June 10, 2008

COMPUTER ENGINEERING DEPARTMENT

ICS 233

COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE

Final Exam

Second Semester (072)

Time: 7:30-10:30 AM

Student Name : _KEY_____

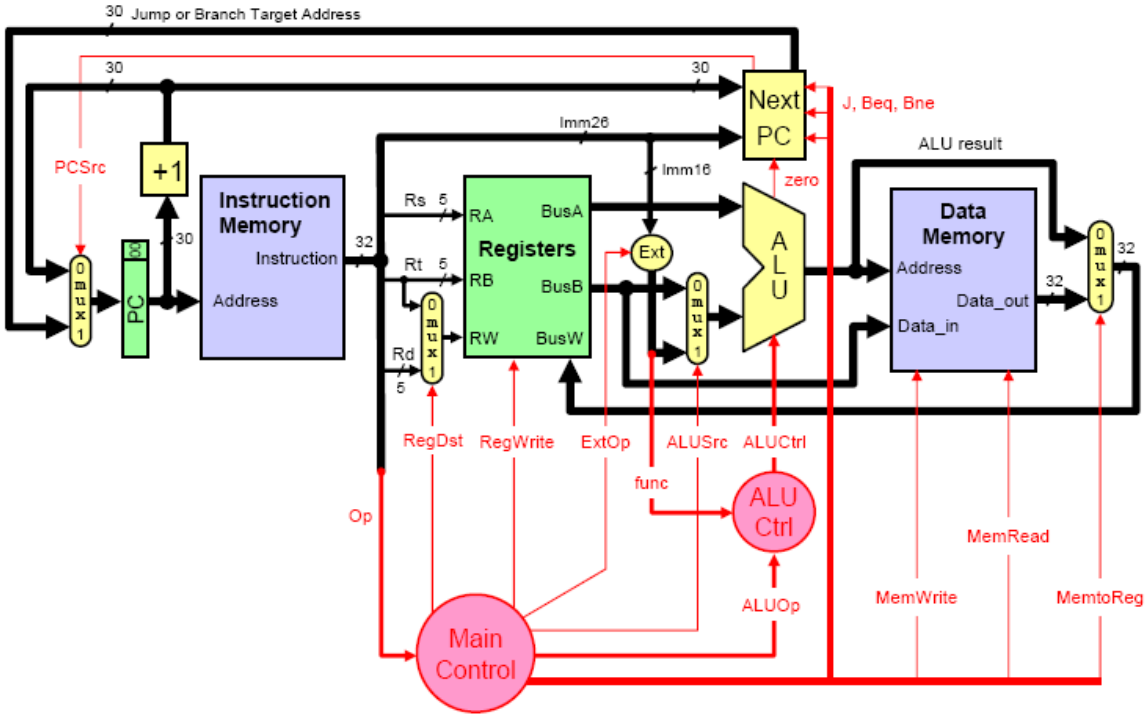
