Almost all questions should be answered in this word document directly. Only Problem 4: Question 1 should be answered in Prob4Question1.xlsx.

Before we begin, here is the diagram of the LC-2200 pipelined datapath you should reference:

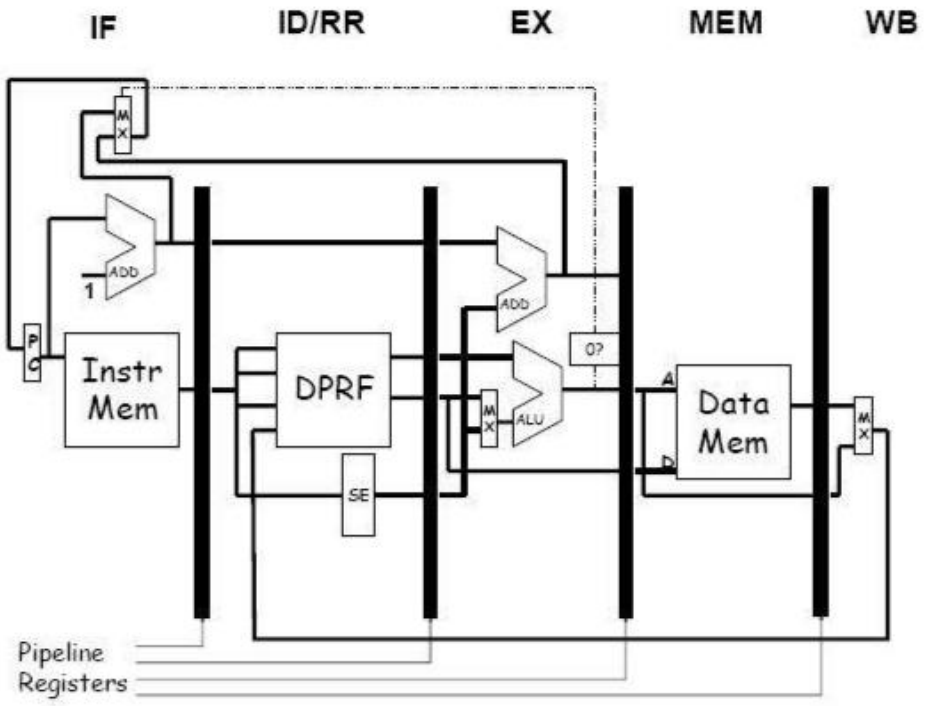

Figure 1: The Pipelined Datapath for LC-2200

Problem 1: Pipelining - Branch Prediction

1. **[10 points]** Regardless of whether we use the conservative approach or branch prediction (where we assume that the branch is not taken), explain why there is always a two cycle delay in the pipeline if the branch is taken (i.e. why two NOPs are injected into the pipeline) before normal execution resumes.

If branch is taken, we will start fetching the instruction from new address, but this new address is loaded into pc at the end of EX stage for BEQ instruction. Therefore there will be stall(noop) in ID/RR and EX ➔ two noop. if the branch is taken.

Problem 2: Pipelining - Data Hazards

Note that you may ONLY assume the following:

- Branching is handled conservatively (or in other words, stop new instructions from entering the pipeline when a branch is detected in ID/RR)

- Branch instructions can be completely determined (i.e. if the branch is taken or not, as well as the

branch's target address) in the EX stage.

• ADD/NAND/ADDI/LW/SW results are not available until their WB stage. Meaning there is **No Data Forwarding**. Must insert NOOPs accordingly.

• All reads in ID/RR stage are done after Writes in WB Stage.

Now, consider the following code fragment that increments the elements of an array stored in memory. At the beginning of each iteration, the register $a0 contains the address of the element to be incremented and $a1 contains the length of the array in bytes plus $a0. Here is the code listing:

```
LOOP: LW $t0, 0($a0)              ; I1
      ADD $t0, $a0, $t0           ; I2
      SW $t0,0($a0)               ; I3
      ADDI $a0, $a0, 1            ; I4
      BEQ $a1, $a0, DONE          ; I5
      BEQ $zero, $zero, LOOP      ; I6
DONE: HALT                        ; I7
```

i. **[10 points]**

(a) Simulate the state of the pipeline for twenty clock cycles and show which instruction is in each stage at each clock cycle. Use one line per cycle and one column per stage. Assume the instructions immediately before I1 were effectively NOOPs. Furthermore, assume that the first branch is not taken for several iterations. Assume $a1 = 10, and $a0 = 0 initially.

