August 23, 2007

COMPUTER ENGINEERING DEPARTMENT

ICS 233

COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE

Final Exam

Summer Semester (063)

Time: 7:30-10:30 AM

Student Name : _KEY_____

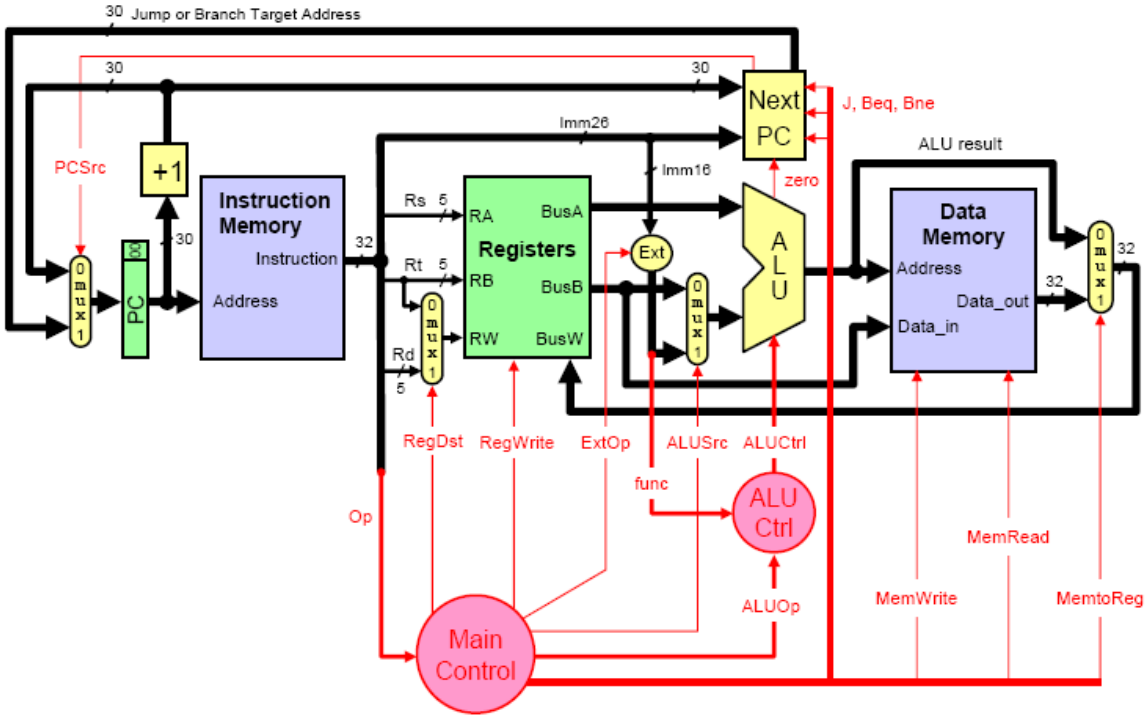
Student ID. : _____

Question	Max Points	Score
Q1	30	
Q2	14	
Q3	10	
Q4	14	
Q5	18	
Q6	14	
Total	100	

Dr. Aiman El-Maleh

[30 Points]

(Q1) Consider the single-cycle datapath and control given below for the MIPS processor implementing a subset of the instruction set:



(i) Show the control signals generated for the execution of the following instructions by filling the table given below:

Op	RegDst	RegWrite	ExtOp	ALUSrc	ALUOp	Beq	Bne	J	MemRead	MemWrite	MemtoReg
R-type	1 = Rd	1	x	0=BusB	R-type	0	0	0	0	0	0
Addi	0 = Rt	1	1=sign	1=Imm	ADD	0	0	0	0	0	0
Andi	0 = Rt	1	0=zero	1=Imm	AND	0	0	0	0	0	0
Lw	0 = Rt	1	1=sign	1=Imm	ADD	0	0	0	1	0	1
Sw	x	0	1=sign	1=Imm	ADD	0	0	0	0	1	x
Beq	x	0	x	0=BusB	SUB	1	0	0	0	0	x
J	x	0	x	x	x	0	0	1	0	0	x

The format of these instructions is given below for your reference:

