COE 301 / ICS 233
Computer Organization

Final Exam – Term 172

Tuesday, May 15, 2018
8:00 am – 10:30 am

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

SOLUTION

☐ Dr. Aiman El-Maleh	☐ COE 301	☐ ICS 233
☐ Dr. Marwan Abu-Amara	☐ COE 301	☐ ICS 233
☐ Dr. Muhamed Mudawar		

Q1	/ 17	Q2	/ 8
Q3	/ 14	Q4	/ 19
Q5	/ 10	Q6	/ 12
Total	/ 80		

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.

Q1. [17 points] General Understanding of Topics

a) **(1 point)** Does pipelining improve the latency of individual instructions? Explain.

Pipelining does NOT improve the latency of individual instructions. However, it improves the throughput.

b) **(2 points)** What causes control hazards in a pipelined datapath and how control hazards can be eliminated?

Control hazards are caused by jump and branch instructions that are delayed in a pipelined datapath. They can be eliminated by converting the next (one or two) instructions that appear after a jump or a taken branch into NOPs.

c) **(2 points)** Explain the difference between static RAM and dynamic RAM.

Static RAM: Cell is made out of 6 transistors and does not require refreshing.

Dynamic RAM: Cell is made out of 1 transistor and 1 capacitor, requires refreshing, but denser (cheaper) than SRAM.

d) **(2 points)** Is it possible to use only one memory for both instructions and data in the single-cycle datapath? Explain why or why not. Is it possible to use only one memory for both instructions and data in a multi-cycle datapath? Explain.

In a single-cycle datapath, a load instruction must be fetched and must read data during the same cycle. Using only one memory is NOT possible to fetch the instruction and load the data during the same cycle.

In a multi-cycle datapath, using only one memory IS possible because fetching the instruction and loading the data can occur in two different cycles.

e) **(2 points)** Why do we need cache memory, and why do we have two separate cache memories (I-cache and D-cache) in a pipelined processor?

We need cache memory to reduce memory latency.

Two separate caches (I-cache and D-cache) are needed to access both of them during the same cycle by two different instructions.

f) **(2 points)** Explain the concepts of temporal locality and spatial locality of reference in cache memory.

Temporal Locality: if a program references an instruction (or data) at a given address then it might reference the same address again in the future.

Spatial Locality: if a program references an instruction (or data) at a given address then it might reference the next address in the memory.

g) **(2 points)** What needs to be stored inside a cache for block identification? How does a cache know whether there is a cache hit or miss?

A cache stores tags for block identification.

The tag stored in the cache is compared against the tag in the memory address to determine whether there is a cache hit or miss.

h) **(2 points)** Suppose a 4-way set-associative cache has a capacity of 32 KiB (1 KiB = 1024 bytes) and each block consists of 64 Bytes. What is the total number of blocks in the cache? What is the number of sets?

