Homework 2

CDA 4102/ CDA 5155: Fall 2024

**Due:** October 16, 2024 at **11:30 pm Total Points:** 20 points

You are not allowed to take or give help in completing this assignment. Submit the PDF version of the submission in e-learning before the deadline. Late submission (by email attachment to prabhat@ufl.edu) is allowed (up to 24 hours) with a 20% penalty (irrespective of whether it is late for 10 minutes or 10 hours). No grades for late submissions after 24 hours from the deadline.Handwritten (scanned PDF) submissions will NOT be accepted. For example, write it in Microsoft Word (or LaTeX) and convert it to PDF.*Please do not include the questions in your solution (PDF) since it affects the plagiarism checker.*Please include the following sentence on top of your submission (PDF):

**I have neither given nor received any unauthorized aid on this assignment.**

1. Consider the following RISC-V instructions. This code sequence does not have any delay slot instruction.

Loop: fld f0, 32(x1)

fld f2, 64(x1)

fmul.d f4, f0, f2

fadd.d f4, f2, f0

fsd f4, 64(x1)

addi x1, x1, #-8

bne x1, x0, Loop

The third column in the following table shows the number of cycles of latency between a source instruction (first column) and any subsequent instruction (of any type) consuming the result of the source instructions. The fourth column indicates the number of functional units available for executing the respective type of source instruction.

|  |  |  |  |
| --- | --- | --- | --- |
| **Function Unit** | **Related Instruction** | **Latency Cycles** | **Number of Units** |
| ALU1 | addi, bne | 1 | 1 |
| ALU2 | fld, fsd (Address Calculation) | 2 | 1 |
| Memory Unit | fld, fsd | 3 | 1 |
| FP Adder | fadd.d | 4 | 1 |
| FP Multiplier | fmul.d | 5 | 1 |

Assume that the reservation station and the reorder buffer both have infinite size. ALU1 is used for execution of *addi* and *bne* instructions, and ALU2 is used for effective address calculation for load/store instructions. Assume that you can make at most two writes to CDB in one clock cycle. Create two tables showing when each instruction issues, begins execution, accesses memory and writes its result to the CDB for the first two iterations for the following two scenarios using **Tomasulo’s** **algorithm.** Assume that WAW and WAR dependencies are eliminated due to implicit register renaming. Indicate other dependencies (*Control*, *RAW*, *Structural*, and *in-order*) in the entry (if applicable). For *fld* and *fsd* operations, the “Execute” stage is used by ALU2 (address calculation). “Memory” column implies memory access and therefore it will be blank for all instructions except for *fld* and *fsd*. Similarly, “Write-CDB” would be empty for *fsd*. Also, *fsd* needs the value to be stored only in the “Memory” stage.

**a) [4]** Use a RISC-V pipeline with **two-issue** and **without speculation**. Assume branch prediction is perfect. Some of the entries are filled already to show how to indicate the dependencies/hazards. You need to fill the remaining entries. [*Slide* ***90*** *of paralleism.ppt solves a similar problem with different set of assumptions.*]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Iter.** | **Instruction** | | **Issue** | **Execute** | **Memory** | **Write-CDB** |
| 1 | fld | f0, 32(x1) | 1 | 2-3 | 4-6 | 7 |
| 1 | fld | f2, 64(x1) | 1 | 4-5  (Structural) | 7-9  (Structural) | 10 |
| 1 | fmul.d | f4, f0, f2 | 2 | 11-15  (RAW) |  | 16 |
| 1 | fadd.d | f4, f2, f0 |  |  |  |  |
| 1 | fsd | f4, 64(x1) |  |  |  |  |
| 1 | addi | x1, x1, #-8 |  |  |  |  |
| 1 | bne | x1, x0, Loop |  |  |  |  |
| 2 | fld | f0, 32(x1) |  |  |  |  |
| 2 | fld | f2, 64(x1) |  |  |  |  |
| 2 | fmul.d | f4, f0, f2 |  |  |  |  |
| 2 | fadd.d | f4, f2, f0 |  |  |  |  |
| 2 | fsd | f4, 64(x1) |  |  |  |  |
| 2 | addi | x1, x1, #-8 |  |  |  |  |
| 2 | bne | x1, x0, Loop |  |  |  |  |

