March 21, 2022

Lab 7 Part 1 related preparation for the Midterm Exam

Review of general concepts and recent midterm exam questions

Links to original labs/exams

1. Lab 7 Part 1 the 3-element pipeline adder (slides 3-14)
ee457_lab7_P1 (dir)
ee457_pipe_3elem_adder_Verilog.pdf

2. Q#1 from the midterm of Fall 2018 (slides 15-20)
ee457_MT_Fall2018.pdf
ee457_MT_Fall2018_sol.pdf

3. Q#1 from the midterm of Fall 2019 (slides 21-22)
ee457_MT_Fall2019.pdf
ee457_MT_Fall2019_sol.pdf

4. Q#3 from the midterm of Spring 2020 (slides 23-26)
ee457_MT_Spring2020.pdf
ee457_MT_Spring2020_sol.pdf

5. Q#1 from the midterm of Fall 2020 (slides 27-39)
ee457_MT_Fall2020.pdf
ee457_MT_Fall2020_sol.pdf

6. Q#1 from the midterm of Spring 2021 (slides 40-44)
ee457_MT_Sp2021.pdf
ee457_MT_Sp2021_sol.pdf

7. Q#1 from the midterm of Fall 2021 (slides 45-52)
ee457_MT_Fall2021.pdf
ee457_MT_Fall2021_sol.pdf

Lab 7 Part 1

3-element adder

Lab 7 Part 1 Block Diagram

3-element adder

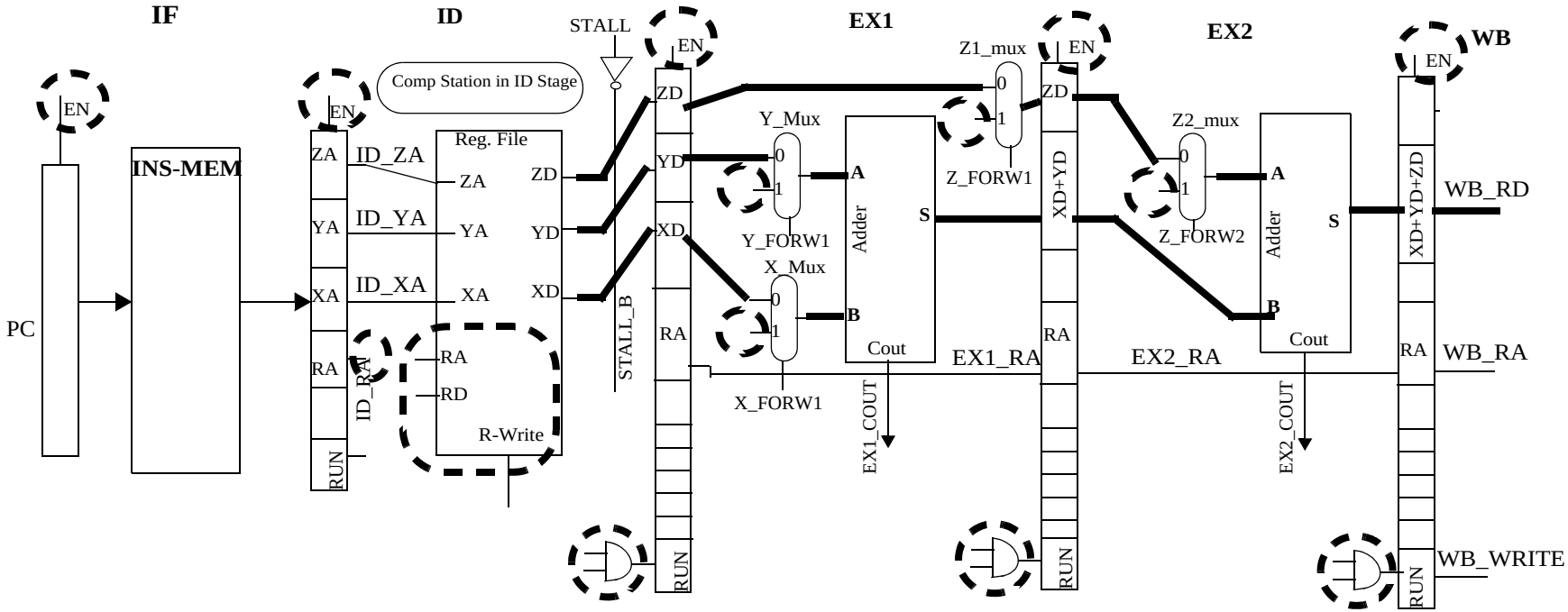
Seniors can only help from the WB stage.

Z-reg does not cause stalling.

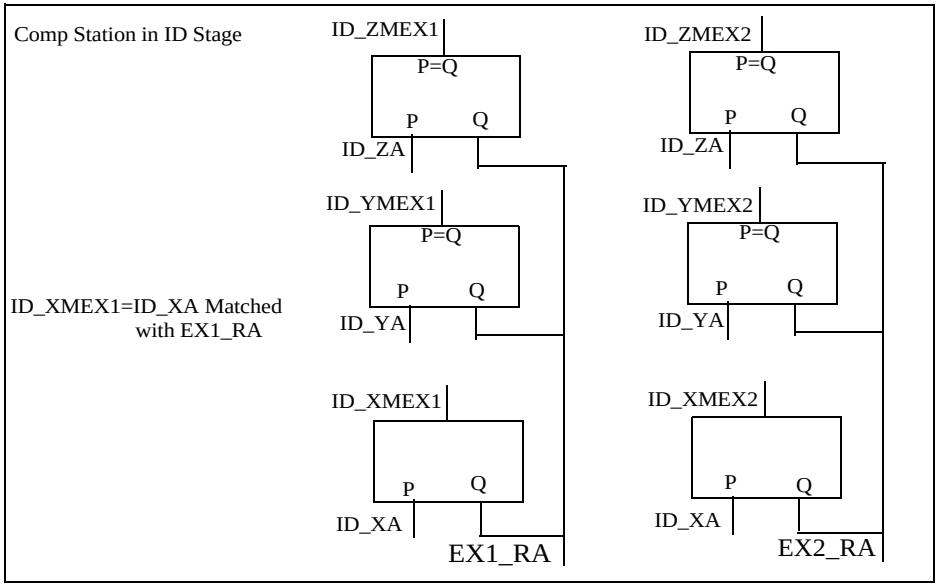
X and/or Y registers cause stalling
if they are dependent on their senior #1.

Priority in forwarding? Since only one senior can forward, there is no priority in forwarding.

Redundancy in forwarding: Since there is only one stage to receive forwarding help, there cannot be any redundant FMP (forwarding multiplexer pair).

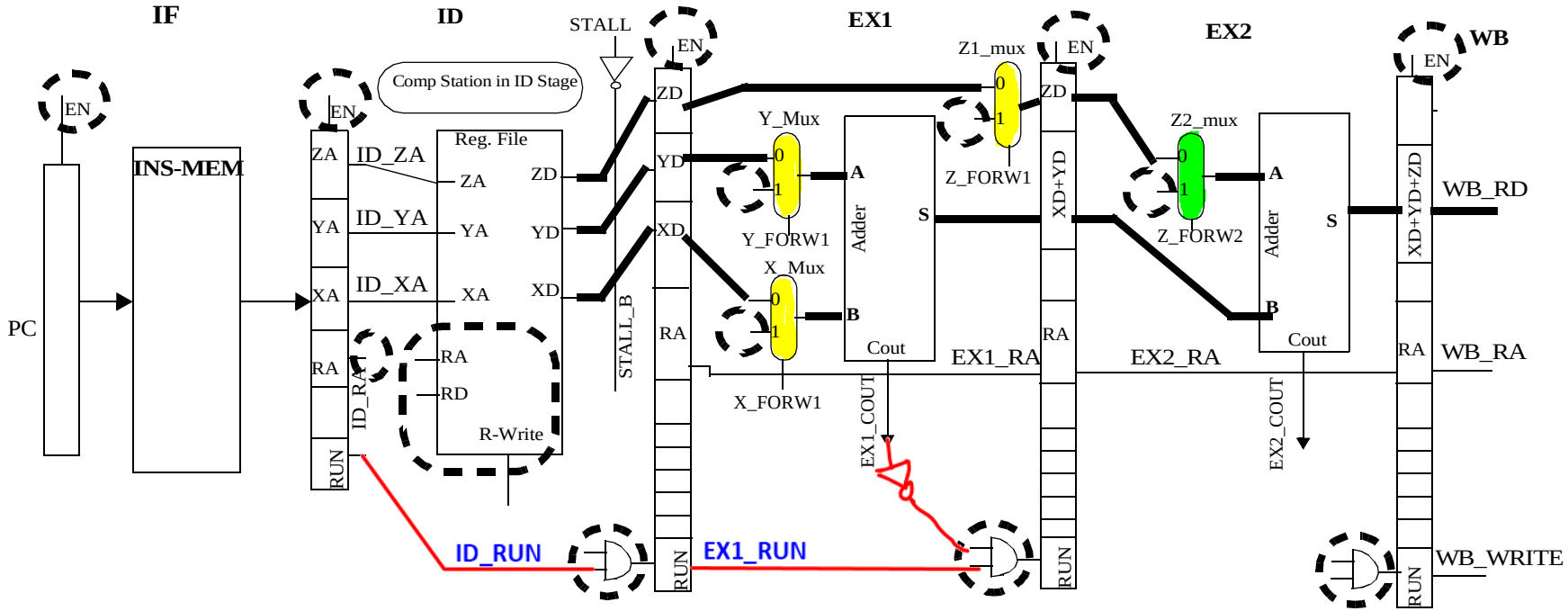


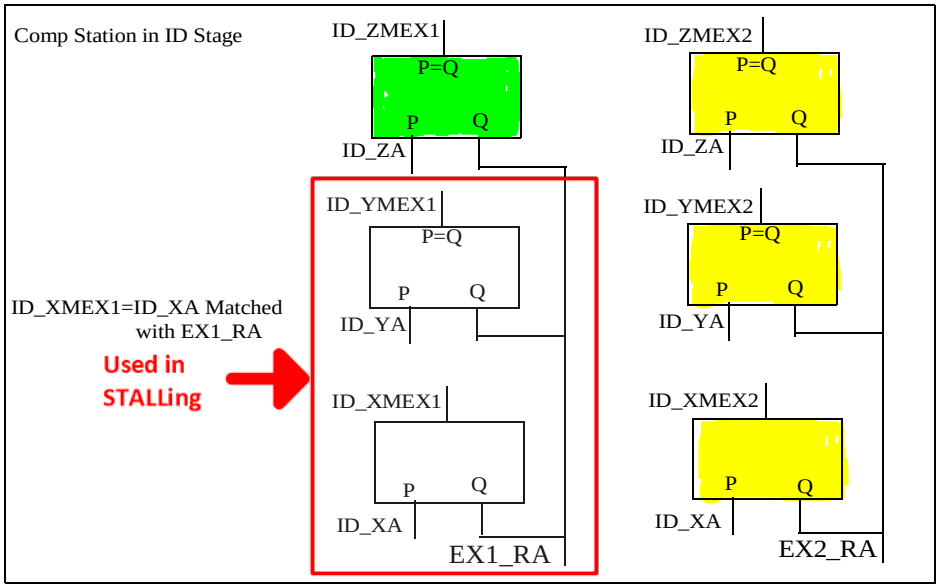
1. Complete all missing connections marked in dotted lines
   Also complete the RA(Result Address) connection in ID stage (ID_RA).
2. Complete all five enable (EN) controls on the pipeline registers (including PC).
3. Complete the forwarding paths into the four forwarding muxes.
4. Complete logic to inject bubble into the next stage if the current instruction is being stalled or being flushed.
5. Draw the logic on a separate page for generating **STALL, X_FORW1,Y_FORW1, Z_FORW1, Z_FORW2**.



Pinelined 3-element Adder Block Diagram
LAB 7 Part1

Fig. 1

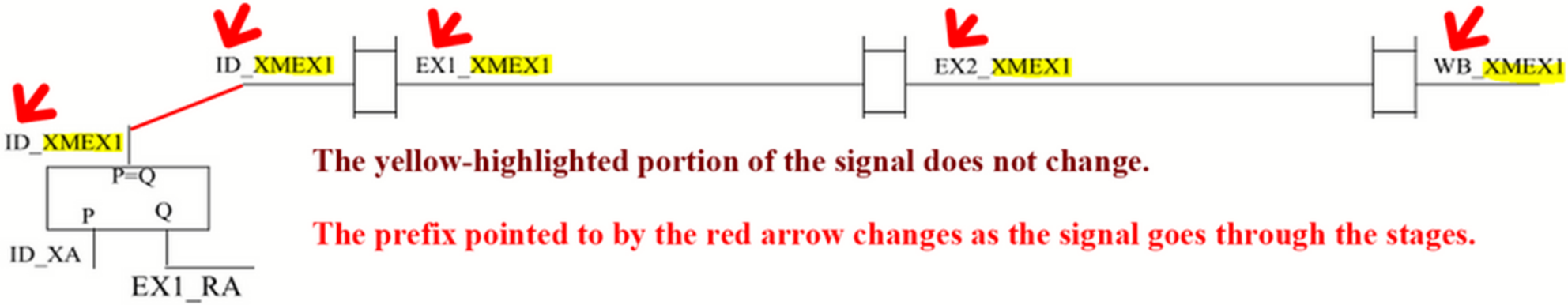




Pinelined 3-element Adder
Block Diagram
LAB 7 Part1

Comparison units
3 in IFRF + 6 Comp station in ID = 9 total

ee457_lab7_P1_comparator_inference_signal_naming_convention.pdf

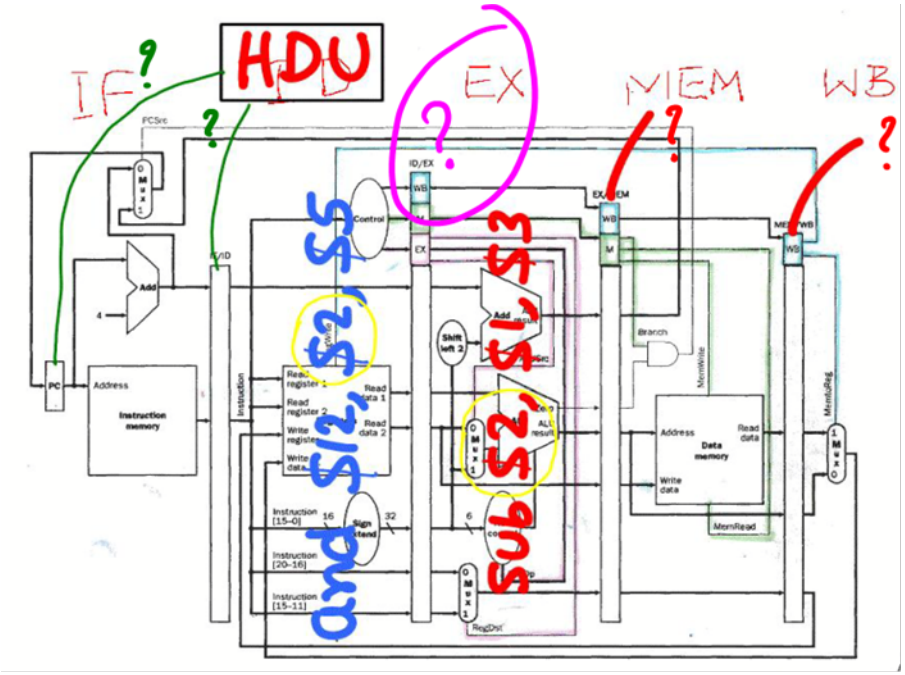


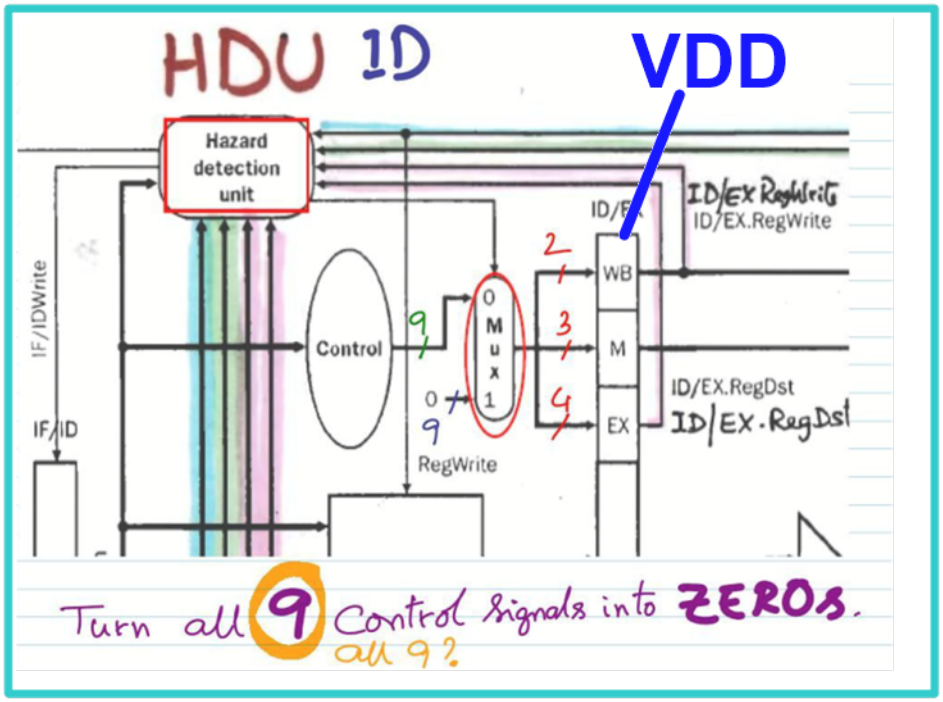
The yellow-highlighted portion of the signal does not change.

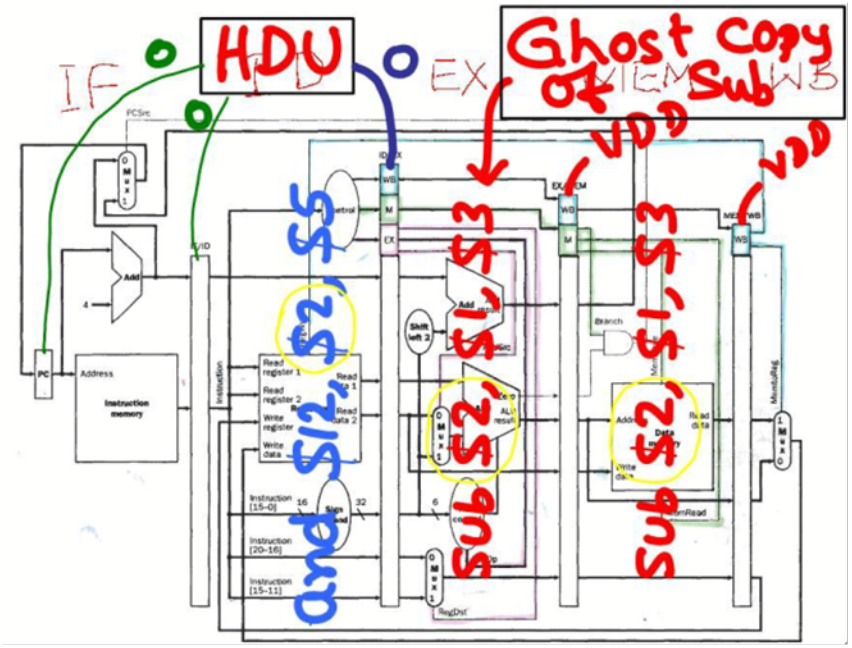
The prefix pointed to by the red arrow changes as the signal goes through the stages.

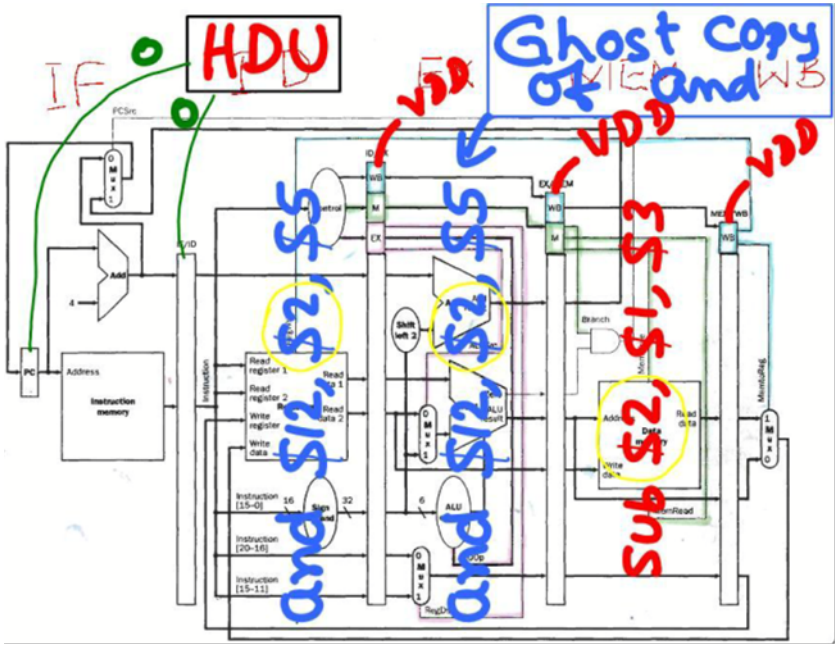
In some of the exams, we ask students to have Forwarding units in each of the EX stages, even through area wise and timing wise it is not a good idea.