Instruction		Meaning	Format					
add	rd, rs, rt	addition	$Op^6 = 0$	rs^5	rt^5	rd^5	0	0x20
addi	rt, rs, imm^{16}	add immediate	0x08	rs^5	rt^5	imm^{16}		
andi	rt, rs, imm^{16}	and immediate	0x0c	rs^5	rt^5	imm^{16}		
lw	rt, imm^{16}(rs)	load word	0x23	rs^5	rt^5	imm^{16}		
sw	rt, imm^{16}(rs)	store word	0x2b	rs^5	rt^5	imm^{16}		
beq	rs, rt, label	branch if equal	0x04	rs^5	rt^5	imm^{16}		
j	label	jump	0x02	imm^{26}				

(ii) We wish to add the following instructions to the single-cycle datapath. Add any necessary datapath and control signals needed for the implementation of these instructions. Show only the modified and added components to the datapath. Show the values of the control signals to control the execution of each instruction.

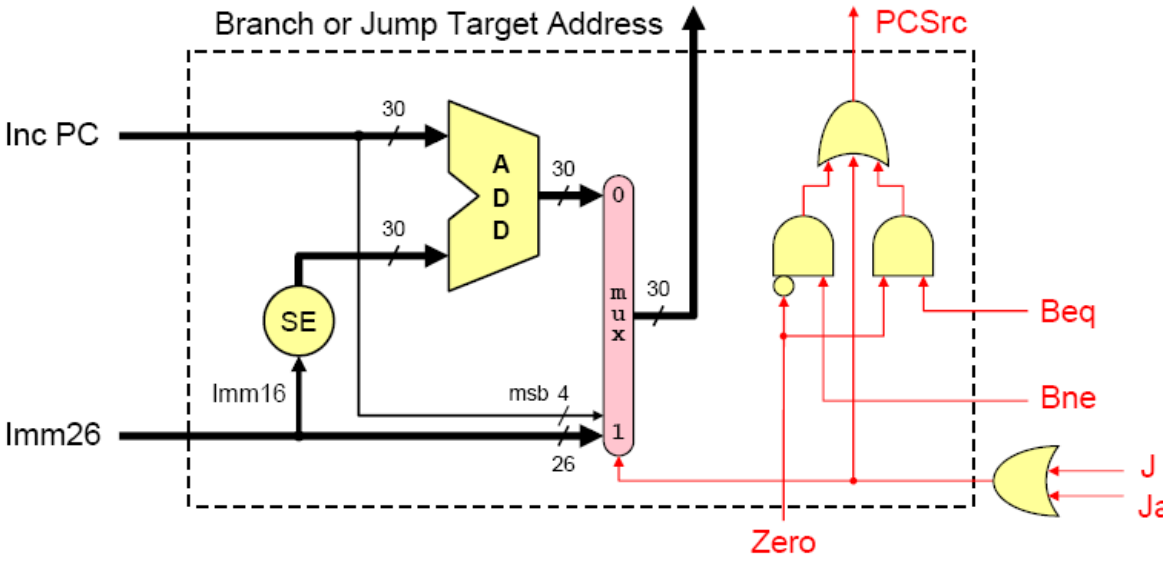
a. jal

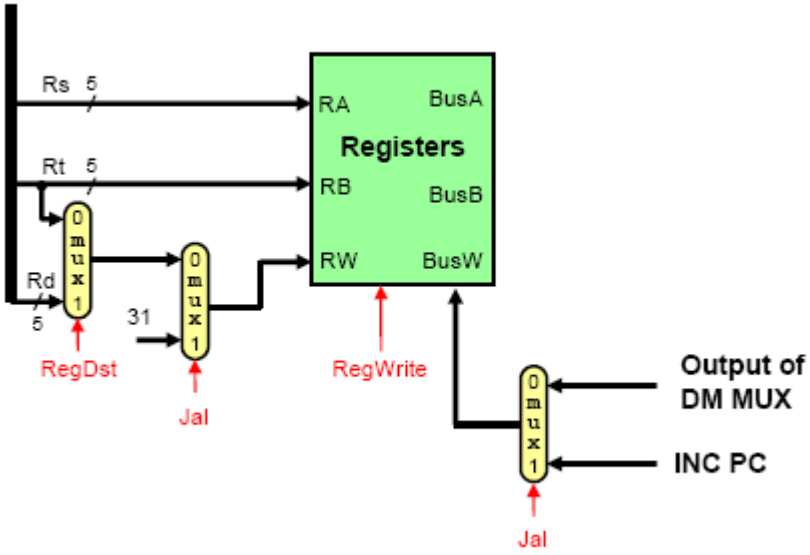
Instruction		Meaning	Format	
jal	label	$31=PC+4, jump	$op^6 = 3$	imm^{26}

