HW 5 Solutions

Manoj Mardithaya

Question 1: Processor Performance

The critical path latencies for the 7 major blocks in a simple processor are given below.

	IMem	Add	Mux	ALU	Regs	DMem	Control
a	400ps	100ps	30ps	120ps	200ps	350ps	100ps
b	500ps	150ps	100ps	180ps	220ps	1000ps	65ps

For each part, answer the following questions:

1. What is the critical path for a MIPS ADD instruction? Explain your break-up.

   Using the diagram in the book, three of the many possible paths through the circuit are:

   Add + Add + Mux Imem + Control + Mux + ALU + DMem + Mux Imem + Mux + Reg + Mux + ALU + DMem + Mux

   The longest of these (and the other shorter combinations that I didn't list) is the critical path. Note that the instruction type doesn't affect the critical path – the critical path through the circuit is the same because it determines how quickly you can do the next operation – even if an instruction finishes meaningful work 'early', it still has to go through the rest of the circuit so your timing and signals stay in sync.

   For both a. and b., the longest is the third path:

   $$400 + 30 + 200 + 30 + 120 + 350 + 30 = 1160$$

   $$500 + 100 + 220 + 100 + 180 + 1000 + 100 = 2200$$

2. If the number of registers is doubled, this increases Regs by 100ps and Control by 20ps. This results in 10% fewer instructions due to fewer load/stores. What is the new critical path for a MIPS ADD instruction?

   Again, the *10% fewer instructions* doesn't affect the critical path. (*it would affect program runtime, because of fewer instructions.*). The changes don't change the critical path, they only increase the length of it to $1260ps$and$2300ps$ respectively.

Question 2: Pipelining

The 5 stages of the processor have the following latencies:

	Fetch	Decode	Execute	Memory	Writeback
a.	300ps	400ps	350ps	500ps	100ps
b.	200ps	150ps	120ps	190ps	140ps

Assume that when pipelining, each pipeline stage costs 20ps extra for the registers between pipeline stages.

1. Non-pipelined processor: what is the cycle time? What is the latency of an instruction? What is the throughput?

   Cycle-time: there is no pipelining, so the cycle-time has to allow an instruction to go through all the stages each cycle. Therefore:

   a. CT = 1650ps

   b. CT = 800ps

   The latency for an instruction is also the same, since each instruction takes 1 cycle to go from beginning fetch to the end of writeback. The throughput similarly is $\frac{1}{cycle \quad time}$ instructions per second.

2. Pipelined processor: What is the cycle time? What is the latency of an instruction? What is the throughput?

   Pipelining to 5 stages reduces the cycle time to the length of the longest stage. Additionally, the cycle time needs to be slightly longer to accomodate the register at the end of the stage.

   a. CT = 520ps

   b. CT = 220ps

   The latency for both is $5 * (cycle \quad time)$, since an instruction needs to go through 5 pipeline stages, spending 1 cycle in each, before it commits.

   The throughput for both is still 1 instruction/cycle. Throughput increases because the cycle time reduces.

3. If you could split one of the pipeline stages into 2 equal halves, which one would you choose? What is the new cycle time? What is the new latency? What is the new throughput?

   Splitting the longest stage is the only way to reduce the cycle time. After splitting it, the new cycle time is based on the new longest stage.

   a. Old longest stage is Memory. New CT = 420ps

   b. Old longest stage is Fetch. New CT = 210ps

   The new latency is $6*(cycle \quad time)$, since an instruction needs to go through 6 pipeline stages now.

The throughput for both is still 1 instruction/cycle. Throughput increases because the cycle time reduces.

4. Assume the distribution of instructions that run on the processor is:

   - 50% ALU
   - 25%: BEQ
   - 15%: LW
   - 10%: SW

   Assuming there are no stalls or hazards, what is the utilization of the data memory? What is the utilization of the register block's write port? (Utilization in percentage of clock cycles used)

   Data memory is utilized only by LW and SW instructions in the MIPS ISA. So the utilization is 25% of the clock cycles.

   The write port may be utilized by ALU and LW instructions. The utilization is 65% of the clock cycles.

   Calculating utilization in terms of *% time* though gives different results, because unless the corresponding stage (data memory or writeback) is the stage dictating cycle time, the circuit is idle for some period of the clock cycle in *all* cycles.

Question 3: Stalling

```
Sequence of instructions:
   1.  lw $s2, 0($s1)
   2.  lw $s1, 40($s6)
   3.  sub $s6, $s1, $s2
   4.  add $s6, $s2, $s2
   5.  or $s3, $s6, $zero
   6.  sw $s6, 50($s1)
```

1. Data dependencies: A data dependence is a dependence of one instruction B on another instruction A because the value produced by A is read by B. So when listing data dependencies, you need to mention A, B, and the location (in this case, the register that causes the dependence). It also doesn't matter how far apart dependencies are. Don't confuse data dependencies and hazards!