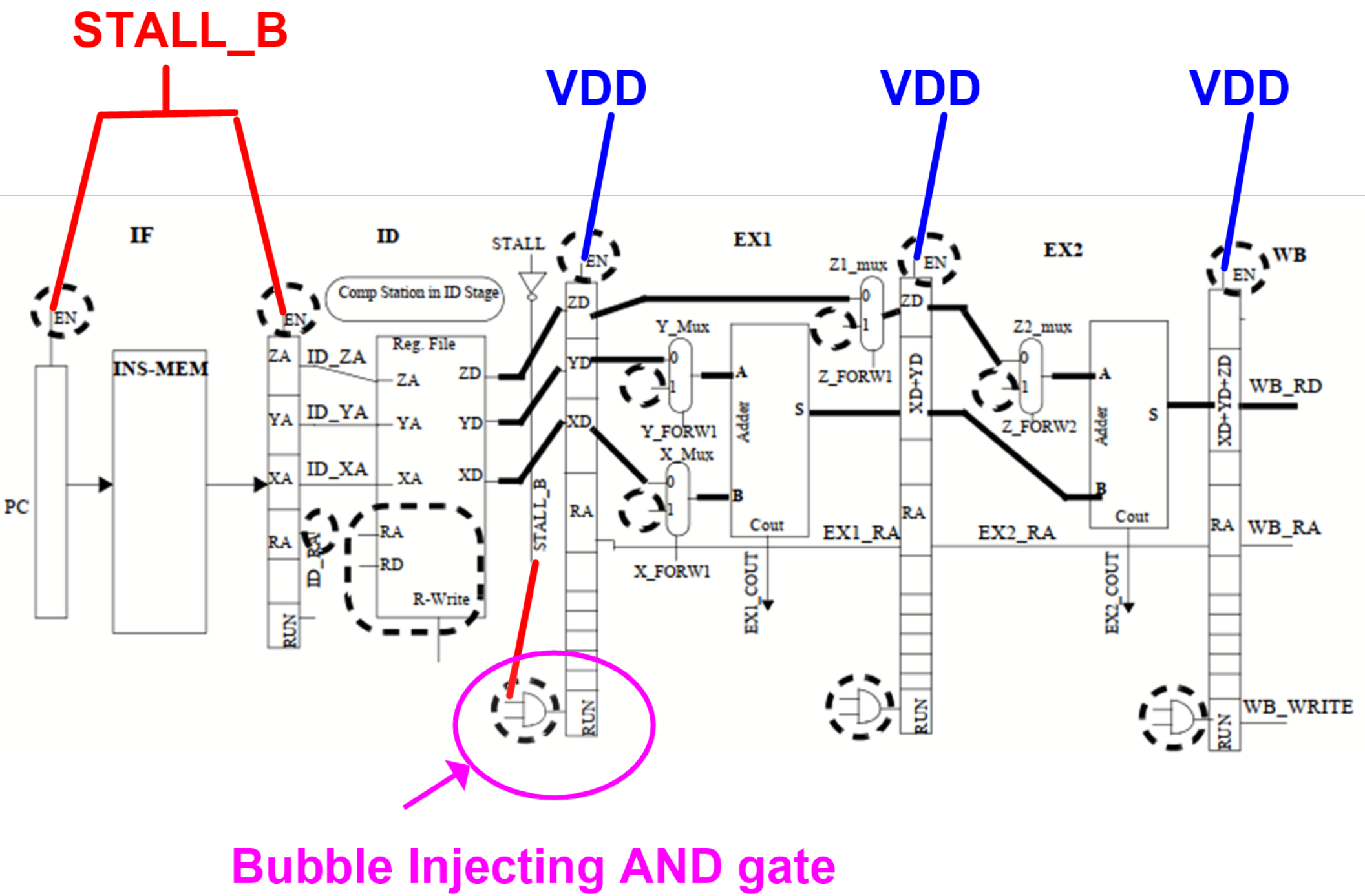
Do not create GHOST copies of the stalled or the senior instruction when stalling a dependent junior! Inject a bubble!



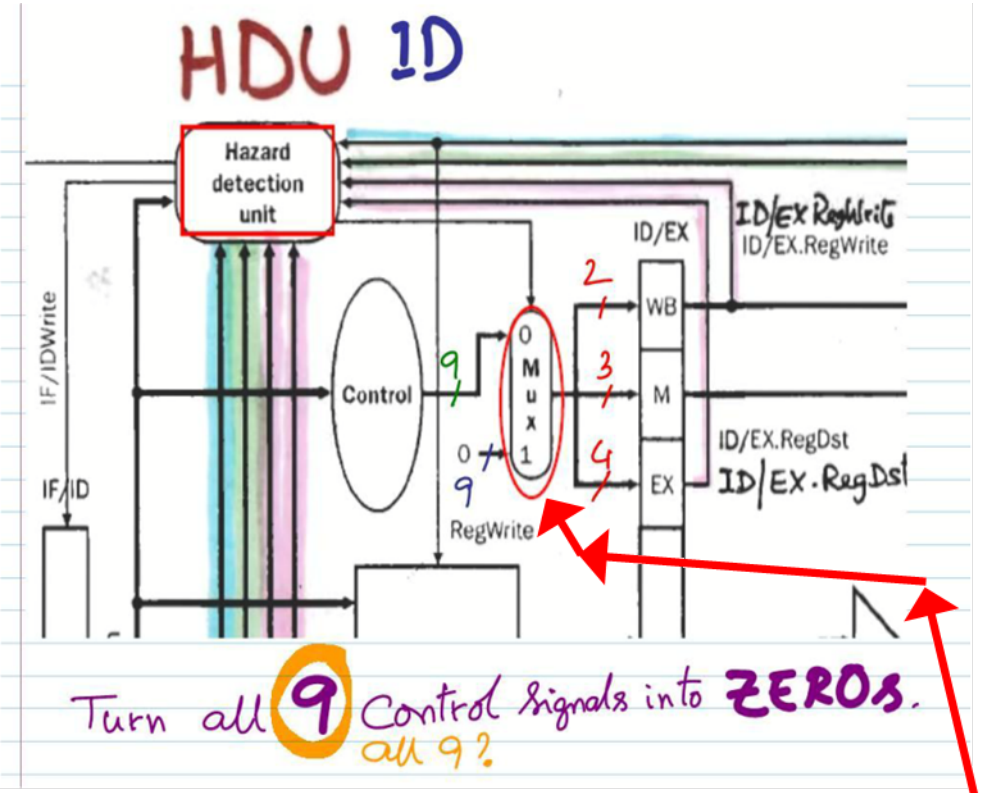


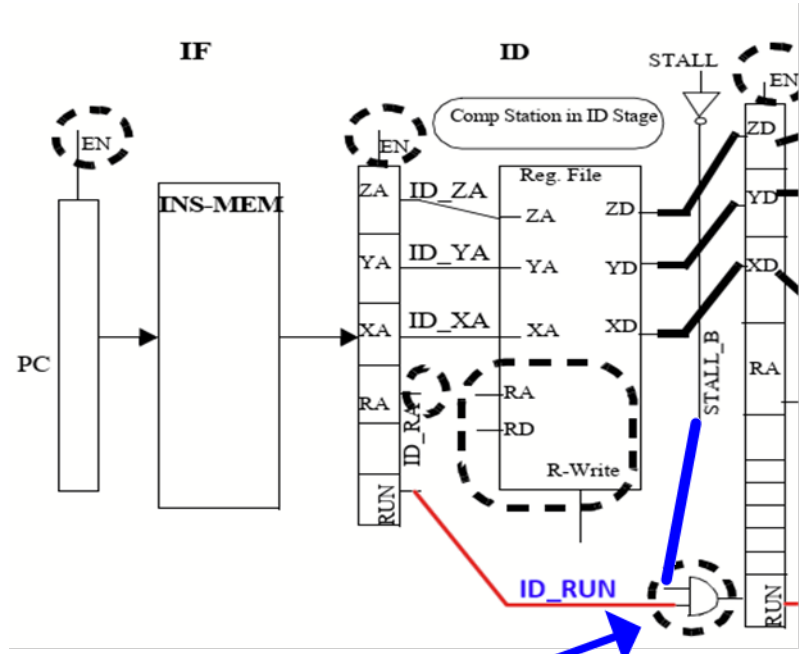




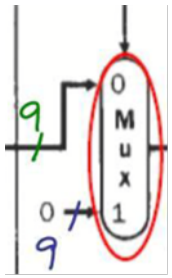

STALL_B
VDD
VDD
VDD
IF
ID
EX1
EX2
WB
STALL
EN
Comp Station in ID Stage
INS-MEM
PC
ZA
YA
XA
RA
RUN
ID_ZA
ID_YA
ID_XA
ID_RA
Reg. File
RD
R-Write
ZD
YD
XD
STALL_B
Y_Mux
Y_FORW1
X_Mux
X_FORW1
Adder
A
B
S
Cout
EX1_COUT
Z1_mux
Z_FORW1
XD+YD
EX1_RA
Z2_mux
Z_FORW2
EX2_RA
EX2_COUT
XD+YD+ZD
WB_RD
WB_RA
WB_WRITE
Bubble Injecting AND gate

HDU in our Lab 6 Part 4 (or any pipelined CPU) injects a bubble into the next (EX) stage, when it stalls an instruction in the ID stage.

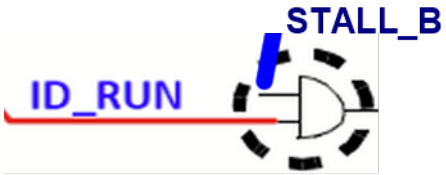




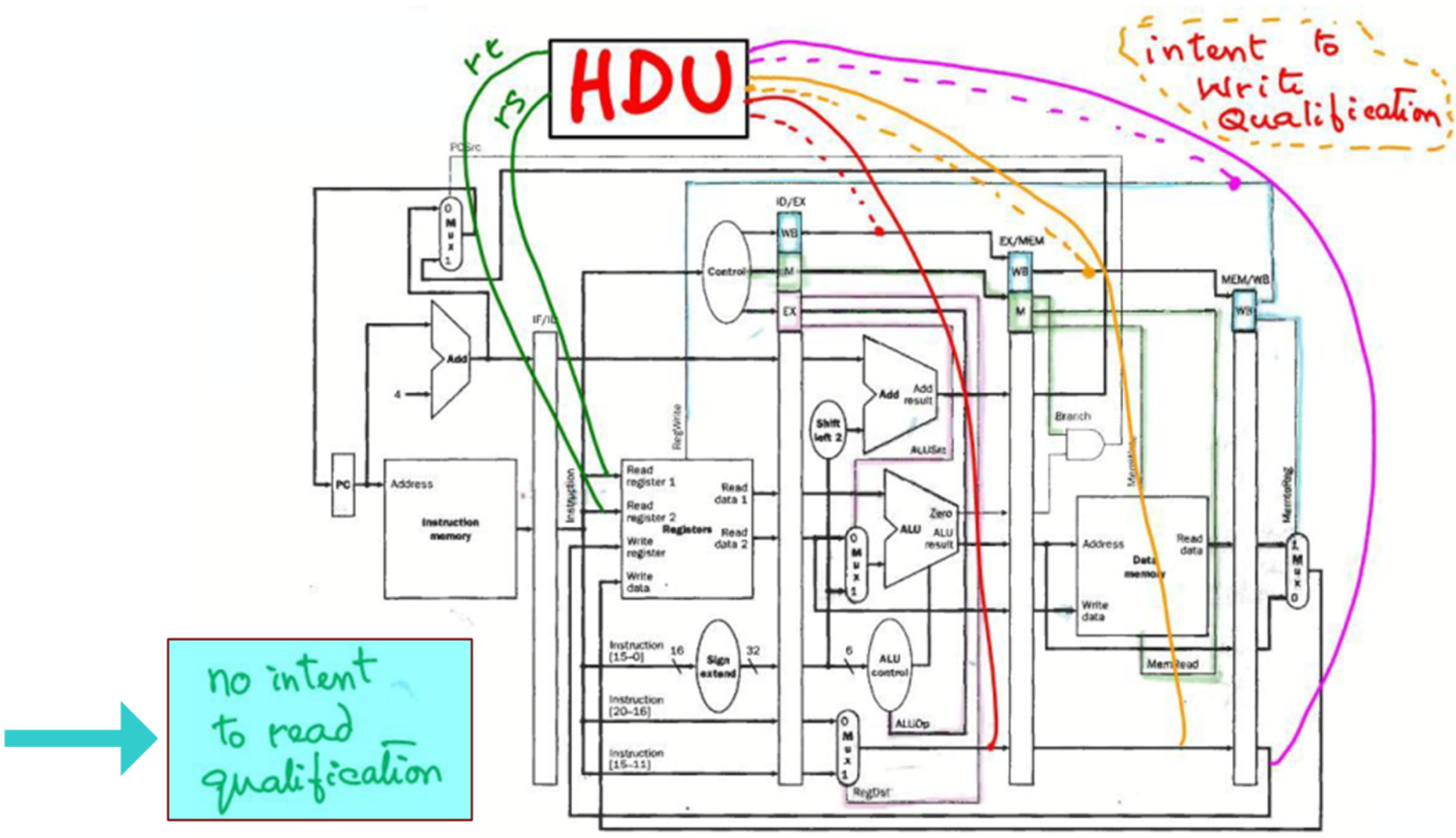
Mux with a zeros input is to illustrate the Bubble injection concept



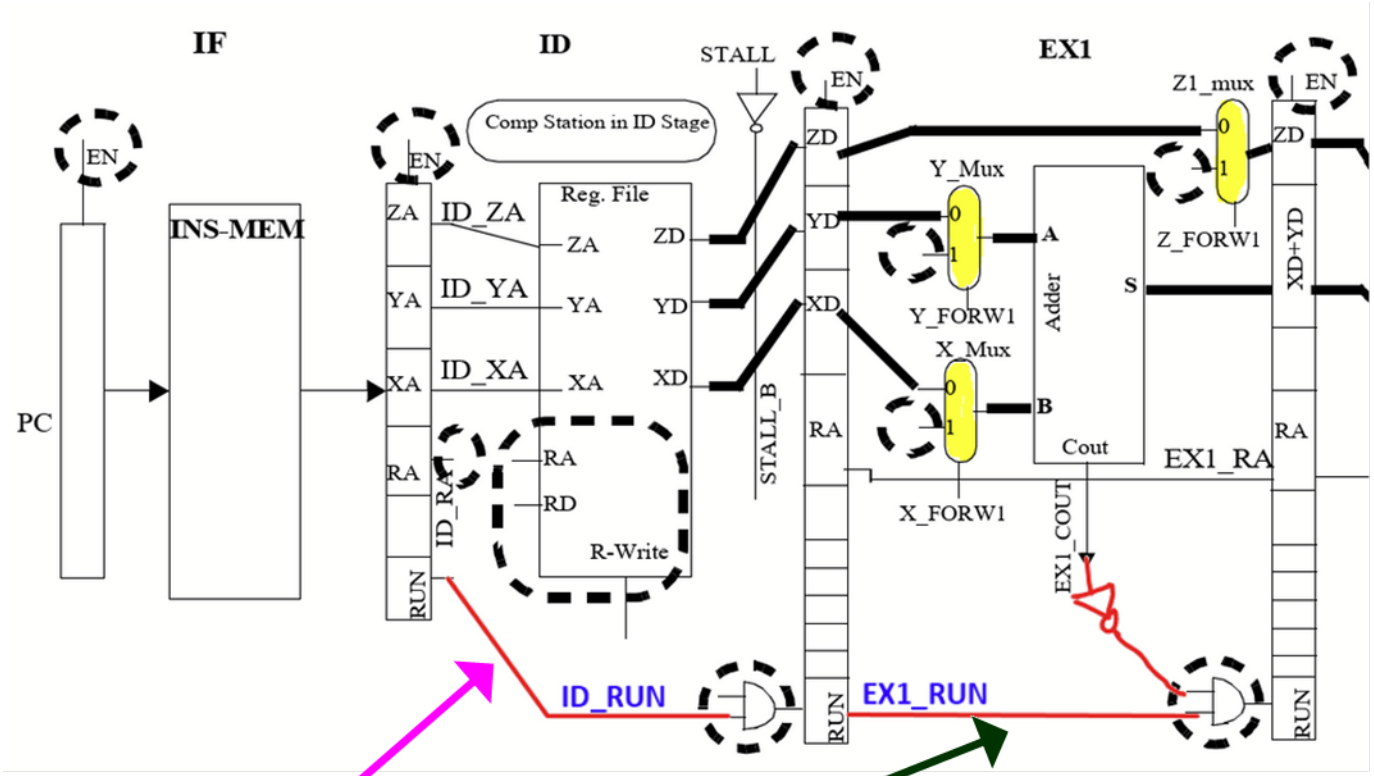
Actual implementation can be as simple as a bunch of AND gates (here one AND gate.

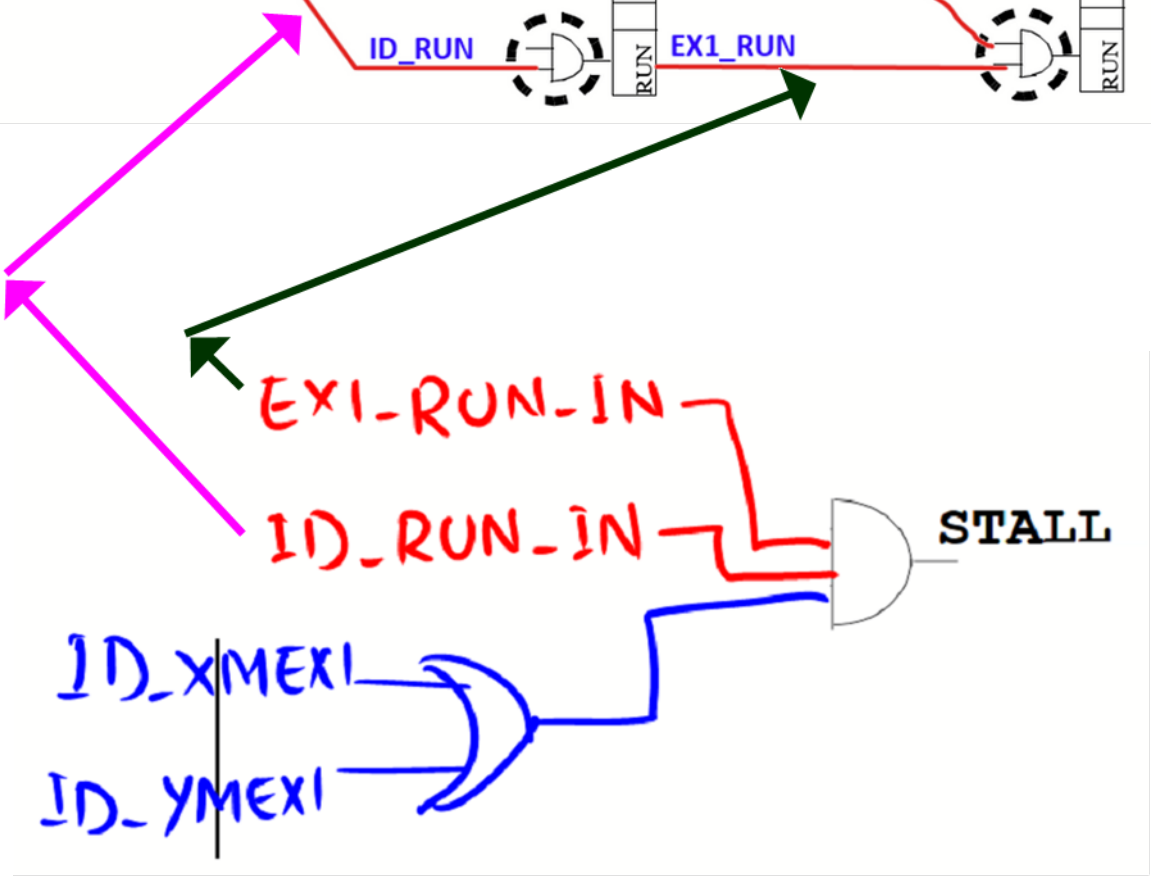


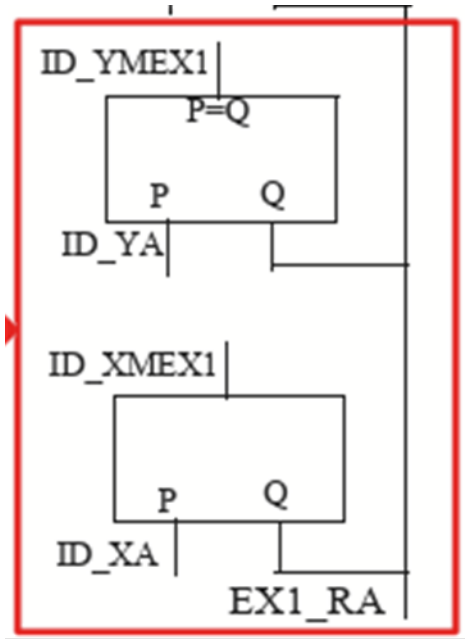
Do spurios stalls occur in the 5-stage CPU of Lab 6 Part 4 and/or this Lab 7 Part 1?



