guard**, **round** and **sticky** bits:

```
A= + 1.011 1110 0100 0000 0000 0000 000          x 2^32
B = +1.111 1000 0000 0000 0000 0000 000          x 2^4
C = -1.011 1110 0100 0000 0000 0000 000          x 2^32
```

Is the obtained result intuitive? Justify your answer.

```
        1.011 1110 0100 0000 0000 0000 000       x 2^32
  +     1.111 1000 0000 0000 0000 0000 000       x 2^4
-----------------------------------------------------------
        1.011 1110 0100 0000 0000 0000 000       x 2^32
  +     0.000 0000 0000 0000 0000 0000 001       x 2^32 (align)
  =     1.011 1110 0100 0000 0000 0000 001       x 2^32
-----------------------------------------------------------
```

Rounding the result to the nearest even gives the result:

```
  =    +1.011 1110 0100 0000 0000 0000           x 2^32 (round)
-----------------------------------------------------------

        1.011 1110 0100 0000 0000 0000           x 2^32
  -     1.011 1110 0100 0000 0000 0000           x 2^32
-----------------------------------------------------------

  =    +0.000 0000 0000 0000 0000 0000           x 2^32
```

Note that this number is equivalent to 0.

The result is counterintuitive because C=-A, and one expects that the result we will obtain will be equal to B. However, we got the result as 0. The main reason is that when we added A and B the result was A due to rounding as B is very small compared to A. Thus, when we add C=-A, we get 0.

[30 Points]

(Q4)

(i) **[3 Points]** The components of a Single Cycle Datapath have the following delays**:**

1. 150 ps for fetching the instruction from the Instruction Memory,
2. 100 ps for reading or writing the register file (in parallel with instruction decoding),
3. 50 ps for any ALU operation,
4. 200 ps for loading or storing using the data memory.

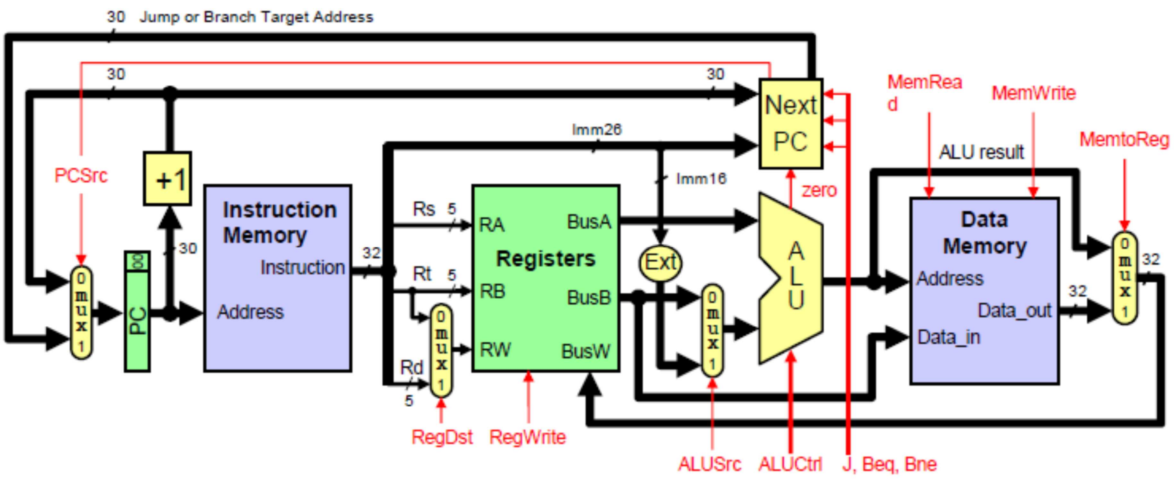
The datapath setup time is 30 ps, the hold time is 45 ps, and the clock skew time is 20 ps. Ignore the delay through the multiplexers and other logic. What is the shortest clock period for correct operation of MIPS assembly instructions. Evaluate the highest possible clock rate.

<u>**Solution:**</u>

The single cycle period is bounded: T =< Critical path (longest instruction execution) + Tsetup+ Thold +Tskew.

