Lincoln Sand

1. Consider an unpipelined or single-stage processor design like the one discussed in slide 19 of lecture 15. At the start of a cycle, a new instruction enters the processor and is processed completely within a single cycle. It takes 7,000 ps to navigate all the circuits in a cycle (including latch overheads). Therefore, for this design to work, the cycle time has to be at least 7,000 pico seconds.

a. What is the clock speed of this processor? (5 points)

$$\text{Clock Speed (Hz)} = \frac{1}{\text{Cycle Time (Seconds)}}$$

Plugging in the values (after converting the picoseconds to seconds) gives: $142,857,142.86\text{Hz} \approx 142.86\text{MHz}$.

b. What is the CPI of this processor, assuming that every load/store instruction finds its instruction/data in the instruction or data cache? (5 points)

The CPI is 1, since every instruction completes in a single cycle.

c. What is the throughput of this processor (in billion instructions per second)? (10 points)

The throughput is the clockspeed multiplied by the CPI. Since the CPI is 1, it's just the clockspeed. This is approximately 143 million instructions per second.

2. The processor in Question 1 above is converted into a 14-stage pipeline. The slowest of these 14 stages takes 600 ps (including latch overheads).

a. What is the clock speed of this processor? (5 points)

The clock speed in a pipelined processor is determined by the longest stage in the pipeline. Since the slowest stage takes 600 ps, this is the limiting factor.

The clock speed works out to approximately 1.67 GHz ($1,666,666,666.67$ Hz).

b. What is the CPI of this processor, assuming that every load/store instruction finds its instruction/data in the instruction or data cache, and there are no stalls from data/control/structural hazards? (5 points)

The CPI (assuming no stalls or hazards) is 1.

c. What is the throughput of this processor (in billion instructions per second)? (10 points)

The throughput is approximately 1.67 billion instructions per second.

d. What is the speedup, relative to the unpipelined processor in Q1? Why is the speedup less than 14X? (10 points)

The speedup is approximately 11.67 times. The speedup is less than 14 because it is constrained by the slowest stage in the pipeline. In an ideal scenario, the speedup would be equal to the number of stages exactly (14). However, the actual speedup is lower due to the fact that the pipeline's speed is dictated by its slowest stage, not the average speed of all stages.

3. Show how the following three consecutive instructions move through each stage of the five stage pipeline, similar to the example on slide 10 of lecture 18. This pipeline does not support any bypassing. Make sure the decode stage does not advance an instruction through the pipeline unless all data dependences are correctly resolved. (25 points)

```
I1: add $s1, $s2, $s3
I2: lw $s4, 4($s1)
I3: add $s5, $s4, $s1
```

I1: add $s1, $s2, $s3 - This instruction adds the contents of $s2 and $s3 and stores the result in $s1.

I2: lw $s4, 4($s1) - This instruction has a data dependency on I1 as it requires the address in $s1 to be updated before it can proceed.

I3: add $s5, $s4, $s1 - This instruction has a dependency on I2 (for $s4) and on I1 (for $s1).

```
Cycle 1:
I1: IF

Cycle 2:
I1: ID
I2: IF

Cycle 3:
I1: EX
I2: ID  (Stalls due to dependency on I1's result)
I3: IF
```

```
Cycle 4:
I1: MEM
I2: ID  (Stalls due to dependency on I1's result)
I3: IF

Cycle 5:
I1: WB
I2: ID  (Can now proceed as I1 has written back the result)
I3: IF

Cycle 6:
I1: -
I2: EX
I3: ID  (Stalls due to dependency on I2's result)

Cycle 7:
I1: -
I2: MEM
I3: ID  (Stalls due to dependency on I2's result)

Cycle 8:
I1: -
I2: WB
I3: ID  (Can now proceed as I2 has written back the result)

Cycle 9:
I1: -
I2: -
I3: EX

Cycle 10:
```

I1: -
I2: -
I3: MEM

Cycle 11:
I1: -
I2: -
I3: WB

4. Show how the same three instructions move through each stage of the five stage pipeline, similar to the example on slide 12 of lecture 18. This pipeline does support bypassing. Make sure the decode stage does not advance an instruction through the pipeline unless all data dependences are correctly resolved. You don't need to show the latch involved in every bypass (but feel free to ponder this question for your own understanding). (25 points)

```
I1: add $s1, $s2, $s3
I2: lw $s4, 4($s1)
I3: add $s5, $s4, $s1
```

With bypassing (also known as forwarding) in the pipeline, the instructions can proceed without waiting for the dependent values to be written back to the registers. This reduces the number of stalls. However, the load word instruction (lw) has a load-use data hazard, which means even with bypassing, the subsequent instruction (add $s5, $s4, $s1) cannot immediately use the value loaded by lw. This is because the data is available only after the memory access stage of the load instruction.

```
Cycle 1:
I1: IF

Cycle 2:
I1: ID
I2: IF

Cycle 3:
```

```
I1: EX
I2: ID (Stalls due to dependency on I1's result for $s1)
I3: IF

Cycle 4:
I1: MEM
I2: EX  (Bypassing allows $s1 to be used here from I1's EX st
I3: ID

Cycle 5:
I1: WB
I2: MEM
I3: ID (Stalls due to dependency on I2's result for $s4)

Cycle 6:
I1: -
I2: WB
I3: EX  (Bypassing allows $s1 to be used here from I2's WB st

Cycle 7:
I1: -
I2: -
I3: MEM

Cycle 8:
I1: -
I2: -
I3: WB
```

I1 can proceed without interruption as it has no dependencies.

I2 is delayed in the ID stage waiting for I1 to complete its EX stage, but then it can proceed in the next cycle due to bypassing from I1's EX stage to I2's ID stage for $s1.

I3 must wait until I2 completes its MEM stage because I2 is a load and I3 needs the loaded value. Once I2 reaches MEM, I3 can use the value being written back (bypassed) to proceed with its EX stage.