Total number of blocks = 32 x 1024 / 64 = 512 blocks

Number of sets = 512 / 4 = 128 sets

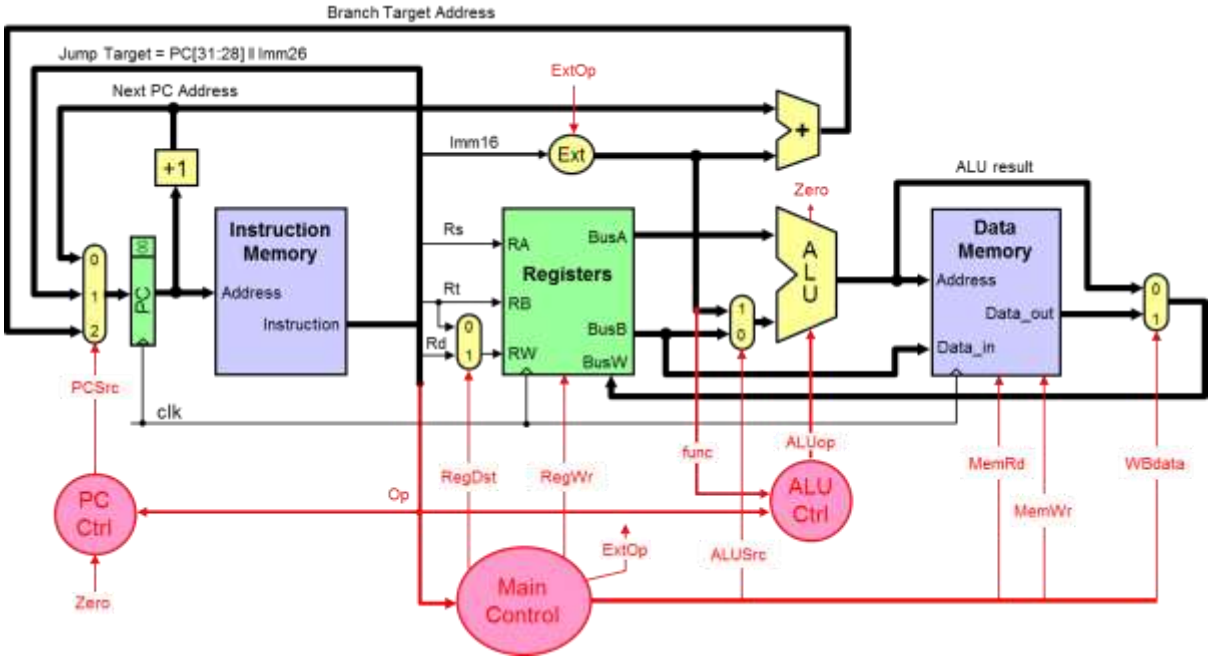
i) **(2 points)** Explain the difference between a write-through and a write-back cache.

Write-through cache: every write to the cache is written to the lower-level memory.

Write-back cache: the write is done in the cache only. A modified bit is needed to indicate whether a block has been modified. Modified blocks are written back to memory when replaced.

Q2. [8 points] Single-Cycle Processor

The single-cycle datapath and control of a MIPS-like processor is shown below. However, this datapath and control lacks the implementation of many important instructions.



Consider adding the following two new instructions to the above datapath: **JLR** and **LWI**. The **JLR** instruction is I-type and has a unique opcode. The **LWI** instruction is R-type and has a unique function code. The least-significant 2 bits of register **PC** are hardwired to **00**, and not stored in **PC**. Therefore, it is sufficient to increment **PC** by **1** to point to the next instruction in memory.

Instruction	**Format**	**Meaning**
Jump and Link Register Op = JLR	Op, Rs, Rt, Imm16	Reg[Rt] = PC + 1 PC = Reg[Rs] + Imm16
Load Word Indexed Op = R-type, Func = LWI	Op, Rs, Rt, Rd, Func	Reg[Rd] = Mem[Reg[Rs] + Reg[Rt]]

a) **(4 points)** Redraw the necessary changes to the above datapath to implement the above two instructions. Draw only the modified parts and explain why they are needed.