Cycle	**IF**	**ID/RR**	**EX**	**MEM**	**WB**
1	I1				
2	I2	I1			
3	I3	I2	I1		
4	I3	I2	NOOP	I1	
5	I3	I2	NOOP	NOOP	I1
6	I4	I3	I2	NOOP	NOOP
7	I4	I3	NOOP	I2	NOOP
8	I4	I3	NOOP	NOOP	I2
9	I5	I4	I3	NOOP	NOOP
10	I6	I5	I4	I3	NOOP
11	I6	I5	NOOP	I4	I3
12	I6	I5	NOOP	NOOP	I4
13	I6	NOOP	I5	NOOP	NOOP
14	I7	I6	NOOP	I5	NOOP

15	I7	NOOP	I6	NOOP	I5
16	I1	NOOP	NOOP	I6	NOOP
17	I2	I1	NOOP	NOOP	I6
18	I3	I2	I1	NOOP	NOOP
19	I3	I2	NOOP	I1	NOOP
20	I3	I2	NOOP	NOOP	I1

(b) What is the average CPI (Cycles Per Instruction) for the loop?

17/6

(c) If the processor operates at 1 GHz (that's 1×10^9 Hz), how long will it take to operate on an array of 5 million words?

5000000 * 17 / 6 * (10^-9) = 0.014 sec ➔ 1.4 * 10^-2 seconds

ii. **[5 points]** Identify the hazards in the code if it needed to run on a generic pipeline that does not support fixes for dependencies

(a) Data Hazards

RAW – Read after write

WAW – Write after write

WAR – write after read

(b) Structural Hazards

DPRF instead of SPRF in the ID/RR(at most two register values are needed by any of instructions)

Two ALUs in Execution stage(when BEQ, we can do comparison and address computation using two ALU's in one cycle)

Having a memory in each IF and MEM stages (IF memory is to get instruction address, fetch or store memory operand is MEM stage(to get data))

(c) Control Hazards

Branch prediction(taken or not taken)

Conservative approach

Delayed Branch

iii. **[10 points]** Answer these questions about the LC-2200 pipelined data path from Problem 1.

(a) What is the number of IPC of the LC-2200 pipelined data path?

1

(b) Why do we need a memory in both the Fetch and Memory stages?

At fetch, we want to get instruction into IR(to get instruction address)

At MEM, we want to fetch or store memory operand(to get data)

(c) Why is it important for the register file to be dual ported in a pipelined processor?

In LC2200, at most two register values are needed by any of instructions, Therefore DPRF is important because it can simultaneously take two register addresses and contents of the two registers can be read out in the same time.

Problem 3: Pipelining – Waterfall Diagram with Data Forwarding:

You are given a five stage pipeline with the following features: **[25 Pts]**

- There is Data forwarding from EX and MEM to ID/RR
- There is a static predictor in the IF stage, that always predicts the branch as not taken
- Branches are resolved in the I/RR stage
- Writes happen before reads (i.e. Register writes precede register reads)

The following sequence of instructions is run on this pipeline, fill out the cycle by cycle waterfall diagram. Assume all registers are initialized to zero, and the static predictor knows which addresses are branches.

```
addi R1, R0, 10                 ; I1
lea  R3, arr                    ; I2
loop:   beq  R2, R1, end        ; I3
        ldr  R4, 0(R3)          ; I4
        addi R4, R4, 1          ; I5
        STR  R4, 0(R3)          ; I6
        addi R3, R3, 1          ; I7
        addi R2, R2, 1          ; I8
        beq  R0, R0, loop       ; I9
end:    halt                    ; I10

arr: 0x4000
```