   Dependencies:

   `3 depends on 1 ($s2)`

   `3 depends on 2 ($s1)`

   `4 depends on 1 ($s2)`

   `5 depends on 4 ($s6)`

```
6 depends on 2 ($s1)
6 depends on 4 ($s6)
```

2. Assume the 5-stage MIPS pipeline with no forwarding, and each stage takes 1 cycle. Instead of inserting nops, you let the processor stall on hazards. How many times does the processor stall? How long is each stall (in cycles)? What is the execution time (in cycles) for the whole program?

   Ignoring the stalls for a moment, the program takes 10 cycles to execute – not 6, because the first 4 cycles, it does not commit (finish) an instruction – those 4 cycles, the pipeline is still filling up. It is also not 30, because when the first instruction commits, the 2nd instruction is nearly done, and will commit in the next cycle. Remember, pipelines allow multiple instructions to be executing at the same time.

   With the stalls, there are only two stalls – after the 2nd load, and after the add – both are because the next instruction needs the value being produced. Without forwarding, this means the next instruction is going to be stuck in the fetch stage until the previous instruction writes back. These are 2 cycle stalls (when in doubt, draw diagrams like the ones on the Implementing MIPS slides, slide 63). So to answer the question, 2 stalls, 2 cycles each, and the total is $10 + 2 * 2 = 14$ cycles to execute the program.

3. Assume the 5-stage MIPS pipeline with full forwarding. Write the program with nops to eliminate the hazards. (Hint: time travel is not possible!)

   Again, draw a diagram – in the second stall, the result of the add is available at the register at the end of the execute stage when the next instruction wants to move to the execute stage, so you can forward the value of $s6 as the input to the execution stage (as the argument for the or) – this removes the second 2-cycle stall. However, the loaded value is not ready until the end of the memory stage, so you cannot using forwarding to remove both cycles – you still need to wait 1 cycle. The solution is to place a NOP after the second load. (Note: this is also the reason why MIPS has load delay slots).

Speaking of delay slots, the question was ambiguous in whether delay slots were used or not. If they are, all the answers are slightly different:

Part 1: You have one fewer dependence - 3 does not depend on 2, because it is in the delay slot.

Part 2: The first stall is only 1 cycle, so the program executes in 13 cycles.

Part 3: You do not require any nops, because of the delay slot.

Question 4: More Pipelines

You are given a non-pipelined processor design which has a cycle time of 10ns and average CPI of 1.4. Calculate the latency speedup in the following questions.

The solutions given assume the base CPI = 1.4 throughput. Since the question is ambiguous, you could assume pipelining changes the CPI to 1. The method for computing the answers still apply.

1. What is the best speedup you can get by pipelining it into 5 stages?

   Since IC and CPI don't change, and, in the best case, pipelining will reduce CT to 2ns:

$$Speedup = \frac{CT_{old}}{CT_{new}} = \frac{10ns}{2ns} = 5\text{x Speedup}$$

2. If the 5 stages are 1ns, 1.5ns, 4ns, 3ns, and 0.5ns, what is the best speedup you can get compared to the original processor?

   The cycle time is limited by the slowest stage, so CT = 4 ns.

$$Speedup = \frac{CT_{old}}{CT_{new}} = \frac{10ns}{4ns} = 2.5\text{x Speedup}$$

3. If each pipeline stage added also adds 20ps due to register setup delay, what is the best speedup you can get compared to the original processor?

   Adding register delay to the cycle time because of pipeline registers, you get CT = 4.02 ns.

$$Speedup = \frac{CT_{old}}{CT_{new}} = \frac{10ns}{4.02ns} = 2.49\text{x Speedup}$$

4. The pipeline from Q4.3 stalls 20% of the time for 1 cycle and 5% of the time for 2 cycles (these occurences are disjoint). What is the new CPI? What is the speedup compared to the original processor?

   20% of the time: $CPI = 1.4 + 1 = 2.4$

   5% of the time: $CPI = 1.4 + 2 = 3.4$

   75% of the time: $CPI = 1.4$

   New average CPI: $2.4 * 0.2 + 3.4 * 0.05 + 1.4 * 0.75 = 1.7$

$$Speedup = \frac{CT_{old} * CPI_{old}}{CT_{new} * CPI_{new}} = \frac{10 * 1.4}{4.02 * 1.7} = 2.049\text{x Speedup}$$