Hence, T =< (150+100+50+200+100) + 30 +45 +20 = 695 ps
The highest Clock Frequency is bounded by: CR =< 10^12 /695 Hz= 1.44 GHz

(ii) Consider the following MIPS datapath:



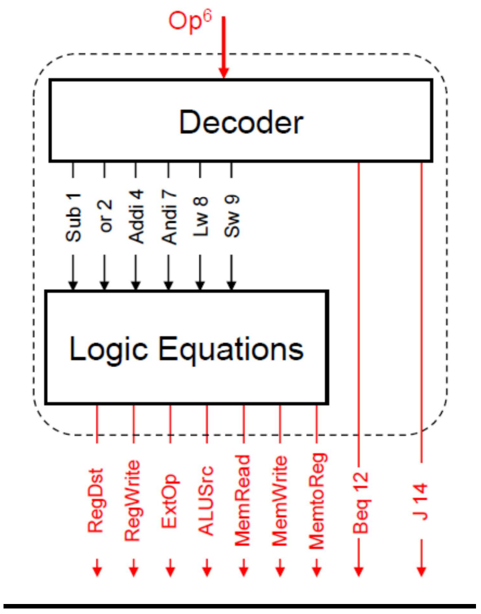
1. **[8 Points]** List the values of the datapath controls in the following Table depending for each instruction.

Type	Instr.	OPCODE $X_5 X_4 X_3 X_2 X_1 X_0$	RegDst	RegWrite	Ext	ALUSrc	ALUCtrl	MemRead	MemWrite	MemtoReg
R	**Sub**	**000001**	1	1	x	0	sub	0	0	0
	Or	**000010**	1	1	x	0	or	0	0	0
I	**Addi**	**000100**	0	1	1	1	add	0	0	0
	Andi	**000111**	0	1	0	1	and	0	0	0
	Lw	**001000**	0	1	1	1	add	1	0	1
	Sw	**001001**	x	0	1	1	add	0	1	x
	Beq	**001100**	x	0	x	0	sub	0	0	x
J	**J**	**001110**	x	0	x	x	x	0	0	x

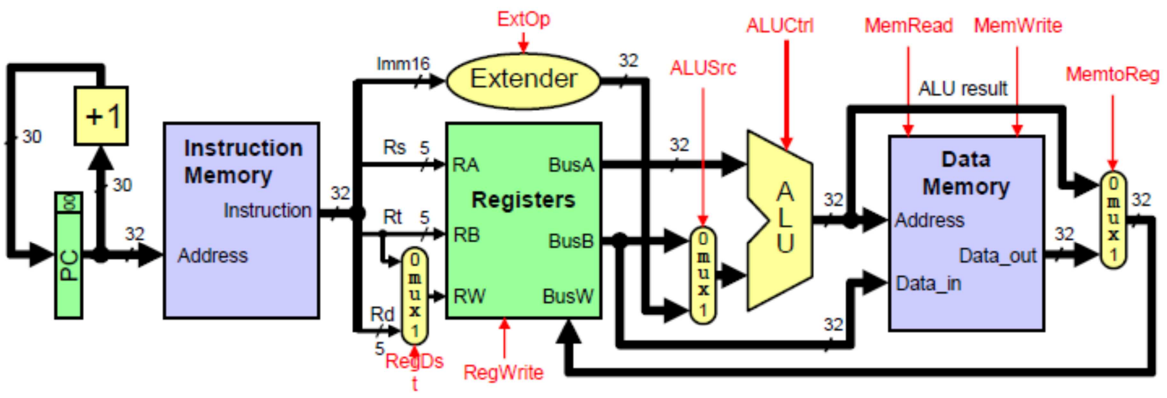
2. **[4 Points]** Design the control unit to generate the above control signals (except ALUCtrl) using the simplest logic.

Here are the controls:

RegDst =	sub + or
RegWrite =	Not (sw + beq + j)
Ext =	Not (andi)
ALUSrc =	Not (sub + or + Beq)
MemRead =	lw
MemWrite =	sw
MemtoReg =	lw