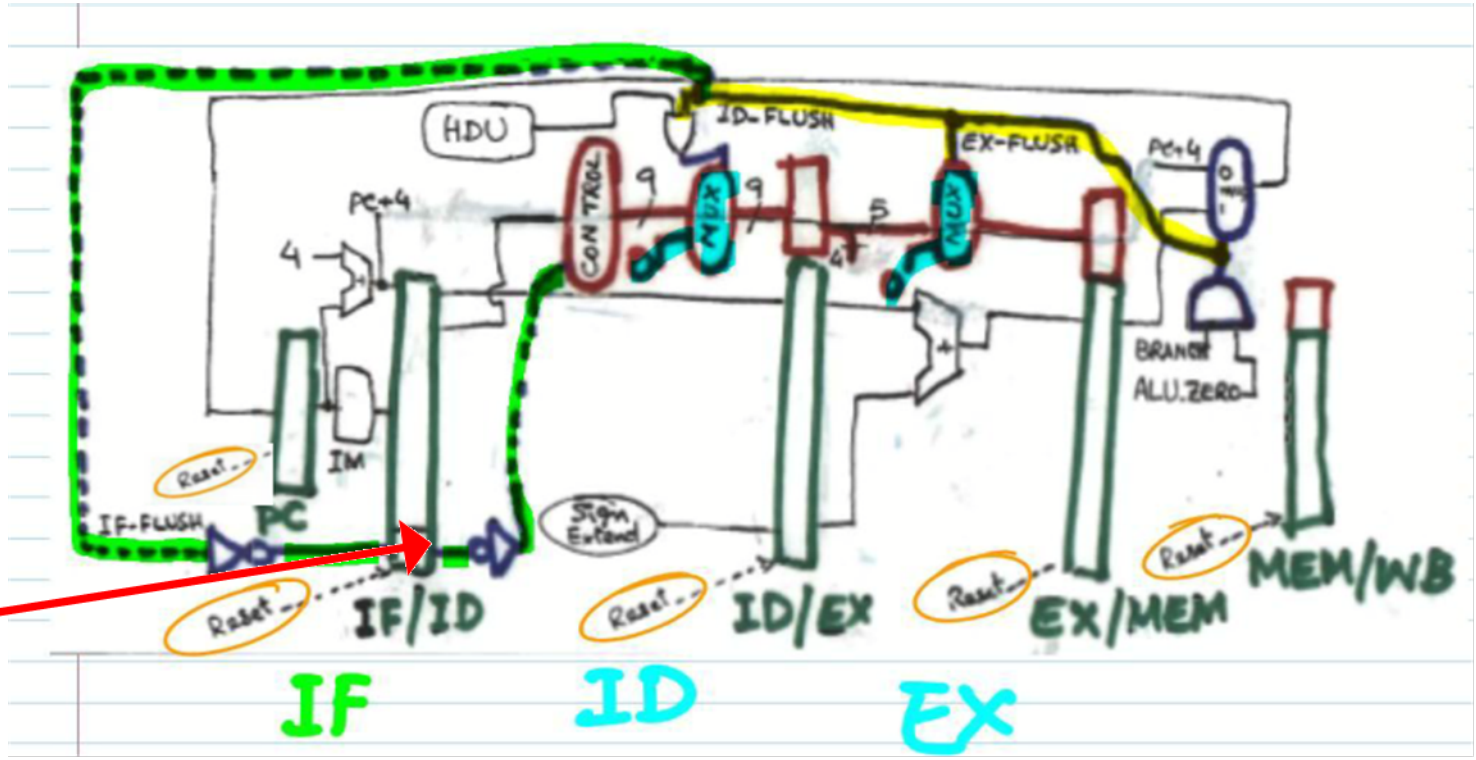
Stall logic in Lab 7 (all parts) will check to see that the ID_RUN_IN is true.


IF
ID
EX1
STALL
EN
Comp Station in ID Stage
Z1_mux
PC
INS-MEM
ZA
YA
XA
RA
RUN
ID_ZA
ID_YA
ID_XA
ID_RA
Reg. File
ZD
YD
XD
RD
R-Write
STALL_B
Y_Mux
Y_FORW1
X_Mux
X_FORW1
Adder
A
B
S
Cout
Z_FORW1
XD+YD
EX1_RA
EX1_COUT
ID_RUN
EX1_RUN


EX1_RUN_IN
ID_RUN_IN
STALL
ID_XMEX1
ID_YMEX1


ID_YMEX1
P=Q
P
Q
ID_YA
ID_XMEX1
P
Q
ID_XA
EX1_RA

On power-on reset, you want to make sure that there are no RANDOM instructions in the pipeline.



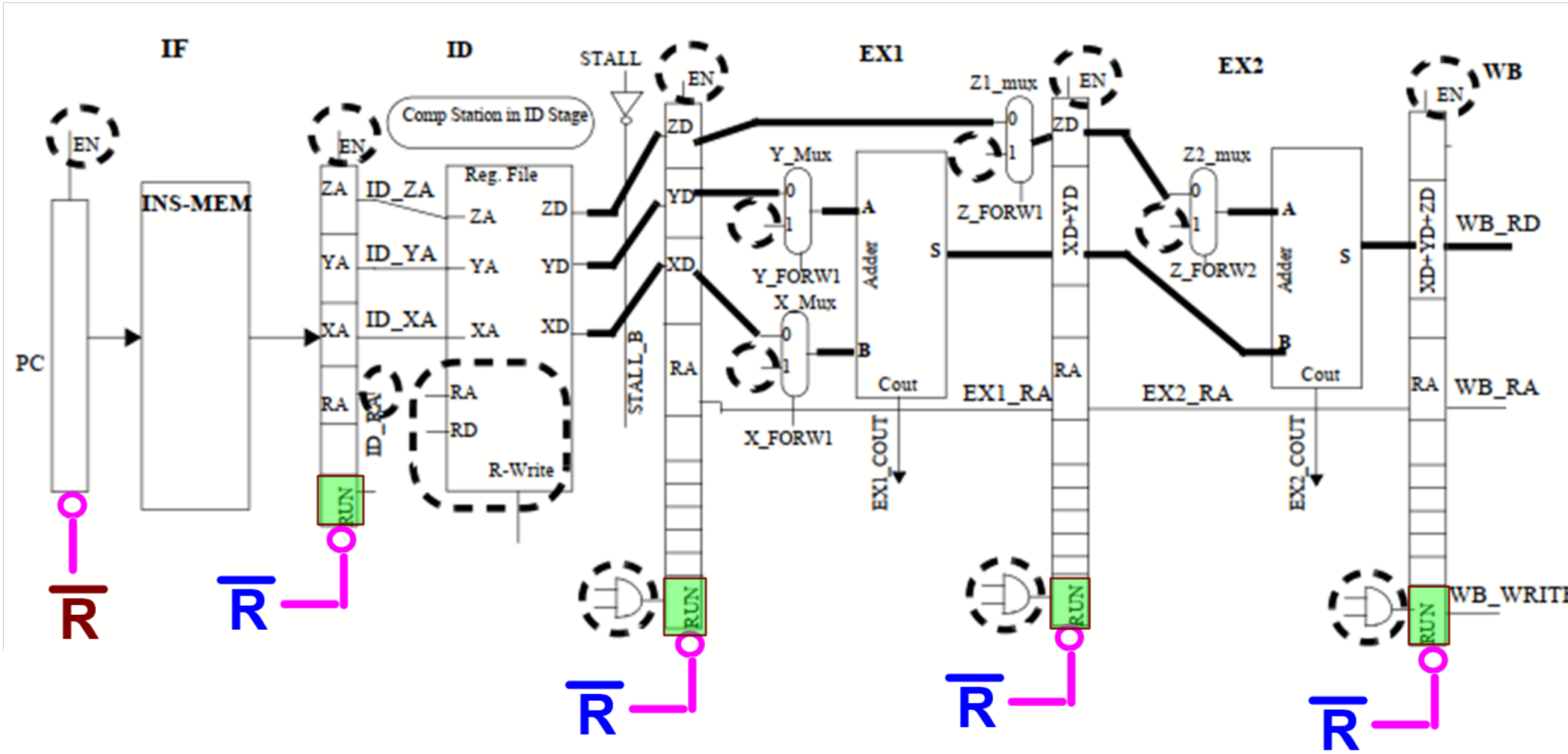
Dual purpose of the WBFF in Lab 6 Part 4

Why don't we need a WBFF in Lab 7 Part 1?

Answer: No opcode here. Single-bit active-high RUN signal! So, no difficulty in converting the RANDOM instr in ID stage to a bubble. There is no branch instruction to flush the IF stage fall-through junior through wrist banding. Even if there is a branch instruction, we can inject a bubble into ID from the IF stage to flush the IF junior!

The reset signal (during power-on reset) asynchronously clears the RUN signal to replace the random instruction in the right 4 stages with a bubble.

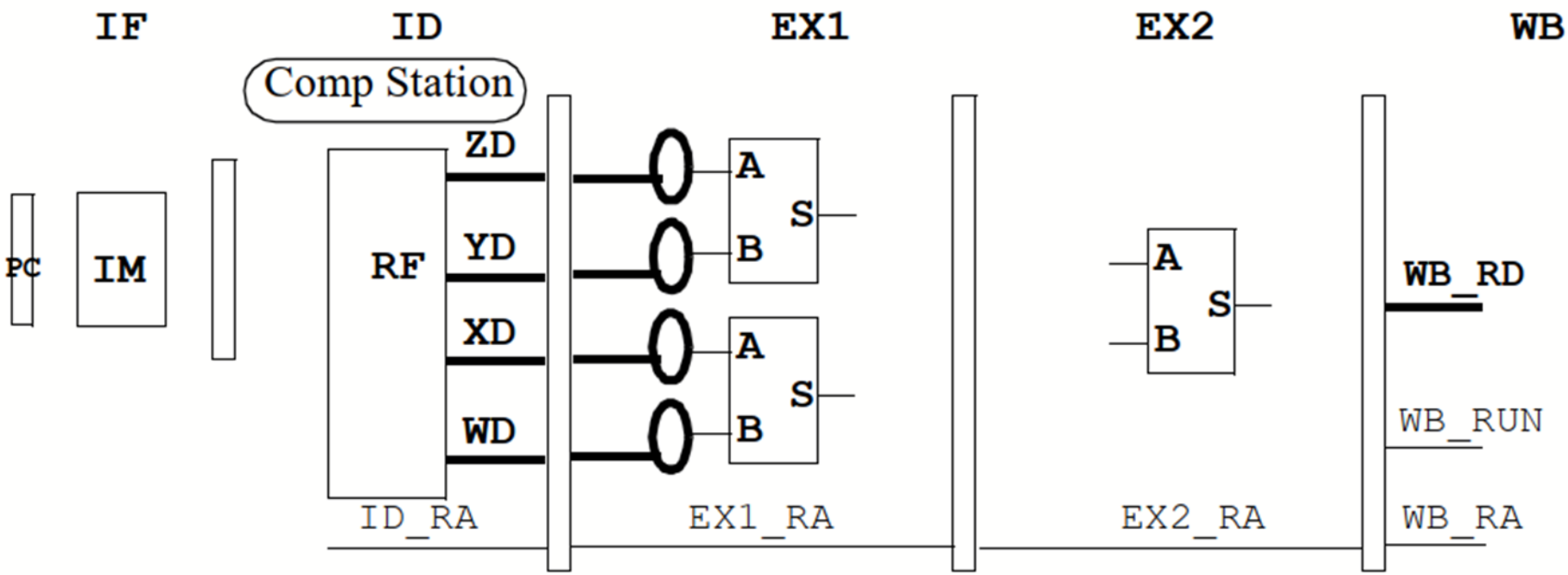
The RESET signal also clears the PC so that the very first boot-up instruction at (PC=0) is fetched in IF stage.



Q#1 Fall 2018 Midterm

A 4-element adder

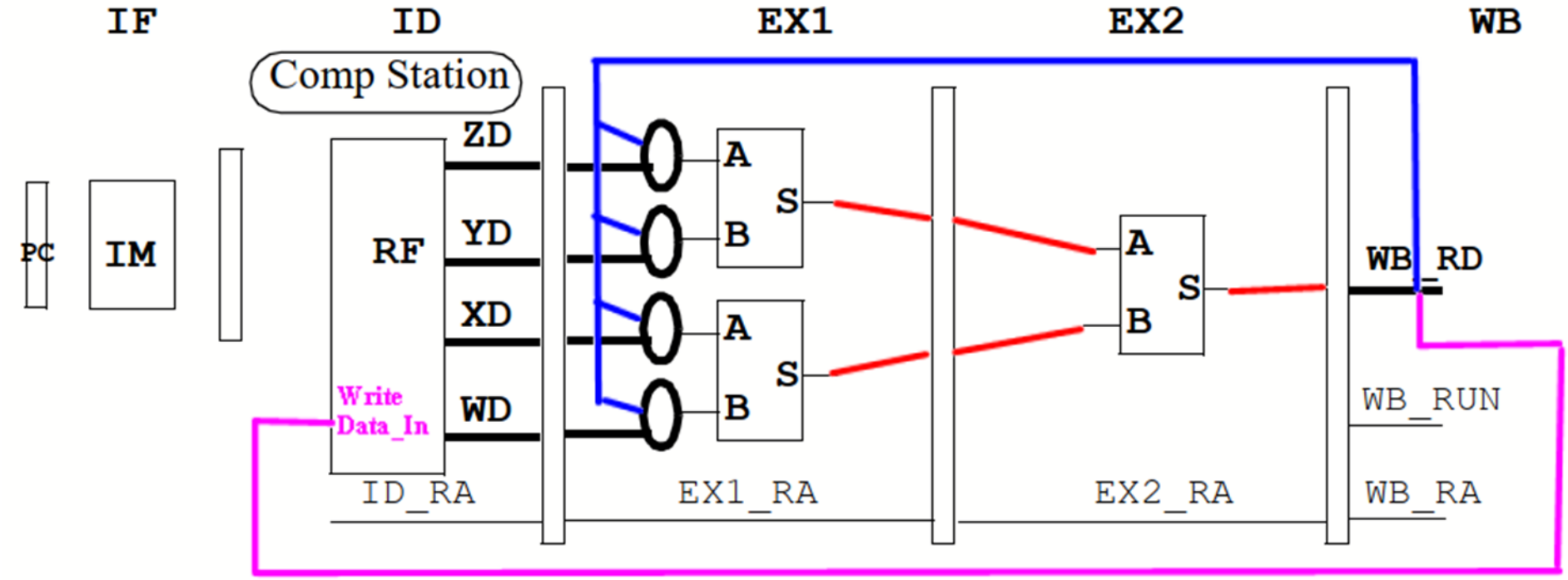
The 5-stage design:



Comparison units in the IFRF: _____

Comparison units in the comp station: _____

____ used in stalling and _____ used for forwarding

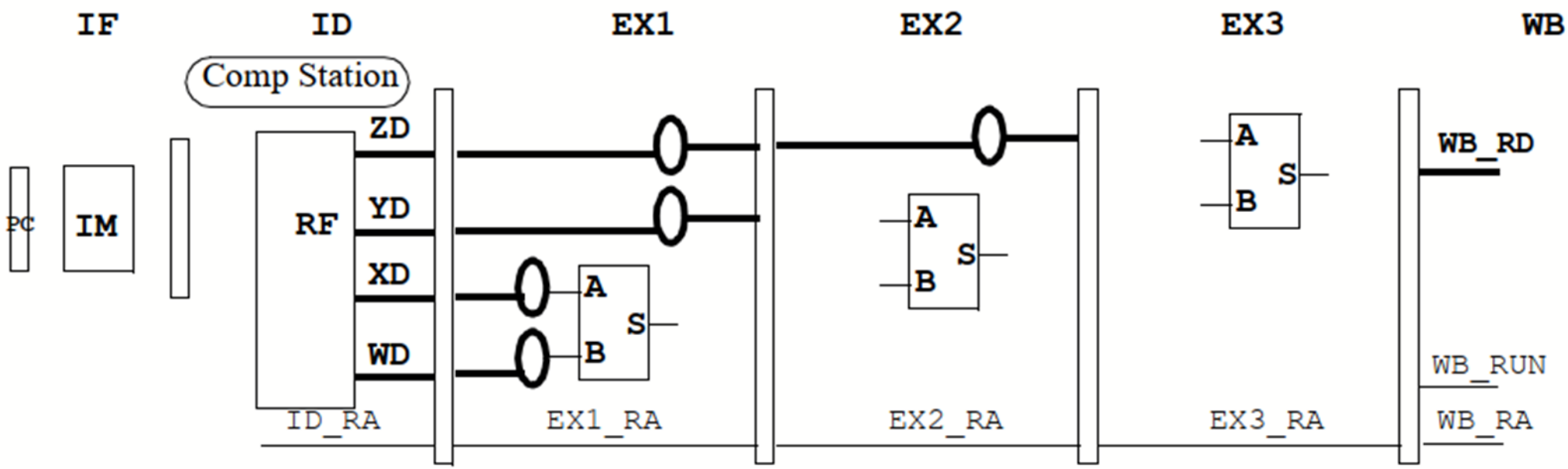


Comparison units in the IFRF: 4

Comparison units in the comp station: 8

4 **used in stalling and** 4 **used for forwarding**

The 6-stage design:



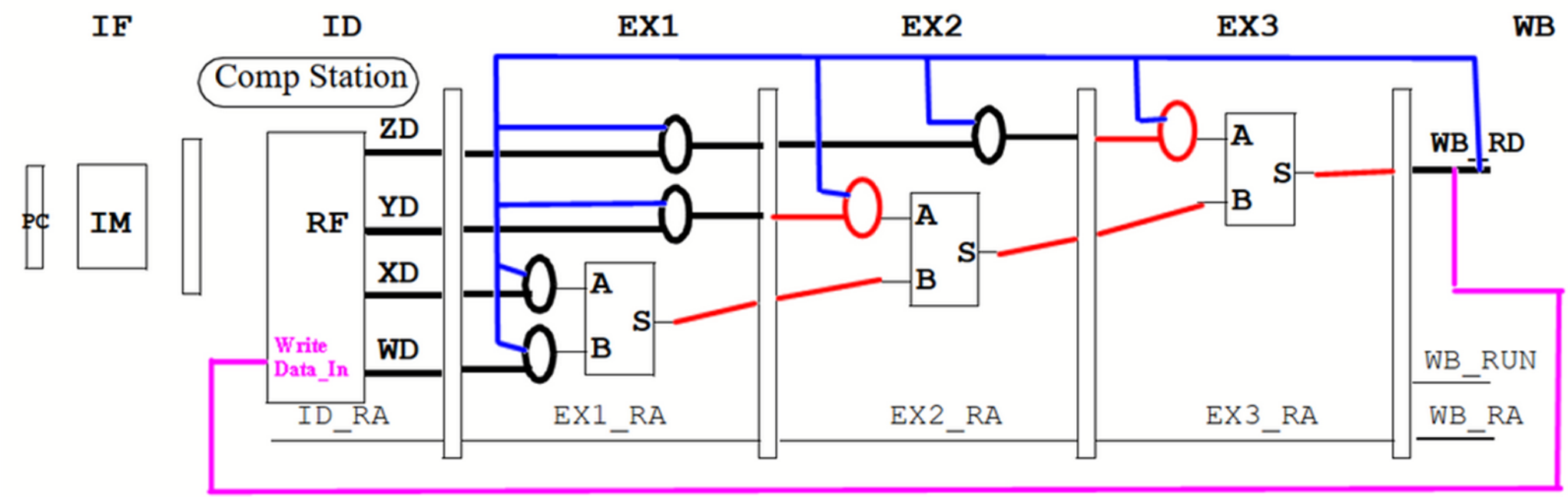
Comparison units in the IFRF: _____

Comparison units in the comp station: _____

____ used in stalling and _____ used for forwarding

Q#1 F2018 MT

The 6-stage design:



Comparison units in the IFRF: 4

Comparison units in the comp station: 12

5 used in stalling and 7 used for forwarding

The 5-stage and the 6-stage
Each has its own advantage and disadvantage!