This instruction is similar to the jump instruction (J) with the difference that register $31 should be loaded with the incremented PC value. Thus, we need to add a MUX at the input of RW input to the register file to select the value 31 when executing this instruction. We also need to add a MUX at the input of BusW in the register file to select the incremented PC value to be loaded instead of the value coming from the output of the data memory MUX. In addition, we need to make changes to the NextPC block to perform the same operation needed by the J instruction for Jal instruction. These changes are shown below.

The values of the control signals to control the execution of this instruction are given below:

Op	RegDst	RegWrite	ExtOp	ALUSrc	ALUOp	Jal	Beq	Bne	J	MemRead	MemWrite	MemtoReg
jal	x	1	x	x	x	1	0	0	0	0	0	x





b. blez

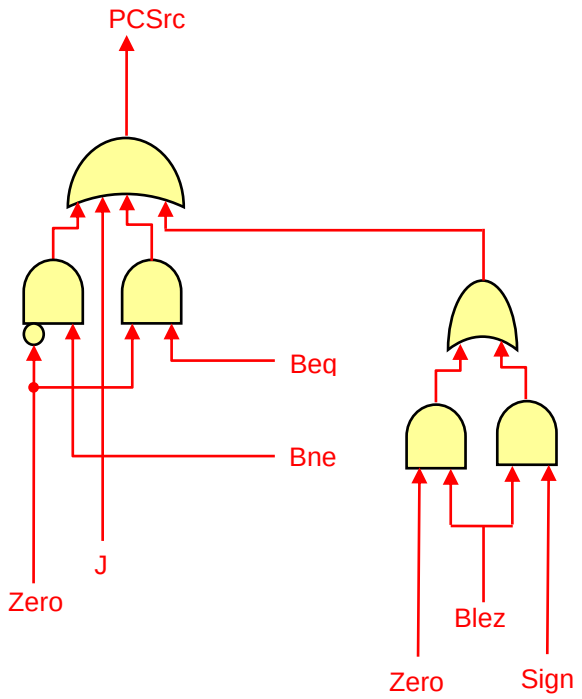
Instruction	Meaning	Format			
blez rs, label	Branch if (rs<=0)	$Op^6 = 1$	rs^5	0	imm^{16}

Since the first source operand specified by RS comes on BusA and the second operand which is the Zero register specified by the RT filed comes on BusB, all we need is to get the operand on BusA to appear at the output of the ALU as we just need to check the sign bit (i.e. most significant bit of the result). Performing an addition, subtraction, xoring, oring operations will work. Let us assume that we will do an ALU addition operation.

If the sign bit is 1, then it is less than zero but if it is zero, then we also need to check if the result is equal to zero. Thus, the changes needed to be done are in the NextPC block as shown below.

The values of the control signals to control the execution of this instruction are given below:

Op	RegDst	RegWrite	ExtOp	ALUSrc	ALUOp	Blez	Beq	Bne	J	MemRead	MemWrite	MemtoReg
blez	x	0	x	0= BusB	ADD	1	0	0	0	0	0	x



(iii) Assume that the propagation delays for the major components used in the datapath are as follows:

- Instruction and data memories: 200 ps
- ALU and adders: 180 ps
- Register file access (read or write): 150 ps
- Main control: 50 ps
- ALU control: 30 ps

Ignore the delays in the multiplexers, PC access, extension logic, and wires. What is the cycle time for the single-cycle datapath given above?