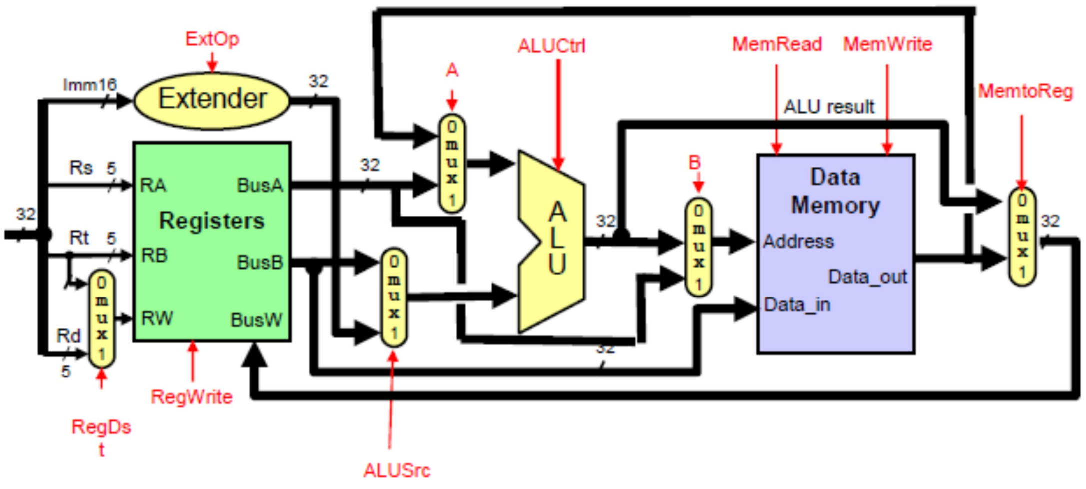
(iii) **[5 Points]** We would like to add a new instruction to the MIPS instruction set: **Addm rd, rt, rs** that performs (rd) ← DM[(rs)]+(rt). Draw the additional changes on the MIPS datapath shown below to enable the execution of Addm instruction and give the values of all the control signals in the modified datapath by filling the given table.



Two extra Multiplexers are needed:

a. DM Address input needs a MUX [BusA, ALUout] with control B=1 for Addm and 0 otherwise,
b. Upper Input of ALU needs a MUX[BusA, Data-out(DM)] with control A=0 for addm.

Here is the revised datapath with the above Muxes:

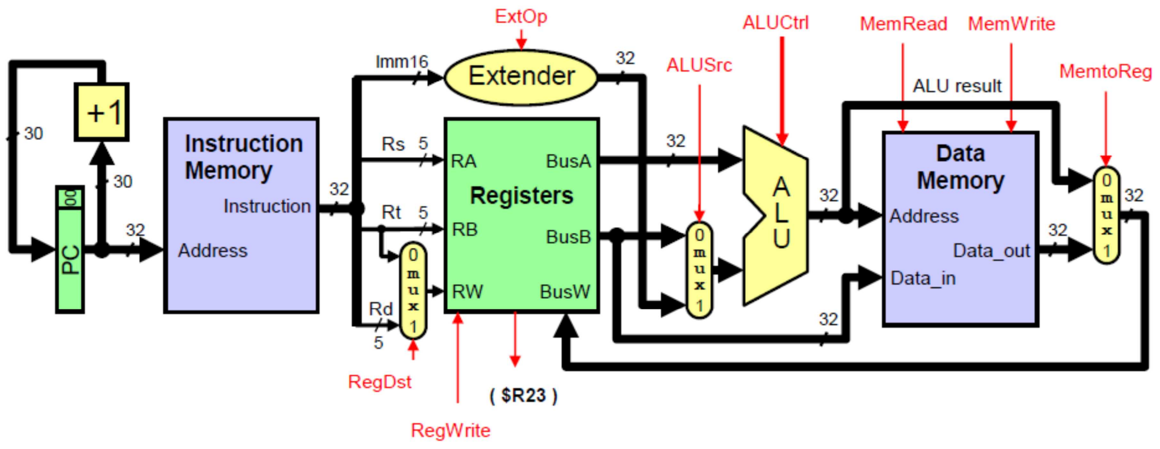


RegDst	RegWrite	Ext	ALUSrc	ALUCtrl	MemRead	MemWrite	MemtoReg	PCSrc	A	B
1	1	x	0	Add	1	0	0	0	**0**	**1**

c. **[5 Points]** Assume we want to add instructions to MIPS such as: Addp rd, rt, rs that performs (rd) ← (rs)+(rt) if $23=1, else register rd remains unchanged. This is called a predicated instruction that executes only if a predicate is true (register $23=1). We assume that:

1. Control signals generated by the control unit for Addp are identical to those generated for Add rd, rt, rs instruction.
2. The Control unit generates a signal S=1 only for predicated instructions.
3. The content of register $23 is always output by the Registers (see the below drawing).

