|  |
| --- |
| **Assignment # 4**  **SYSC 5704 – Elements of Computer Systems** |
| Fall 2014  Submitted To  Dr. R. Gregory Franks  By  **Ferhan Jamal (100 953 487)**  Carleton University |

**4.3 When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance trade-off. In the following three problems, assume that we are starting with a datapath from Figure 4.2, where I-Mem, Add, Mux, ALU, Regs, D-Mem, and Control blocks have latencies of 400 ps, 100 ps, 30 ps, 120 ps, 200 ps, 350 ps, and 100 ps, respectively, and costs of 1000, 30, 10, 100, 200, 2000, and 500, respectively.**

**Consider the addition of a multiplier to the ALU. This addition will add 300 ps to the latency of the ALU and will add a cost of 600 to the ALU. The result will be 5% fewer instructions executed since we will no longer need to emulate the MUL instruction.**

**1. [10] <§4.1> What is the clock cycle time with and without this improvement?**

The value of clock cycle time is determined by the critical path.

Critical path is :

400(I-Mem)+200 (Regs)+30(Mux)+120(ALU)+350(D-Mem)+30(Mux) = 1130ps

*With the improvement clock cycle time is: 1130 ps*

When ALU is added, Critical path will be 1130+120=1250ps

*Without the improvement clock cycle time is: 1130 ps*

**2. [10] <§4.1> What is the speedup achieved by adding this improvement?**

The cycle time will be increased because of the addition of the MUL instructions (5% fewer instructions can be performed) and there will be no changes in the speed -up in the critical path.

The cycle time will be increased because of the addition of the MUL instructions(5% fewer instructions can be performed) and there will be no changes in the speed up in the critical path.

Let us consider 700 instructions at 1130ps, then the value of run time will be 791,000 ps. With the improvement, we have 700-(5% of 700)= 665 instructions at 1430 ps with a run time of 950,950 which implies that the performance is decrease *with 5% few lesser instructions and with the clock cycle improvement*.

**3. [10] <§4.1> Compare the cost/performance ratio with and without this improvement.**

*Case a: Without the improvement*

Let us consider the value of instruction count as 50,000.

The value of run-time will be 50,000\*1130 = 56,500,000ps or 56,500,000 x 10^-12 seconds or 0.0000565 seconds.

Performance (1/Execution Time) = 1/ 0.0000565 = 17,699.1150

The value of cost without the improvement is: 3890

[Registers (200) + ALU(100) + Instruction memory(1000) + Data Memory(2000)+ Control(500)+Add (30\*2 = 60)+Mux(10\*3 = 30) : 3890]

Cost/Performance will be: 3890/17,699.1150= 0.2197

*Case b: With the improvement*

In this case, the value of run-time will be 50,000\*1430 = 71,500,000ps or 71,500,000 x 10^-12 seconds or 0.0000715 seconds.

Performance (1/Execution Time) = 1/ 0.0000715 = 13,986.013

The value of cost with the improvement is: 3890+600 = 4490

Cost/Performance will be: 4490/13,986.013= 0.321

In the last case, the cost and the performance ratio is increased by a little bit margin in *with the improvement case* in comparison with *without the improvement case* so cost/performance ratio is much better in *with the improvement case* in comparison with *without the improvement case.*

**4.8 In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise**

**Also, assume that individual stages of the datapath have the following latencies:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IF** | **ID** | **EX** | **MEM** | **WB** |
| **250ps** | **250ps** | **150ps** | **300ps** | **200ps** |

**Also, assume that instructions executed by the processor are broken down as follows:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Alu** | **Beq** | **Lw** | **Sw** |
| **45%** | **20%** | **20%** | **15%** |

**1. [5] <§4.5> What is the clock cycle time in a pipelined and non-pipelined processor?**

The value of clock cycle time in a non-pipelined processor is the sum of all latencies which is 250+250+150+300+200=1150

The value of clock cycle time in a pipelined processor is the largest latency among all stages which is 300

**2. [10] <§4.5> What is the total latency of an LW instruction in a pipelined and non-pipelined processor?**

**In pipeline processor:**

In a pipelined processor, LW instruction uses all 5 stages with a total latency of 5\*300 = 1500 ps where 300ps is the clock cycle time of a pipeline processor.

**Non-pipelined processor:**

In non-pipelined processor, the total of an LW instruction is the sum of latency of all stages which is 250+250+150+300+200=1150 ps

**3. [10] <§4.5> If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?**

I will Split MEM stage into two stages of 150 ps.

New clock cycle time of the processor will be 250 ps.

**4. [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the data memory?**

Data memory is utilized only by lw and sw instructions in the MIPS ISA if there are no stalls or hazards, so the data memory utilization will be 35% of the clock cycles

**5. [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the write-register port of the “Registers” unit?**

The write-register port is utilized by alu and lw instructions, so the utilization will be 65%.

**6. [30] <§4.5> Instead of a single-cycle organization, we can use a multi-cycle organization where each instruction takes multiple cycles but one instruction finishes before another is fetched. In this organization, an instruction only goes through stages it actually needs (e.g., ST only takes 4 cycles because it does not need the WB stage). Compare clock cycle times and execution times with single-cycle, multi-cycle, and pipelined organization.**

The value of clock cycle times is already calculated for pipe lined and single cycle instructions in the 1st part of this question and the multi-cycled organization has the same clock cycle as the pipelined organization so the value of clock cycle time for the multi-cycled organization would be the same as pipelined organization as obtained in 1st of this question.