Cycle Time = IM + max(Main Control+ALU Control, Register Reading) +
ALU + DM + Register Writing
= 200 ps + 150 ps + 180 ps + 200 + 150 ps = 880 ps

(iv) Suppose that we want to add a variant of the lw (load word) instruction to the single-cycle datapath, which increments the index register by 4 after loading a word from memory. This instruction **lw_inc $rt, imm^{16}($rs)** corresponds to the following two instructions:

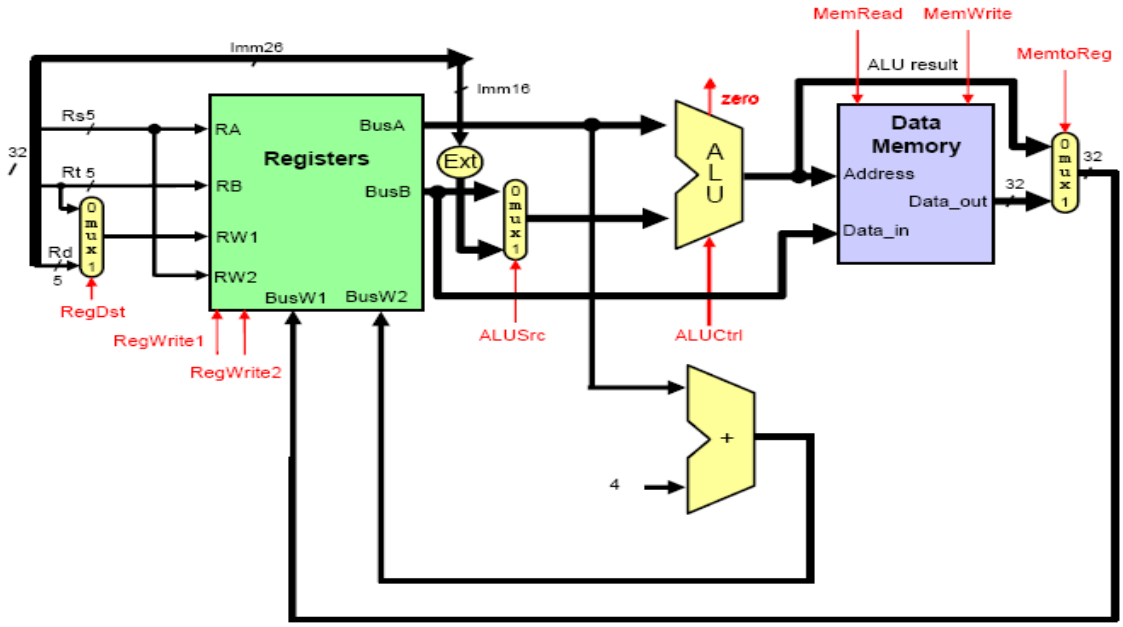
lw $rt, imm^{16}($rs)

addi $rs, $rs, 4

This instruction would be useful in array manipulation.

a. Add any necessary datapath and control signals needed for the implementation of this instruction. Show only the modified and added components to the datapath. If there are any changes needed to the register file, just indicate the required changes and add the required signals without showing its implementation. Show the values of the control signals to control the execution of this instruction.

The only difference between this instruction and the lw instruction is that we need to increment the $rs register by 4 and update its value. This requires that we have a register file that has two write ports as we need to write to $rs the incremented value and also write the loaded value to $rt register. We also need an additional adder to increment the $rs register by 4 since the ALU is used by the instruction for address calculations. The required changes are shown below:



The values of the control signals to control the execution of this instruction are given below:

Op	RegDst	RegWrite1	RegWrite2	ExtOp	ALUSrc	ALUOp	Beq	Bne	J	MemRead	MemWrite	MemtoReg
lw_inc	0 = Rt	1	1	1=sign	1=Imm	ADD	0	0	0	1	0	1

b. Consider the following code for adding the words of an n-word Array. The procedure receives the array address in $a0 and the number of words in $a1 and returns the sum in $v0.

```
AddArray:
        xor $v0, $v0, $v0       # Array sum=0
Next:
        lw $t0, ($a0)
        addi $a0, $a0, 4
        add $v0, $v0, $t0
        addi $a1, $a1, -1
        bne $a1, $zero, Next
        jr $ra
```

Determine the instruction count as a function of the Array size, n. What will be the instruction count if the new instruction **lw_inc** is utilized in this code. Assuming that due to the implementation of this instruction, the clock cycle is increased by 5%. What will be the maximum speedup in executing the procedure with the **lw_inc** instruction?