Student ID. : _____

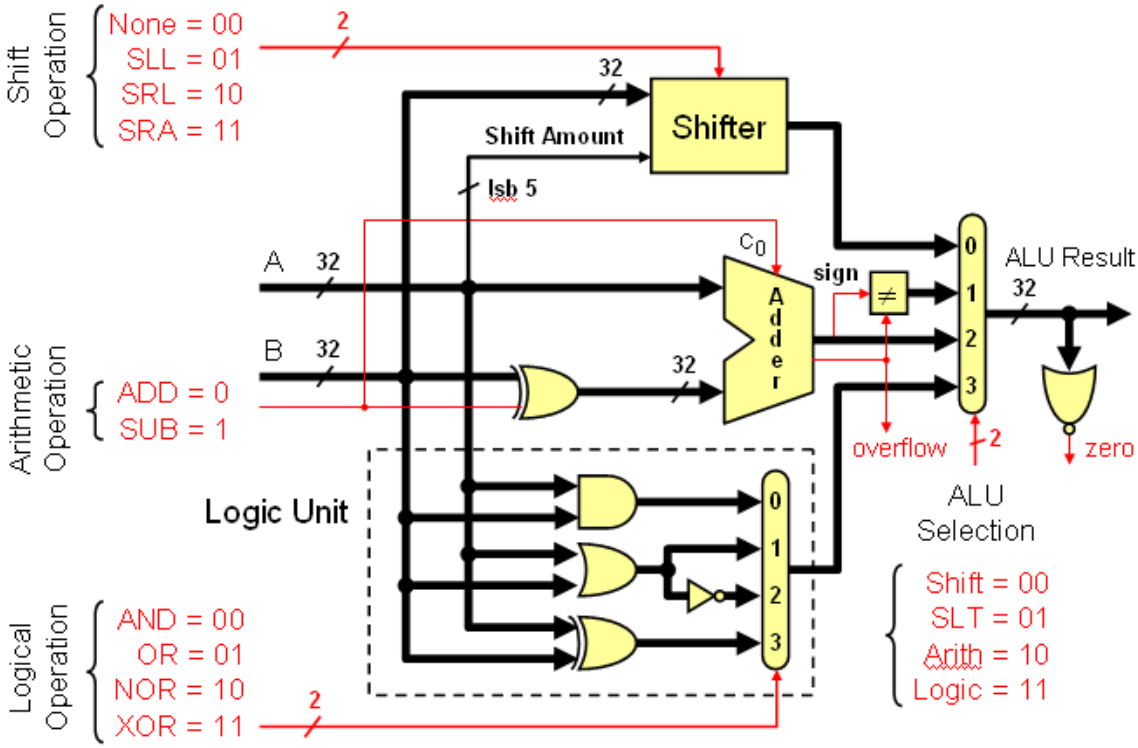
Question	Max Points	Score
Q1	30	
Q2	15	
Q3	15	
Q4	10	
Q5	18	
Q6	12	
Total	100	

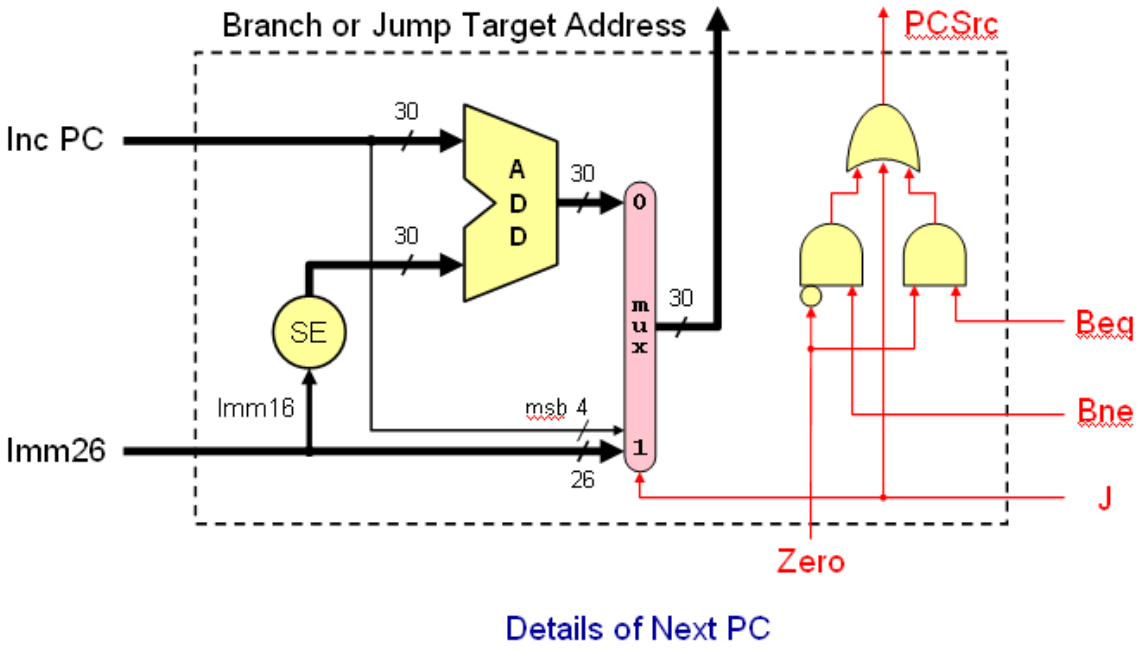
Dr. Aiman El-Maleh

[30 Points]

