Computer Architecture Project 4 Report

2021-14284 Taehyun Yang

Part 1 Pop and Push implementation

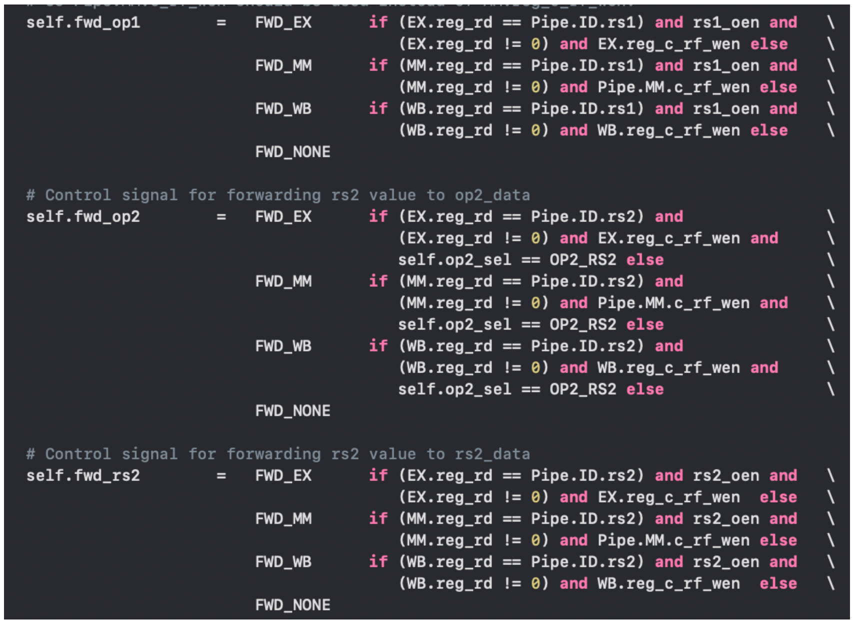
Using the basic schematics, I have implemented Pop and Push like so:

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:



```
self.fwd_op1        =   FWD_EX      if (EX.reg_rd == Pipe.ID.rs1) and rs1_oen and   \
                                       (EX.reg_rd != 0) and EX.reg_c_rf_wen else     \
                        FWD_MM      if (MM.reg_rd == Pipe.ID.rs1) and rs1_oen and   \
                                       (MM.reg_rd != 0) and Pipe.MM.c_rf_wen else    \
                        FWD_WB      if (WB.reg_rd == Pipe.ID.rs1) and rs1_oen and   \
                                       (WB.reg_rd != 0) and WB.reg_c_rf_wen else     \
                        FWD_NONE

# Control signal for forwarding rs2 value to op2_data
self.fwd_op2        =   FWD_EX      if (EX.reg_rd == Pipe.ID.rs2) and                \
                                       (EX.reg_rd != 0) and EX.reg_c_rf_wen and      \
                                       self.op2_sel == OP2_RS2 else                  \
                        FWD_MM      if (MM.reg_rd == Pipe.ID.rs2) and                \
                                       (MM.reg_rd != 0) and Pipe.MM.c_rf_wen and     \
                                       self.op2_sel == OP2_RS2 else                  \
                        FWD_WB      if (WB.reg_rd == Pipe.ID.rs2) and                \
                                       (WB.reg_rd != 0) and WB.reg_c_rf_wen and      \
                                       self.op2_sel == OP2_RS2 else                  \
                        FWD_NONE

# Control signal for forwarding rs2 value to rs2_data
self.fwd_rs2        =   FWD_EX      if (EX.reg_rd == Pipe.ID.rs2) and rs2_oen and   \
                                       (EX.reg_rd != 0) and EX.reg_c_rf_wen   else   \
                        FWD_MM      if (MM.reg_rd == Pipe.ID.rs2) and rs2_oen and   \
                                       (MM.reg_rd != 0) and Pipe.MM.c_rf_wen else    \
                        FWD_WB      if (WB.reg_rd == Pipe.ID.rs2) and rs2_oen and   \
                                       (WB.reg_rd != 0) and WB.reg_c_rf_wen   else   \
                        FWD_NONE
```