If you want the **5**-stage to look better, you write testbench with more instruction sequences such that

If you want the **6**-stage to look better, you write testbench with more instruction sequences such that

If you want the **5**-stage to look better, you write testbench with more instruction sequences such that **either $W or $X is dependent on its senior #2. Then the 5-stage design does not stall but the 6-stage design would stall.**

Basically 5-stage does better than the 6-stage for solving dependencies of $W and $X. $Y is same. But for $Z, the 6-stage is better.

If you want the **6**-stage to look better, you write testbench with more instruction sequences such that **$Z is dependent on its senior #1. Then the 6-stage design does not need to stall but the 5-stage design would stall.**

Basically 5-stage does better than the 6-stage for solving dependencies of $W and $X. $Y is same. But for $Z, the 6-stage is better.

Q#1 Fall 2019 Midterm

A 4-element adder

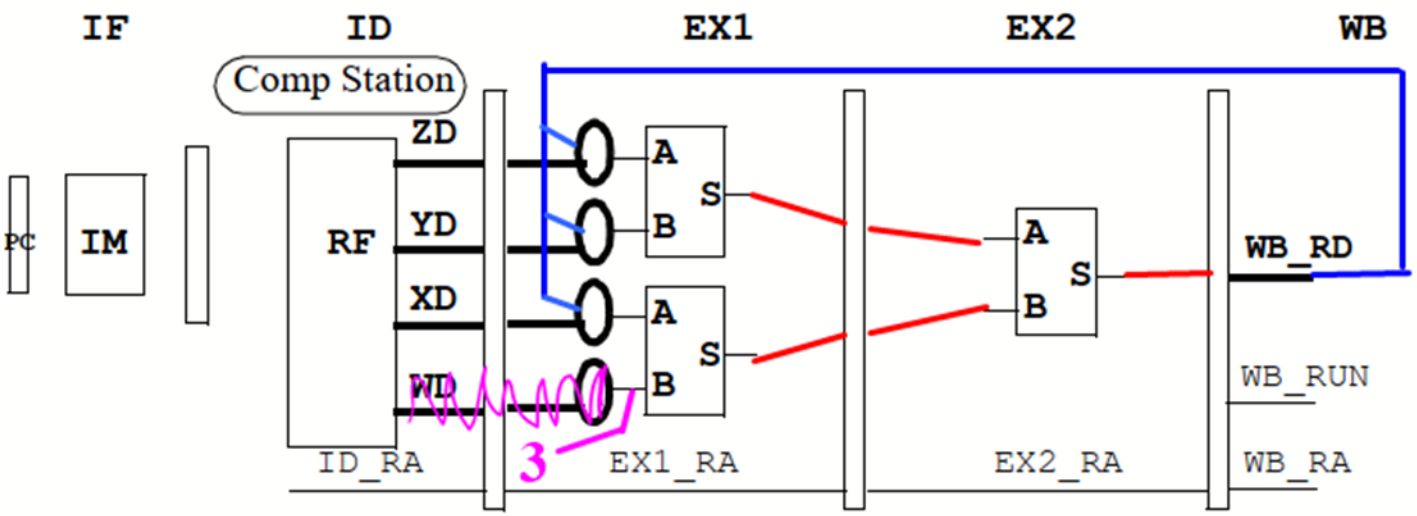
One of the four is a constant 3

Do you want that to be the W or the X or the Y or the Z?

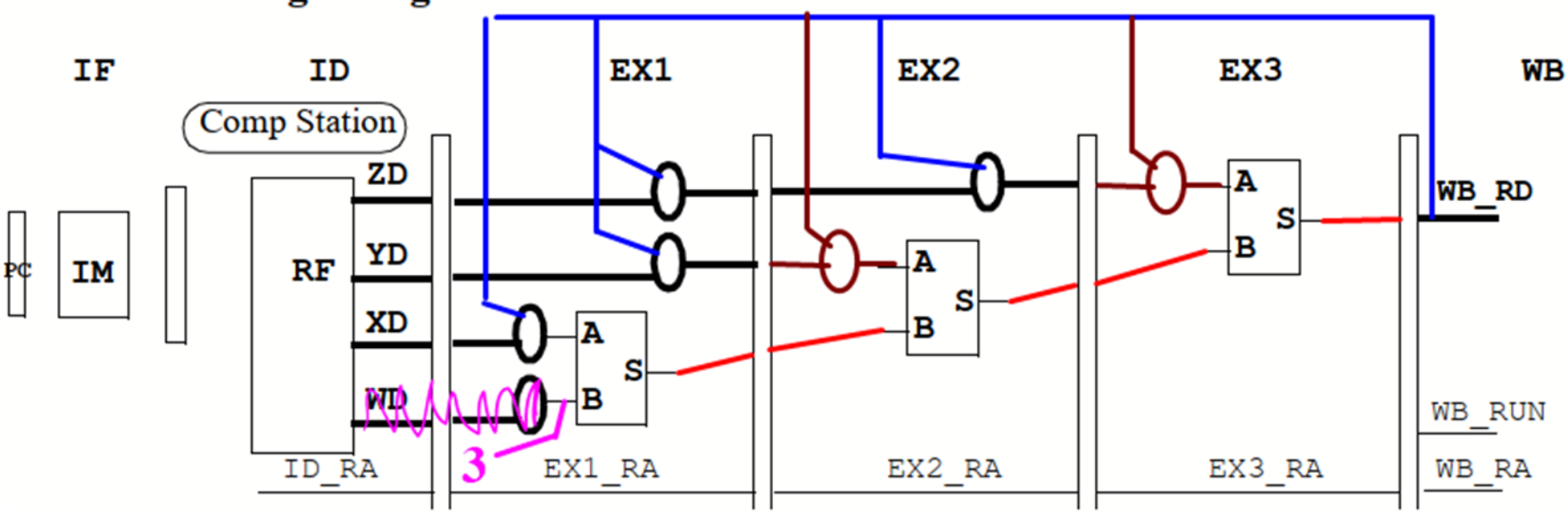
Ans: W or X to avoid as many stalls as possible

W or X can be replaced by constant 3

The 5-stage design:



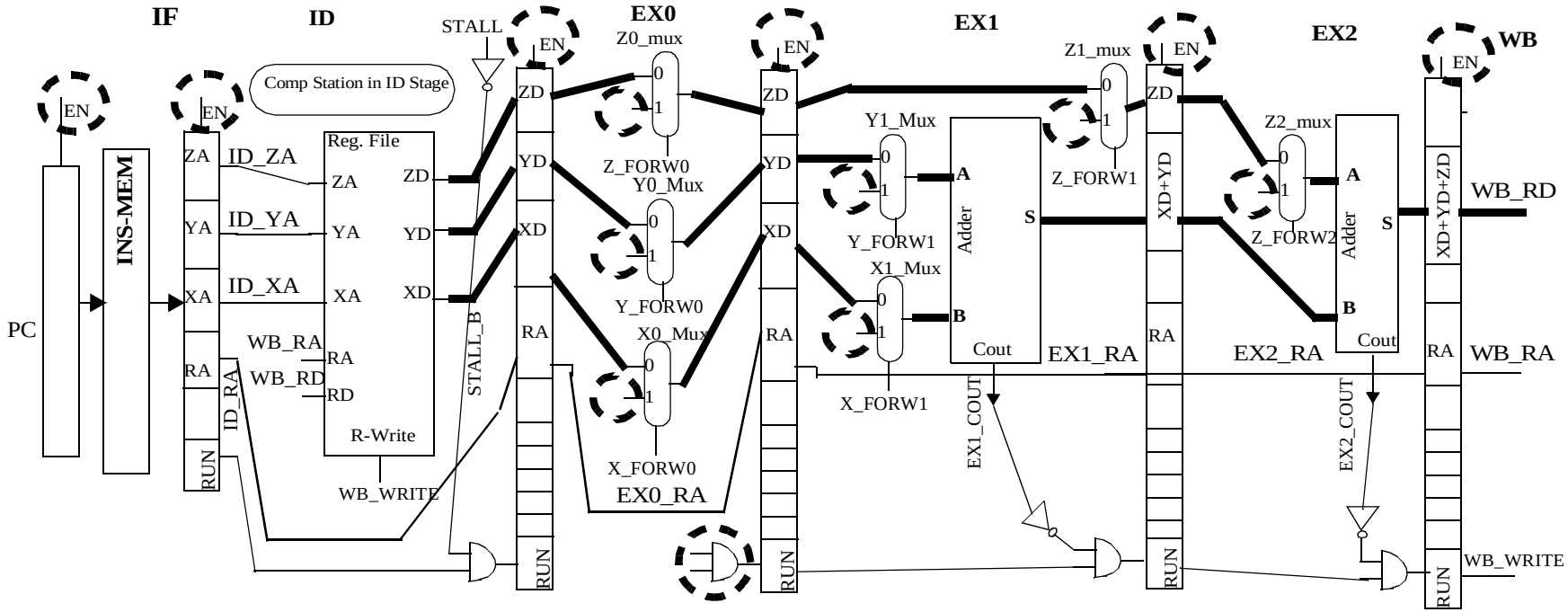
The 6-stage design:



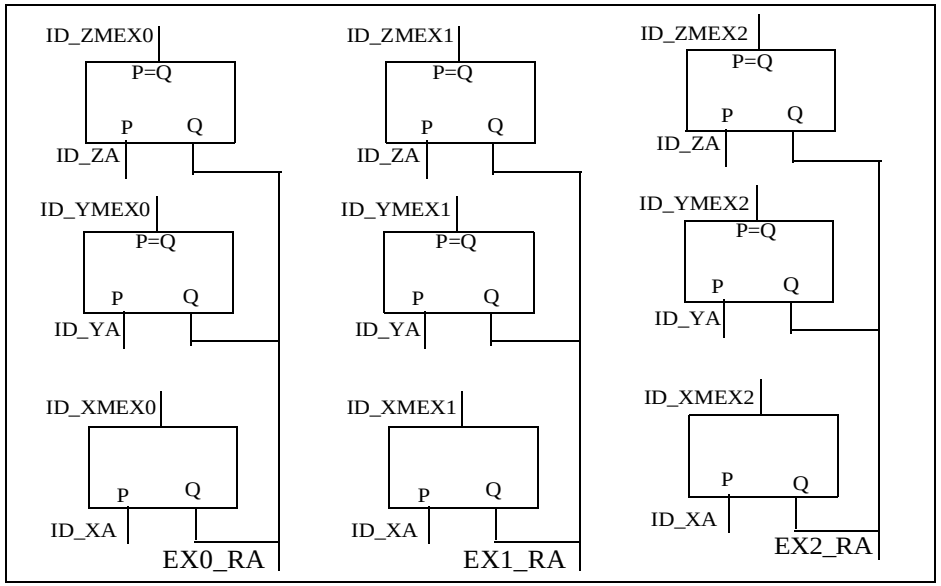
Q#3 Spring 2020 MT

The 3-element adder

With a Dummy Stage EX0

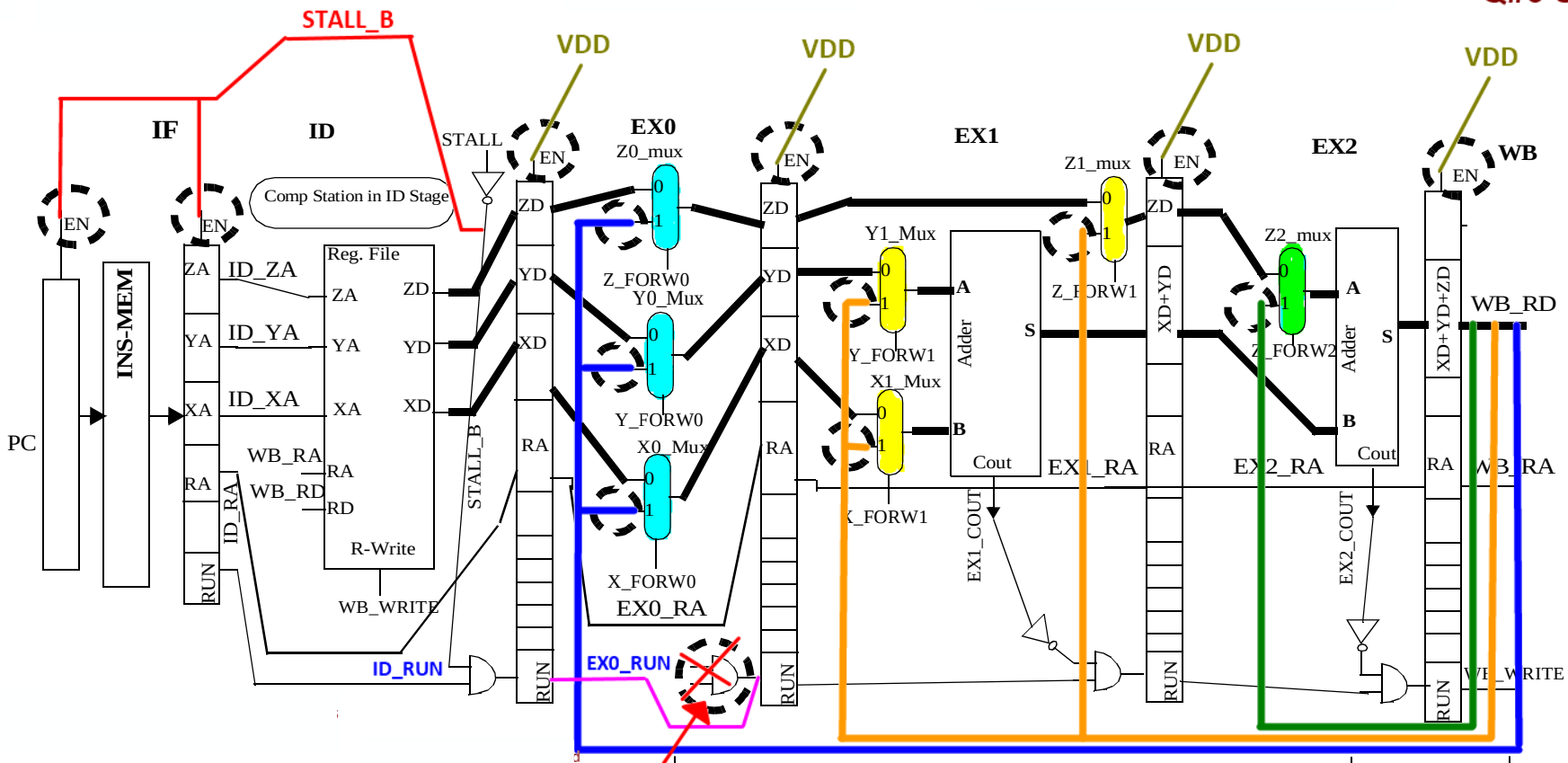


Cross out any unneeded muxes, comparison units, etc. and complete the remaining connections marked as



Pinelined 3-element Adder
Block Diagram
LAB 7 Part1 modified (EX0 added)

Q#3 S2020 MT



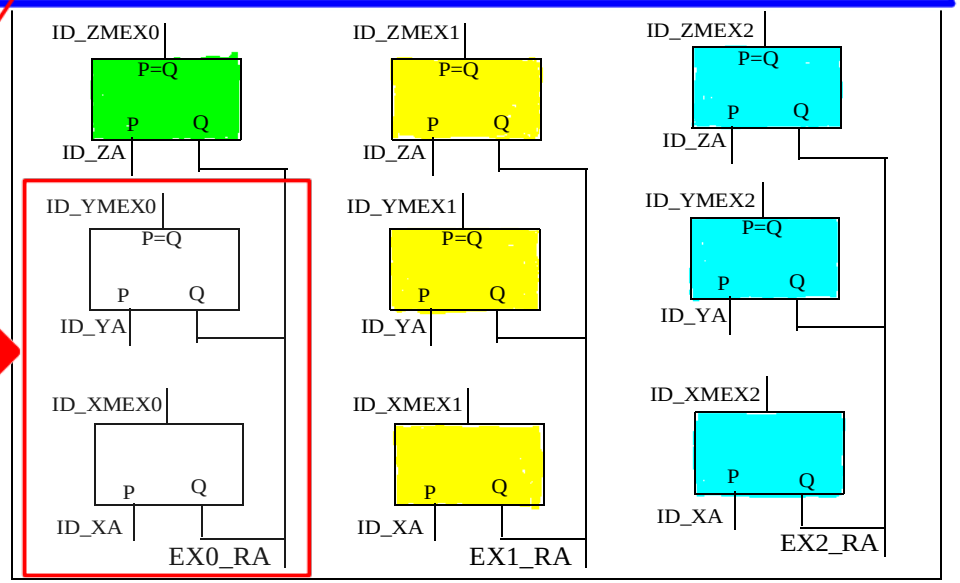
Only thing that needed crossing off is the 2-input AND gate in the EX0 stage.

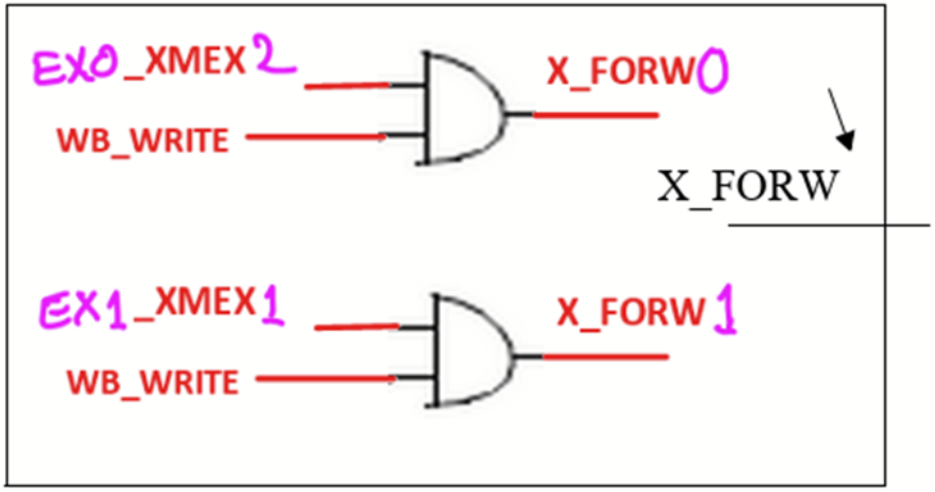
Cross out any unneeded muxes, comparison units, etc. and complete the remaining connections marked as

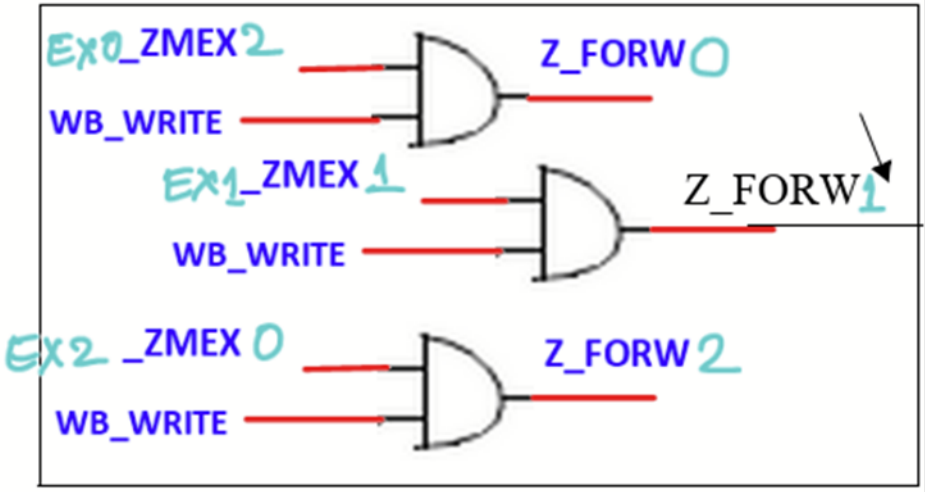
All three new muxes and all three new comparison units are all needed!