(Q1) Consider the single-cycle datapath and control given below along with ALU and Next PC blocks design for the MIPS processor implementing a subset of the instruction set:







Details of Next PC

(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
xori	0 = Rt	1	0=zero	1=Imm	XOR	0	0	0	0	0	0
lw	0 = Rt	1	1=sign	1=Imm	ADD	0	0	0	1	0	1
bne	x	0	x	0=BusB	SUB	0	1	0	0	0	x

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

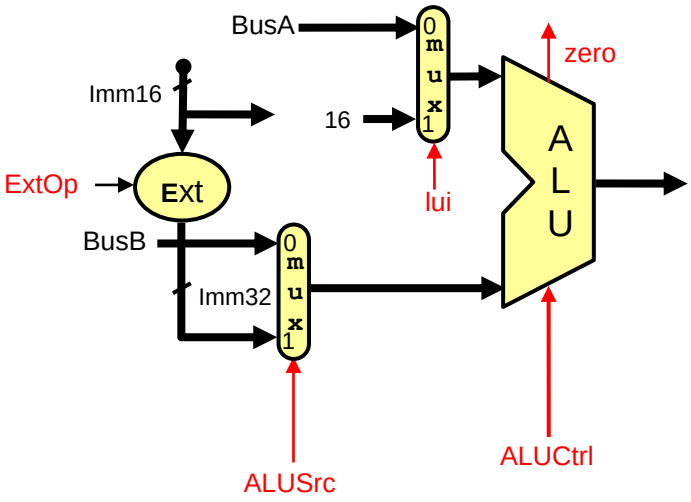
Instruction		Meaning	Format					
sub	rd, rs, rt	rd = rs + rt	$Op^6 = 0$	rs^5	rt^5	rd^5	0	0x22
addi	rt, rs, imm^{16}	rt = rs + imm^{16}	0x08	rs^5	rt^5	imm^{16}		
xori	rt, rs, imm^{16}	rt = rs ^ imm^{16}	0x0e	rs^5	rt^5	imm^{16}		
lw	rt, imm^{16}(rs)	rt = MEM[rs+imm16]	0x23	rs^5	rt^5	imm^{16}		
bne	rs, rt, label	branch if (rs != rt)	0x05	rs^5	rt^5	imm^{16}		

(ii) We wish to add the following instructions to the MIPS single-cycle datapath. Add any necessary datapath modifications 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. lui

Instruction	Meaning	Format			
lui rt, imm^{16}	rt= imm^{16} << 16	Op^6 = 0xf	0	rt^5	imm^{16}

For this instruction, the shift amount is 16 and the operand to be shifted is the immediate value selected on the B-input of the ALU. Thus, we need to add another MUX to select the shift amount as 16 for this instruction. The modified parts of the datapath to support the execution of this instruction is given below:



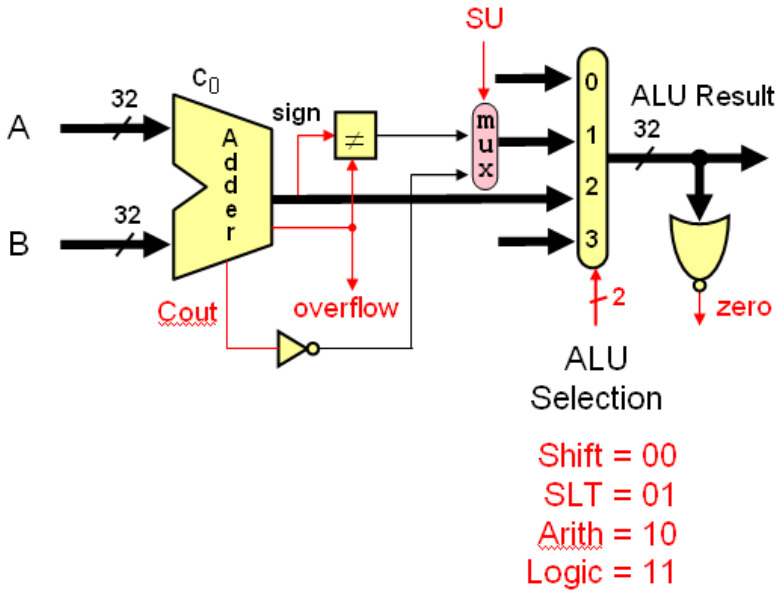
The values of the control signals to control the execution of this instruction are given below:

Op	lui	RegDst	RegWrite	ExtOp	ALUSrc	ALUOp	Beq	Bne	J	MemRead	MemWrite	MemtoReg
lui	1	0 = Rt	1	x	1=Imm	SLL	0	0	0	0	0	0

b. sltiu

Instruction	Meaning	Format			
sltiu rt, rs, imm^{16}	rt=(rs<imm?1:0)	Op^6 = 0xb	rs^5	rt^5	imm^{16}

To execute this instruction the ALU needs to be modified such that when there is a borrow i.e. carry out = 0 then the result is 1. We can add a MUX to select between SLT and SLTU instructions. This MUX is controlled by the signal SU. When SU=1, it will assume it signed comparison otherwise it will assume it unsigned. The modified ALU is shown below:



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

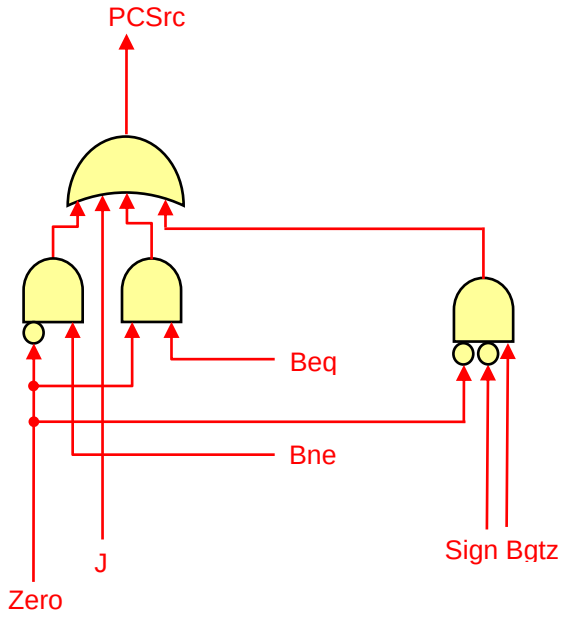
Op	SU	RegDst	RegWrite	ExtOp	ALUSrc	ALUOp	Beq	Bne	J	MemRead	MemWrite	MemtoReg
sltiu	0	0 = Rt	1	0	1=Imm	SLT	0	0	0	0	0	0

c. bgtz

Instruction	Meaning	Format			
bgtz rs, label	branch if (rs>0)	$Op^6 = 7$	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.

To check that the result is greater than 0, we need to check that the sign bit is 0 and that the result is not equal to zero. Thus, the changes needed to be done are in the NextPC block as shown below:



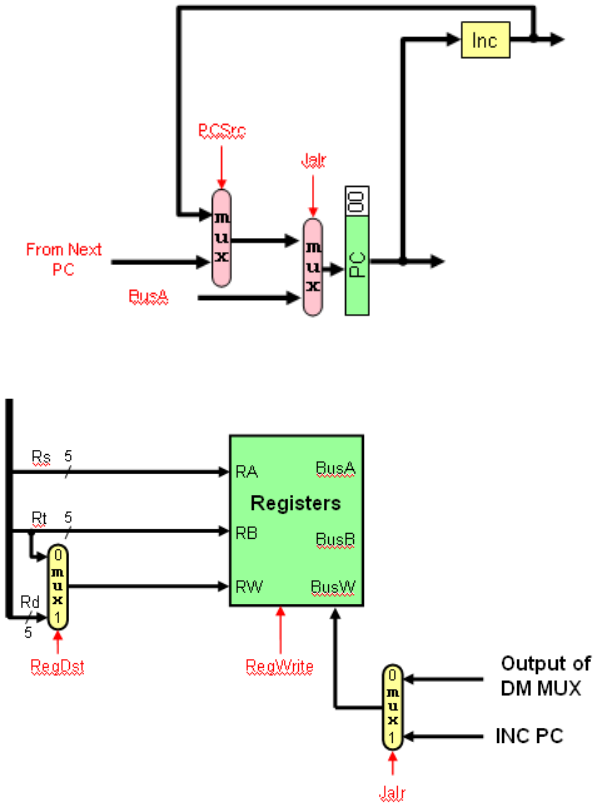
The values of the control signals to control the execution of this instruction are given below:

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

d. jalr

Instruction	Meaning	Format					
jalr rd, rs	rd=pc+4, pc=rs	$op^6 = 0$	rs^5	0	rd^5	0	9

To execute this instruction, we need to load the PC with the content of rs register and load rd register with the incremented value of the PC. Thus, we 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 add a mux to select whether to load the PC from BusA or the MUX selecting between the incremented PC value or the address generated by the Next PC block 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	Jalr	Beq	Bne	J	MemRead	MemWrite	MemtoReg
jalr	1=Rd	1	x	x	x	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: 150 ps
- ALU and adders: 100 ps
- Register file access (read or write): 60 ps
- Main control: 20 ps
- ALU control: 20 ps

Ignore the delays in the multiplexers, PC access, extension logic, and wires.

a. 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
= 150 ps + 60 ps + 100 ps + 150 + 60 ps = 520 ps

b. A friend of yours suggested modifying the MIPS instruction set architecture to remove the ability to specify an offset for memory access instructions. Specifically, all load-store instructions with nonzero offsets would become pseudo instructions and would be implemented using two instructions. For example: the instruction `lw $t0, 4($t1)` is implemented as:

```
addi  $at, $t1, 4          # add the offset to a temporary
lw    $t0, $at             # new way of doing lw $t0, 4($t1)
```

What will be the cycle time if the proposed simplified architecture were to be used? Under what conditions, the proposed architecture will be faster?

In the proposed architecture, since the load and store instructions do not need to perform an ALU operation, the Data Memory block can be made in parallel with the ALU. Thus, the new cycle time will be:

Cycle Time = IM + max(Main Control+ALU Control, Register Reading) +
max(ALU, DM) + Register Writing
= 150 ps + 60 ps + 150 + 60 ps = 420 ps

Execution time of existing architecture = IC*CPI*CLK Cycle= IC * 520 ps

Execution time of proposed architecture = IC(1+LS) * 420 ps , where LS is the percentage of Load-Store instructions as for each such instruction an extra instruction is added.

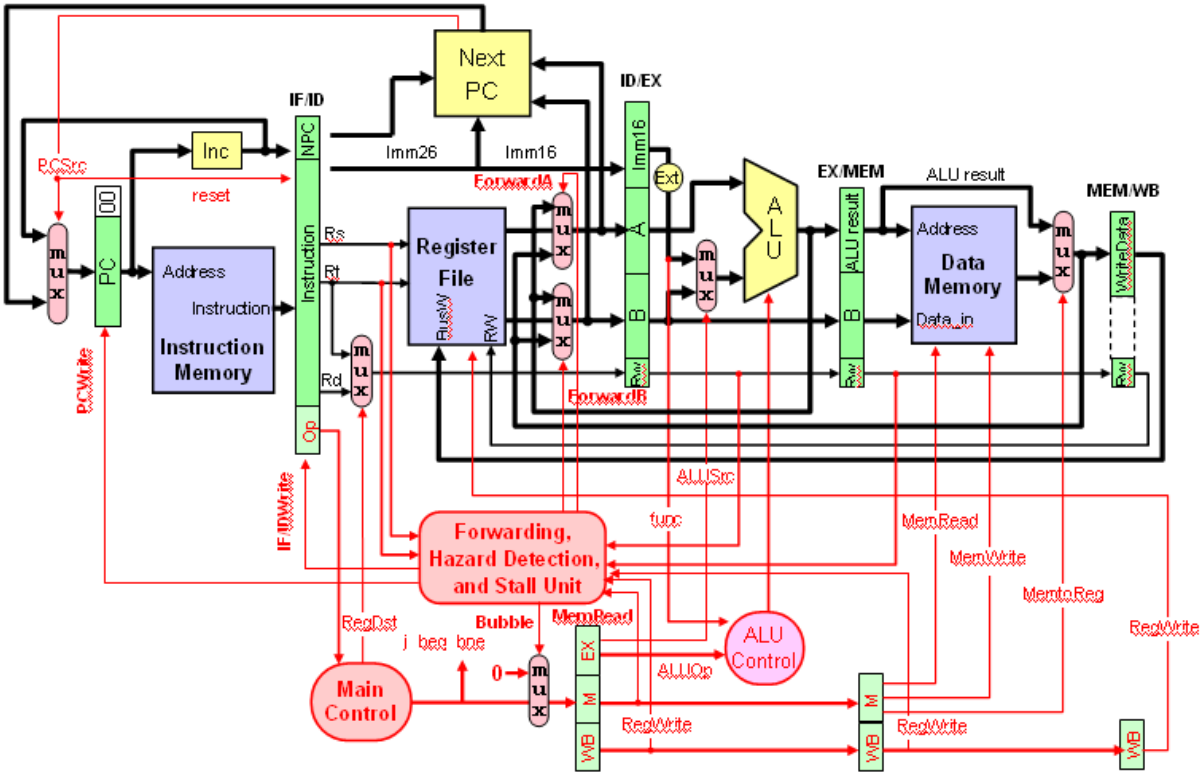
Thus, the proposed architecture will be faster if IC(1+LS)*420ps < IC*520 ps
$\Rightarrow$1+LS < 520/420=1.238 $\Rightarrow$ LS < 0.238.

