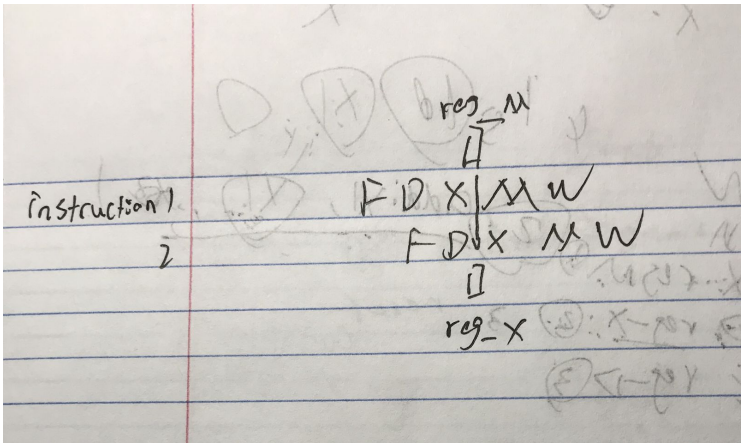
Design Doc: Detailing the structural and logical design of your program

DIAGRAM: interface, flow chart

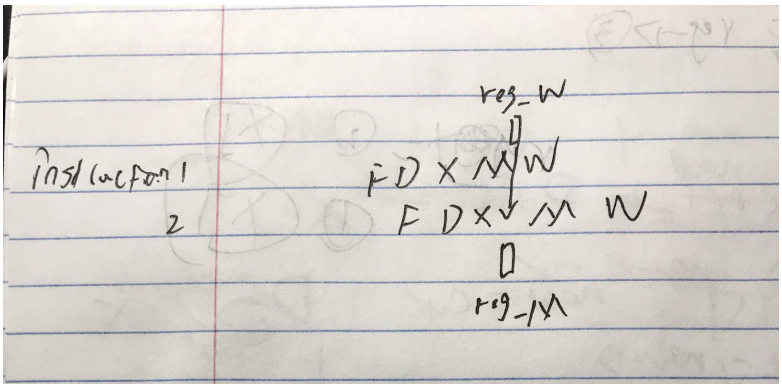
Design: In this programming we need to implement 5 stages the same as PA1. In addition, we have to implement stall, forwarding and branch prediction buffer in order to execute pipeline within multiple cycles.

- **Forwarding:** As the assignment instruction specifies, we need to use struct stage_reg_(d,x,m,w) to communicate between stages. Fowardings are included in those structures. There are 3 forwardings: (MX, WX, WM)

    1. MX forwarding: We need to foward the value of destination register in a instruction to to the source register of its next instruction if these two register numbers are same. We Used a boolean forward tag in the execution stage to justify if the fowarding will happen or not. When instruction. Rd number= nextinstructoin.r1 or r2 number, we forward value from cur_m_reg to the corresponding source register. In order to do that, we implement a fowarding value field in stage_reg_m.



    2. WM forwarding: Usually happens between two memory instrucitons. Similarly, We need to foward the value of destination register in a instruction to to the source register of its next instruction if these two register numbers are same. We still use fowardtag to justify if the fowarding is needed or not. We implement a unsigned_passvalue field in stage_reg_m in order to store the forwarding value of the first instruction's correct destination value.

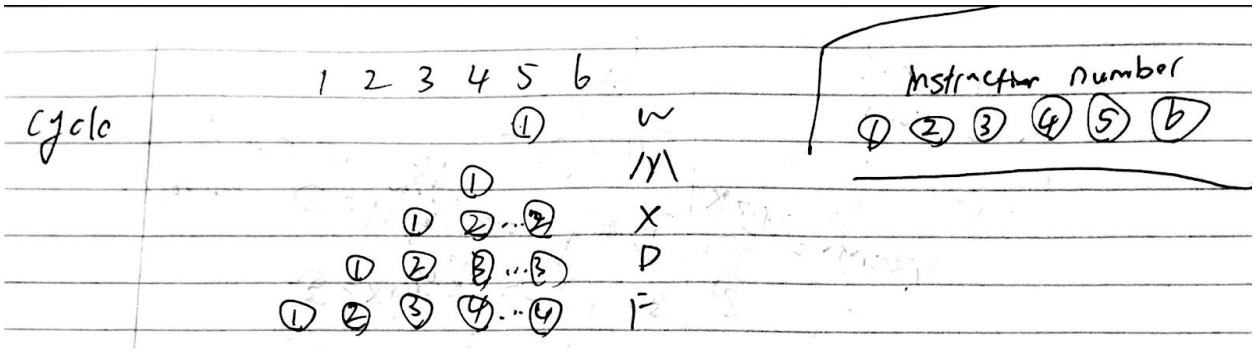


3. WX forwarding: since the program execute the loop from W to F, this forwarding can be done by just calling the available value. No further action needed.