**b**) **[4]** Use a RISC-V pipeline with **two-issue** and **with speculation**. In this case, you also need to specify when each instruction commits. Please follow ***in-order commit*** for instructions. Assume that up to two instructions of any type can commit per cycle. Branch prediction is perfect. Note that *fsd* will spend 3 cycles in the commit stage, because its memory access occurs during commit. However, *fld* will spend 3 cycles in “Memory Read” and one cycle in commit. Some of the entries are filled already to show how to indicate the dependencies/hazards. You need to fill the remaining entries. [*Slide* ***91*** *of parallelsim.ppt solves a similar problem with different set of assumptions.*]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Iter** | **Instruction** | | **Issue** | **Execute** | **Memory Read** | **Write-CDB** | **Commit** |
| 1 | fld | f0, 32(x1) | 1 | 2-3 | 4-6 | 7 | 8 |
| 1 | fld | f2, 64(x1) | 1 | 4-5  (Structural) | 7-9  (Structural) | 10 | 11 |
| 1 | fmul.d | f4, f0, f2 | 2 | 11-15  (RAW) |  | 16 | 17 |
| 1 | fadd.d | f4, f2, f0 |  |  |  |  |  |
| 1 | fsd | f4, 64(x1) |  |  |  |  |  |
| 1 | addi | x1, x1, #-8 |  |  |  |  |  |
| 1 | bne | x1, x0, Loop |  |  |  |  |  |
| 2 | fld | f0, 32(x1) |  |  |  |  |  |
| 2 | fld | f2, 64(x1) |  |  |  |  |  |
| 2 | fmul.d | f4, f0, f2 |  |  |  |  |  |
| 2 | fadd.d | f4, f2, f0 |  |  |  |  |  |
| 2 | fsd | f4, 64(x1) |  |  |  |  |  |
| 2 | addi | x1, x1, #-8 |  |  |  |  |  |
| 2 | bne | x1, x0, Loop |  |  |  |  |  |

1. This question has three parts. Consider two interleaved branch instructions b1 and b2 with the sequence of outcomes shown in the table (where **T**: Taken, **NT**: Not-taken). **Fill** in the following tables based on the respective instructions, and **compute** the prediction accuracy, i.e., percentage of correct predictions. Please show major steps in your computation. Use the convention (*what to predict if the last branch was not taken*)/(*what to predict if the last branch was taken*) for the correlated predictor. [*Slide* ***40*** *of parallelism.ppt solves a similar problem with different assumptions.*]
2. **[2]** Local + Local Predictors:Use a 1-bit (local) predictor for b1 and a 1-bit local predictor for b2. In this case, they are independent branches.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| b1 prediction | b1 action | New b1 prediction | b2 prediction | b2 action | New b2 prediction |
| T | T | T | T | T | T |
|  | NT |  |  | NT |  |
|  | T |  |  | T |  |
|  | NT |  |  | NT |  |

**Prediction Accuracy:**

1. **[2]** Local + Correlated Predictors:Use a 1-bit (local) predictor for b1 and a (1, 1) correlated predictor for b2. In this case, b2 depends (correlated) on b1 but b1 does not depend on b2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| b1 prediction | b1 action | New b1 prediction | b2 prediction | b2 action | New b2 prediction |
| T | T | T | T/T | T | T/T |
|  | NT |  |  | NT |  |
|  | T |  |  | T |  |
|  | NT |  |  | NT |  |

**Prediction Accuracy:**

1. **[2]** Both Correlated Predictors: Assume that both branches are correlated to each other.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| b1 prediction | b1 action | New b1 prediction | b2 prediction | b2 action | New b2 prediction |
| T/T | T | T/T | T/T | T | T/T |
|  | NT |  |  | NT |  |
|  | T |  |  | T |  |
|  | NT |  |  | NT |  |

**Prediction Accuracy:**

1. **[1]** Does the prediction accuracy match your expectations? For example, if (a) provides better results than (b) and/or (c), argue that both branches are independent based on their actions (or vice versa).
2. **[3 pts]** Write a sequence of RISC-V assembly instructions to count the number of ‘1’s in the value stored in x1. Store the result in x2. For example, if we provide x1 as 0x1234ABCD, then x2 must be 0xF after executing your code since the value in x1 has 15 ‘1’s. Please use only the opcodes from Project 1. Please use only four registers (x0, x1, x2, and x3). Assume that x0 is always 0. Please do not initialize x1, assume that some positive value is already there (leftmost bit is 0).
3. **[2]** Consider a simple 5-stage pipeline, where each stage takes different times to complete: IF 3.2 ns, ID 2.3 ns, EX 4.5 ns, MEM 3 ns, and WB 1 ns. There is 0.5 ns delay associated with other design constraints (latches and clock skew) due to pipelining. Assume that we would like to execute **only 100 instructions**. Compute the ***clock period***, ***speedup***, ***efficiency***, and ***throughput*** for this simple pipelined processor. Please show major steps. [*Slide* ***10*** *of pipelining.ppt may be helpful for this problem.*]