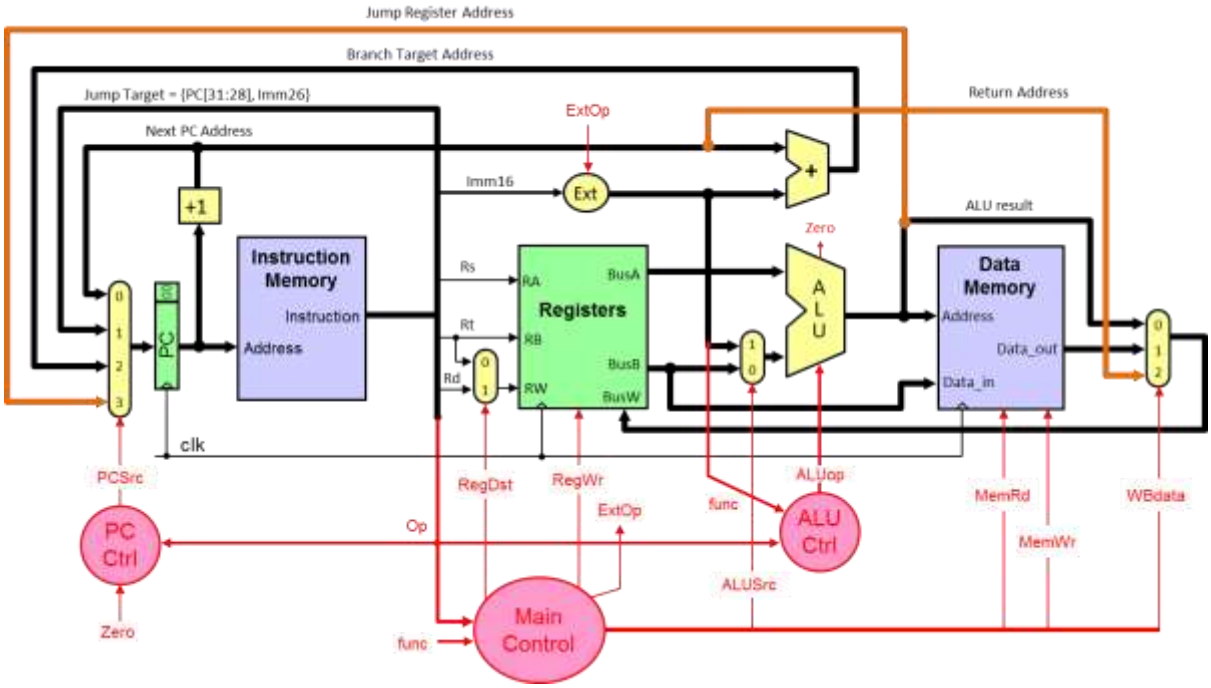
b) **(4 points)** Identify any new control signal needed to implement the above two instructions. Draw a table showing the values of all control signals to implement the above two instructions.

Q2 Solution

a) Changes needed to implement JLR and LWI instructions:

Add a 4th input to the mux at the input of the PC register and add a bus connecting the ALU result (Jump Register Address) back to the PC.

Add a 3rd input to the WB mux and add a bus connecting the Return Address (PC + 1) back to the register file.



b) Same control signals are used, except that Main control logic now depends on the opcode and function code for LWI.

	PCSrc	RegDst	RegWr	ExtOp	ALUSrc	ALUOp	MemRd	MemWr	WBdata
JLR	3=JRA	0=Rt	1	1=Sign	1=Imm	ADD	0	0	2=RA
LWI	0=PC+1	1=Rd	1	X	0=BusB	ADD	1	0	1=Mem

Q3. [14 points] Performance of Single-Cycle, Multi-Cycle, and Pipelined CPU

Compare the performance of a **single-cycle processor** and a **multi-cycle processor**. The delay times are as follows:

Instruction memory access time = 500 ps
Instruction Decode and Register read = 200 ps
ALU delay = 100 ps
Data memory access time = 500 ps
Register write = 200 ps

Ignore the other delays in the multiplexers, wires, etc. Assume a program has the following instruction mix: **40%** ALU, **5%** load, **5%** store, **30%** branch, and **20%** jump.

a) (6 points) Compute the delay for each instruction class and the clock cycle for the **single-cycle processor**.

Instruction Class	Instruction Memory	Decode and Register Read	ALU	Data Memory	Write Back	Total Delay
ALU	**500**	**200**	**100**		**200**	**1000**
Load	**500**	**200**	**100**	**500**	**200**	**1500**
Store	**500**	**200**	**100**	**500**		**1300**
Branch	**500**	**200**	**100**			**800**
Jump	**500**	**200**				**700**

Clock cycle for the single-cycle processor = **1500 ps = 1.5 ns**

b) (2 points) Compute the **clock cycle** and the **average CPI** for the **multi-cycle processor**.

Clock cycle for the multi-cycle processor = **max(500,200,100) = 500 ps**

Average CPI for the multi-cycle processor =

$0.4 \times 4 + 0.05 \times 5 + 0.05 \times 4 + 0.3 \times 3 + 0.2 \times 2 = 3.35$

c) (2 points) Determine **quantitatively** if there is a speedup when using the multi-cycle processor with respect to the single-cycle.