IC = 2 + 5*n

Instruction count after using lw_inc instruction will be = 2 + 4*n

Speedup = (2+5*n) / (2+4*n)*1.05

Maximum speedup = 5/4*1.05 (for large n)
= 1.19

[14 Points]

(Q2) Consider the code given below:

```
lw $2, ($1)
addi $2, $2, 1
sw $2, ($1)
addi $1, $1, 4
```

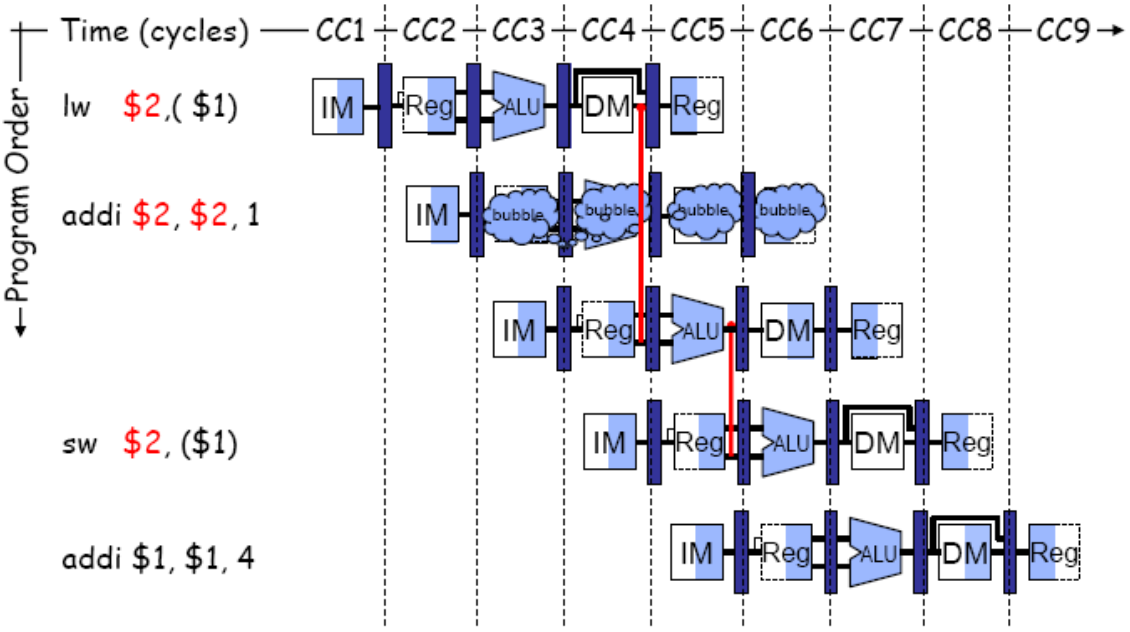
(i) Identify all the **RAW** data dependencies in the above code. Which dependencies are data hazards that will be resolved by forwarding? Which dependencies are data hazards that will cause a stall?

RAW dependencies:

lw $2, ($1) **and** **addi $2, $2, 1** **(stall 1 cycle & forwarding)**

addi $2, $2, 1 **and** **sw $2, ($1)** **(forwarding)**

(ii) Using a multiple-clock-cycle graphical representation, show the instruction execution across the pipeline including forwarding paths and stalled cycles if any. How many clock cycles will be needed to execute the instructions?



Number of clock cycles = 9.

(iii) Can you modify the instructions without changing their execution behavior while reducing the number of clock cycles needed for their execution? What will be the speedup if any?

Yes, the program sequence can be written as follows without changing its execution behavior while eliminating the stall cycle and reducing the number of clock cycles from 9 to 8.

```
lw $2, ($1)
addi $1, $1, 4
addi $2, $2, 1
sw $2, -4($1)
```