Thus, if the percentage of Load-Store instructions in any program is less that 23.8% the proposed architecture will be faster.

[15 Points]

(Q2) Consider the pipelined MIPS processor design given below:

(i) Make all the necessary changes to the given pipelined design to overcome data and control hazards by showing the design of hazard detection, forwarding and stall unit.



(ii) Show the control signals that will be used for forwarding along with their conditions.

Control Signal	Explanation
ForwardA = 00	First ALU operand comes from the register file
ForwardA = 01	Forwarded from the previous ALU result
ForwardA = 10	Forwarded from data memory or 2nd previous ALU result
ForwardB = 00	Second ALU operand comes from the register file
ForwardB = 01	Forwarded from the previous ALU result
ForwardB = 10	Forwarded from data memory or 2nd previous ALU result

Forwarding Conditions:

if (IF/ID.Rs == ID/EX.Rw and ID/EX.Rw ≠ 0 and ID/EX.RegWrite)
ForwardA = 01
elseif (IF/ID.Rs == EX/MEM.Rw and EX/MEM.Rw ≠ 0 and EX/MEM.RegWrite)
ForwardA = 10
else ForwardA = 00
if (IF/ID.Rt == ID/EX.Rw and ID/EX.Rw ≠ 0 and ID/EX.RegWrite)
ForwardB = 01
elseif (IF/ID.Rt == EX/MEM.Rw and EX/MEM.Rw ≠ 0and EX/MEM.RegWrite)
ForwardB = 10
else ForwardB = 00

(iii) Show the control signals that will be used for stalling the pipeline along with their conditions.

Condition for Stalling the pipeline:

1. Stalling the Pipeline due to Load Instruction:

if ((ID/EX.MemRead == 1) and (ID/EX.Rw ≠ 0) and
((ID/EX.Rw == IF/ID.Rs) or (ID/EX.Rw == IF/ID.Rt))) Stall

Stall means that the signals PCWrite=0 and IF/IDWrite=0, which will freeze the content of PC and IF/ID registers and bubble=1 which will introduce a bubble in the ID/EX register by setting the control signals to 0.

2. Stalling the Pipeline due to taken branch Instruction:

Also, when PCSrc=1, reset=1 and the content of IF/ID register will be reset to 0 to make the fetched instruction a NOP.

[15 Points]

(Q3) Consider the code given below:

```
add $1, $2, $3
lw $2, 8($1)
sub $2, $2, $1
sw $2, 8($1)
```

(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?