Speedup = **$(1500 \times 1) / (500 \times 3.35) = 0.8955 \Rightarrow$** ***No speedup***

d) (2 points) Assume that the processor is **pipelined**. Furthermore, assume that a program has the following instruction mix: **40%** ALU, **5%** load, **5%** store, **30%** branch, and **20%** jump. Moreover, assume that **90%** of the branches will be **taken**. The CPU stalls **1** cycle for each jump and **2** cycles for each taken branch. Compute the **average CPI** for the **pipelined** processor due to control hazards only.

Average CPI for the **pipelined** processor for control hazards =

$\mathbf{CPI_{base} + CPI_{stalls} = 1 + (0.3 \times 0.9 \times 2) + (0.2 \times 1) = 1.74}$

e) (2 points) Assume that the processor is **pipelined** and that load instructions are **5%** of the instruction count and store instructions are also **5%** as given above. However, the program spends **30%** of its execution time executing load instructions and **15%** of its execution time executing store instructions. The designers discovered that the Data cache is producing many cache misses causing the CPU to stall. They decided to improve the design of the data cache and improve the execution time of the load instructions by a factor of **3x** (3 times faster) and the store instructions by a factor of **2x**. Determine the overall speedup of the program due to the improvements done to the data cache.

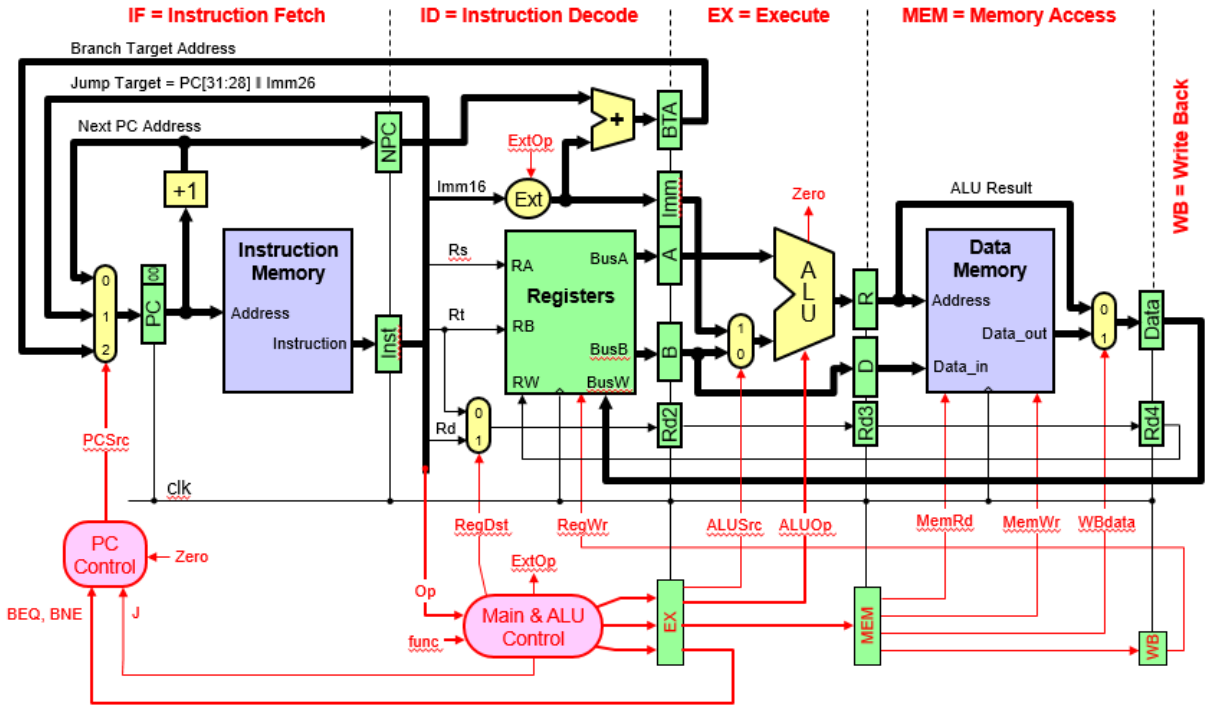
Speedup due to data cache improvement =

$$\frac{1}{\frac{0.3}{3} + \frac{0.15}{2} + (1 - 0.3 - 0.15)} = 1.379$$

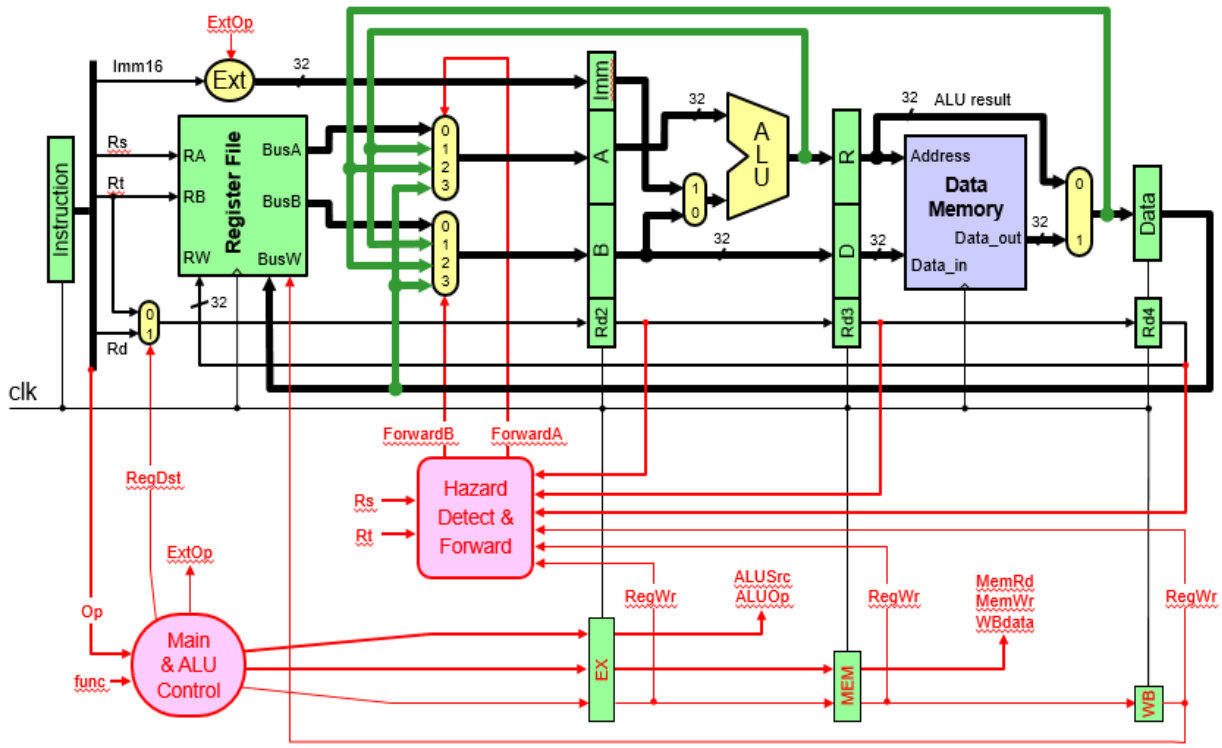
The program will run faster by a factor of 1.379x due to data cache improvement.

Q4. [19 points] Pipelined CPU Design

I. Consider the 5-stage pipelined CPU design given below.



a) **(5 points)** Show the design changes needed for handling data hazards using **<u>forwarding</u>** including a block diagram for data hazard detection and forwarding unit.



b) **(4 points)** Show the control signals that will be used for stalling the pipeline for data hazards due to load instructions along with their conditions. Show the necessary changes that need to be done to the design.