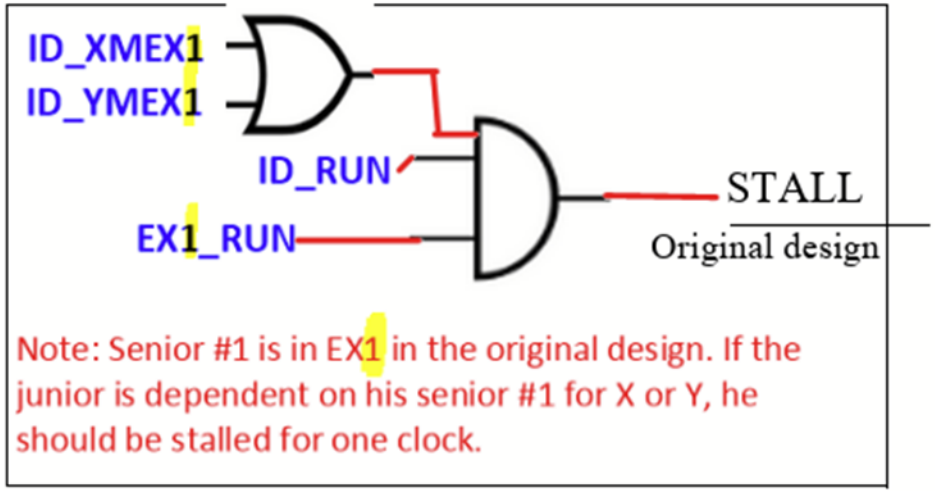
Pinelined 3-element Adder
Block Diagram
LAB 7 Part1 modified (EX0 added)

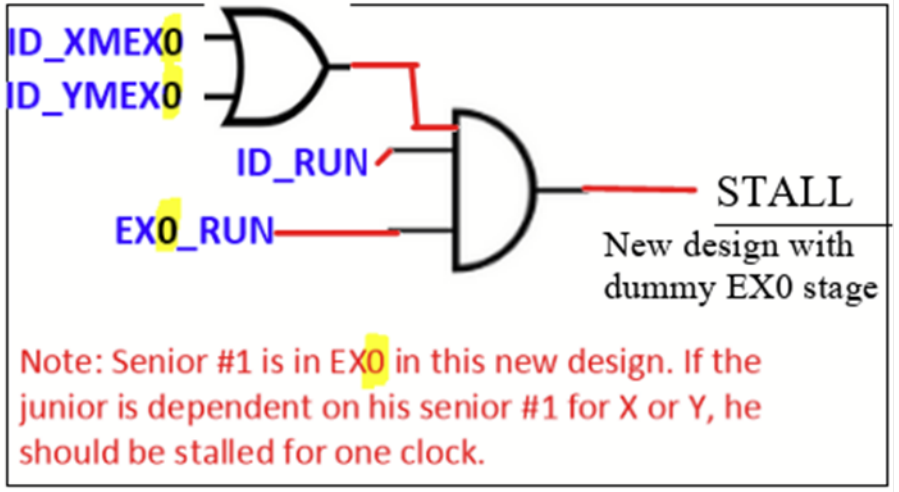
Used in STALLing




EX0_XMEX2
WB_WRITE
X_FORW0
X_FORW
EX1_XMEX1
WB_WRITE
X_FORW1

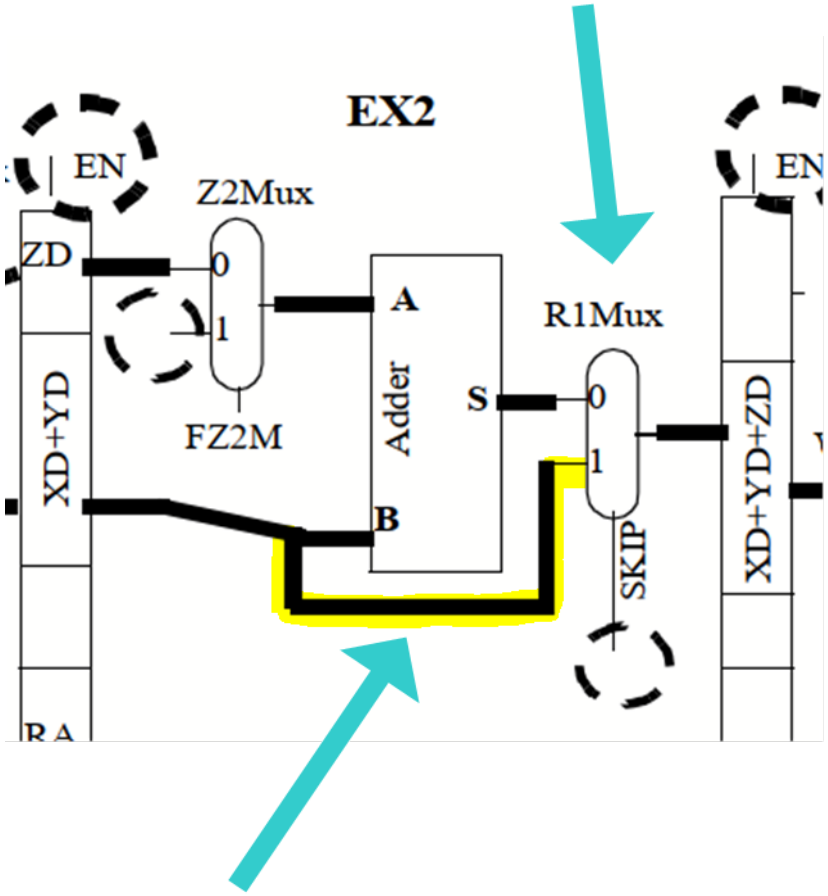

Ex0_ZMEX2
WB_WRITE
Z_FORW0
Ex1_ZMEX1
WB_WRITE
Z_FORW1
Ex2_ZMEX0
WB_WRITE
Z_FORW2


ID_XMEX1
ID_YMEX1
ID_RUN
EX1_RUN
STALL
Original design
Note: Senior #1 is in EX1 in the original design. If the junior is dependent on his senior #1 for X or Y, he should be stalled for one clock.

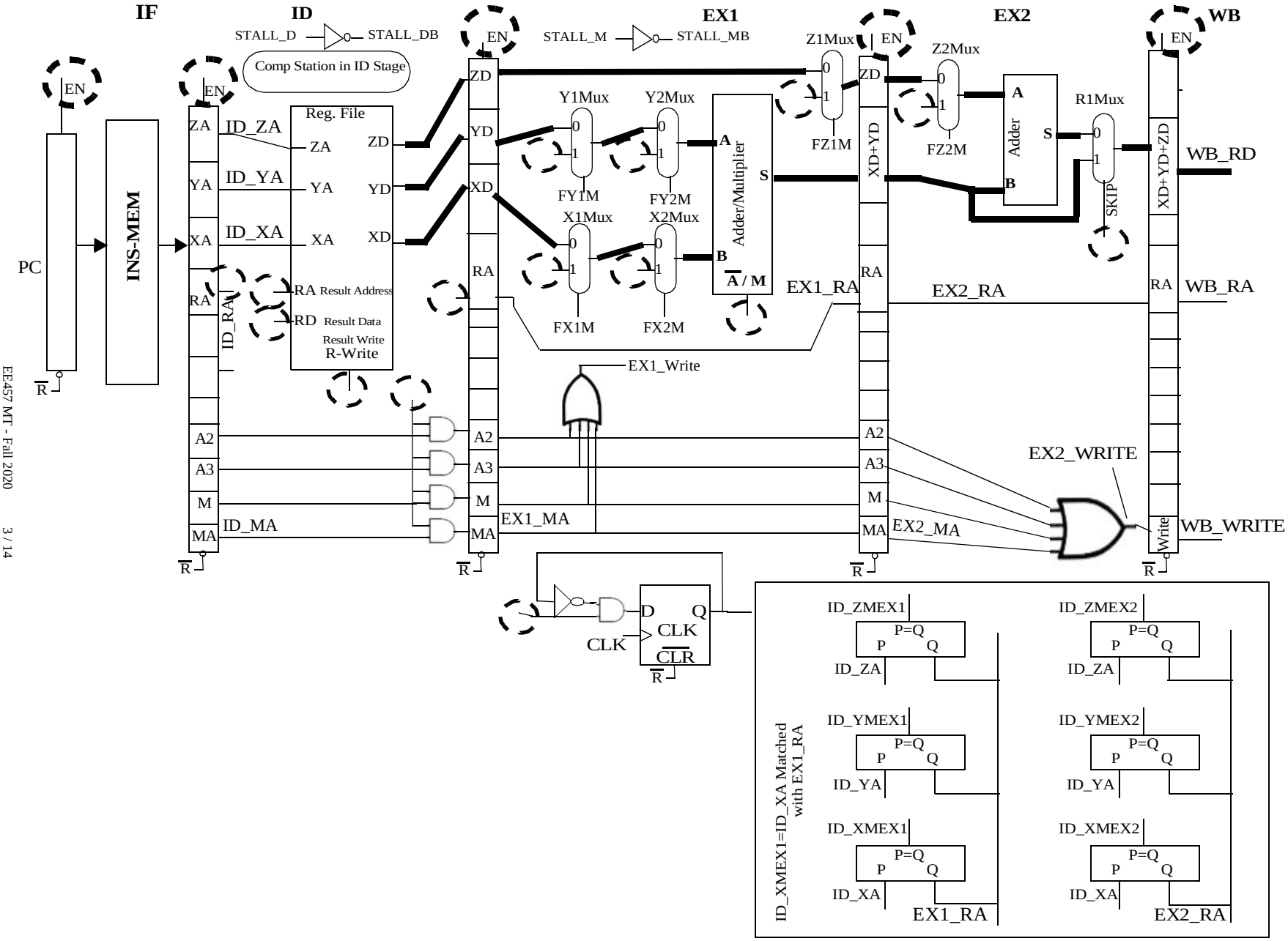

ID_XMEX0
ID_YMEX0
ID_RUN
EX0_RUN
STALL
New design with dummy EX0 stage
Note: Senior #1 is in EX0 in this new design. If the junior is dependent on his senior #1 for X or Y, he should be stalled for one clock.

Q#1 Fall 2020 MT

The 3-element adder with a multiplier and a bypass around the 2nd adder

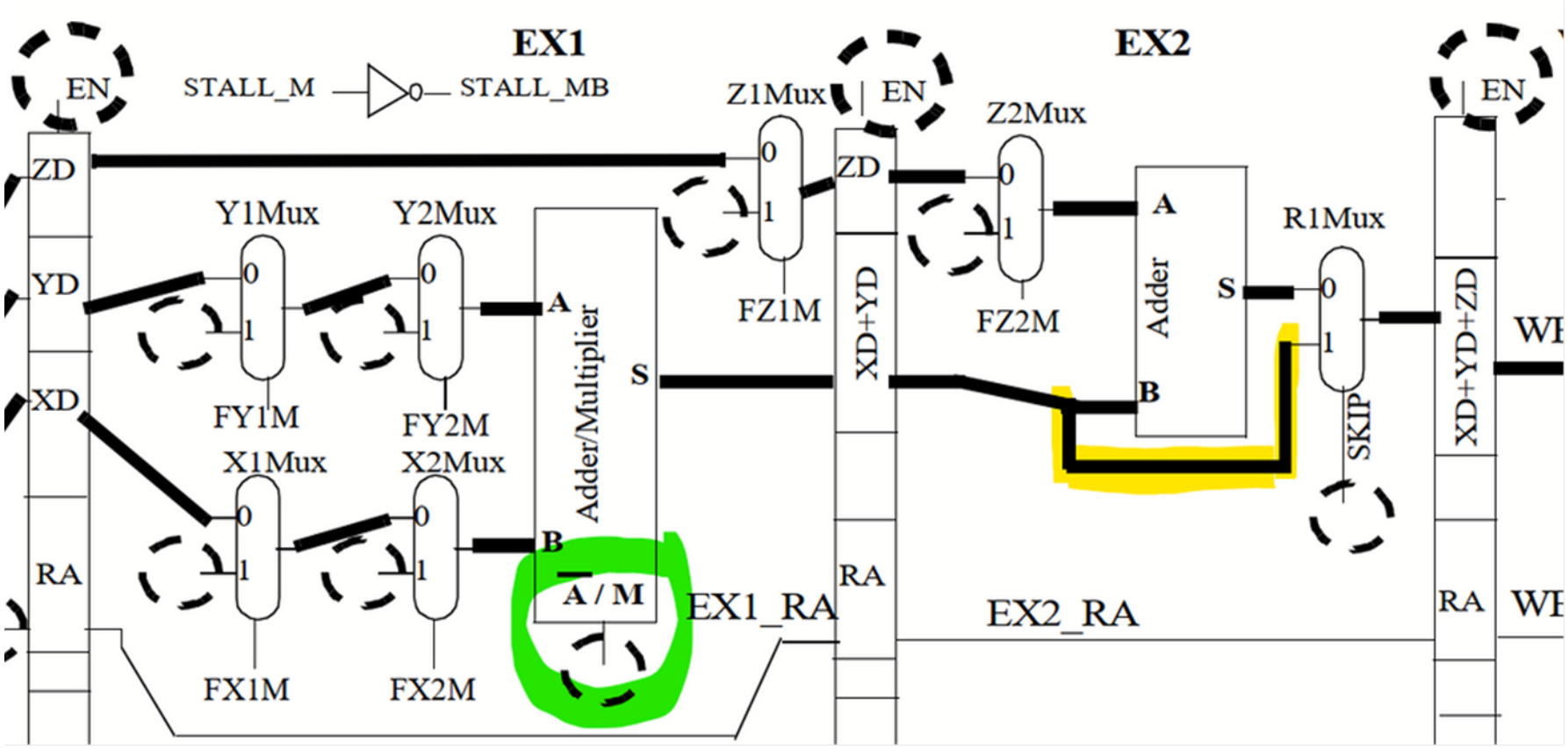


Q#1 Fall 2020 MT



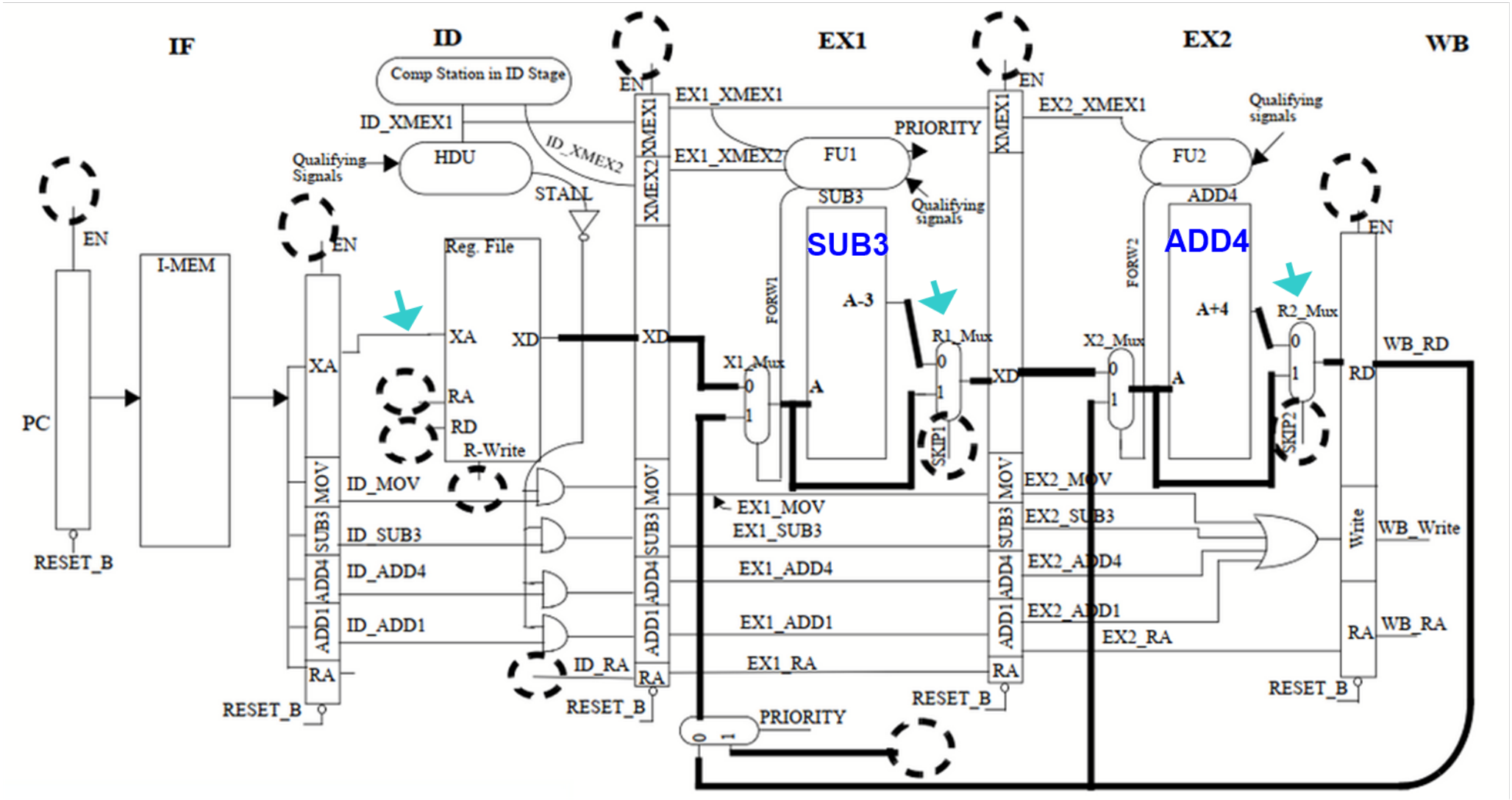
Q#1 F2020 MT

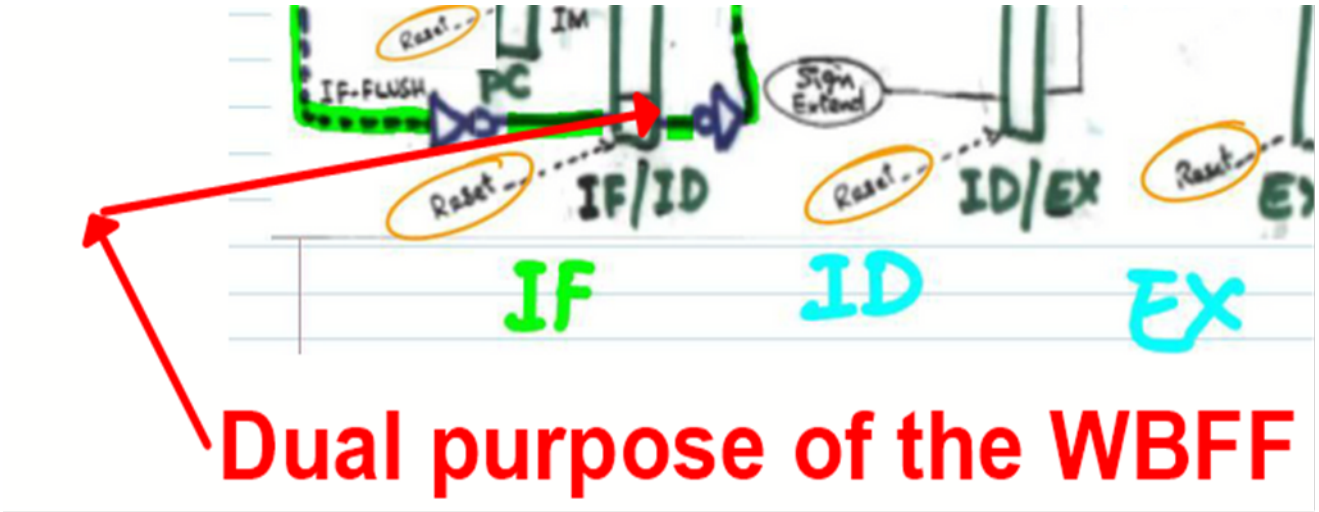
Instruction	Operation	Opcode			3
		MA	**M**	**A3**	**A2**
NOP		0	0	0	0
A2 $R, $X, $Y;	($R) <= ($X)+($Y)	0	0	0	**1**
A3 $R, $X, $Y, $Z;	($R) <= ($X)+($Y)+($Z)	0	0	**1**	0
M $R, $X, $Y;	($R) <= ($X)*($Y)	0	**1**	0	0
MA $R, $X, $Y, $Z;	($R) <= ($X)*($Y)+($Z)	**1**	0	0	0