Draw the additional changes on the MIPS datapath to enable correct execution of Addp instruction and give the values of all the control signals in the modified datapath by filling the given table.



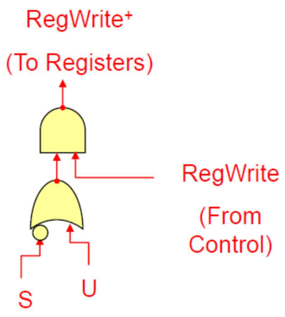
Solution:

The RegWrite that is generated by the control needs to be modified (call it RegWrite$^+$). Let's AND the content of $23 with constant 0x1 and let the output be U. Hence, U=1 only if #23 contains 0x1. Now RegWrite$^+$ is function of RegWrite as generated by control for the previous set of instructions, S and U. RegWrite$^+$ is designed as follows:

(S, U)

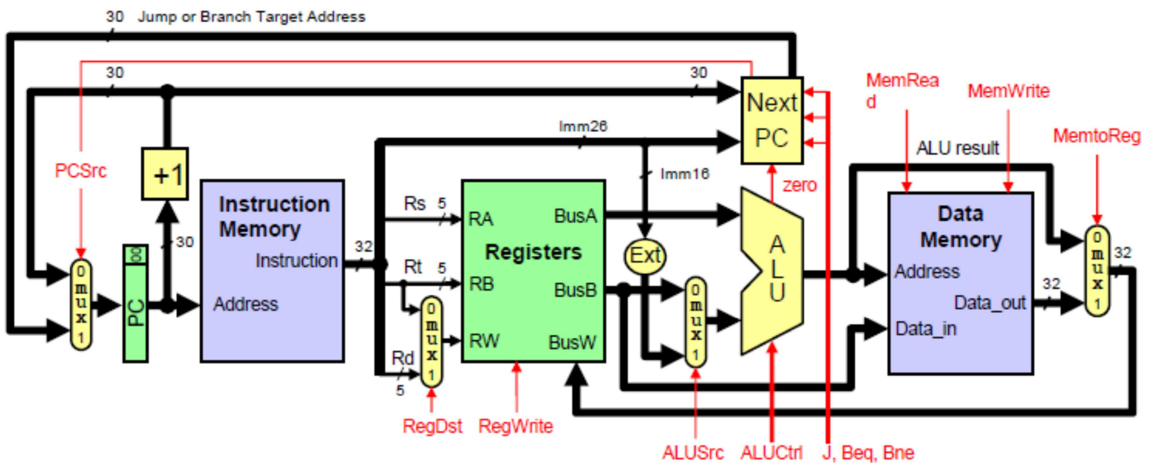
RegWrite	00	01	11	10
0	0	0	0	0
1	1	1	1	0

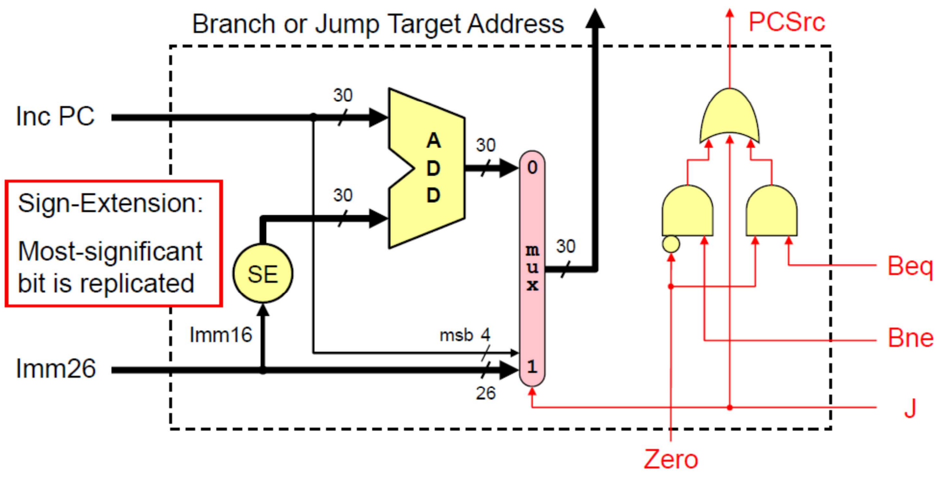
Hence, RegWrite$^+$ must be implement as RegWrite$^+$ = RegWrite AND [Not(S) OR U] to enable implementing the predicated instructions.