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

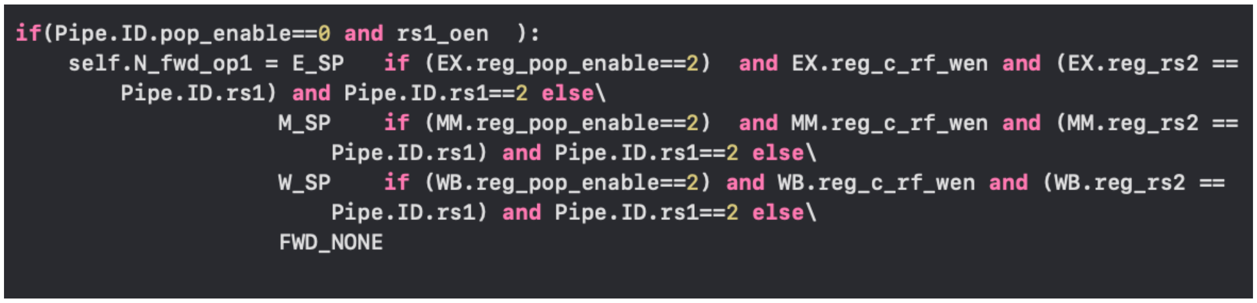
ID.RS2==any register	**A normal function(if ID.RS2 == Stage.RD)**	**Push (if ID.RS2 == Stage.RS2)**	**Pop (if ID.RS2 == Stage.RD)**
EX	Taken care of with given forwarding implementation	Does not matter as push t0 does not interfere with PUSH. Rs2	Load hazard, stall once
MM	Taken care of with given forwarding 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

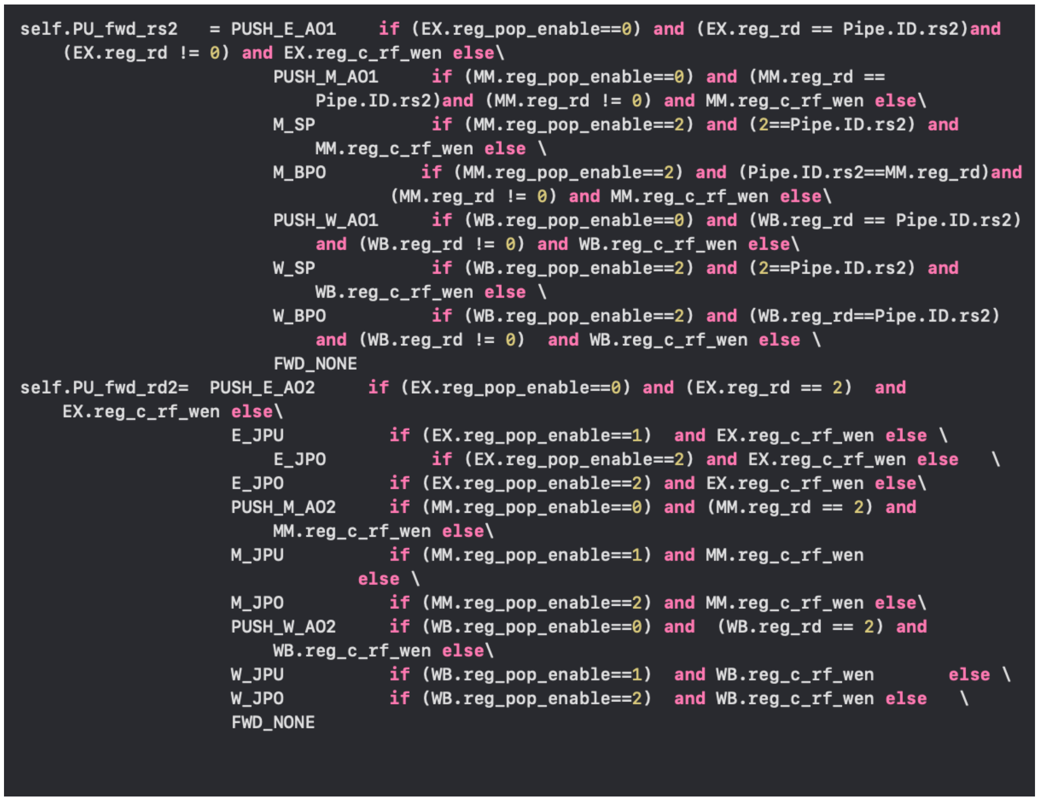
The only exception from this table is when PUSH SP is called. That is simply placed as an exception so that SP is stored before the -4 decrement.