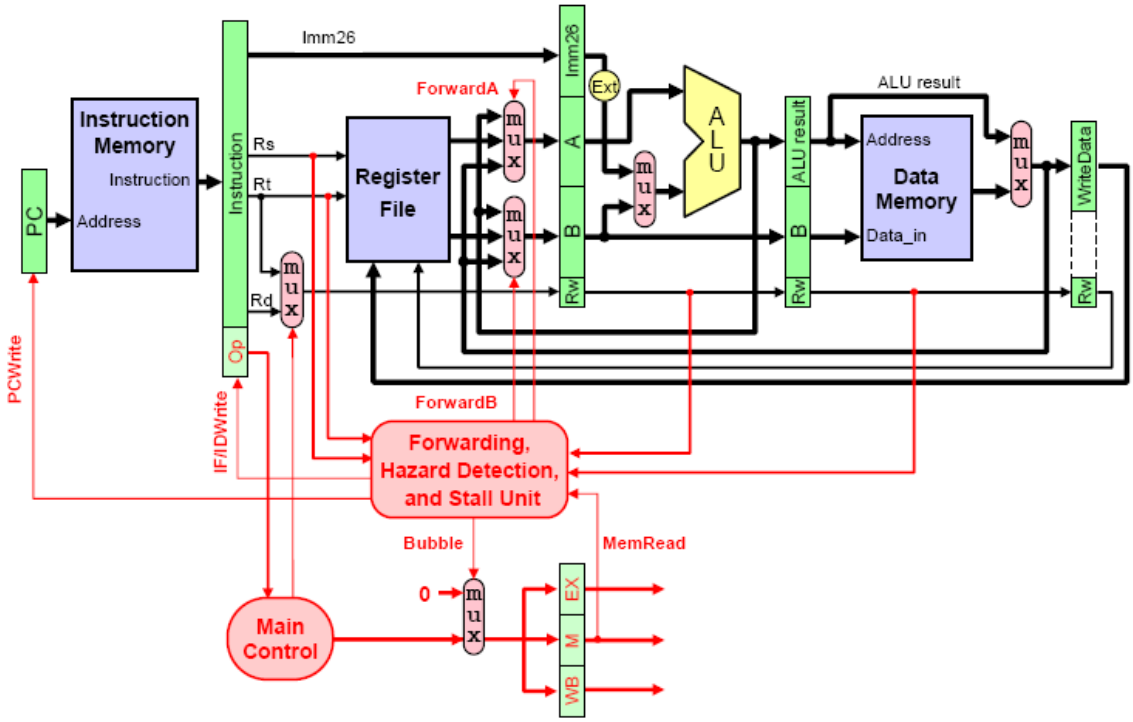
Speedup = 9 / 8 = 1.125.

[10 Points]

(Q3) Consider the instruction sequence given below:

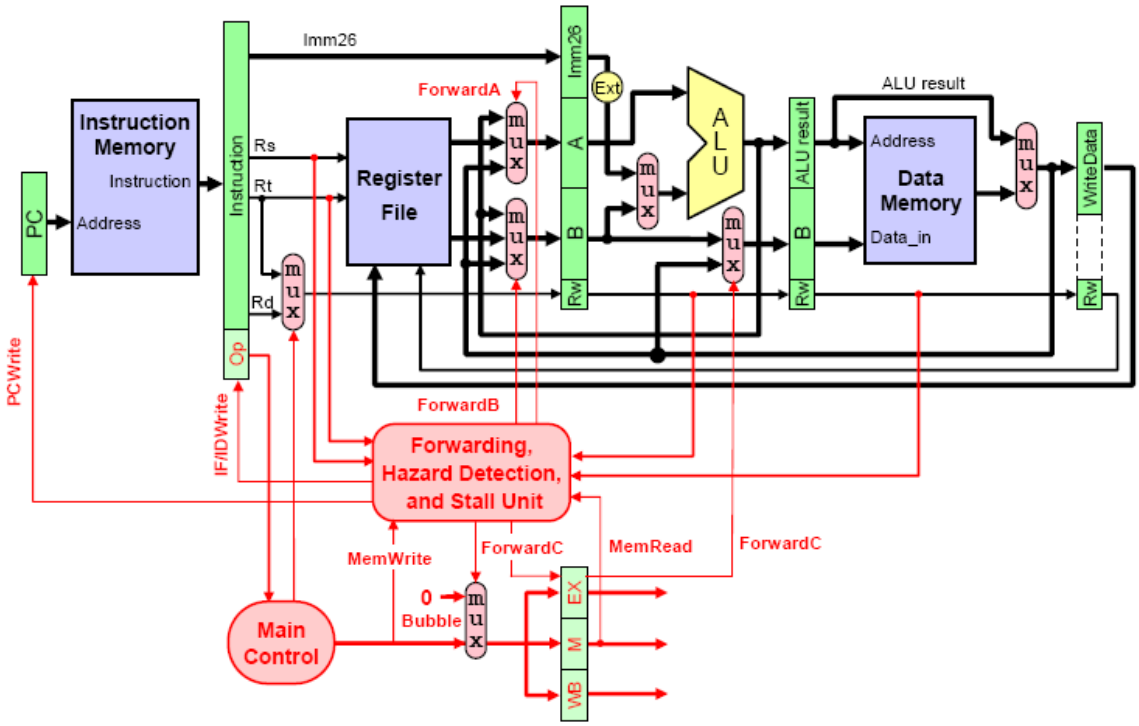
```
lw $t1, ($s0)
sw $t1, ($s1)
```

We can forward that data of **lw** instruction to the next **sw** instruction as required by the above example. However, such forwarding is not supported by the design given below.