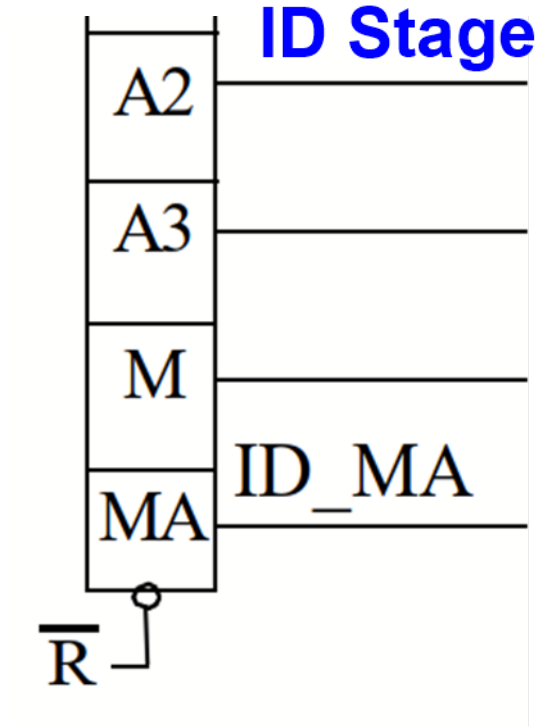


Bypassing stage logic is used in Lab 7 Part 3

NOP	
MOV $R, $X;	($R) <= ($X)
SUB3 $R, $X;	($R) <= ($X) - 3
ADD4 $R, $X;	($R) <= ($X) + 4
ADD1 $R, $X;	($R) <= ($X) + 1





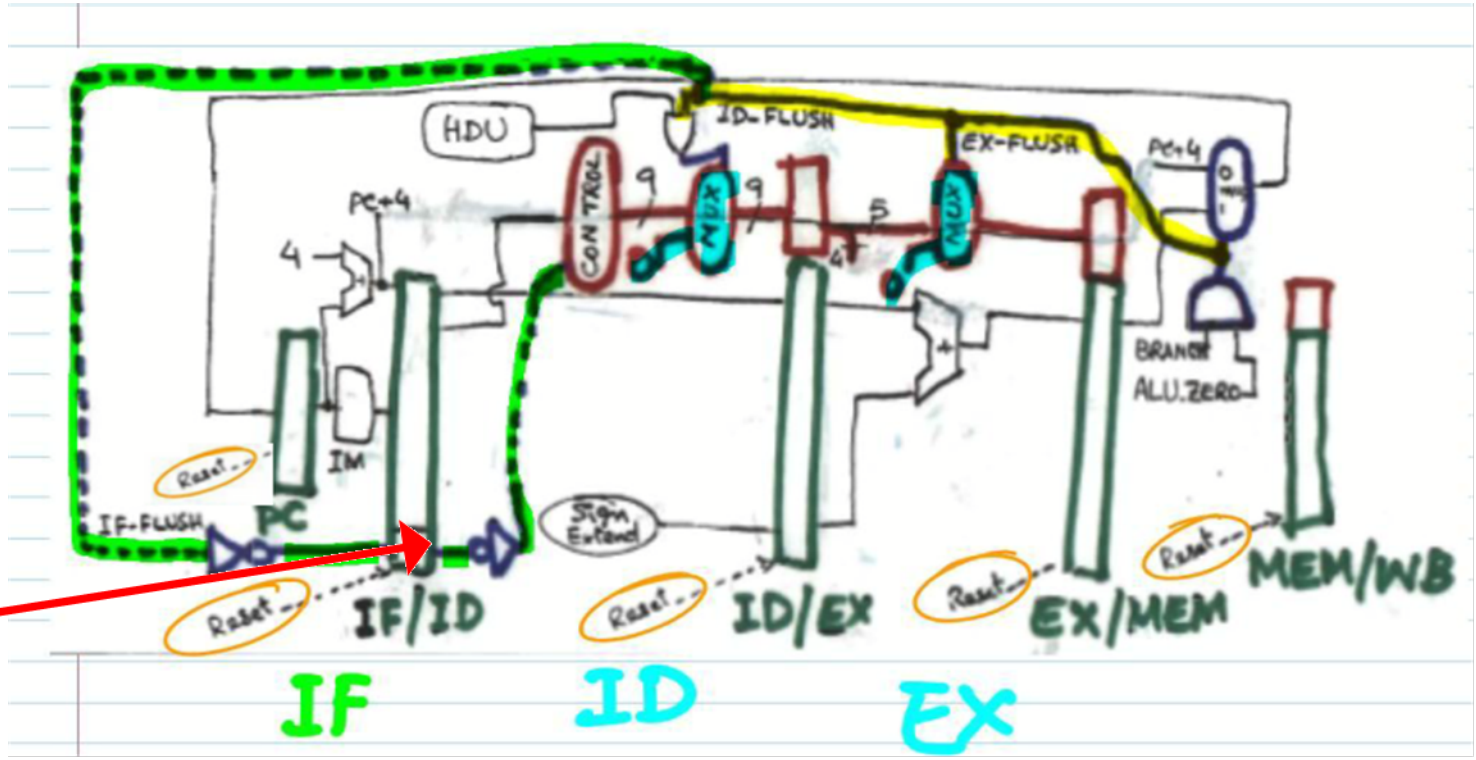


1.1. In our 5-stage early branch pipeline, if ISA declares one branch delay slot, then a successful branch in **ID** stage _____ (does/doesn't) flush its junior in **IF** stage. _____ (Hence/Even then) you _____ (need/do not need) a **WBFF** on **IF/ID** stage register. When we switch-on power, the random instruction in the ID stage is converted to a bubble due to **asynchronously setting or resetting this WBFF and using that information to tell the CU (control unit) that, the random instruction is the ID stage is "destined to be flushed". Then the control unit ignores the opcode of the random instruction and generates 9 zeros for this instruction.**

In the modified Lab 7 design here, we _____ (do/don't) need a **WBFF** on **IF/ID** stage register to convert the power-on random instruction in the **ID** stage. Explain:

Here, the opcode is one-hot coded. It is like the opcode is already decoded into individual control signals, A2, A3, M, and MA. So, NOP is all zeros. Hence we are able to simply clear the IF/ID, ID/EX1, EX1/EX2 on reset to get rid of the RANDOM INSTRUCTIONS in ID, EX1, EX2, and WB stages. Hence we do not need any WBFF here!

On power-on reset, you want to make sure that there are no RANDOM instructions in the pipeline.



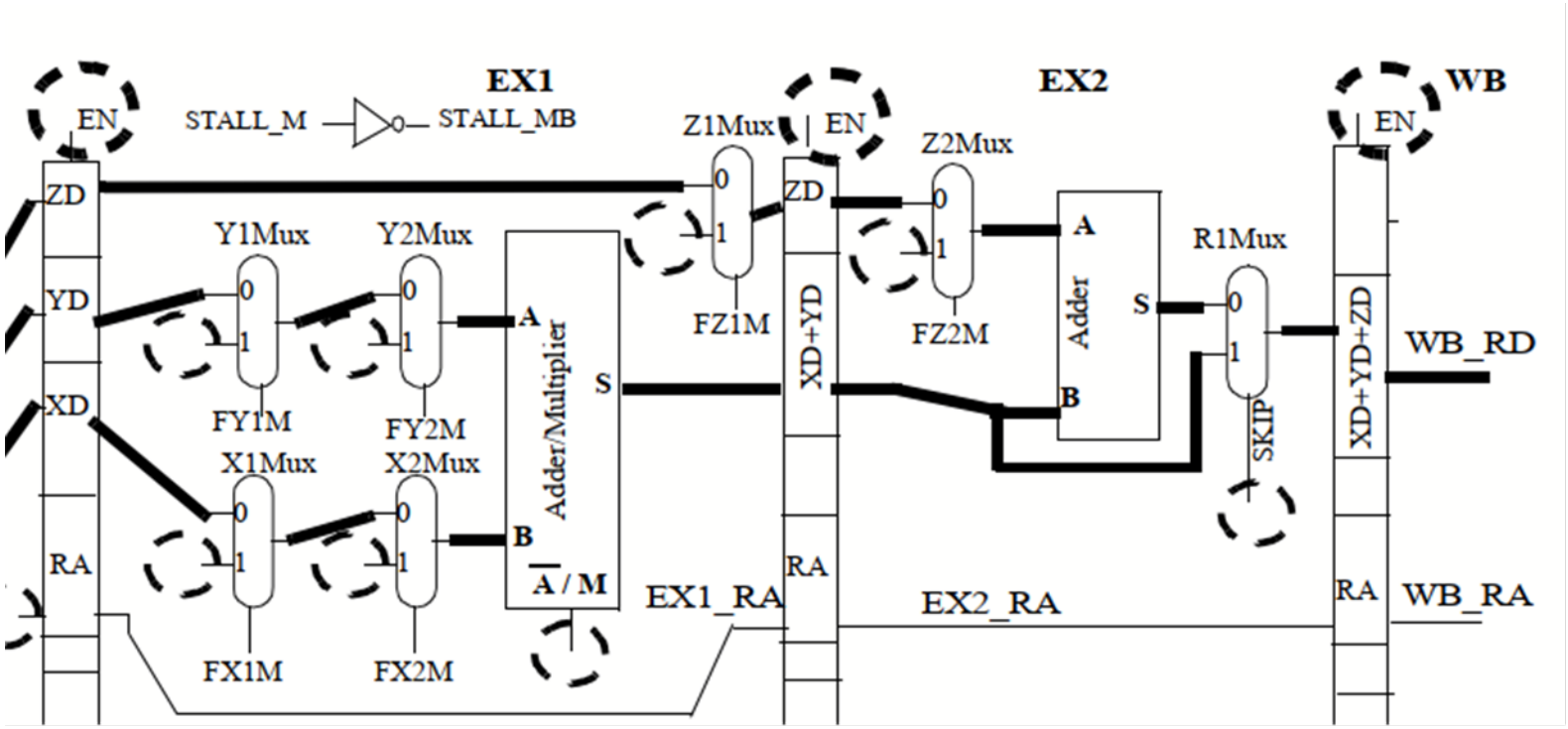
Dual purpose of the WBFF in Lab 6 Part 4

Why don't we need a WBFF in Lab 7 Part 1?

Answer: No opcode here. Single-bit active-high RUN signal! So, no difficulty in converting the RANDOM instr in ID stage to a bubble. There is no branch instruction to flush the IF stage fall-through junior through wrist banding. Even if there is a branch instruction, we can inject a bubble into ID from the IF stage to flush the IF junior!

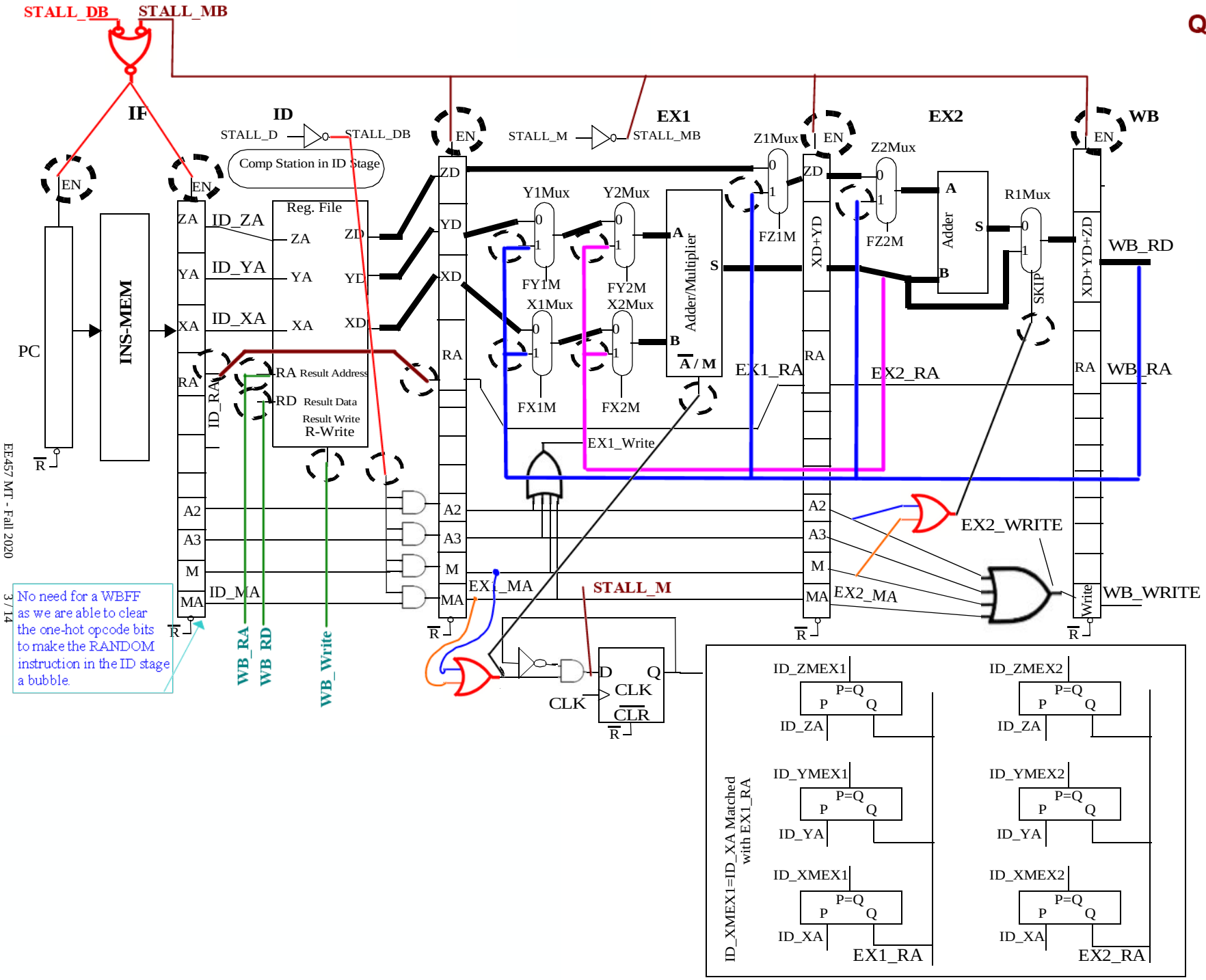
_____ (Like/Unlike) the **X** and **Y** operands, which need *two pairs* of forwarding muxes in **EX1**, the **Z** operand _____ (needs/doesn't need) *a pair* of forwarding muxes in **EX1**. Explain

Unlike the X and Y, the Z does not participate in any computation in the EX1 stage. So, unlike the X and Y, Z will not stall in the ID stage, even if his immediate senior (S#1) is a A3 or MA who cannot help from EX2. So, Z decides to receive help from his senior #1 in the next clock, when Z is in EX2 and his senior #1 is in WB stage. Hence the second mux of Z (Z2Mux) is in EX2 unlike X2Mux and Y2 Mux which are in EX1.



Z has the luxury of waiting until it reaches EX2.

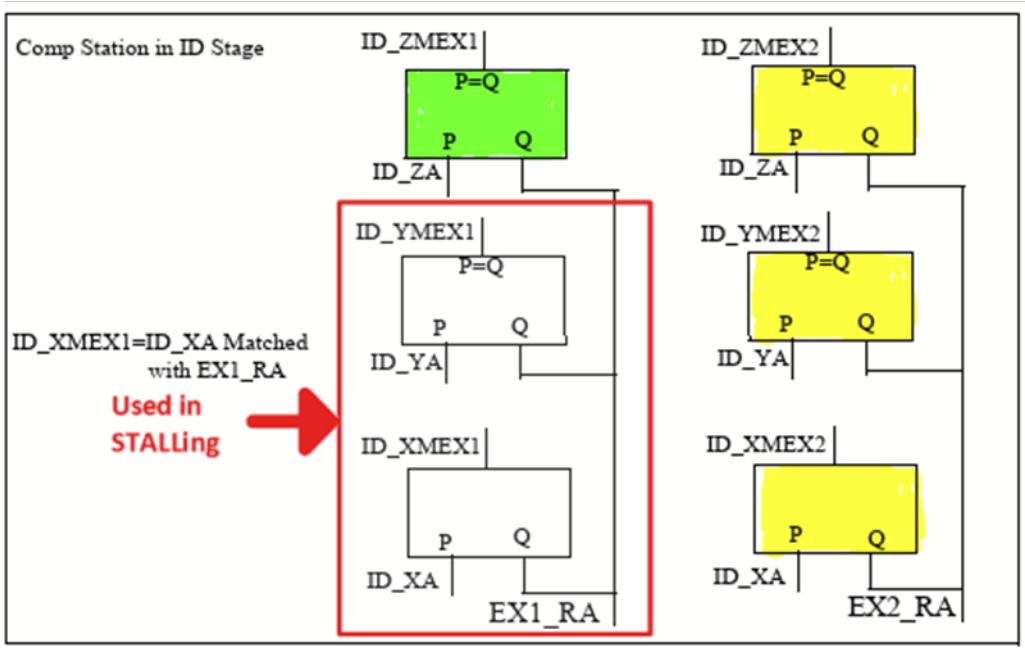
Q#1 F2020 MT


STALL_DB
STALL_MB
IF
ID
EX1
EX2
WB
STALL_D
STALL_DB
STALL_M
STALL_MB
Comp Station in ID Stage
EN
PC
INS-MEM
R
ZA
YA
XA
RA
A2
A3
M
MA
ID_ZA
ID_YA
ID_XA
ID_RA
ID_MA
Reg. File
RA Result Address
RD Result Data
Result Write
R-Write
ZD
YD
XD
Y1Mux
Y2Mux
FY1M
FY2M
X1Mux
X2Mux
FX1M
FX2M
Adder/Multiplier
A / M
S
Z1Mux
FZ1M
Z2Mux
FZ2M
Adder
A
B
R1Mux
SKIP
XD+YD
XD+YD+ZD
EX1_RA
EX2_RA
WB_RD
WB_RA
EX1_Write
EX2_WRITE
EX1_MA
EX2_MA
STALL_M
WB_WRITE
Write
WB_RA
WB_RD
WB_Write
CLK
D
Q
CLR
No need for a WBFF as we are able to clear the one-hot opcode bits to make the RANDOM instruction in the ID stage a bubble.
ID_XMEX1=ID_XA Matched with EX1_RA
ID_ZMEX1
ID_ZMEX2
ID_YMEX1
ID_YMEX2
ID_XMEX1
ID_XMEX2
P=Q
P
Q
ID_ZA
ID_YA
ID_XA
EX1_RA
EX2_RA

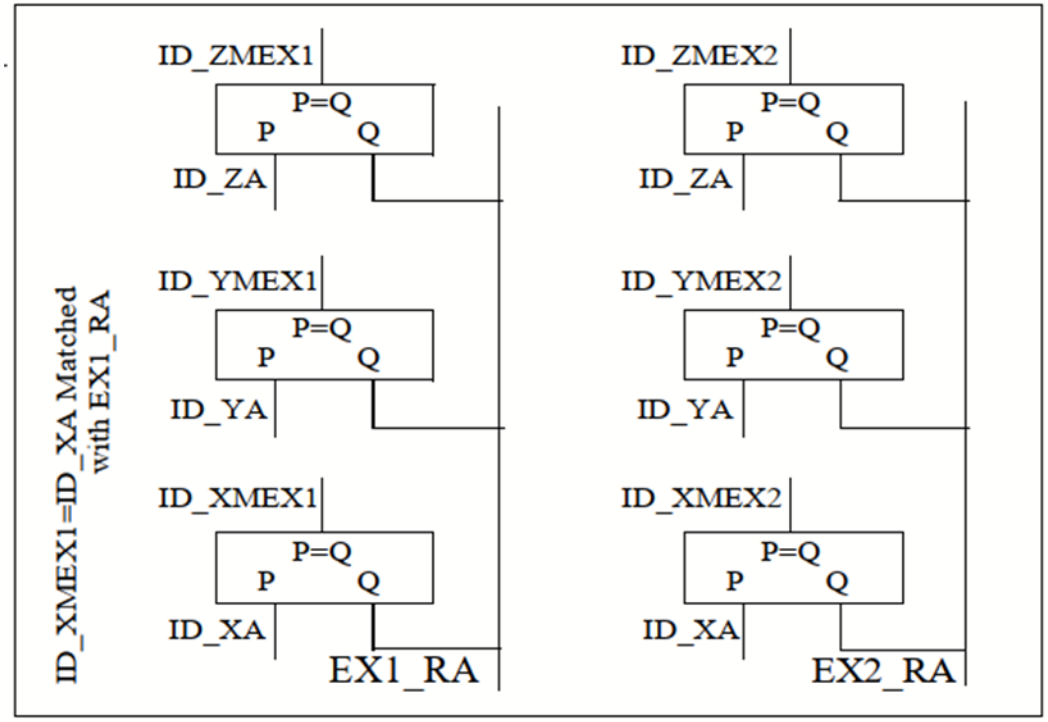
_____ (Like / Unlike) in Lab 7 P1, the comparison units used for stalling and the comparison units used for forwarding _____ (are / aren't) in disjoint groups

Consider different types of seniors: some agreeing to help in time and some asking you to stall.