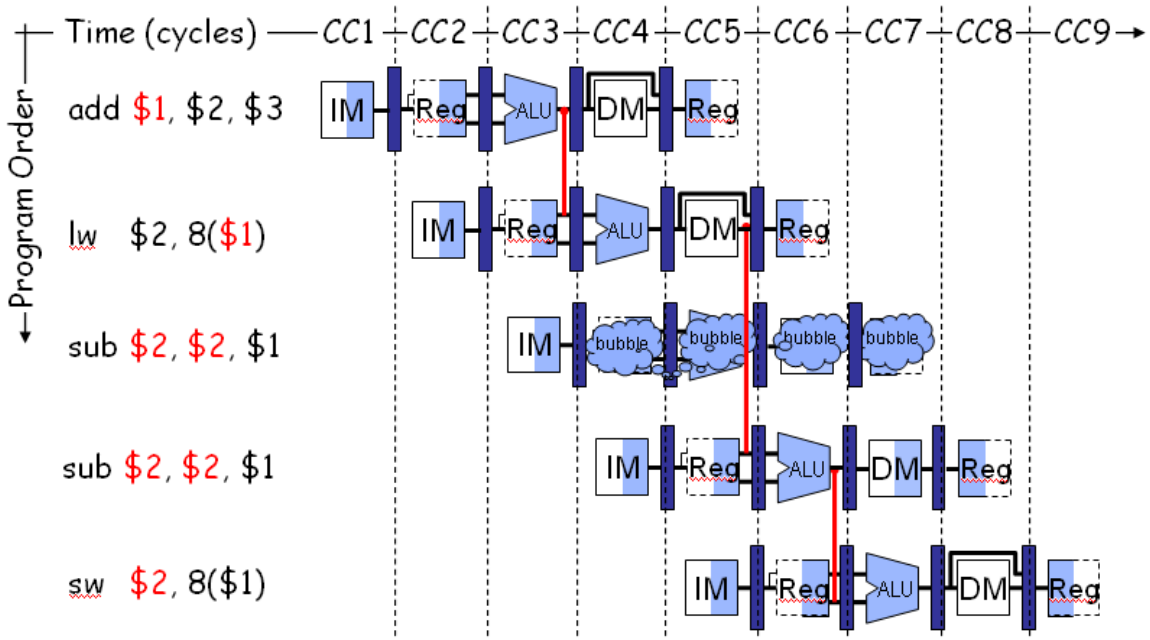
RAW dependencies:

add $1, $2, $3 and lw $2, 8($1) (forwarding)

lw $2, 8($1) and sub $2, $2, $1 (stall 1 cycle & forwarding)

sub $2, $2, $1 and sw $2, 8($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 = 8. Note that in cycle 8, the execution of the sw instruction will be completed.

[10 Points]

(Q4) Consider the MIPS program given below:

```
.data
Table: .word 2, 4, 3, 5, 6, 1
.text
.globl main
main:
        la      $a0, Table      # address of first element
        add     $a1, $a0, 20    # address of last element
max:    move    $v0, $a0        # max pointer = first pointer
        lw      $v1, ($v0)      # $v1 = first value
        move    $t0, $a0        # $t0 = array pointer
loop:   addi    $t0, $t0, 4     # point to next array element
        lw      $t1, 0($t0)     # $t1 = value of A[i]
        slt     $at, $v1, $t1   # if (A[i] ≤ max) then skip
        beq     $at, $0, skip
        move    $v0, $t0        # found new maximum
        move    $v1, $t1
skip:   bne     $t0, $a1, loop  # loop back if more elements
ret:
```

(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 $at, $0, skip	N	T	N	N	T
bne $t0, $a1, loop	T	T	T	T	N

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

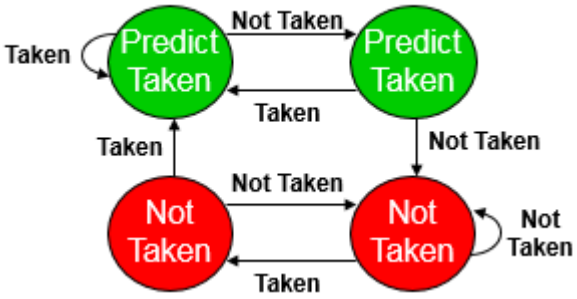
a. 1-bit prediction, initialized to **predict not taken**.