Here are my specific implementations for all of those cases taking account of the textbook.



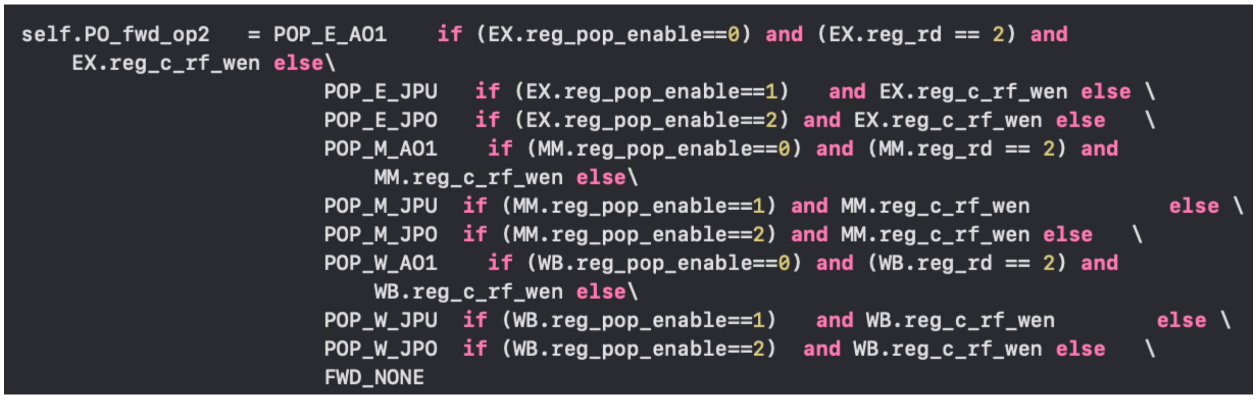
```
if(Pipe.ID.pop_enable==0 and rs1_oen  ):
    self.N_fwd_op1 = E_SP   if (EX.reg_pop_enable==2)  and EX.reg_c_rf_wen and (EX.reg_rs2 ==
        Pipe.ID.rs1) and Pipe.ID.rs1==2 else\
                     M_SP   if (MM.reg_pop_enable==2)  and MM.reg_c_rf_wen and (MM.reg_rs2 ==
                         Pipe.ID.rs1) and Pipe.ID.rs1==2 else\
                     W_SP   if (WB.reg_pop_enable==2) and WB.reg_c_rf_wen and (WB.reg_rs2 ==
                         Pipe.ID.rs1) and Pipe.ID.rs1==2 else\
                     FWD_NONE
```

