Computer Architecture Project 4 Report

2021-14284 Taehyun Yang

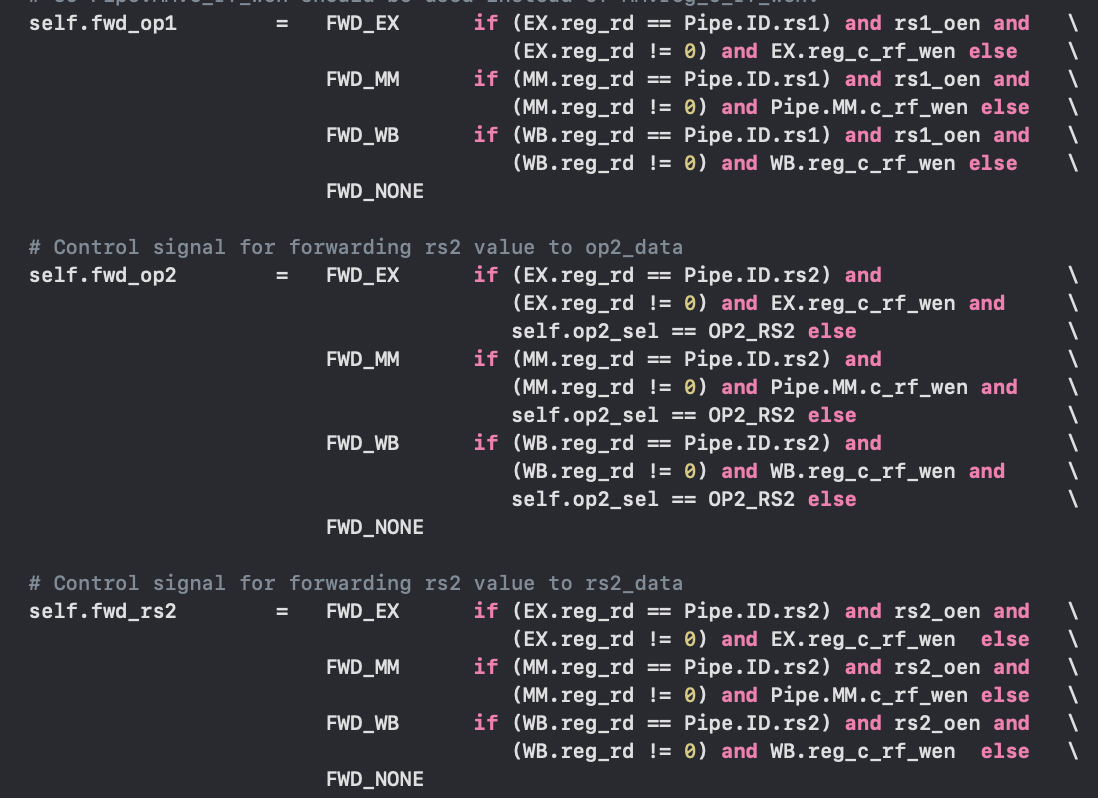
**Part 1 Pop and Push implementation**

Using the basic schematics, I have implemented Pop and Push like so:

PUSH: RS1= 4 RS2= selected register RD= SP

POP: RS1= 4 RS2= SP RD= Selected register

This choice was used because by moving the register and sp to intertwined locations, I could utilize the data hazard forwarding method given to the maximum

Data hazard forwarding method given:

Now here is the possible edge cases of the pop and push implementation and the hazards that follow these scenarios:

When ID= a nonpop or nonpush function

|  |  |  |  |
| --- | --- | --- | --- |
| ID.RS1/2= SP | A normal function(if ID.RS1/RS2 == Stage.RD) | Push (if ID.RS1/RS2 == Stage.RD) | Pop (if ID.RS1/RS2 == Stage.RS2) |
| EX | Taken care of with given forwarding implementation | ID.RS1/RS2==EX.Alu\_out | ID.RS1/RS2==EX.Alu\_out |
| MM | Taken care of with given forwarding implementation | ID.RS1/RS2==MM.wbdata | ID.RS1/RS2==MM.wbdata |
| WD | Taken care of with given forwarding implementation | ID.RS1/RS2==WB.wbdata | ID.RS1/RS2==WB.wbdata |
| ID.RS1/2= any register |  | Push (if ID.RS1/RS2 == Stage.RS2) | POP (if ID.RS1/RS2 == Stage.RD) |
| EX | Taken care of with given forwarding implementation | Does not matter as push t0 does not interfere with RS1 or RS2 | Load Hazard, Stall once |
| MM | Taken care of with given forwarding implementation |  | ID.RS1/RS2==MM.rddata |
| WD | Taken care of with given forwarding implementation |  | ID.RS1/RS2==WB.rddata |

PUSH Function

|  |  |  |  |
| --- | --- | --- | --- |
| ID.RS2==any register | A normal function(if ID.RS2 == Stage.RD) | Push (if ID.RS2 == Stage.RS2) | Pop (if ID.RS2 == Stage.RD) |
| EX | Taken care of with given forwarding implementation | Does not matter as push t0 does not interfere with PUSH. Rs2 | Load hazard, stall once |
| MM | Taken care of with given forwarding implementation | Does not matter as push t0 does not interfere with PUSH. Rs2 | ID.RS2==MM.rddata |
| WD | Taken care of with given forwarding implementation | Does not matter as push t0 does not interfere with PUSH. Rs2 | ID.RS2==WB.rddata |
| ID.RD= sp |  | Push (if ID.RS2 == Stage.RD) | POP (if ID.RS2 == Stage.RS2) |
| EX | Taken care of with given forwarding implementation | ID.RS2==EX.Alu\_out | ID.RS2==EX.Alu\_out |
| MM | Taken care of with given forwarding implementation | ID.RS2==MM.wbdata | ID.RS2==MM.wbdata |
| WD | Taken care of with given forwarding implementation | ID.RS2==WB.wbdata | ID.RS2==WB.wbdata |

POP function:

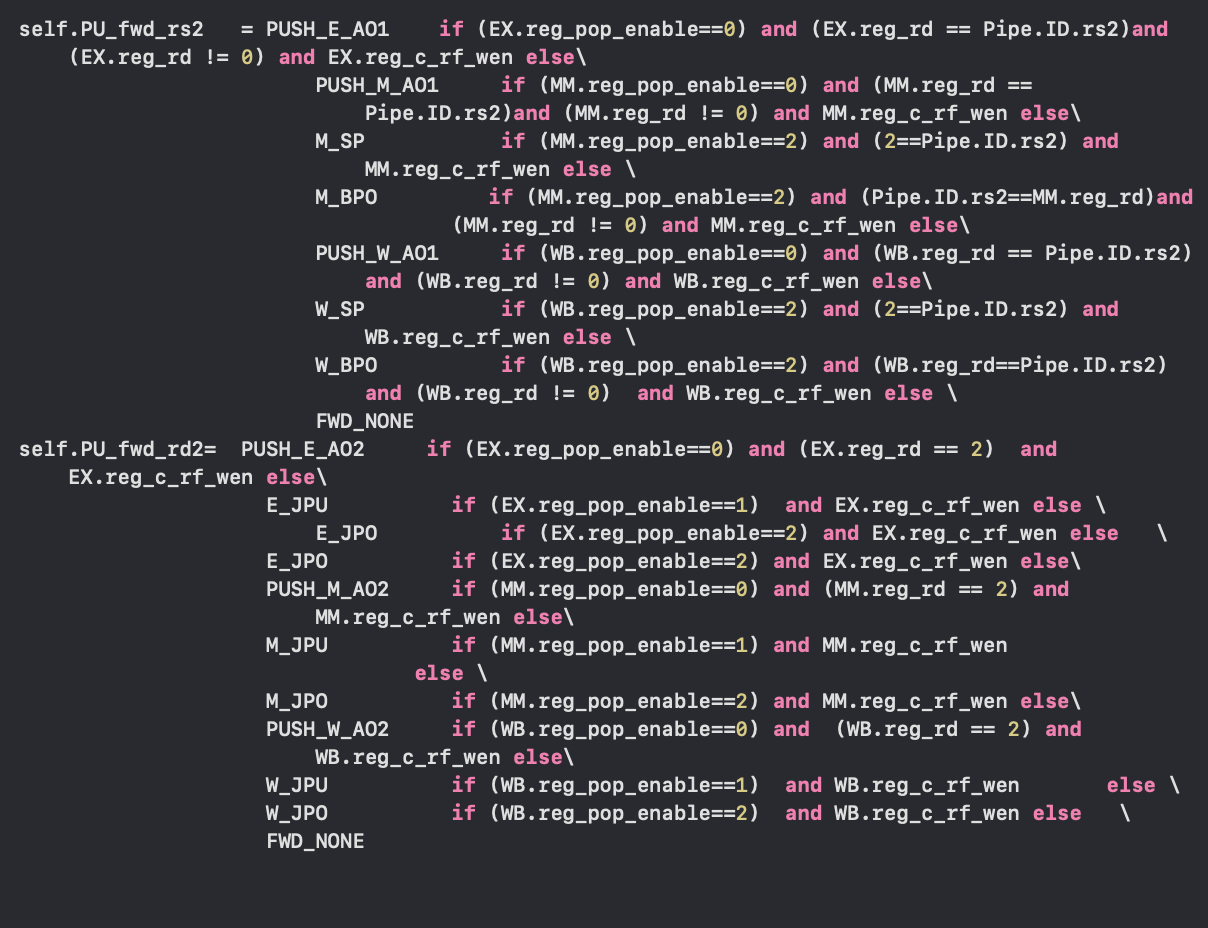
|  |  |  |  |
| --- | --- | --- | --- |
| ID.RS2==SP | A normal function(if ID.RS2 == Stage.RD) | Push (if ID.RS2 == Stage.RS2) | Pop (if ID.RS2 == Stage.RD) |
| EX | Taken care of with given forwarding implementation | ID.RS2==EX.Alu\_out | Load hazard, stall once |
| MM | Taken care of with given forwarding implementation | ID.RS2==MM.wbdata | ID.RS2==MM.rddata |
| WD | Taken care of with given forwarding implementation | ID.RS2==WB.wbdata | ID.RS2==WB.rddata |
| ID.RD= any register |  |  | POP (if ID.RS2 == Stage.RS2) |
| EX | Unneeded as data wont collide | Does not matter as pop t0 does not interfere with PUSH. Rs2 | POP followed by a pop wont collide |
| MM | Unneeded as it wont collide | Does not matter as pop t0 does not interfere with PUSH. Rs2 | POP followed by a pop wont collide |
| WD | Unneeded as it wont collide | Does not matter as pop t0 does not interfere with PUSH. Rs2 | POP followed by a pop wont collide |

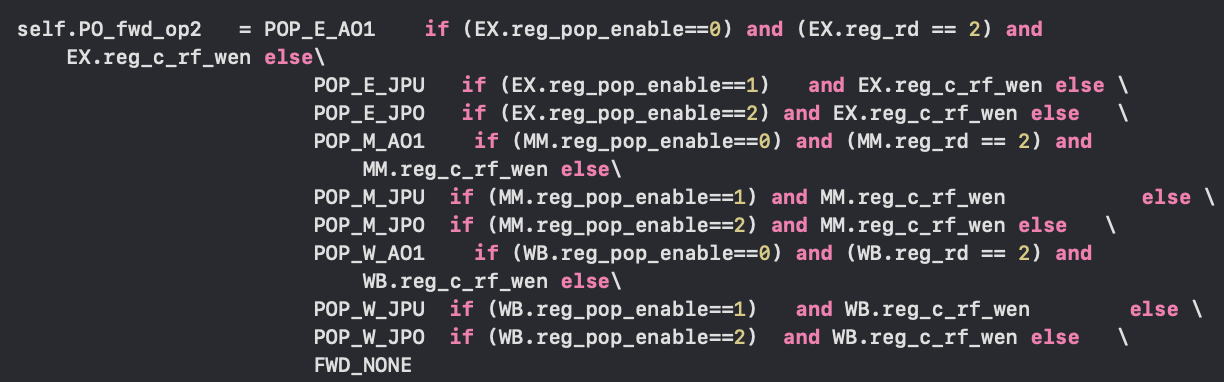
The only exception from this table is when PUSH SP is called. That is simply placed as an exception so that SP is stored before the -4 decrement.