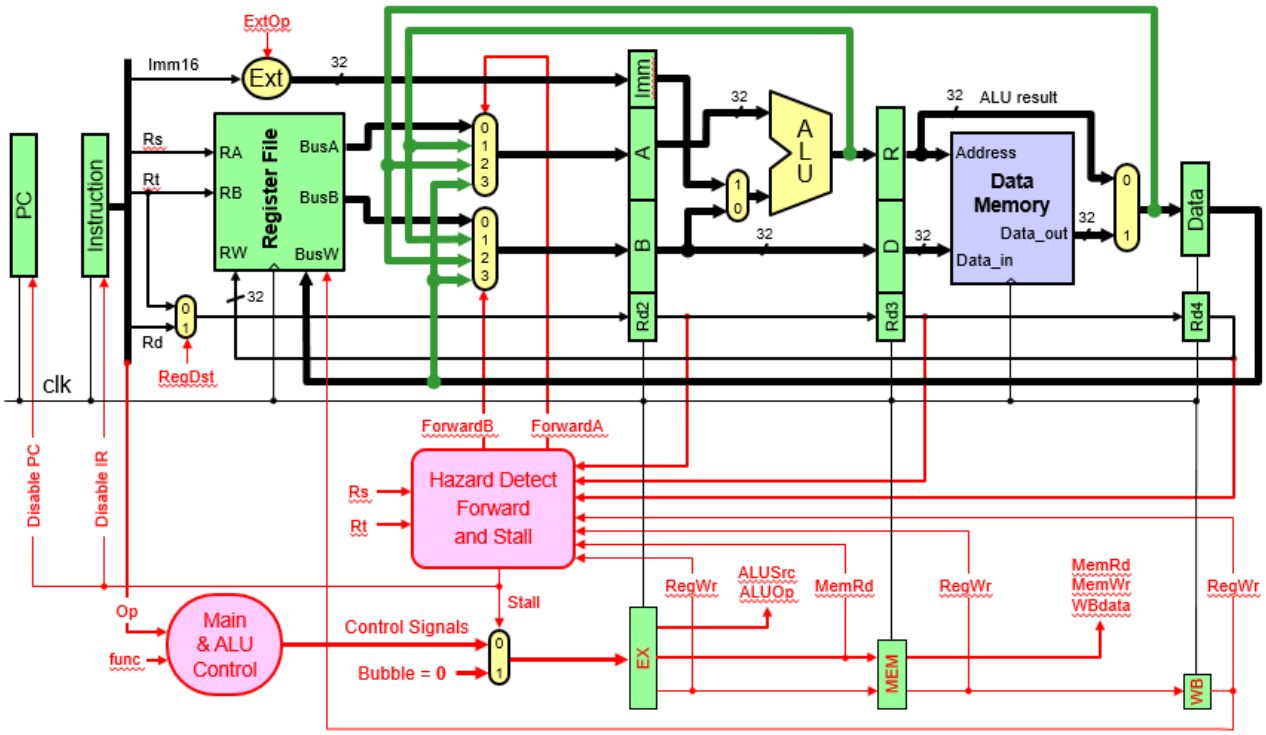
Condition for Stalling the pipeline due to Load Instruction:

```
if    ((EX.MemRd == 1)      // Detect Load in EX stage
and (ForwardA==1 or ForwardB==1))     Stall  // RAW Hazard
```

OR:

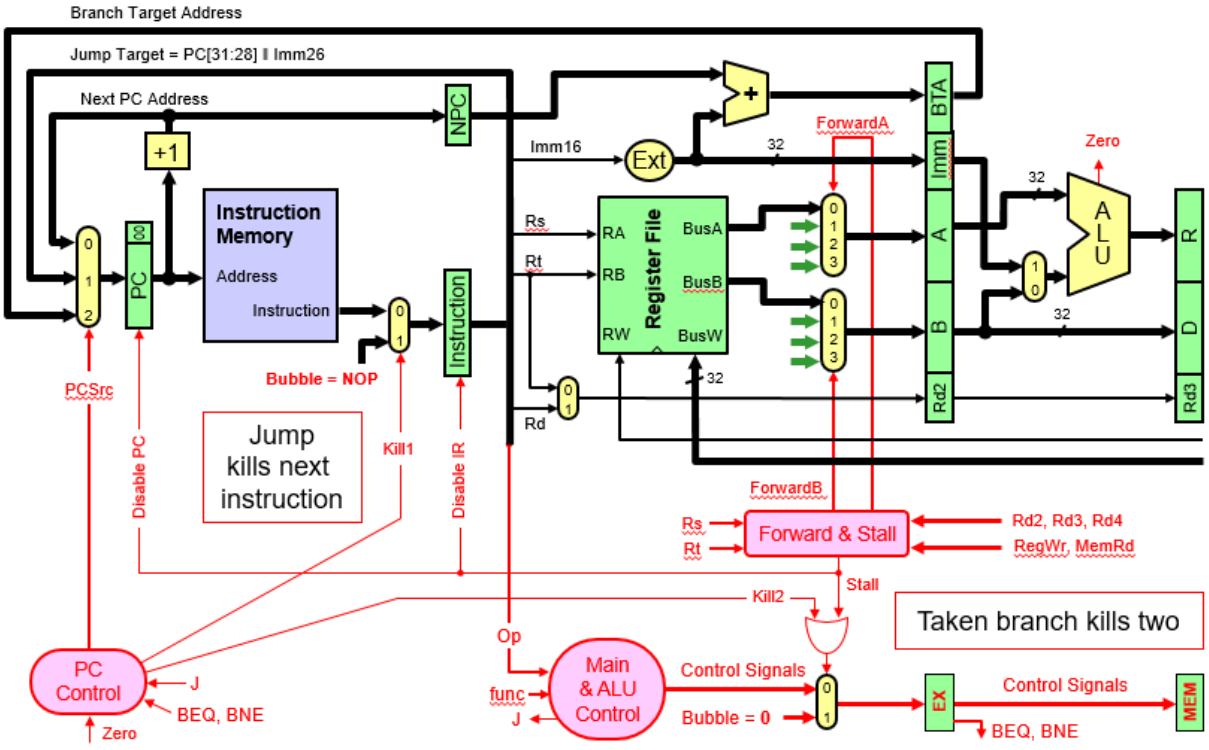
```
if    ((EX.MemRd == 1)
and (Rd2 != 0) and ((Rs == Rd2) or (Rt == Rd2))) Stall
```

Stall will Disable PC and Disable IR (i.e. the signals PCWrite=0 and IRWrite=0), which will freeze the content of PC and IR registers and will introduce a bubble in stage 2 control register by setting the control signals to 0.

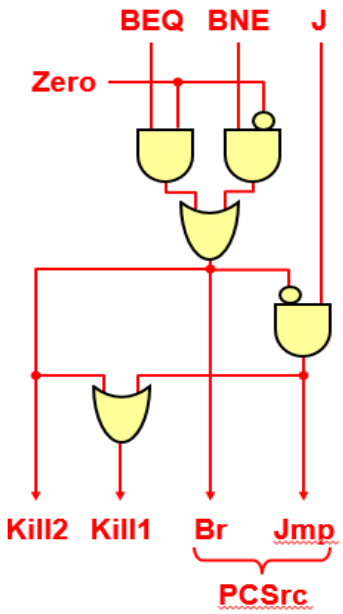


c) **(2 points)** Show the control signals that will be used for handling control hazards. Show the necessary changes that need to be done to the design.

When a jump instruction is at stage 2, Kill1=1 will replace that instruction by a NOP and when a taken branch is at stage 3, Kill1=1 and Kill2=1 will replace both instructions by NOPs.



d) **(3 points)** Show the design of the PC control logic that includes the handling of control hazards assuming that only BEQ, BNE, and J instructions are implemented.



II. **(5** Points**)** Consider the following MIPS assembly language code:

```
I1:    ORI  $s0, $0, 5
I2:    ADDI $s1, $0, 10
I3:    ADD  $s1, $s0, $s1
I4:    LW   $s0, -4($s1)
I5:    ADD  $s0, $s0, $s0
I6:    SW   $s0, -4($s1)
```

Complete the following table showing the timing of the above code on the 5-stage pipeline given in part (i) (IF, ID, EX, MEM, WB) supporting **forwarding** and **pipeline stall**. Draw an arrow showing forwarding between the stage that provides the data and the stage that receives the data. Show all stall cycles (draw an X in the box to represent a stall cycle). Determine the number of clock cycles to execute this code.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1: ORI	**IF**	**ID**	**EX**	-	**WB**										
I2: ADDI		**IF**	**ID**	**EX**	-	**WB**									
I3: ADD			**IF**	**ID**	**EX**	-	**WB**								
I4: LW				**IF**	**ID**	**EX**	**M**	**WB**							
I5: ADD					**IF**	**X**	**ID**	**EX**	-	**WB**					
I6: SW							**IF**	**ID**	**EX**	**M**	-				

Total number of clock cycles to execute the above code = **10**

Q5. [10 points] Cache Memory

a) **(3 points)** Given that the memory address consists of 64 bits, consider a 64 KiB **fully associative** cache (1 KiB = 1024 bytes) with 64-byte cache blocks and a write back policy is used. Compute the **total number of bits** required to store the **valid**, **modified**, and **tag** bits in the cache.