Lab 7 P1

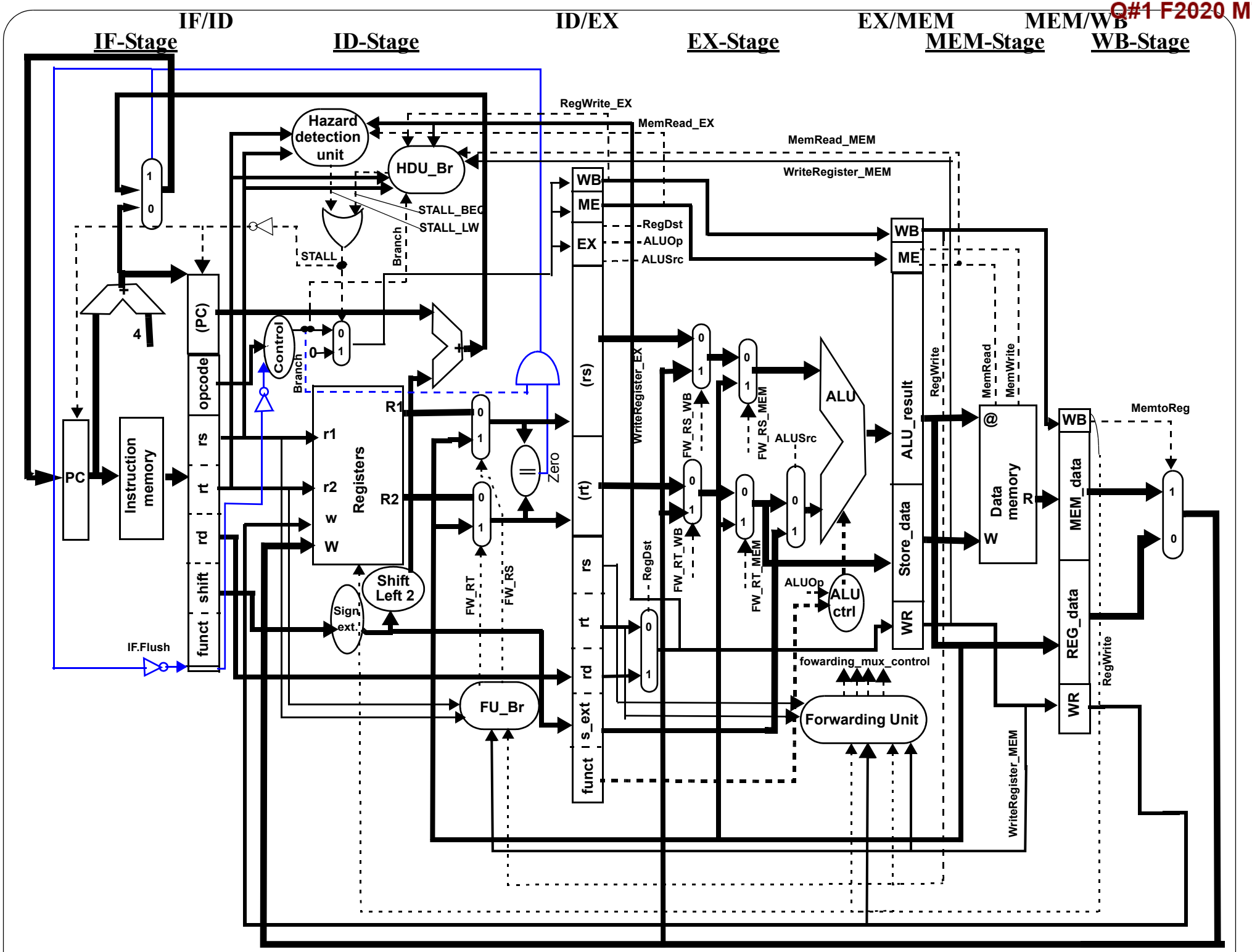


Q#1 Fall 2020 MT



How about Lab 6 Par 4 early branch?

The pair of comparison units in the FU_Br are also in HDU_Br. A branch instr. in ID stage can get forwarding help from R-type senior #2 in MEM stage but he has to stall if that senior #2 in MEM is a LW instr. There are other such cases. Please see Lab 6 Part 5.



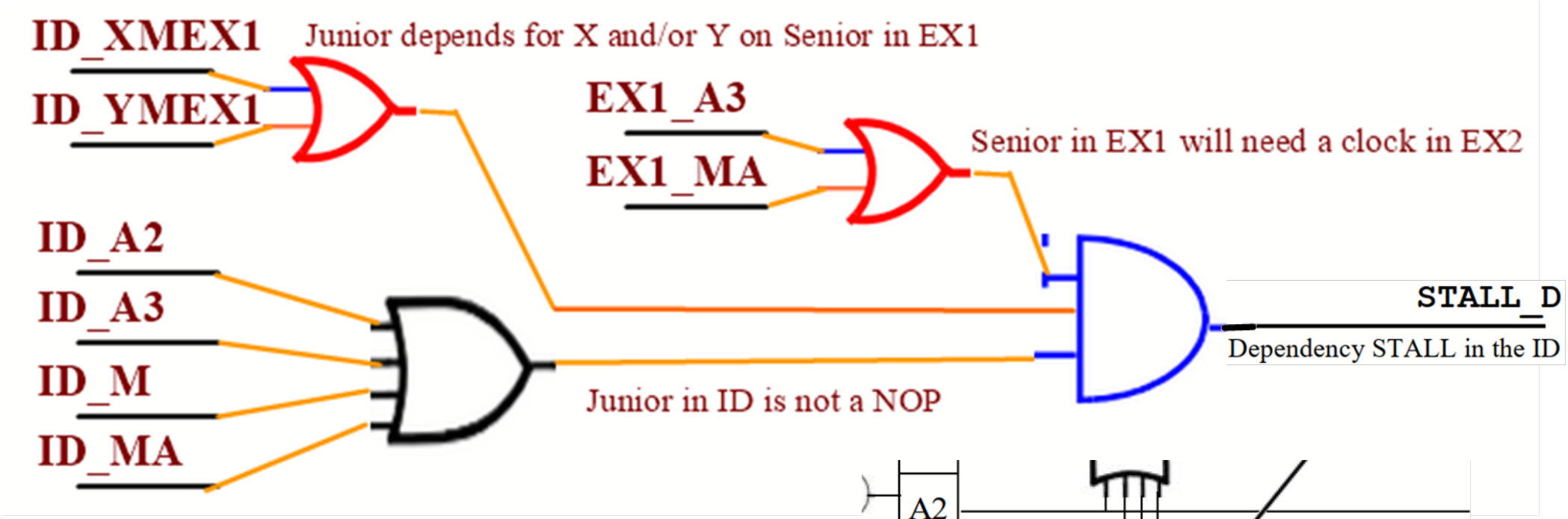
10/18/06

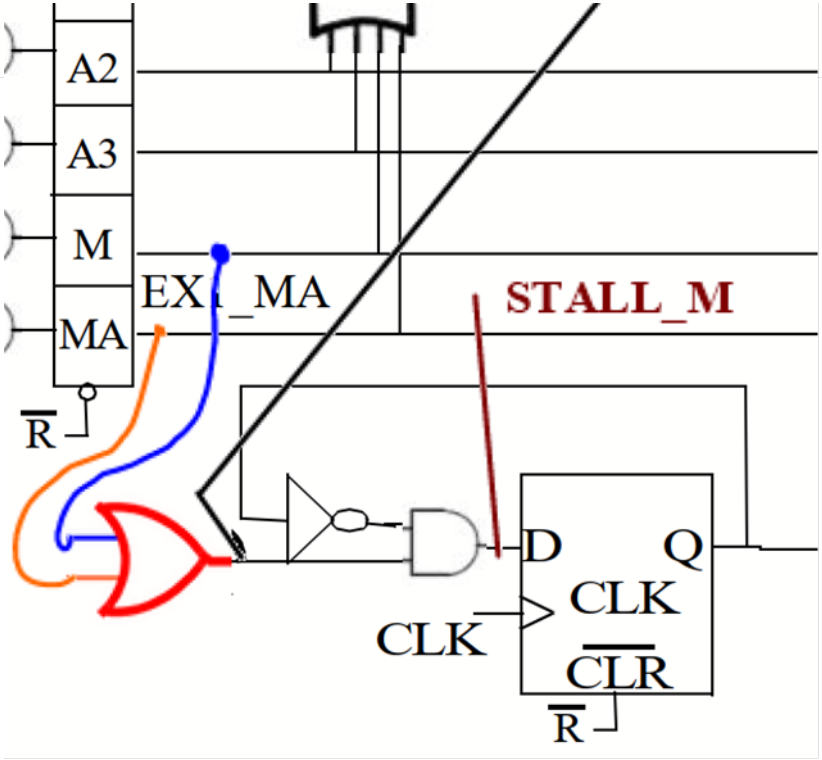
Detailed implementation of Early Branch suggested in 3rd Ed.

Drawn by: Wei-jen Hsu

Designed by: Gandhi Puvvada

STALL_D and STALL_M





A stall in EX stage or beyond results in stalling the entire pipe!

DCM (Data Cache Miss) in the early or the late branch designs of the CPU.

Stall initiated by MULTIPLY operation in EX1 in this Q#1 of Fall 2020 MT.

If you do not stall the entire pipe, any forwarding help from the WB stage senior to the junior in EX stage would be lost :(

But the senior in WB has already helped the junior in EX stage, so why do we need to stall the entire pipe?

Well, combinational logic in EX stage cannot hold (store) the help received in a prior clock!

Time-Space Diagram

Complete the Time-Space diagram below. If there are excess rows in the table, just leave them blank. If there aren't enough rows to complete the given sequence, stop at the last row. Write "**Bubble**" or draw a bubble when needed.

instr1: MA#1	MA $4, $3, $2, $1
instr2: M#2	M $4, $4, $4
instr3: A3#3	A3 $7, $6, $5, $4
instr4: A2#4	A2 $8, $7, $4
instr5: M#5	M $4, $8, $7

CC# = Clock Cycle #

CC#	IF	ID	EX1	EX2	WB
CC1	MA#1				
CC2	M#2	MA#1			
CC3	A3#3	M#2	MA#1		
CC4	A3#3	M#2	MA#1		
CC5	A3#3	M#2	Bubble	MA#1	
CC6	A2#4	A3#3	M#2	Bubble	MA#1
CC7	A2#4	A3#3	M#2	Bubble	MA#1
CC8	M#5	A2#4	A3#3	M#2	Bubble
CC9	M#5	A2#4	Bubble	A3#3	M#2
CC10		M#5	A2#4	Bubble	A3#3
CC11			M#5	A2#4	Bubble
CC12			M#5	A2#4	Bubble

I am trying to check to see that students are careful not to keep the STALL_M signal active in the 2nd clock when a MA or M is preparing to leave EX1.

In what clocks (write CC numbers) did **STALL_D** go active? CC3, CC4, due to M#2's dependency on MA#1 CC8 due to A2#4's dependency on A3#3

In what subsequent clocks did **STALL_D** go inactive? CC5 CC9

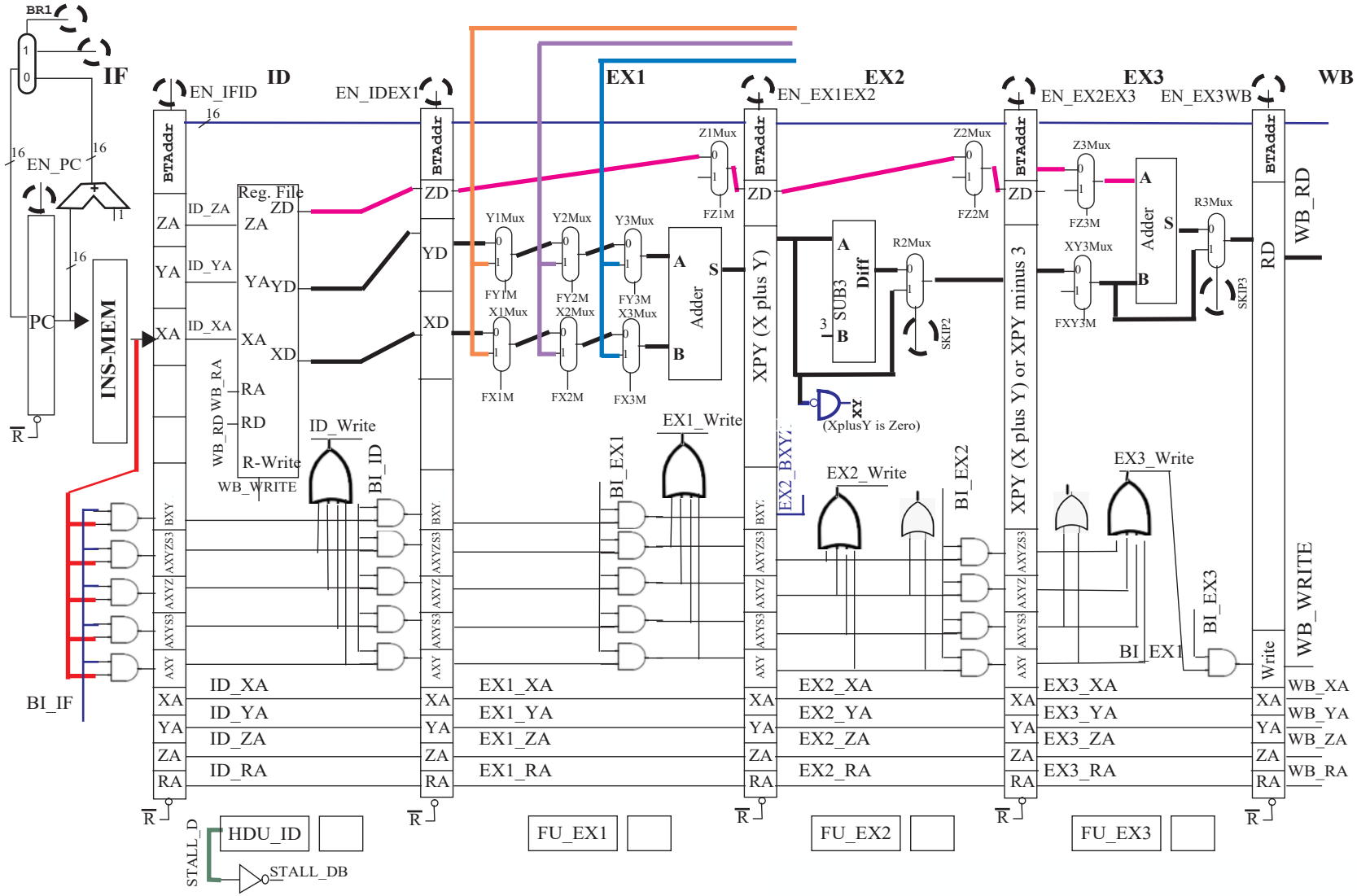
In what clocks (write CC numbers) did **STALL_M** go active? CC3 for MA#1 CC6 for M#2 CC11 for M#5

In what subsequent clocks did **STALL_M** go inactive? CC4 CC7 CC12

Q#1 Spring 2021 MT

The 3-element adder

with a SUB3 and a BXY

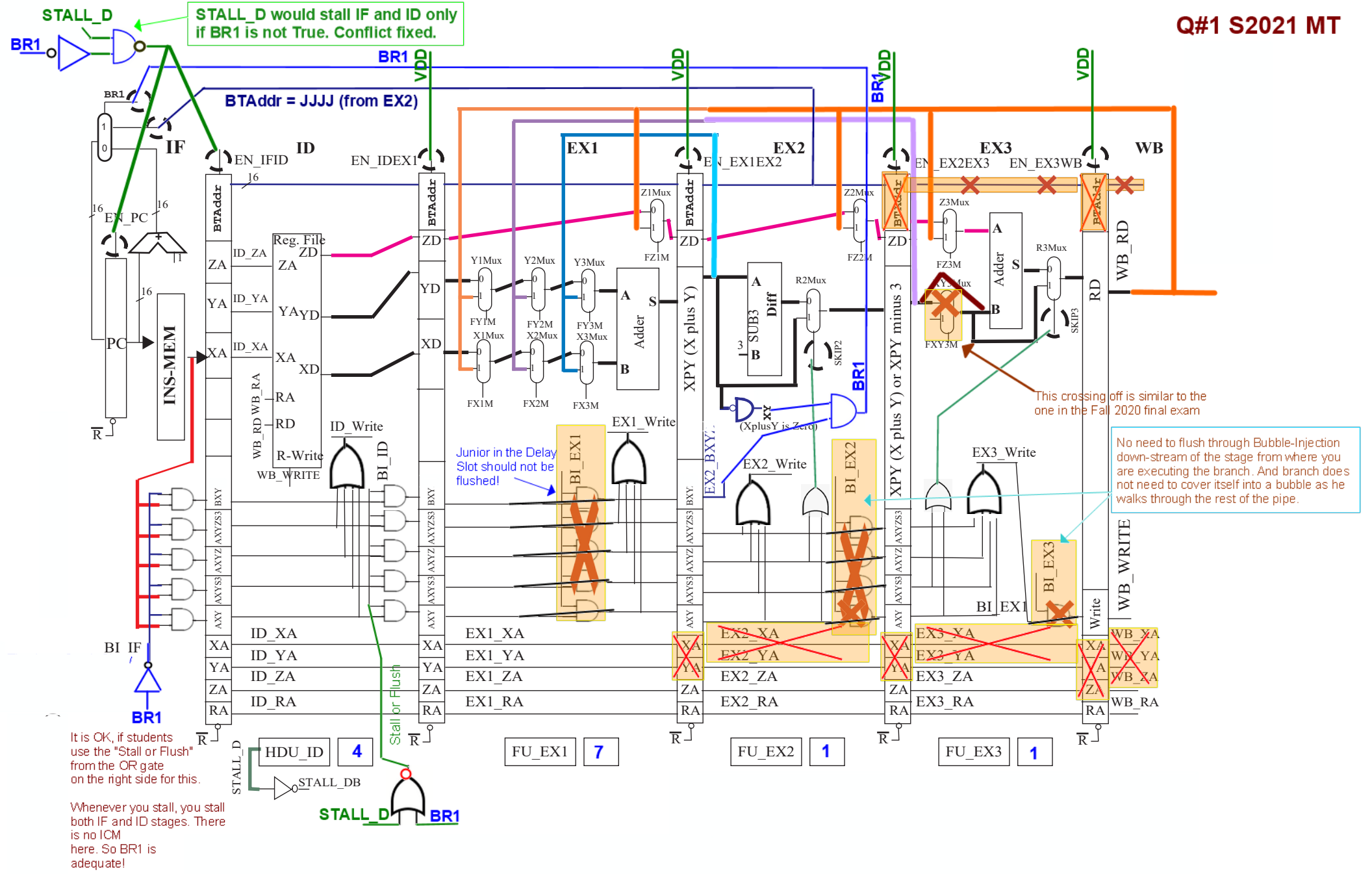


Separate HDU and FUs. So, instructions need to bring their source operands to later stages to compare with the destinations of their seniors.

Count of comparison units in boxes next to the HDU and FUs.

Instruction		Operation
NOP		
AXY	$R, $X, $Y;	($R) <= ($X)+($Y)
AXYS3	$R, $X, $Y;	($R) <= ($X)+($Y)-3
AXYZ	$R, $X, $Y, $Z;	($R) <= ($X)+($Y)+($Z)
AXYZS3	$R, $X, $Y, $Z;	($R) <= ($X)+($Y)-3+($Z)
BXY:	$X, $Y, JJJJ;	(PC) <= JJJJ if (XplusY == 0)

BXY has one delay slot
= do not flush Junior #1 even if BXY is successful
= do not initiate bubble-injection
from the Junior #1's stage


Q#1 S2021 MT
STALL_D
BR1
STALL_D would stall IF and ID only if BR1 is not True. Conflict fixed.
BTAddr = JJJJ (from EX2)
IF
ID
EX1
EX2
EX3
WB
EN_IFID
EN_IDEX1
EN_EX1EX2
EN_EX2EX3
EN_EX3WB
EN_PC
PC
INS-MEM
Reg. File
R-Write
WB_WRITE
ID_Write
BI_ID
EX1_Write
EX2_Write
EX3_Write
BI_EX1
BI_EX2
BI_EX3
BI_IF
Junior in the Delay Slot should not be flushed!
This crossing off is similar to the one in the Fall 2020 final exam
No need to flush through Bubble-Injection down-stream of the stage from where you are executing the branch. And branch does not need to cover itself into a bubble as he walks through the rest of the pipe.
XPY (X plus Y)
XPY (X plus Y) or XPY minus 3
SUB3
Diff
Adder
(XplusY is Zero)
Stall or Flush
HDU_ID
4
FU_EX1
7
FU_EX2
1
FU_EX3
1
STALL_DB
It is OK, if students use the "Stall or Flush" from the OR gate on the right side for this.
Whenever you stall, you stall both IF and ID stages. There is no ICM here. So BR1 is adequate!

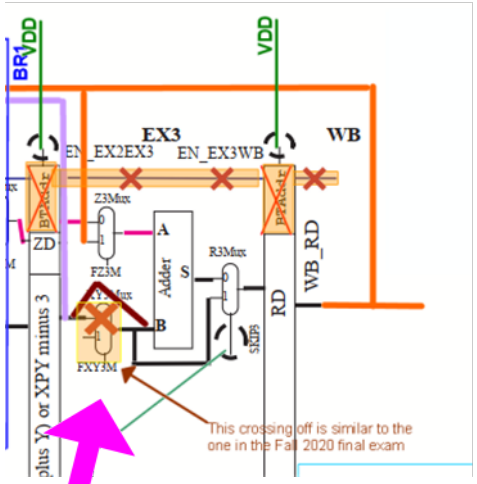
Write a number in each of the 4 small boxes indicating the number of register ID comparison units on the next page and transcribe them here. $0 is not a special register here.

HDU_ID	4	FU_EX1	7	FU_EX1	1	FU_EX1	1
	4+7+1+1 = 13			FU_EX2		FU_EX3	

Now arrive at the bare minimum number of comparison units we would need if we followed our Lab 7 Part 1 method of pooling all comparison units in a comparison station in ID stage.

comp station in ID stage | 9 | The 4 comparison units in the HDU are replicated in the FU_EX1.

Do not help an instruction in the result path!
You need to help in its source path!