Text, application

Description automatically generatedHere are my specific implementations for all of those cases taking account of the textbook.

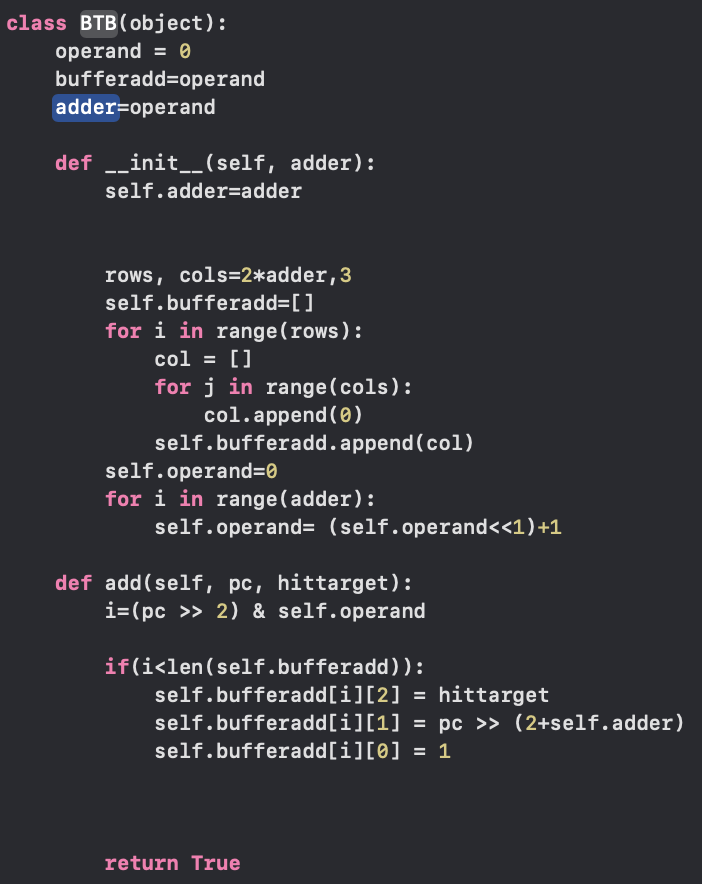
This is for the normal functions. These are all repeated based on op1, op2 and rs2 and changed accordingly. How the data is forwarded specifically is written on the table above.