RegDst	RegWrite	Ext	ALUSrc	ALUCtrl	MemRead	MemWrite	Memto Reg	
1	1	x	0	r	0	0	0	

d. **[5 Points]** Assume we want to add the instruction JAL to the MIPS datapath. Make all the necessary modifications to the MIPS Datapath for implementing the JAL instruction including the NextPC block. The NextPC block implementation is given below.



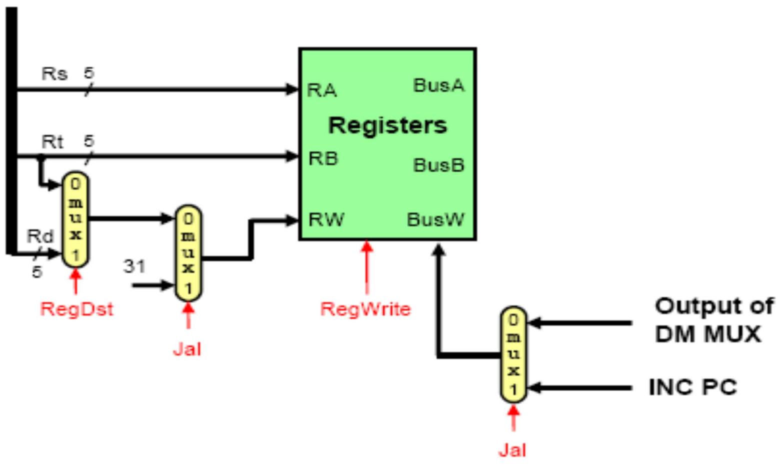


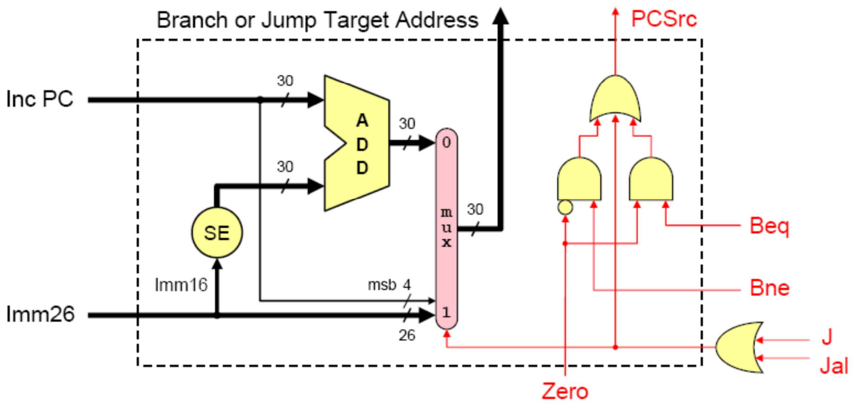
Solution

JAL implement the following register transfers: \$31 ← PC+4 and PC ← [PC31-28, IMM-26, 00]. We need two MUXes to implement \$31 ← PC+4:

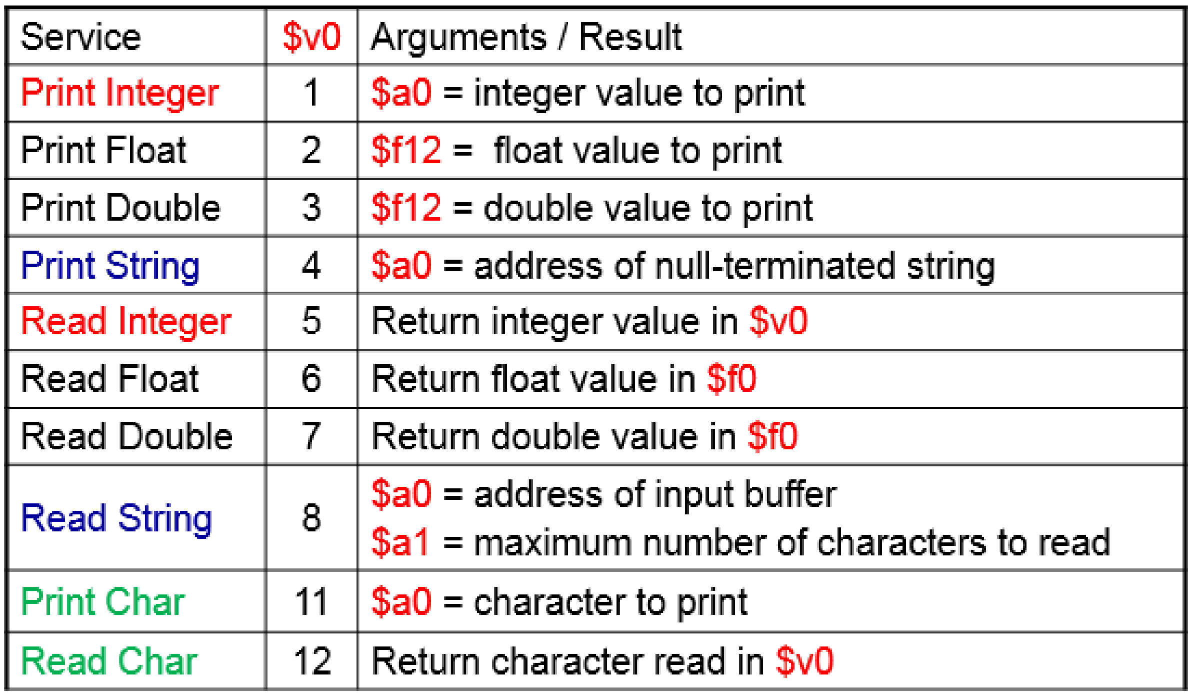
1. A MUX is needed to write register \$31 as input address 31 on the R/W of Registers,
2. A MUX is needed to input PC+4 on BusW of Registers when JAL=1.

Since JAL and J update PC in the same way. JAL (decoder output) is to be ORed with J for updating PCSrc.


Rs 5
Rt 5
Rd 5
31
RA
RB
RW
BusA
BusB
BusW
Registers
0 mux 1
RegDst
Jal
RegWrite
Output of DM MUX
INC PC
Jal


Branch or Jump Target Address
PCSrc
Inc PC
Imm26
Imm16
SE
ADD
30
msb 4
26
0 mux 1
Beq
Bne
J
Jal
Zero

<u>Syscall Services:</u>



Service	$v0	Arguments / Result
Print Integer	1	$a0 = integer value to print
Print Float	2	$f12 = float value to print
Print Double	3	$f12 = double value to print
Print String	4	$a0 = address of null-terminated string
Read Integer	5	Return integer value in $v0
Read Float	6	Return float value in $f0
Read Double	7	Return double value in $f0
Read String	8	$a0 = address of input buffer $a1 = maximum number of characters to read
Print Char	11	$a0 = character to print
Read Char	12	Return character read in $v0

MIPS Instructions:

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

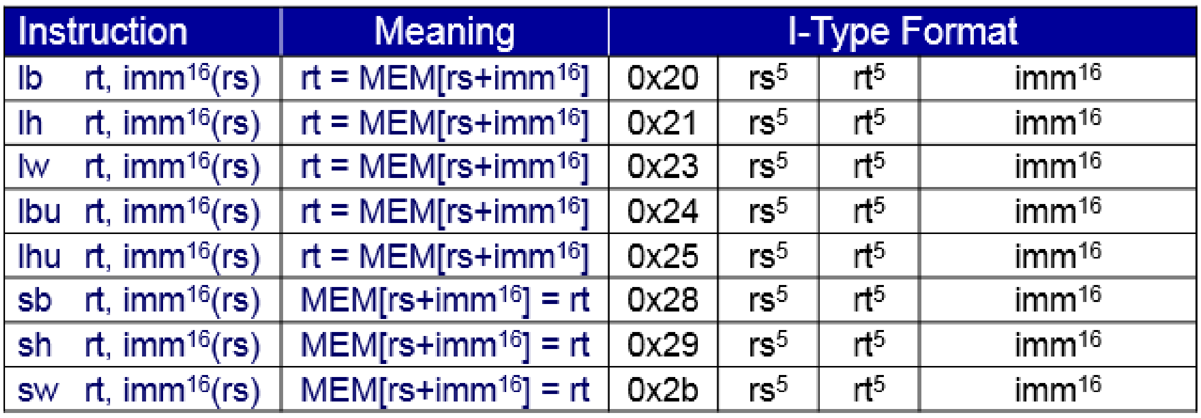
Instruction	Meaning	R-Type Format					
and $s1, $s2, $s3	$s1 = $s2 & $s3	op = 0	rs = $s2	rt = $s3	rd = $s1	sa = 0	f = 0x24
or $s1, $s2, $s3	$s1 = $s2 | $s3	op = 0	rs = $s2	rt = $s3	rd = $s1	sa = 0	f = 0x25
xor $s1, $s2, $s3	$s1 = $s2 ^ $s3	op = 0	rs = $s2	rt = $s3	rd = $s1	sa = 0	f = 0x26
nor $s1, $s2,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

