James Bickerstaff & Jefferson Boothe

**ECE 2162 P1 Report**

**Section 1**

* The code is written in python, and must be executed on a system in which it is installed (Windows OS, Linux OS, WSL, etc.)
* The name of the main file that will be run is “tomasulo.py”, and all supporting code is located within the same repository
* To execute the code, run the following command (replacing the paths as needed):

*python3* ***tomasulo****.py* *path/to/****config\_file****.txt path/to/****instruction\_file****.txt*

* The output of the program will be sent to the terminal window, including the ARF values, memory values, and timing information, as well as per-cycle information for debugging or further investigation

**Section 2**

**Initializing FUs to the following spec:**

Using Integer Adder with 2 reservation stations, 1 cycle in EXE, and 1 FU

Using Floating Point Adder with 3 reservation stations, 3 cycles in EXE, and 1 FU

Using Floating Point Multiplier with 2 reservation stations, 20 cycles in EXE, and 1 FU

Using Load/Store Unit with 250 (infinite) reservation stations, 1 cycle in EXE, 4 cycles in MEM, and 1 FU

Using an instruction buffer of size 12

Using a ROB of size 128

Using 1 CDB with buffers of size 1 per FU in the event of multiple instructions trying to WB

**Initial Values:**

For conciseness of the report, all screenshots/test cases shown here have the same initial values loaded from the config file:

R1=10, R2=20, R3=16, R5=12, R6=32, R7=0, R8=2

F1=123.123, F2=30.1, F3=6.9, F5=1

MEM[4]=5, MEM[8]=2, MEM[12]=3.4, MEM[44]=2048

**Tier 1: Straight-line cases where no dependencies exist among instructions**

*Case1\_noDeps.txt*

|  |  |
| --- | --- |
| Instructions: | Output: |
| LD F1,4(R0)  SUB.D F2,F3,F4  ADD R1,R2,R3  MULT.D F5,F6,F7  SD F8,8(R0)  ADDI R4,R4,5 |  |

**Tier 2: Straight-line code where there are dependencies (false and true):**

*Case2\_deps.txt*

|  |  |
| --- | --- |
| Instructions: | Output: |
| SUB.D F1,F2,F3  ADD.D F1,F2,F3  MULT.D F4,F1,F5  LD F6,4(R0)  ADD.D F3,F6,F6  ADD.D F6,F4,F5  SD F6,8(R0) |  |

This test consists of both false and true dependencies among Floating Point registers.

**Tier 3: Forwarding among load/store instructions:**

*Case3\_forward.txt*

|  |  |
| --- | --- |
| Instructions: | Output: |
| ADDI R9,R0,4  LD F8,0(R9)  MULT.D F1,F3,F2  SD F1,8(R9)  LD F7,0(R5)  ADD.D F5,F7,F2  ADDI R7,R5,4  SD F5,4(R5) |  |

This test contains a forward from store, performed in cycle 25, the LD finishes the MEM stage in only 2 cycles (which would normally take 4), before writing back on 26 and the subsequent ADD.D begins executing on 27 with the loaded value.

**Tier 4: Structural Hazards in Reservation Stations and Functional Units**

*Case4\_hazards.txt*

|  |  |
| --- | --- |
| Instructions: | Output: |
| ADD.D F0,F1,F2  ADDI R1,R0,4  ADDI R2,R0,8  ADD R3,R1,R2  LD F8,0(R3)  MULT.D F5,F8,F8  ADD.D F4,F5,F8  ADD.D F6,F5,F5  SD F4,4(R3)  SD F6,8(R3) |  |

This test contains many hazards: the 1st and 3rd instruction finish executing on cycle 3 and contest for CDB access to write back. ADD R3,R1,R2 cannot issue right away due to the reservation stations being filled. Next, the two ADD.D at the end both resolve dependencies on the same cycle and contest for the privilege to begin executing on cycle 35.