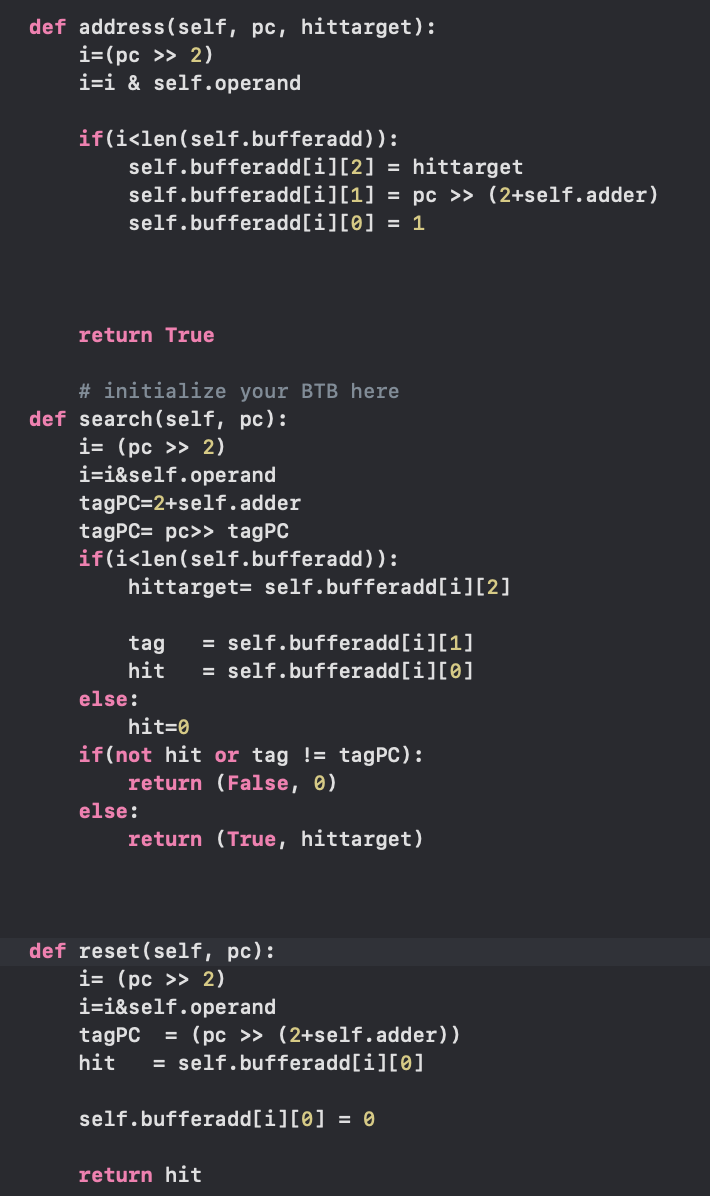
Function for Push. As push function interacts and has to constantly review two registers, RS2, and RD, each registers are always compared with RD when it is a normal function, and RS2 and RDs for other pop/push functions.

This is the pop function. Notice how comparison for RD is not necessary as they either don’t collide or are already dealt with in the given function. This function also compares the RS2s to other possible functions.

Part 2 BTB implementation

For our BTB implementation, we initialize it in our BTBclass, as so.

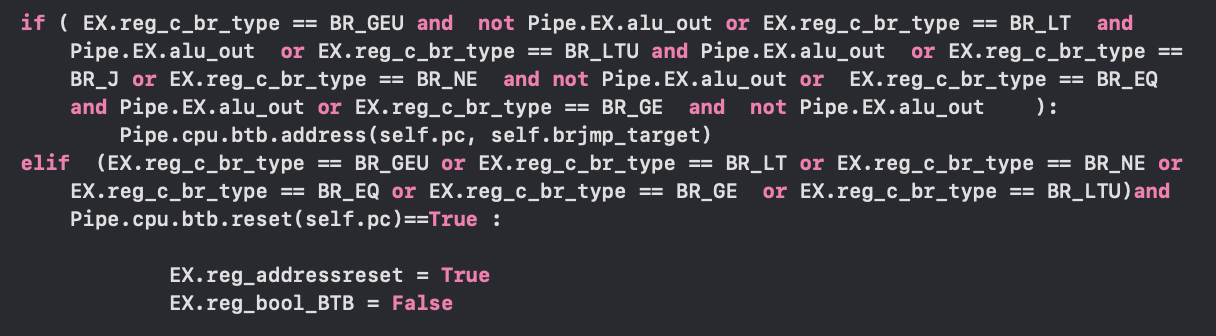
Now we create a field for the operand, the buffer, and the adder while creating a 3 by 2^adder matrix for our branch target buffer.

Then we create an address function, which depending on the pc and whether the address exists, it alters our buffer matrix to the corresponding results.

Search function helps us whether the address exists in our buffer while reset helps us sift through the buffer to see whether the target has been a hit or a miss. When there is a miss, we are

Able to reset the buffer so that the new branch

Target can be added.



Here is the case when the BTB is fully

Utilized. The functions we have implemented are all realized. However, I was not

Able to fully implement the BTB as an

Error occurs when the buffer goes

Beyond the capcity. I hope to fix this later.