With forwardings, it is still possible for stall to happen when:

- ALU instructions happen right after memory instruction, one stall needed.
- Two consecutive memory instructions when the destination of the second one needs the value of the destination of first value.
- Stall happens when we did branch prediction wrong.

- **Stall implementation:** It is possible for pipeline to stall at fetch, decode, execution and memory stages. Therefore, in each of these functions, we have to include a stall check to check if the stall happens or not(like a flag). If in the current cycle we found a stall in a specific stage, we need to inform the previous stage of next cycle to stall. The case of stall in pipeline looks like:



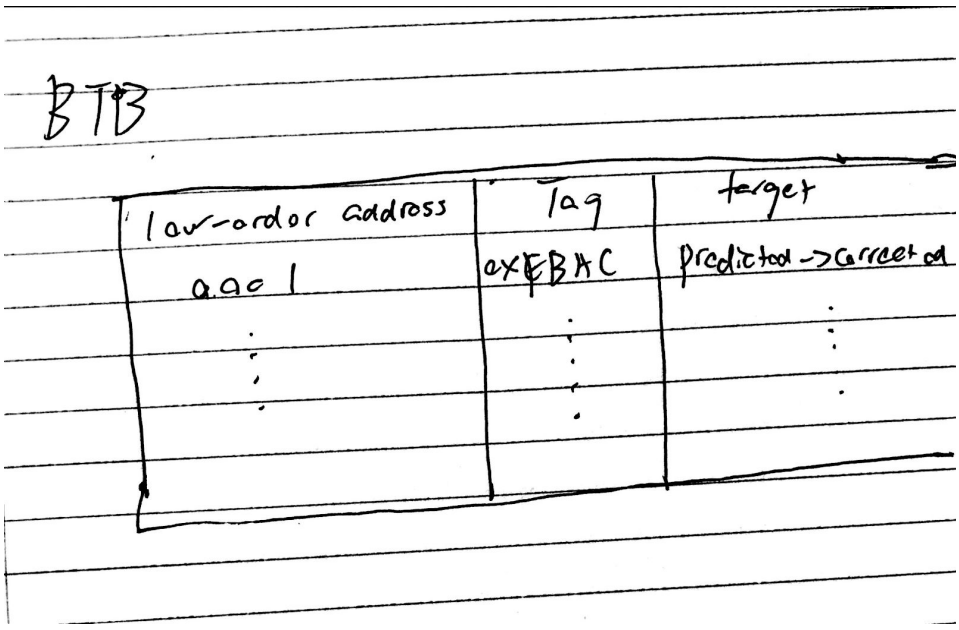
In order to let stages stall,

1. we inform the stage preceeding to reload the value from cur_reg to new_reg by using pointer.
2. We let fetch not to fetch a new instruction.

- **Branch prediction:** We will have a specific function for branch prediction. An array called **branch target buffer(BTB)** should be defined to store the address that the branch will jump to. Forwarding branches are predicted not-taken and backward branches are predicted taken.
    - Correct prediction: There is no need for a stall because the branch goes directly to the next correct instruction.

    - Incorrect prediction: There will be a control hazard stall. For example if there are two instructions after an incorrect branch prediction, the two instructions still go through fetch and decode. These instructions must be flushed, requiring a two cycle stall before the next instruction from branching can be fetched.

The array BTB will store the branch address and predicted target address at first. If the prediction is right, nothing changes. However, if we predicted wrong, the predicted target address will be overwritten by the correct target address.



Task assignment

- Hang Yuan
  - Transfer the design into code.
  - Test&Debug
- Zhewei Wang
  - Design structure of code
  - Test&debug
- Andrew Tsai
  - Design structure of code
  - Test&Debug

o lb 1
o lh 2
o lw 3
o ld 4
o lbu 5
o lhu 6
o lwu 7
(X) fence 8
(X) fence.i 9
o addi 10
o slli 11
o slti 12
o sltiu 13
o xori 14
o srli 15
o srai 16
o ori 17
o andi 18
x auipc 19
addiw 20
slliw 21
srliw 22
sraiw 23
o sb 24
sh 25
sw 26
sd 27
add 28
sub 29
sll 30
slt 31

sltu 32
xor 33
srl 34
sra 35
or 36
o and 37
o lui 38
o addw 39
o subw 40
sllw 41
srlw 42
sraw 43
beq 44
bne 45
blt 46
bge 47
bltu 48
bgeu 49
jalr 50
jal 51
(X) ecall 52
(X) ebreak 53
(X) csrrw 54
(X) csrrs 55
(X) csrrc 56
(X) csrrwi 57
(X) csrrsi 58
(X) csrrci 59
mul 60
div 61
rem 62