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.



```
self.PU_fwd_rs2   = PUSH_E_AO1    if (EX.reg_pop_enable==0) and (EX.reg_rd == Pipe.ID.rs2)and
    (EX.reg_rd != 0) and EX.reg_c_rf_wen else\
                        PUSH_M_AO1     if (MM.reg_pop_enable==0) and (MM.reg_rd ==
                            Pipe.ID.rs2)and (MM.reg_rd != 0) and MM.reg_c_rf_wen else\
                        M_SP           if (MM.reg_pop_enable==2) and (2==Pipe.ID.rs2) and
                            MM.reg_c_rf_wen else \
                        M_BPO         if (MM.reg_pop_enable==2) and (Pipe.ID.rs2==MM.reg_rd)and
                                   (MM.reg_rd != 0) and MM.reg_c_rf_wen else\
                        PUSH_W_AO1     if (WB.reg_pop_enable==0) and (WB.reg_rd == Pipe.ID.rs2)
                            and (WB.reg_rd != 0) and WB.reg_c_rf_wen else\
                        W_SP           if (WB.reg_pop_enable==2) and (2==Pipe.ID.rs2) and
                            WB.reg_c_rf_wen else \
                        W_BPO          if (WB.reg_pop_enable==2) and (WB.reg_rd==Pipe.ID.rs2)
                            and (WB.reg_rd != 0)  and WB.reg_c_rf_wen else \
                        FWD_NONE
self.PU_fwd_rd2=  PUSH_E_AO2     if (EX.reg_pop_enable==0) and (EX.reg_rd == 2)  and
    EX.reg_c_rf_wen else\
                    E_JPU          if (EX.reg_pop_enable==1)  and EX.reg_c_rf_wen else \
                        E_JPO          if (EX.reg_pop_enable==2) and EX.reg_c_rf_wen else    \
                    E_JPO          if (EX.reg_pop_enable==2) and EX.reg_c_rf_wen else\
                    PUSH_M_AO2     if (MM.reg_pop_enable==0) and (MM.reg_rd == 2) and
                        MM.reg_c_rf_wen else\
                    M_JPU          if (MM.reg_pop_enable==1) and MM.reg_c_rf_wen
                            else \
                    M_JPO          if (MM.reg_pop_enable==2) and MM.reg_c_rf_wen else\
                    PUSH_W_AO2     if (WB.reg_pop_enable==0) and  (WB.reg_rd == 2) and
                        WB.reg_c_rf_wen else\
                    W_JPU          if (WB.reg_pop_enable==1)  and WB.reg_c_rf_wen       else \
                    W_JPO          if (WB.reg_pop_enable==2)  and WB.reg_c_rf_wen else    \
                    FWD_NONE
```

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.

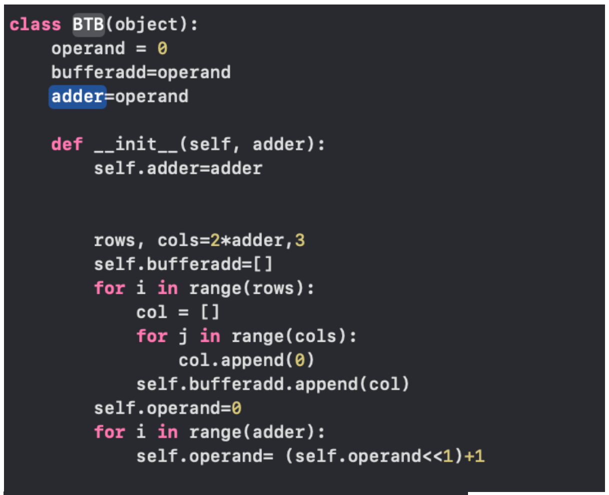


```
self.PO_fwd_op2    = POP_E_AO1    if (EX.reg_pop_enable==0) and (EX.reg_rd == 2) and
    EX.reg_c_rf_wen else\
                        POP_E_JPU   if (EX.reg_pop_enable==1)   and EX.reg_c_rf_wen else \
                        POP_E_JPO   if (EX.reg_pop_enable==2) and EX.reg_c_rf_wen else   \
                        POP_M_AO1    if (MM.reg_pop_enable==0) and (MM.reg_rd == 2) and
                            MM.reg_c_rf_wen else\
                        POP_M_JPU  if (MM.reg_pop_enable==1) and MM.reg_c_rf_wen           else \
                        POP_M_JPO  if (MM.reg_pop_enable==2) and MM.reg_c_rf_wen else   \
                        POP_W_AO1    if (WB.reg_pop_enable==0) and (WB.reg_rd == 2) and
                            WB.reg_c_rf_wen else\
                        POP_W_JPU  if (WB.reg_pop_enable==1)   and WB.reg_c_rf_wen        else \
                        POP_W_JPO  if (WB.reg_pop_enable==2)  and WB.reg_c_rf_wen else   \
                        FWD_NONE
```