Total Valid bits = **# of blocks = 1024 bits**

Total Modified bits = **# of blocks = 1024 bits**

Total Tag bits = **# of blocks × Tag bits = 1024 × (64 – 6) bits = 59,392 bits**

b) **(3 points)** Assume that the memory address consists of 64 bits, and a 64 KiB **4-way set associative** cache with 64-byte cache blocks is used. Find the number of **tag** bits, **index** bits, and **offset** bits needed.

Offset bits = **6 bits**

Index bits = **$\log_2[64K/(4 \times 64)] = 8$ bits**

Tag bits = **64 – 6 – 8 = 50 bits**

c) **(4 points)** Given a 2-way set-associative cache that uses 32-bit memory addresses divided into 4 bits of offset, 12 bits of index, and 16 bits of tag. Starting with an empty cache, show the **tag**, **index**, and **way** (block 0 or 1) for each of the following sequentially referenced addresses and indicate whether the reference resulted in a **hit** or a **miss**. The replacement policy used is **FIFO**.

Address	Tag	Index	Way	Hit / Miss
0x00553F0F	0x0055	0x3F0	0	Miss
0x00773F01	0x0077	0x3F0	1	Miss
0x00553F02	0x0055	0x3F0	0	Hit
0x005530AC	0x0055	0x30A	0	Miss
0x00773F07	0x0077	0x3F0	1	Hit
0x005530AA	0x0055	0x30A	0	Hit
0x009930AB	0x0099	0x30A	1	Miss
0x00993F05	0x0099	0x3F0	0	Miss

Q6. [12 points] Cache Performance

A processor runs at 2.5 GHz and has a CPI=1.7 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 for both caches. Assume that the time required to transfer a block of data from the main memory to the cache, i.e. miss penalty, is 40 ns.

a) **(4 Points)** Compute the number of stall cycles per instruction and the overall CPI.

Combined misses per instruction = 4% + 15% × 6% = 0.049
Miss Penalty = 40 ns × 2.5 GHz = 100 cycles
Memory stall cycles per instruction = 0.049 × 100 = 4.9 cycles
Overall CPI = 1.7 + 4.9 = 6.6 cycles

b) **(4 Points)** Compute the average memory access time (AMAT) in ns.

AMAT(I-Cache) = Hit time + Miss rate(I-Cache) × Miss penalty
= (1 cycle / 2.5 GHz) + 0.04 × 40 ns = 2 ns

AMAT(D-Cache) = Hit time + Miss rate(D-Cache) × Miss penalty
= (1 cycle / 2.5 GHz) + 0.06 × 40 ns = 2.8 ns

AMAT = 1/(1+ %LS) × AMAT(I-Cache) + %LS/(1+%LS) × AMAT(D-Cache)
= 1/(1+0.15) × 2 ns + 0.15/(1+0.15) × 2.8 ns = 2.104 ns

Alternative Solution:

Combined Miss Rate = Combined Misses per Instruction / (1 + %LS)
Combined Miss Rate = 0.049 / (1 + 0.15) = 0.0426

AMAT = 0.4 ns + 0.0426 × 40 ns = 2.104 ns

c) **(4 Points)** Discuss how the average memory access time (AMAT) can be reduced by mentioning all the factors that could reduce it and for each factor explaining how it can be done.

The average memory access time can be reduced by:

1. **Reducing the Hit time by using Small and simple caches**
2. **Reducing the Miss Rate by using Larger cache size, higher associativity, and larger block size**
3. **Reducing the Miss Penalty by using Multilevel caches**