EEC 170

Assignment 3 – Chapter 4

1. **(2 pts)** In a single-cycle CPU (no pipelined stages), what is the value of the program counter after each of the following instructions execute?
   1. 0x4 addi x2 x0 0xF
   2. 0x8 jal x0 0xF

**The RISC-V ISA has extensions that support 16-bit instructions (which would occupy 2 bytes). Therefore, branch/jump immediates are left-shifted by 1 to improve the instruction memory space accessible by branch and jump instructions. Otherwise, the LSB of branch and jump immediates would never be used. See Section 4.3 for more information on branching instructions.**

1. **(2 pts)** What are the five stages in the pipelined processor presented in Chapter 4? What is the purpose of each stage?

**Instruction Fetch (IF) : Fetches the instruction from instruction memory at the address provided by the program counter. The instruction is sent to the IF/ID pipeline register.**

**Instruction Decode (ID) : Drives the fetched instruction to control logic and the register file. The control logic, register file outputs, and various control signals from the instruction (immediate, ALU control, and the write register used in the writeback stage) are sent to the ID/EX pipeline register.**

**Execute (EX) : Performs any ALU operations such as calculating a result from an arithmetic or logic operation, an address in the data memory for a load/store operation. Additionally, for use with branch/jump instructions, the PC’ is calculated. The result is sent to the EX/MEM pipeline register.**

**Memory Access (MEM) : If a memory access is needed (i.e. load or store operation) it is performed in this stage. The address to be accessed (generated by the ALU) is provided from the EX/MEM register and driven to the Data Memory along with the write signal to be activated if a store instruction is at this stage of the pipeline.**

**Write Back (WB) : A result may need to be sent back to the register file. This result can come from memory or the result of an ALU operation. The result will be passed the to the register file and trigger a write to occur to the register file on the next clock cycle.**

1. **(6 pts)** Show the stages of each instruction in the following code assuming a 5-stage pipeline with forwarding and stalling capabilities. If necessary, add new rows to accommodate nops. If more time is needed add more columns.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Time (ps)** | **CC1** | **CC2** | **CC3** | **CC4** | **CC5** | **CC6** | **CC7** | **CC8** |
| sub x4, x5, x3 | F | D | EX | MEM | WB |  |  |  |
| lw x8, 12(x4) |  | F | D | EX | MEM | WB |  |  |
| sub x7, x5, x4 |  |  | F | D | EX | MEM | WB |  |
| beq x4, x8, L0 |  |  |  | F | D | EX | MEM | WB |

**Note: This example shows the power of forwarding, there were no additional nop instructions needed. While the CPI for this series of instructions was 2 (due to the delay getting the first instruction to the WB stage), if we assume the previous 4 instructions were not stalled, then the CPI improves to 1 (best case CPI for single-issue). Without forwarding there would have been 3 nops needed changing this potentially best-case CPI to 1.375!**

* 1. Describe what types of hazards you identified above.

**All 4 hazards are data dependencies.**

**Note: Forwarding is possible for all four hazards (see table above). The first hazard can be mitigated by forwarding the ALU result of the first sub instruction to the ALU input in the EX stage of the lw instruction. Two other hazards can be mitigated by forwarding the Rd data from the WB stage of the first sub and lw instructions to the ALU input in the EX stage of the sub and beq instructions, respectively. Finally, the register file can forward x4 from the WB stage of the first sub instruction to the decode stage of the beq instruction. See section 4.7 for more details.**

1. **(10 pts)** Dynamic branch prediction is a powerful means to, on average, reduce the overhead associated with mis-predicting branches. Assume a processor implements the 2-bit dynamic branch predictor described below.

|  |  |  |
| --- | --- | --- |
| **Prediction State** | **Branch Taken or Not?** | **Next State** |
| Strong Predict Taken | Taken | Strong Predict Taken |
| Strong Predict Taken | Not | Weak Predict Taken |
| Weak Predict Taken | Taken | Strong Predict Taken |
| Weak Predict Taken | Not | Weak Predict Not Taken |
| Weak Predict Not Taken | Taken | Weak Predict Taken |
| Weak Predict Not Taken | Not | Strong Predict Not Taken |
| Strong Predict Not Taken | Taken | Weak Predict Not Taken |
| Strong Predict Not Taken | Not | Strong Predict Not Taken |

* 1. What will the prediction accuracy for the code segment shown below be? Assume all branch instructions have unique prediction buffers and start out in the strong predict taken state. Show your work.



**The first branch instruction is evaluated on whenever a6 is an even number. It will only be taken once. If this loop were longer, the prediction accuracy would improve, but it is best suited for a predict not-taken policy. The 2-bit predictor performs the following branch prediction:**

* **Predicted : Actual 🡪 Prediction Accuracy**
* **SPT : NT 🡪 0/1 , a6 = 0**
* **WPT : NT 🡪 0/2, a6 = 2**
* **WPNT : NT 🡪 1/3, a6 = 4**
* **SPNT : T 🡪 1/4, a6 = 6**

**Branch Instruction 2 is evaluated every iteration and taken when the result of the evenness check is zero. If a predict not taken policy were used the same accuracy would occur. The 2-bit predictor performs the following branch prediction:**

* **Predicted : Actual 🡪 Prediction Accuracy**
* **SPT : T 🡪 1/1 , a28 = 0**
* **SPT : NT 🡪 1/2, a28 = 1**
* **WPT : T 🡪 2/3, a28 = 0\**
* **SPT : NT 🡪 2/4, a28 = 1**
* **WPT : T 🡪 3/5, a28 = 0**
* **SPT : NT 🡪 3/6, a28 = 1**

**2-bit predictor accuracy: 4/10 = 40%**

* 1. What is the prediction accuracy if the predictor instead assumes the branch is always not taken? In this case did the dynamic branch predictor help or hurt the prediction accuracy?

**If the predict not taken policy is used instead, the first branch instruction is correct for 3/4 decisions and the second branch instruction is correct for 3/6. Therefore, the prediction accuracy is 6/10 = 60%. In this case an always not taken policy is better and the 2-bit predictor wound up hurting performance.**

**Note: In a large program, there is very likely a mix of branches that tend toward both always taken and always not taken and the 2-bit predictor would be preferable.**