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:

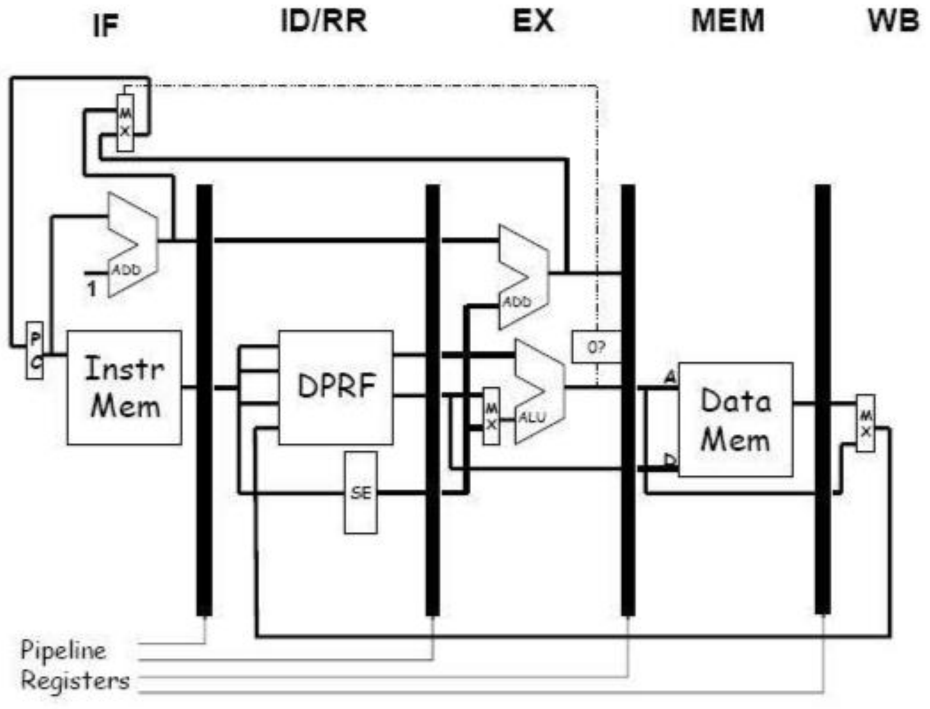


Figure 1: The Pipelined Datapath for LC-2200

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.

Using either conservative or prediction branch not taken will cause the instructions (sequentially) after branch to be loaded into IF and ID while beq moves to EX. But once we know the branch will be taken, we must load the new address into the PC in IF while beq moves to MEM. However, the contents of ID and EX must be squashed with NOOPs because they are no longer applicable since the branch is taken. Two NOOPs cause two cycles of delay.

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	NOP	I1	
5	I3	I2	NOP	NOP	I1
6	I4	I3	I2	NOP	NOP
7	I4	I3	NOP	I2	NOP
8	I4	I3	NOP	NOP	I2
9	I5	I4	I3	NOP	NOP
10	I6	I5	I4	I3	NOP
11	I6	I5	NOP	I4	I3
12	I6	I5	NOP	NOP	I4

13	I6	NOP	I5	NOP	NOP
14	I7	I6	NOP	I5	NOP
15	I7	NOP	I6	NOP	I5
16	I1	NOP	NOP	I6	NOP
17	I2	I1	NOP	NOP	I6
18	I3	I2	I1	NOP	NOP
19	I3	I2	NOP	I1	NOP
20	I3	I2	NOP	NOP	I1

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

16 cycles to complete the six-instruction loop. So avg CPI = 15/6.

(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 * 15/6 * 1/(1 GHz) = 0.0125 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 (I2, I1)

RAW (I3, I2)

WAR (I4, I3)

RAW (I5, I4)

WAW (I2, I1)

(b) Structural Hazards

The EX stage needs two ALUs for branch instructions, because it needs to compare the two registers' contents and needs to compute the PC += offset. Without two ALUs one NOP will be needed.

(c) Control Hazards

I5 and I6 because they are branch instructions.

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, under ideal circumstances.

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

For efficiency's sake, because the first holds only instructions and the other only data. With only one memory, all reading and writing would have to be done in one stage of the pipeline which would create bubbles.

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

So that it can be read from and written to within the same clock cycle.

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.

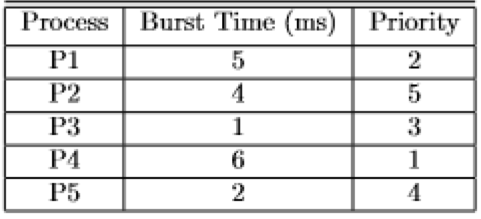
```
addi R1, R0, 10                    ; I1
lea  R3, arr                       ; I2
loop:   beq  R2, R1, end           ; I3
        ldr  R4, 0(R3)             ; I4
        addi R4, R4, 1             ; I5
        STR  R4, 0(R3)             ; I6
        addi R3, R3, 1             ; I7
        addi R2,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

arr: 0x4000
```

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	NOP	I4	I3
8	I7	I6	I5	NOP	I4
9	I8	I7	I6	I5	NOP
10	I9	I8	I7	I6	I5
11	I10	I9	I8	I7	I6
12	I3	NOP	I9	I8	I7
13	I4	I3	NOP	I9	I8
14	I5	I4	I3	NOP	I9
15	I6	I5	I4	I3	NOP
16	I6	I5	NOP	I4	I3
17	I7	I6	I5	NOP	I4
18	I8	I7	I6	I5	NOP
19	I9	I8	I7	I6	I5
20	I10	I9	I8	I7	I6

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

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

Priority
P1: 11
P2: 18
P3: 12
P4: 6
P5: 14

RR
P1: 16
P2: 14
P3: 3
P4: 18
P5: 9

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

SJF, with average of 23/5

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

SJF, because every finished job reduces the overall waiting time (because finished jobs do not wait) and SJF finishes jobs most quickly (by doing the shortest first)
(Priority can also be, depending on implementation)

v. Which scheduling algorithm has the highest variance in turnaround time in general?
FCFS, because it always takes the next job regardless of turnaround time, totally random
(Priority can also be, depending on implementation)

vi. List the scheduling algorithm which suffers from starvation.

SJF, longer algorithms can be forever preempted by shorter ones
(Priority can also be, depending on implementation)

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.

Memory is allocated in fixed size, in chunks large enough to store the process. How the chunk is selected depends on whether best-fit (closest size larger chunk to process) or first-fit (first larger chunk) algorithm is used. Internal fragmentation is common, and occurs when the chunk size is larger than process size. The excess space cannot be used and is wasted. External fragmentation also occurs when sufficient space is available but the space is in several non-contiguous chunks. It can be solved by compaction, relocating memory chunks and merging them into one larger chunk.
Variable size allocation gives each process a chunk exactly the size needed by the process. This removes the need for algorithms like best-fit/first-fit, and also removes internal fragmentation, since there will never be excess space in a chunk. However, external fragmentation is still a problem and must still be solved through compaction.

Submission:

Please upload and submit the following:

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