Conditional Branch	Branch Prediction				
beq $at, $0, skip	N	N	T	N	N
bne $t0, $a1, loop	N	T	T	T	T

Wrong=3+2=5 Right=2+3=5
Accuracy = 5/10 = 50%

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

The 2-bit prediction scheme used is as follows:



Conditional Branch	Branch Prediction				
beq $at, $0, skip	N	N	N	N	N
bne $t0, $a1, loop	N	T	T	T	T

Wrong=2+2=4 Right=3+3=6
Accuracy = 6/10 = 60%

[18 Points]

(Q5) Assume that you have a cache with **64 bytes data size** (i.e. not including tag and valid bits). Consider the following series of address references given as 16-bit addresses:

0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x0ec2, 0x0ec3, 0x0ec4, 0x0ec5, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7, 0xffc6, 0xffc7.

(i) Assuming that the cache is organized as **direct-mapped** with **2-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 (**in binary**) 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 =1 Index =5 Tag=10

Address	Tag	Index	Offset	Hit/Miss
0x00c2	0000000011	00001	0	M
0x00c3	0000000011	00001	1	H
0x00c4	0000000011	00010	0	M
0x00c5	0000000011	00010	1	H
0x0ec2	0000111011	00001	0	M
0x0ec3	0000111011	00001	1	H
0x0ec4	0000111011	00010	0	M
0x0ec5	0000111011	00010	1	H
0x00c2	0000000011	00001	0	M
0x00c3	0000000011	00001	1	H
0x00c4	0000000011	00010	0	M
0x00c5	0000000011	00010	1	H
0x00c6	0000000011	00011	0	M
0x00c7	0000000011	00011	1	H
0xffc6	1111111111	00011	0	M
0xffc7	1111111111	00011	1	H

Miss ratio = 8/16=0.5

(ii) Assuming that the cache is organized as **four-way set associative** with **2-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 (**in binary**) 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 =1 Index =3 Tag=12

Address	Tag	Index	Offset	Hit/Miss
0x00c2	000000001100	001	0	M
0x00c3	000000001100	001	1	H
0x00c4	000000001100	010	0	M
0x00c5	000000001100	010	1	H
0x0ec2	000011101100	001	0	M
0x0ec3	000011101100	001	1	H
0x0ec4	000011101100	010	0	M
0x0ec5	000011101100	010	1	H
0x00c2	000000001100	001	0	H
0x00c3	000000001100	001	1	H
0x00c4	000000001100	010	0	H
0x00c5	000000001100	010	1	H
0x00c6	000000001100	011	0	M
0x00c7	000000001100	011	1	H
0xffc6	111111111100	011	0	M
0xffc7	111111111100	011	1	H

Miss ratio = 6/16=0.375

[12 Points]

(Q6) A processor runs at 2 GHz and has a CPI=1.4 for a perfect cache (i.e. without including the stall cycles due to cache misses). Assume that load and store instructions are 15% of the instructions. The processor has an I-cache with a 4% miss rate and a D-cache with 6% 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 40 ns.

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

Miss penalty in clock cycles = $40 * 10^{-9} * 2 * 10^{9} = 80$
Number of stall cycles per instruction = $0.04 * 80 + 0.15 * 0.06 * 80 = 3.2+0.72=3.92$
Overall CPI = 1.4 + 3.92 = 5.32

(ii) What is the average memory access time (AMAT) in ns?

No of memory accesses = I + 0.15*I=I*(1+0.15)=1.15*I
Access Time = 1*I + 0.04*I*80 + 0.15*I + 0.15*I*0.06*80= I*(1+0.15+(0.04+.15*.06)*80)=5.07*I
AMAT = Access Time / No of memory accesses = 5.07*I/1.15*I=4.409 clock cycles = $4.409 * 0.5 * 10^{-9} = 2.2$ ns

(iii) Discuss how you can reduce the AMAT.

The AMAT can be reduced by reducing the hit time, the miss rate and the miss penalty. The hit time can be reduced by using small and simple caches. The miss rate can be reduced by using larger cache size, higher associativity, and larger block size. The miss penalty can be reduced using a second level cache and also using memory interleaving in transferring the data between the RAM and cache.