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

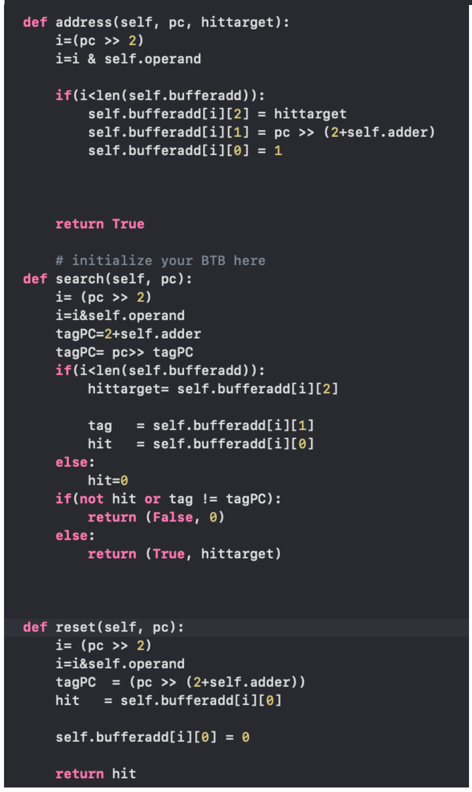
For our BTB implementation, we initialize it in our BTBclass, as so.



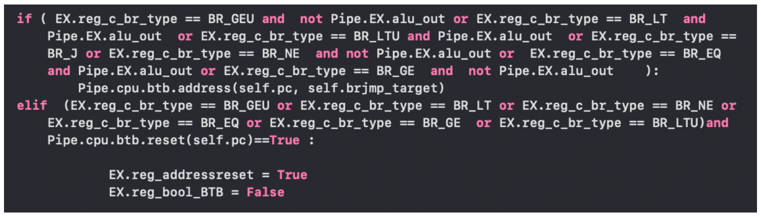
```python
class BTB(object):
    operand = 0
    bufferadd=operand
    adder=operand

    def __init__(self, adder):
        self.adder=adder


        rows, cols=2*adder,3
        self.bufferadd=[]
        for i in range(rows):
            col = []
            for j in range(cols):
                col.append(0)
            self.bufferadd.append(col)
        self.operand=0
        for i in range(adder):
            self.operand= (self.operand<<1)+1
```

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.



```python
def address(self, pc, hittarget):
    i=(pc >> 2)
    i=i & self.operand

    if(i<len(self.bufferadd)):
        self.bufferadd[i][2] = hittarget
        self.bufferadd[i][1] = pc >> (2+self.adder)
        self.bufferadd[i][0] = 1



    return True

    # initialize your BTB here
def search(self, pc):
    i= (pc >> 2)
    i=i&self.operand
    tagPC=2+self.adder
    tagPC= pc>> tagPC
    if(i<len(self.bufferadd)):
        hittarget= self.bufferadd[i][2]

        tag   = self.bufferadd[i][1]
        hit   = self.bufferadd[i][0]
    else:
        hit=0
    if(not hit or tag != tagPC):
        return (False, 0)
    else:
        return (True, hittarget)



def reset(self, pc):
    i= (pc >> 2)
    i=i&self.operand
    tagPC  = (pc >> (2+self.adder))
    hit   = self.bufferadd[i][0]

    self.bufferadd[i][0] = 0

    return hit
```



```python
if ( EX.reg_c_br_type == BR_GEU and  not Pipe.EX.alu_out or EX.reg_c_br_type == BR_LT  and
    Pipe.EX.alu_out  or EX.reg_c_br_type == BR_LTU and Pipe.EX.alu_out  or EX.reg_c_br_type ==
    BR_J or EX.reg_c_br_type == BR_NE  and not Pipe.EX.alu_out or  EX.reg_c_br_type == BR_EQ
    and Pipe.EX.alu_out or EX.reg_c_br_type == BR_GE  and  not Pipe.EX.alu_out    ):
        Pipe.cpu.btb.address(self.pc, self.brjmp_target)
elif  (EX.reg_c_br_type == BR_GEU or EX.reg_c_br_type == BR_LT or EX.reg_c_br_type == BR_NE or
    EX.reg_c_br_type == BR_EQ or EX.reg_c_br_type == BR_GE  or EX.reg_c_br_type == BR_LTU)and
    Pipe.cpu.btb.reset(self.pc)==True :

            EX.reg_addressreset = True
            EX.reg_bool_BTB = False
```

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.