(i) Show the required changes in the datapath and forwarding unit to support such forwarding.

We need a multiplexer at the B input in the EX/MEM register as shown below. The data loaded from the Data Memory should be fed back at the input of the multiplexer. A control signal, ForwardC, is needed to control the selection of this multiplexer. The Forwarding unit in the ID stage will generate the ForwardC signal and pipeline it, after detecting the dependency between a SW and a previous LW instruction.



(ii) Write the condition for generating the forwarding control signal. Identify the pipeline registers and control signals used by the **sw** and **lw** instructions when writing the condition.

ForwardC=0 means no forwarding,
ForwardC=1 means forward the load data from the MEM stage.

<u>Condition for generating ForwardC</u>:

If (ID/EX.MemRead == 1 and MemWrite == 1 and
ID/EX.Rw == IF/ID.Rt and ID/EX.RW != 0) ForwardC = 1
Else ForwardC = 0.

[14 Points]

(Q4) Given that **TABLE** is defined as: **TABLE: .ascii "Salam Alaikom"**. The code given below counts the number of characters 'A' or 'a' found in TABLE and stores the count in register $t0.

```
        xor $t0, $t0, $t0
        li $t1, 13
        la $t2, TABLE
        addi $t2, $t2, -1
Next:   beq $t1, $zero, ENL
        addi $t2, $t2, 1
        lbu $t3, ($t2)
        ori $t3, $t3, 0x20
        li $t4, 'a'
        addi $t1, $t1, -1
        bne $t3, $t4, Next
        addi $t0, $t0, 1
        j Next
ENL:
```

(i) Show the outcomes of each of the conditional branch instructions due to the execution of the program (T for taken, N for not taken).

Conditional Branch	Branch Outcome													
Beq $t1, $zero, ENL	N	N	N	N	N	N	N	N	N	N	N	N	N	T
Bne $t3, $t4, Next	T	N	T	N	T	T	N	T	N	T	T	T	T	

(ii) List the predictions and the accuracies for each of the following dynamic branch predictions schemes:

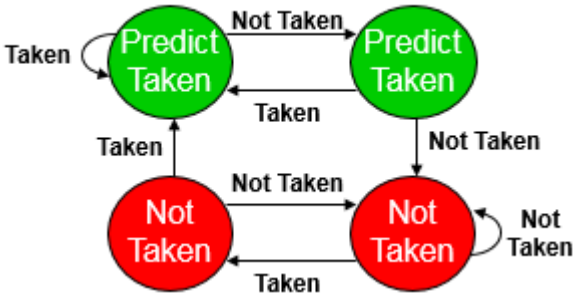
a. 1-bit prediction, initialized to **predict not taken**.

Conditional Branch	Branch Prediction													
Beq $t1, $zero, ENL	N	N	N	N	N	N	N	N	N	N	N	N	N	N
Bne $t3, $t4, Next	N	T	N	T	N	T	T	N	T	N	T	T	T	

Wrong=1+9=10 Right=13+4=17
Accuracy = 100% * 17/27 = 62.96%

b. 2-bit predictor, initialized to **weakly predict not taken**.

The 2-bit predictor used is as follows:



Conditional Branch	Branch Prediction													
Beq $t1, $zero, ENL	N	N	N	N	N	N	N	N	N	N	N	N	N	N
Bne $t3, $t4, Next	N	T	T	T	T	T	T	T	T	T	T	T	T	

Wrong=1+5=6 Right=13+8=21
Accuracy = 100% * 21/27 = 77.78%

[18 Points]

(Q5) Consider the following series of address references given as 16-bit addresses:

0x0000, 0x0001, 0x0002, 0x0040, 0x0041, 0x0000, 0x0001, 0x0002, 0x0040, 0x0041, 0x0042, 0x0062, 0x0063, 0x0043, 0x0045, 0x0046.

