Advanced Lab in Computer Architecture

1. **Harvard Architecture**

Advantages	Disadvantages:
Using two memories (data, instructions) allows using them in more effective way (reduces hazards, variable word size and high bandwidth)	Separating memories can create a wasteful situation where the instruction memory is full, and the data memory isn't but no more instructions can be loaded even when there is free memory.
Setting priority in memory access (fetch Vs load/store) is no longer needed	Control unit is more complex to implement
Using pipeline will increase CPI and reduce execution time	Additional registers and logic will require more area and will increase chip cost
	A more complex chip, with more inputs and outputs will require a more sophisticated board, which will increase board cost

2. **Hazards**

2.1. Structural Hazards

A structural hazard occurs when 2 (or more) pipelined instructions try to access a resource which can provide only 1 of them. In Harvard architecture this is happening when an **LD** operation is following and **ST** operation (both try to access memory in the same cycle). Another option is when **DMA** is activated (again, memory access on same cycle). These structural hazards are resolved in the following ways:

1) LD-ST Hazard: stalling the pipeline until ST finished EXEC1 state.
2) DMA-CPU Hazard: prioritizing CPU over DMA to avoid unnecessary stalls.

Cycle	0	1	2	3	4	5	6	7
Inst 0 – ST	FETCH0	FETCH1	DECODE0	DECODE1	EXEC0	EXEC1		
Inst 1 - LD		FETCH0	FETCH1	DECODE0	DECODE1	(STALL)	EXEC0	EXEC1

2.2. Data Hazards

A data hazard occurs when there are dependencies between pipelined instruction. when instruction A needs to update register Ri and instruction B needs Ri for execution.

To detect data hazards, source registers of instruction j (in decode stage) will be compared to destination registers of previous instructions which are still in the pipeline.

these hazards are resolved stalling the pipeline unstill the sources of the instruction are ready, meaning are not destination registers of previous instructions.

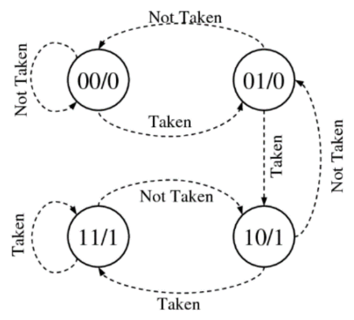
Cycle	0	1	2	3	4	5	6	7	8
ADD r2 r3 r4	FETCH0	FETCH1	DECODE0	DECODE1	EXEC0	EXEC1			
SUB r5 r2 r3		FETCH0	FETCH1	DECODE0	(STALL)	(STALL)	DECODE1	EXEC0	EXEC1

2.3. Control Hazard

A control hazard occurs when a branch was wrongly taken, meaning wrong instructions entered the pipeline.

To decrease the frequency of those hazards we use branch prediction. If , even with prediction, a wrong branch has been taken all wrong instructions have to be flushed out of the pipeline. After flushing, new (and correct) instructions will be fetched.

3. In our design the branch predictor is 2-bit branch predictor.
   It can be described using the following state machine:
   We will use a single predictor for the whole CPU which will be used in every branch and updated in every branch resolution.



4. ***Implemented in code:*** under prog204620702

5. **Files are provided:** under other204620702/Q5

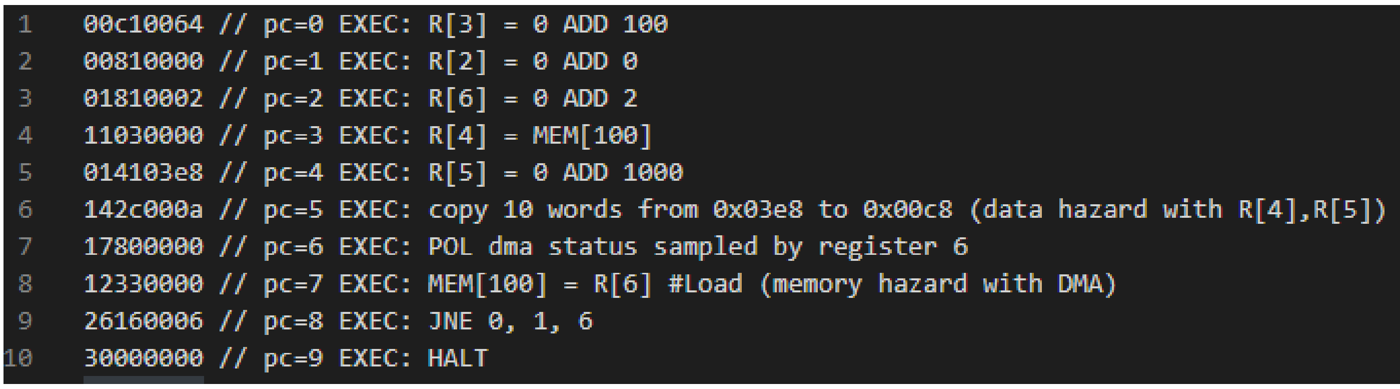
6.

Application	Previous Execution Time	Current Execution Time	Speed Up
Example	366 cycles	155 cycles	**x2.36**
Multiplication	168 cycles	91 cycles	**x1.85**
Fibonacci	1644 cycles	558 cycles	**x2.95**

   The new CPI is not 1 as management hoped since they have been neglecting hazards & stalls. Considering other architectures (speculative architecture which execute instruction Out Of Order such as Tomosulo and Scoreboard) can improve CPI to 1 and beyond.

7. ***DMA is Implemented in code:*** results are under other204620702/Q7

   DMA ASM code:



```
00c10064 // pc=0 EXEC: R[3] = 0 ADD 100
00810000 // pc=1 EXEC: R[2] = 0 ADD 0
01810002 // pc=2 EXEC: R[6] = 0 ADD 2
11030000 // pc=3 EXEC: R[4] = MEM[100]
014103e8 // pc=4 EXEC: R[5] = 0 ADD 1000
142c000a // pc=5 EXEC: copy 10 words from 0x03e8 to 0x00c8 (data hazard with R[4],R[5])
17800000 // pc=6 EXEC: POL dma status sampled by register 6
12330000 // pc=7 EXEC: MEM[100] = R[6] #Load (memory hazard with DMA)
26160006 // pc=8 EXEC: JNE 0, 1, 6
30000000 // pc=9 EXEC: HALT
```