The definition of single-cycle organization, multi-cycle, and pipelined organization are as follows:

**Single-cycled organization**: In single-cycled organization, every instruction takes one clock cycle.

**Multi-cycle organization**: As mentioned in the question that in multi-cycle each instruction takes multi-cycles but one instruction finishes before another is fetched.

**Pipelined-Organization:** This technique is used in the computer processor to increase the throughput and performance of the processor. The instruction cycle is broken into a series of steps called pipeline.

The value of speed-up will eventually find the comparison of execution times of single-cycle, multi-cycle and pipelined organization. *We are calculating the value of speed-up here to see how much performance we can get by increasing number of processors.*

A multi-cycle organization completes a lw in 5 cycles, a sw in 4 cycles, an ALU instruction in 4 cycles, and a beq in 4 cycles.

The speed up in the 1st case will be:

Speed up = Multi-cycle execution time/Pipelined execution time

As multi-cycle execution time is n times Pipelined execution time, therefore:

Speed up = .20 x 5 + .80 x 4 = 1+3.2 = 4.2

*Speed up = 4.2*

The speed up in the 2nd case will be:

Speed up = Single-cycle execution time/Pipelined execution time

Speed up= 1150/300 (Values obtained in the 1st part of this question)

*Speed up= 3.8333*

**4.9 In this exercise, we examine how data dependencies affect execution in the basic 5-stage pipeline described in Section 4.5. Problems in this exercise refer to the following sequence of instructions:**

**OR R1, R2, R3**

**OR R2, R1, R4**

**OR R1, R1, R2**

**Also, assume the following cycle times for each of the options related to forwarding:**

|  |  |  |
| --- | --- | --- |
| **Without Forwarding** | **With Full Forwarding** | **With ALU-ALU Forwarding** |
| **250ps** | **300ps** | **290ps** |

**Data Dependencies:-**

Data dependency describes the situation that the data used by the instructions depend upon the data created or generated by other instructions. The data dependency situation can be in this way that the data stored in locations can be used by other instructions. The 3 types of data dependencies between the instructions are as follows:-

**1. Output dependency**

**2. Anti-dependency**

**3. True data dependency**

**1. Output dependency:** It is also called write-after-write hazard and it occurs when the output register of an instruction is used for write after written by a previous instruction that inturn affect the final output value of a variable)

**2. Anti-dependency:** It is also called write-after-read hazard which occurs when an instruction writes to a location which has been read by a previous instruction.

**3. True data dependency :** It is also read-after-write hazard and it occurs when value produced by an instruction is required by another instruction

**1. [10] <§4.5> Indicate dependencies and their type.**

|  |  |
| --- | --- |
| Instructions | Dependence |
| I1: OR R1, R2, R3  I2: OR R2, R1, R4  I3: OR R1, R1, R2 | RAW on R1 from I1 to I2 and I3  RAW on R2 from I2 to I3  WAR on R2 from I1 to I2  WAR on R1 from I2 to I3  WAW on R1 from I1 to I3 |

* Here RAW is Read after Write, WAR is Write After Read, and WAW is Write after Write.
* I1 is Instruction 1, I2 is Instruction 2 and I3 is instruction 3.

**2. [10] <§4.5> Assume there is no forwarding in this pipelined processor. Indicate hazards and add NOP instructions to eliminate them.**

Below is the code that eliminates hazards by inserting NOP instructions:

|  |  |
| --- | --- |
| Instructions | Delay |
| OR R1,R2,R3  NOP  NOP  OR R2,R1,R4  NOP  NOP  OR R1,R1,R2 | Delay I2 to avoid RAW hazard on R1 from I1  Delay I3 to avoid RAW hazard on R2 from I2 |

* Here RAW is Read after Write, WAR is Write After Read, and WAW is Write after Write.
* I1 is Instruction 1, I2 is Instruction 2 and I3 is instruction 3.

**3. [10] <§4.5> Assume there is full forwarding. Indicate hazards and add NOP instructions to eliminate them.**

**Full forwarding:** In Full Forwarding, an ALU instruction can forward a value to the EX stage of the next instruction without a hazard but load cannot forward to the EX stage of the next instruction

|  |  |
| --- | --- |
| Instruction Sequence | Delay |
| OR R1,R2,R3  OR R2,R1,R4  OR R1, R1, R2 | No RAW hazard on R1 from I1  No RAW hazard on R1 from I2 |

* Here RAW is Read after Write, WAR is Write After Read, and WAW is Write after Write.
* I1 is Instruction 1, I2 is Instruction 2 and I3 is instruction 3.

**4. [10] <§4.5> What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?**

The value of total execution time is calculated by the formula:

Total execution time = clock cycle time \* number of cycles.

|  |  |  |
| --- | --- | --- |
| No Forwarding | With Forwarding | Speedup due to Forwarding |
| 250\*(7 + 2) ps = 2250ps | 300\*(7+1)ps = 2400ps | 0.9375 (2250/2400) |

**5. [10] <§4.5> Add NOP instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).**

Below is the code that eliminates hazards by inserting NOP instructions:

|  |  |
| --- | --- |
| OR R1,R2,R3  OR R2,R1,R4  OR R1,R1,R2 | ALU-ALU forwarding of R1 from I1  ALU-ALU forwarding of R2 from I2 |

**6. [10] <§4.5> What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?**

**ALU-ALU forwarding:** In ALU-ALU-only forwarding, an ALU instruction can forward to the next instruction, but not to the second-next instruction. Similar in this case, a load cannot also be forwarded.

|  |  |  |
| --- | --- | --- |
| No Forwarding | With ALU-ALU Forwarding only | Speedup with ALU Forwarding |
| 250\*(7 + 2) ps = 2250ps | 290\*(7) ps = 2030ps | 1.1083(2030/2250) |