**Tier 5: Simple loop, inclusive of other tiers complexity**

*Case5\_loop.txt*

|  |  |
| --- | --- |
| Instructions: | Output: |
| ADD R1,R0,R0  ADDI R2,R0,16  ADDI R3,R0,256  ADD.D F2,F5,F8  LD F1,8(R0)  # START OF LOOP  ADDI R1,R1,4  MULT.D F2,F2,F1  SD F2,-4(R1)  SUB R3,R3,R2  BNE R1,R2,-5  ADD R5,R1,R2 |  |

This test loops with R1=0,4,8,12 and calculates powers of 2 before storing them in MEM[0]-MEM[12]. There is dependency between each loop iteration requiring the previous loops F2 multiplication to be completed, creating increased complexity, and slowdown.

**Tier 6: Demonstration of Branch Prediction**

*Case6\_notaketake.txt*

|  |  |
| --- | --- |
| Instructions: | Output: |
| ADDI R1,R0,88  ADDI R2,R0,77  # Dont take 1st time  BEQ R1,R2,8  ADD R5,R6,R2  LD F6,4(R0)  LD F7,8(R0)  SUB.D F5,F7,F6  MULT.D F3,F6,F5  ADD R1,R2,R0  # Take  BNE R2,R0,-8  # This instruction should not commit  ADD R9,R2,R3  SD F3,12(R0) |  |

A picture containing graphical user interface

Description automatically generatedHere, note ADD R5,R6,R2 issues on cycle 4 (directly after the branch), showing a correct prediction of not taken. The 2nd branch also predicts not taken on cycle 10, but is found to be incorrect on cycle 11, causing the next instruction to not be issued until cycle 13 which is another branch. This one also predicts not taken (since it was correct last time), but it proves incorrect this time, causing SD F3,12(R0) to not issue until 16. Snippet of execution output:

Points of interest: you can see ADD R5,R6,R2 speculatively issued on cycle 14, but promptly cleared as the branch resolves and indicates misprediction.

*Case6\_take3not1.txt*

|  |  |
| --- | --- |
| Instructions: | Output: |
| ADD R1,R0,R0  ADDI R2,R0,4  SUB.D F2,F1,F5  ADDI R1,R1,1  BNE R1,R2,-3  SD F5,0(R1) |  |

Here, is a for(R1=0, R1<4, R1++) loop to demonstrate the branch predictor adjusting its prediction. At first, it defaults to not taken, which is incorrect, shown by the slowdown in SUB.D not issuing until cycle 8. However, the predictor now predicts taken, and SUB.D issues immediately on cycles 11 and 15. The last iteration also predicts taken, but is now incorrect, and the store instruction does not issue until 21, since the branch resolved on cycle 19. Snippet of execution output:

A screenshot of a computer

Description automatically generated with medium confidenceHere, we can see the branch resolve on cycle 12, confirming it was correctly predicted and no further action must be taken.

Text

Description automatically generated

Here, we see the misprediction resolve on the final iteration. The recovery includes clearing speculative instructions from the execution pipeline as well as reservation stations of each unit.

**Section 3**

James Bickerstaff

Major portions of code written: Branch Predictor, Branch portion of Int Adder Load Store Unit, int/fp ARF, RAT, Memory

Test benchmarks written: case1\_noDeps, case2\_deps, LDchain, LDSD chain, SD chain, 4\_loop\_iterations, random\_instrs

Debugging Effort: worked to debug the components written and related integration issues

Participation Effort: contributed significantly and completed/tested components written

Jefferson Boothe

Major portions of code written: Reservation Station, ROB, Int Adder, FP Adder, FP multiplier, CDB, instruction buffer

Test benchmarks written: case3\_forward, case4\_hazards, case5\_loop, case6\_notaketake, case6\_take3not1, FloatOps, IntOps

Debugging Effort: worked to debug the components written and related integration issues

Participation Effort: contributed significantly and completed/tested components written