(i) Assuming a 32-byte cache organized as a **direct-mapped** cache with 4-byte block size, determine the number of bits in the offset, index and tag fields. Starting with an empty cache, show the offset, index and tag for each address reference in the list and indicate whether it is a hit or a miss. What is the miss ratio for this sequence on this cache?

Offset = 2 bits Index = 3 bits Tag=16-5=11 bits

Address	Tag	Index	Offset	Hit/Miss
0x0000	0x000	000	00	M
0x0001	0x000	000	01	H
0x0002	0x000	000	10	H
0x0040	0x002	000	00	M
0x0041	0x002	000	01	H
0x0000	0x000	000	00	M
0x0001	0x000	000	01	H
0x0002	0x000	000	10	H
0x0040	0x002	000	00	M
0x0041	0x002	000	01	H
0x0042	0x002	000	10	H
0x0062	0x003	000	10	M
0x0063	0x003	000	11	H
0x0043	0x002	000	11	M
0x0045	0x002	001	01	M
0x0046	0x002	001	10	H

Miss ratio = 7 / 16 = 0.4375.

(ii) Assuming a 32-byte cache organized as a **two-way set associative** cache with 4-byte block size, determine the number of bits in the offset, index and tag fields. Starting with an empty cache, show the offset, index and tag for each address reference in the list and indicate whether it is a hit or a miss. Assume that a FIFO replacement policy is used. What is the miss ratio for this sequence on this cache?

Offset = 2 bits Index = 2 bits Tag=16-4=12 bits

Address	Tag	Index	Offset	Hit/Miss
0x0000	0x000	00	00	M
0x0001	0x000	00	01	H
0x0002	0x000	00	02	H
0x0040	0x004	00	00	M
0x0041	0x004	00	01	H
0x0000	0x000	00	00	H
0x0001	0x000	00	01	H
0x0002	0x000	00	10	H
0x0040	0x004	00	00	H
0x0041	0x004	00	01	H
0x0042	0x004	00	10	H
0x0062	0x006	00	10	M
0x0063	0x006	00	11	H
0x0043	0x004	00	11	H
0x0045	0x004	01	01	M
0x0046	0x004	01	10	H

Miss ratio = 4 / 16 = 0.25.

[14 Points]

(Q6) A processor runs at 2 GHz and has a CPI=1.5 for a perfect cache (i.e. without including the stall cycles due to cache misses). Assume that load and store instructions are 20% of the instructions. The processor has an I-cache with a 3% miss rate and a D-cache with 5% miss rate. The hit time is 1 clock cycle. Assume that the time required to transfer a block of data from the RAM to the cache, i.e. miss penalty, is 50 ns.

(i) What is the average memory access time for instruction access in clock cycles?

Miss penalty in clock cycles = $50 * 10^{-9} * 2 * 10^{9} = 100$
AMAT = hit time + miss rate * miss penalty = 1 + 0.03 * 100 = 4

(ii) What is the average memory access time for data access in clock cycles?

AMAT = 1 + 0.05 * 100 = 6

(iii) What is the number of stall cycles per instruction and the overall CPI?

Number of stall cycles per instruction = 1 * 0.03 * 100 + 0.2 * 0.05 * 100 = 4
Overall CPI = 1.5 + 4 = 5.5

(iv) A new technology is proposed that can make the processor run at 4 GHz. The only impact of this technology is that the cache size has to be decreased to keep a hit time of one clock cycle increasing the I-cache miss rate to 4% and the D-cache miss rate to 6%. Assume that the time required to transfer a block of data from the RAM to the cache is still 50 ns. Will the processor be faster using the new technology? What do you suggest to increase the speedup using the new technology?

Miss penalty in clock cycles = $50 * 10^{-9} * 4 * 10^{9} = 200$
Number of stall cycles per instruction = 0.04 * 200 + 0.2 * 0.06 * 200=8+2.4= 10.4
New CPI = 1.5 + 10.4 = 11.9

Speedup = $5.5 * 4 * 10^{9} / 11.9 * 2 * 10^{9} = 0.924$
Thus, the processor with the new technology will not be faster.

To increase the speedup, we need to reduce the miss penalty. This can be done using a second level cache and also using memory interleaving in transferring the data between the RAM and cache.