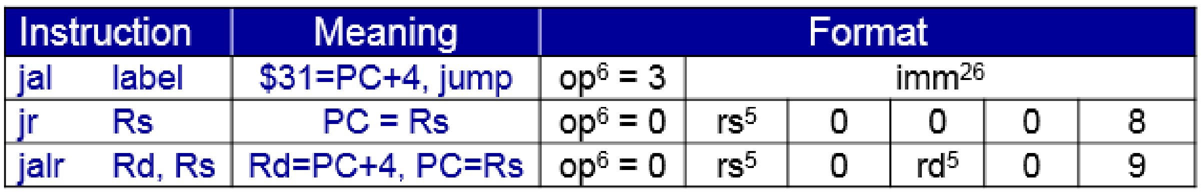
Instruction	Meaning	I-Type Format			
addi $s1, $s2, 10	$s1 = $s2 + 10	op = 0x8	rs = $s2	rt = $s1	imm^{16} = 10
addiu $s1, $s2, 10	$s1 = $s2 + 10	op = 0x9	rs = $s2	rt = $s1	imm^{16} = 10
andi $s1, $s2, 10	$s1 = $s2 & 10	op = 0xc	rs = $s2	rt = $s1	imm^{16} = 10
ori $s1, $s2, 10	$s1 = $s2 | 10	op = 0xd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

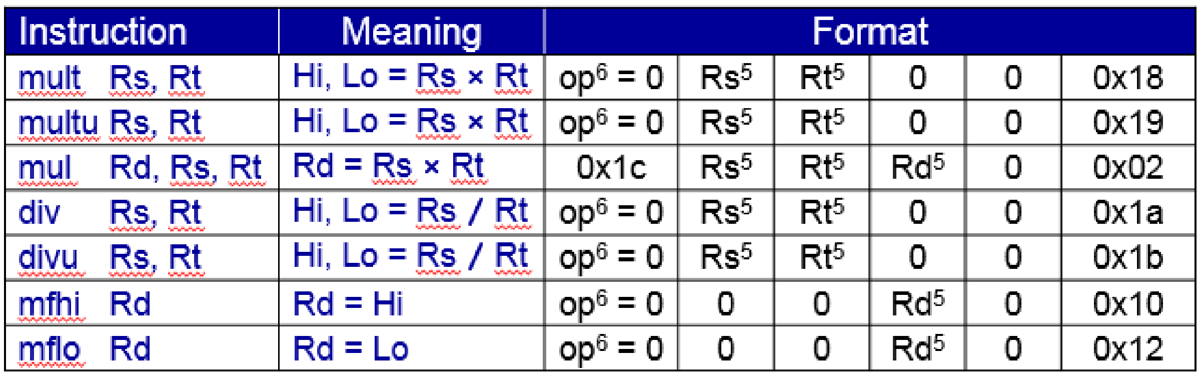
Instruction	Meaning	Format					
slt rd, rs, rt	rd=(rs<rt?1:0)	op^6 = 0	rs^5	rt^5	rd^5	0	0x2a
sltu rd, rs, rt	rd=(rs<rt?1:0)	op^6 = 0	rs^5	rt^5	rd^5	0	0x2b
slti rt, rs, imm^{16}	rt=(rs<imm?1:0)	0xa	rs^5	rt^5	imm^{16}		
sltiu rt, rs, imm^{16}	rt=(rs<imm?1:0)	0xb	rs^5	rt^5	imm^{16}		



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