LEA R3 Arr, do I need noop at I4? Because we gotta write 4000 at r3

Cycle	IF	ID/RR	EX	MEM	WB
1	I1				
2	I2	I1			
3	I3	I2	I1		
4	I4	I3	I2	I1	
5	I5	I4	I3	I2	I1
6	I6	I5	I4	I3	I2
7	I6	I5	NOOP	I4	I3
8	I6	I5	NOOP	NOOP	I4
9	I7	I6	I5	NOOP	NOOP
10	I8	I7	I6	I5	NOOP
11	I8	I7	NOOP	I6	I5
12	I8	I7	NOOP	NOOP	I6
13	I9	I8	I7	NOOP	NOOP
14	I10	I9	I8	I7	NOOP
15	I3	NOOP	I9	I8	I7
16	I4	I3	NOOP	I9	I8
17	I5	I4	I3	NOOP	I9
18	I6	I5	I4	I3	NOOP
19	I6	I5	NOOP	I4	I3
20	I6	I5	NOOP	NOOP	I4

Problem 4: Process Scheduling

1. **[16 points]** Consider the following set of processes, with the length of the CPU burst time in milliseconds. The processes are assumed to have arrived in the order P1, P2, P3, P4, and P4 at time = 0.

Table 1: Processes, their Burst Times, and their Priorities

Process	Burst Time (ms)	Priority
P1	5	2
P2	4	5
P3	1	3
P4	6	1
P5	2	4

Draw four Gantt Charts illustrating the execution of these processes using FCFS (First Come First Serve), SJF (Shortest Job First), Non-Preemptive Priority (lower numbers = higher priority), and RR (Round Robin) with a time quantum of 1 (ignoring priority). Note that a process is no longer waiting once it has completed its burst time. **Submit this question in Prob4Question1.xlsx, question 2 can be answered on here. Look at Prob4Question1.xlsx for the appropriate format to use.**

2. **[14 points]**

i. What is the waiting time for each process for each of the scheduling algorithms in question 1?
FCFS : P1 ➔ 0, P2 ➔ 5, P3 ➔ 9, P4 ➔ 10, P5 ➔ 16
SJF : P1 ➔ 7, P2 ➔ 3, P3 ➔ 0, P4 ➔ 12, P5 ➔ 1
NONPREEMTIVE PRIORITY : P1 ➔ 6, P2 ➔ 14, P3 ➔ 11, P4 ➔ 0, P5 ➔ 12
RR : P1 ➔ 11, P2 ➔ 10, P3 ➔ 2, P4 ➔ 12, P5 ➔ 7

ii. What is the turnaround time of each process for each of the scheduling algorithms in question 1?
FCFS : P1 ➔ 5, P2 ➔ 9, P3 ➔ 10, P4 ➔ 16, P5 ➔ 18
SJF : P1 ➔ 12, P2 ➔ 7, P3 ➔ 1, P4 ➔ 18, P5 ➔ 3
NONPREEMTIVE PRIORITY : P1 ➔ 11, P2 ➔ 18, P3 ➔ 12, P4 ➔ 6, P5 ➔ 14
RR : P1 ➔ 16, P2 ➔ 14, P3 ➔ 3, P4 ➔ 17, P5 ➔ 9

iii. Which of the scheduling policies in question 1 results in the minimum average waiting time over all processes?
SJF

iv. Which scheduling algorithm has provably optimal average waiting time?
SJF

v. Which scheduling algorithm has the highest variance in turnaround time in general?
FCFS

vi. List the scheduling algorithm which suffers from starvation.
SJF

vii. Which of the above scheduling algorithms require a timer interrupt and preemption for their correct operation?
RR

Problem 5: Memory Management

1. **[10 points]** Explain how memory is allocated for processes in varying sized partitions versus how it is in fixed sized partitions. Be sure to mention the presence of either external or internal fragmentation in each method.

For fixed-size partitions, memory is allocated statically with fixed sizes. Therefore when process requests memory, it is given one of those fixed size partition that is equal or greater than the requested memory which can result in wasting memory if requested is smaller than fixed size. This incident is Internal Fragmentation which is poor memory utilization.
Now, even if we can use two fixed size partitions to provide memory requested for example, 6KB is requested, and we have 1KB and 5KB fixed size partitions, its not possible because 1KB, and 5KB memory partitions are not contiguous. And this is External Fragmentation.
For Varying-size partitions, memory is allocated dynamically therefore it fixes the problem of internal fragmentation. However, it does not fix the external fragmentation because available spaces could still be not contiguous to each other upon request.

Submission:

Please upload and submit the following:

1. **This document as a pdf**
2. **Prob4Question1.xlsx**