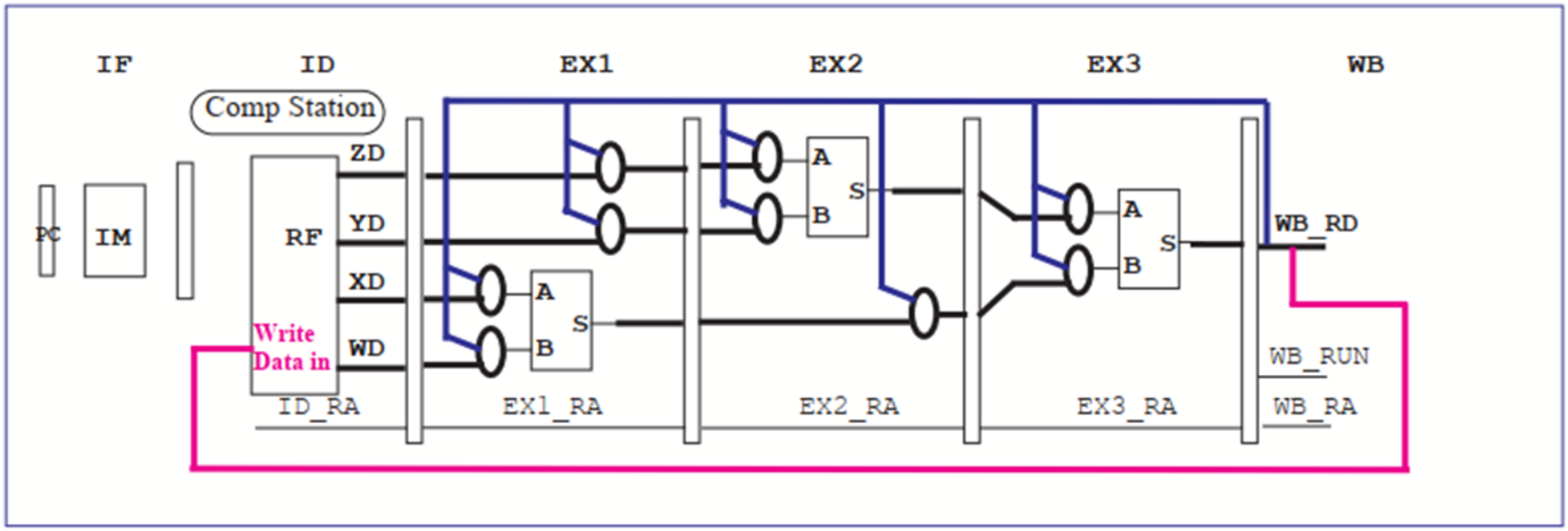
Q#1 Fall 2021 MT

Mr. Bruin created a 4-element adder similar to the Fall 2018 Question but it is neither a tree not a linear cascade!

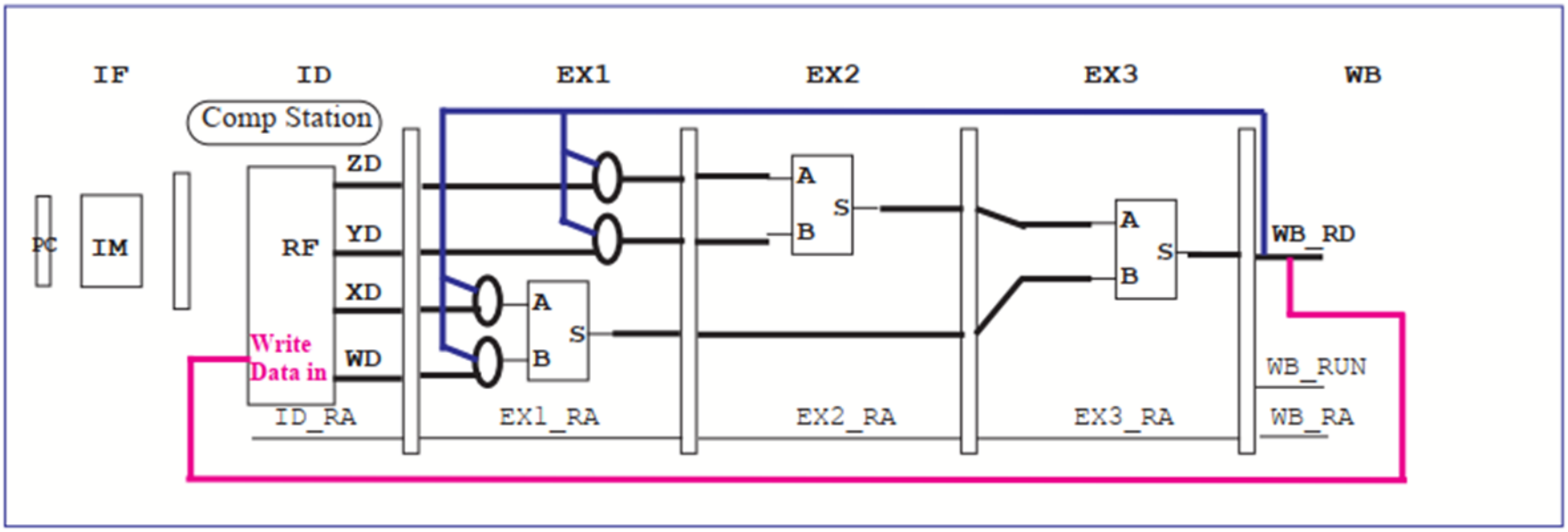
See the MOV instruction support!

1.1 Mr. Bruin and Miss Bruin are the TAs for a similar course at the other school. They looked at our above Fall 2018 midterm question and solution. They wanted to design a midterm question for their course by combining the above two designs in the following fashion. Note that they are adding (W, X) in EX1 and (Y, Z) in EX2 and adding the two intermediate sums in the EX3.

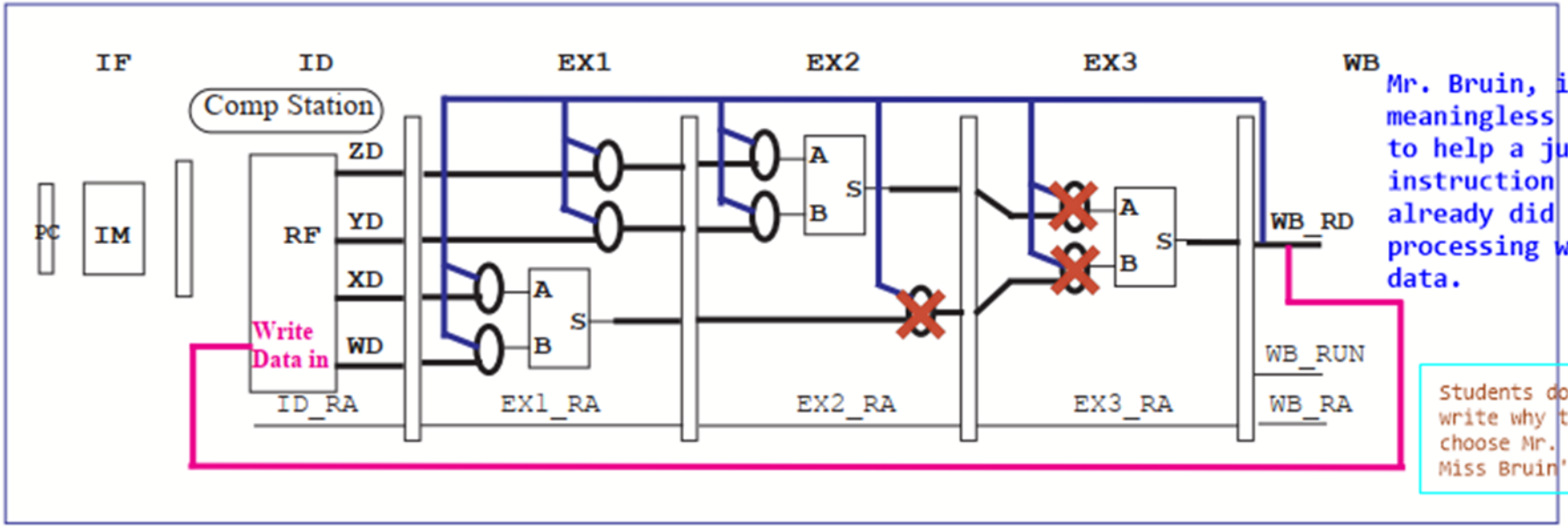
Mr. Bruin's 6-stage solution block diagram:



Miss Bruin told Mr. Bruin that there is no need for any forwarding muxes in the EX2 and EX3 stages. Her design is given below:



Mr. Bruin's 6-stage solution block diagram:

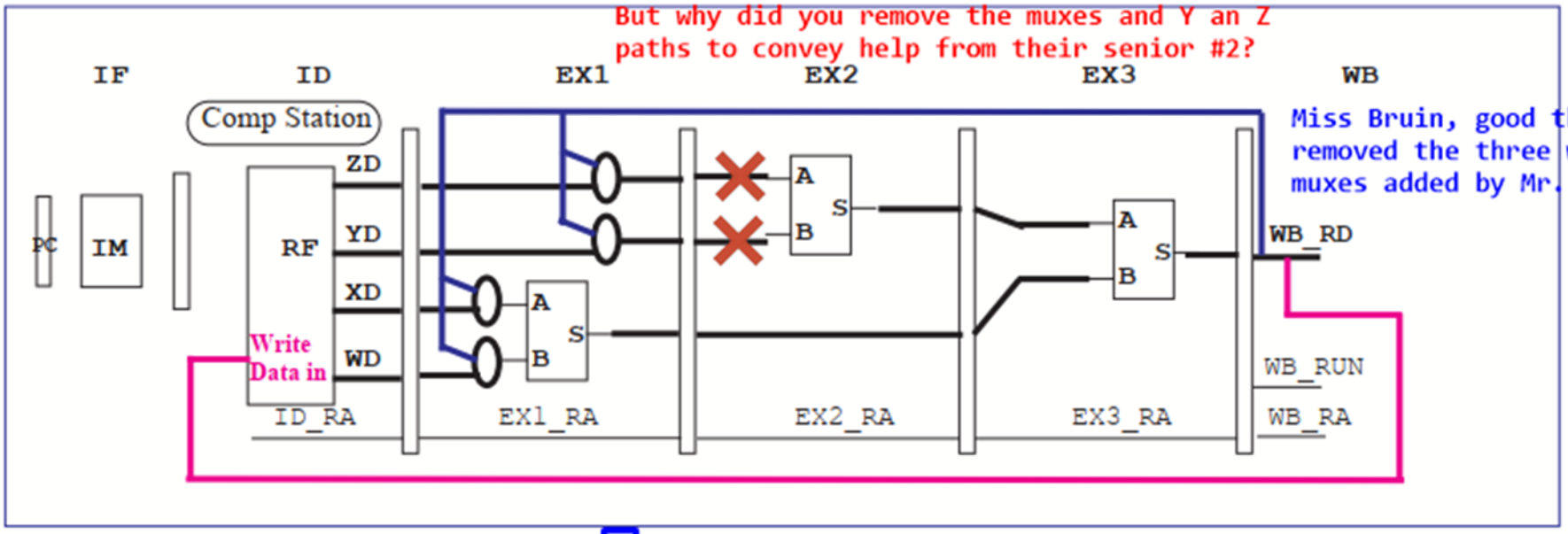


Mr. Bruin, it is meaningless and wrong to help a junior instruction who has already did some processing with his data.

Students do not need to write why they did not choose Mr. Bruin's or Miss Bruin's design.

Miss Bruin told Mr. Bruin that there is no need for any forwarding muxes in the EX2 and EX3 stages. Her design is given below:

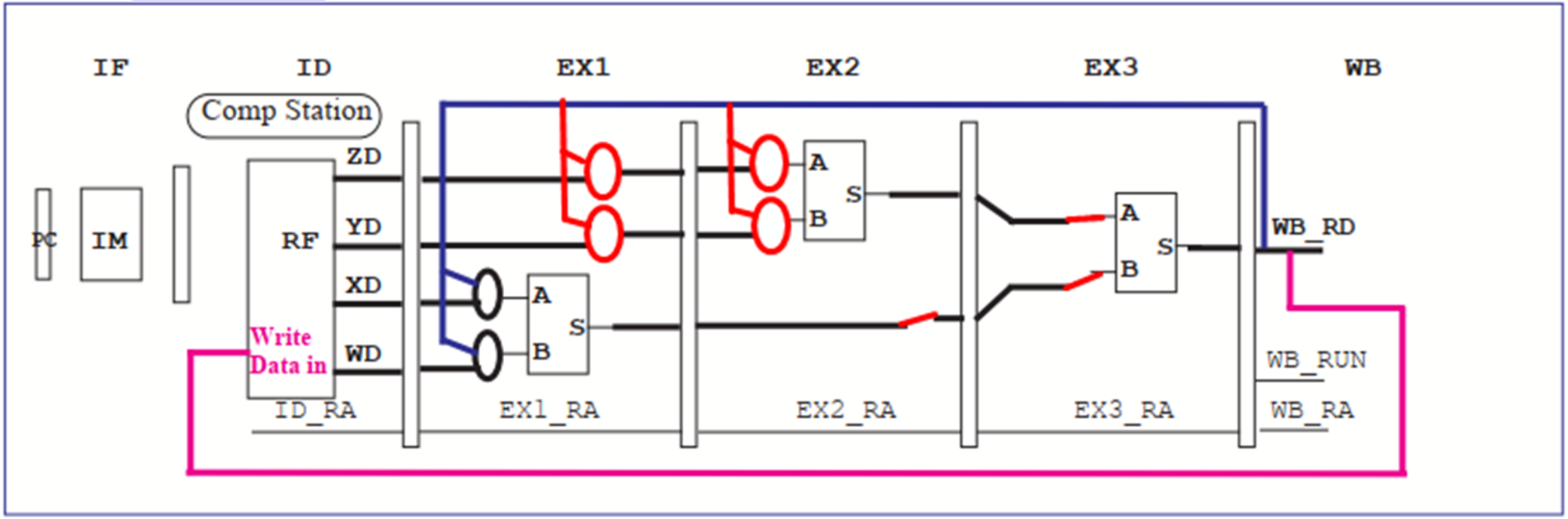
But why did you remove the muxes and Y an Z paths to convey help from their senior #2?



Miss Bruin, good that you removed the three wrong muxes added by Mr. Bruin!

You agree with_____ (A/B/C) where A = Mr. Bruin's design, B = Miss Bruin's design, C = neither. If you chose neither, then complete your design below.

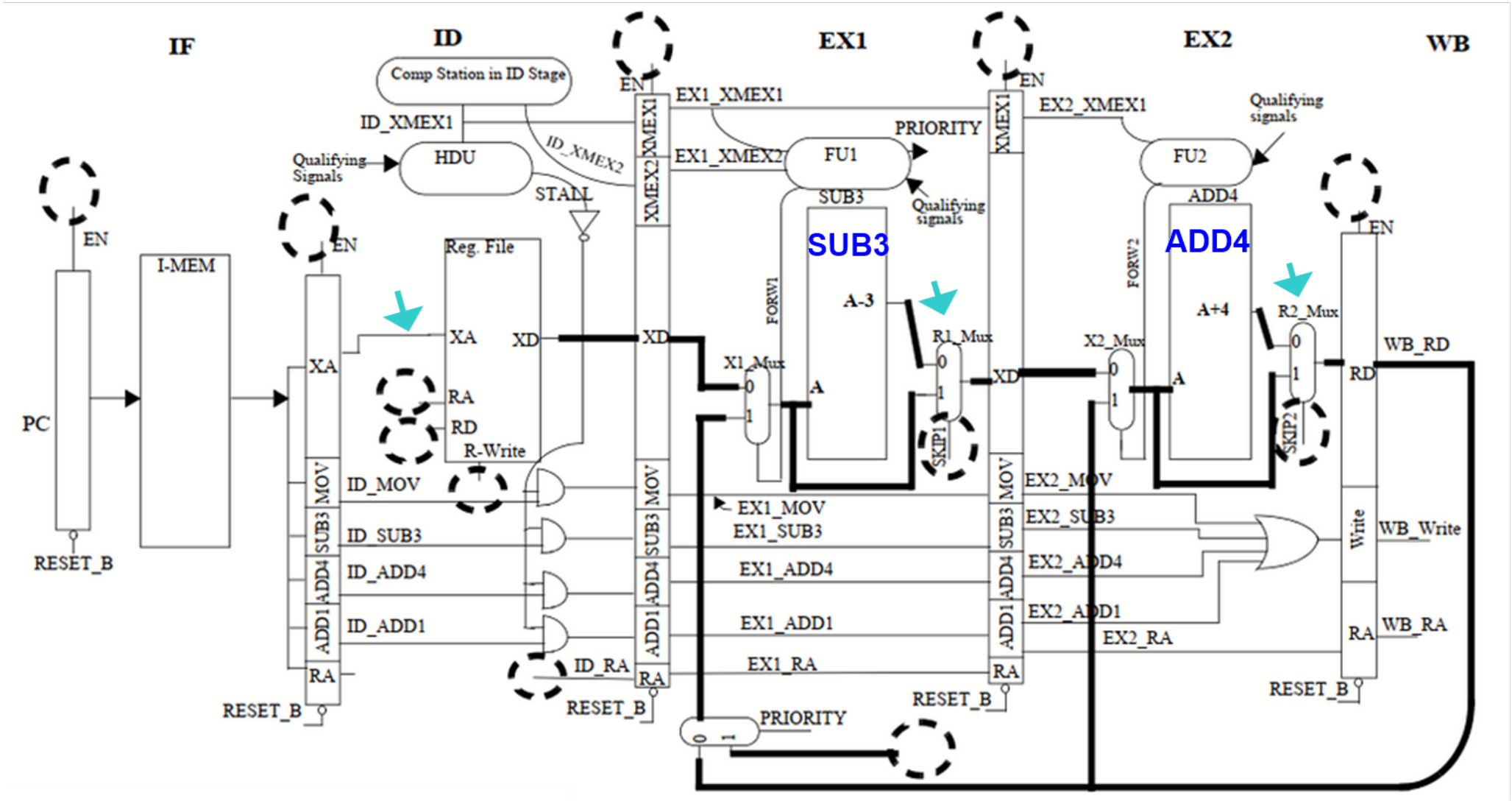
You agree with_____ (A/B/C) where A = Mr. Bruin's design, B = Miss Bruin's design, C = neither. If you chose neither, then complete your design below.



Adding support for a MOV instruction to the Q#1 of Fall 2021 MT

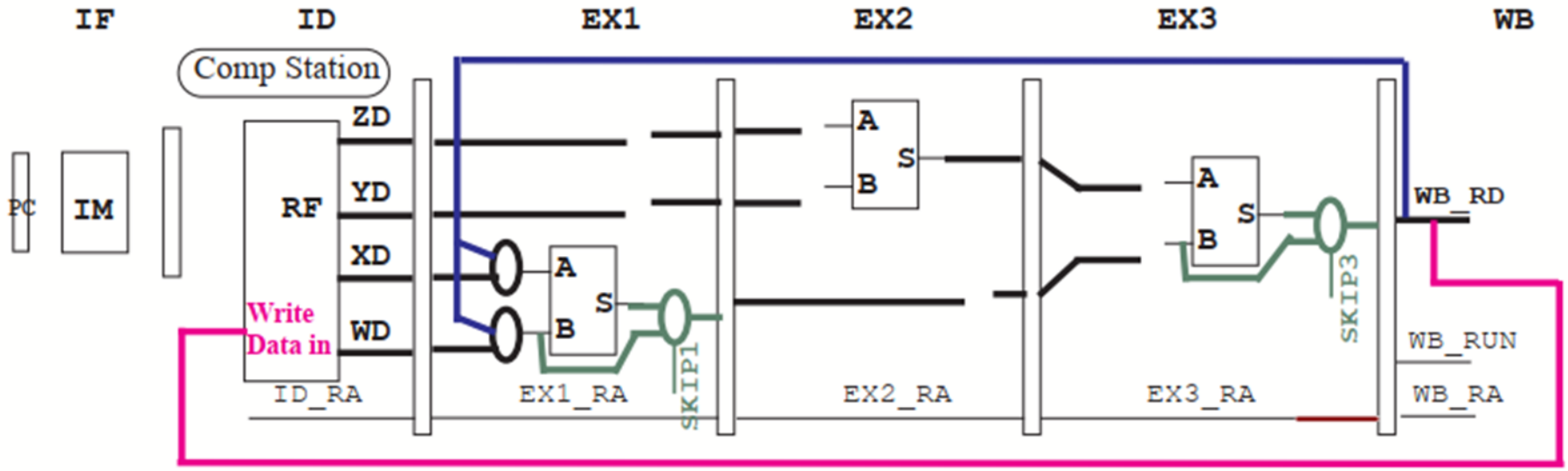
Recollect MOV instruction implementation in Lab7 P3

NOP	
MOV $R, $X;	($R) <= ($X)
SUB3 $R, $X;	($R) <= ($X) - 3
ADD4 $R, $X;	($R) <= ($X) + 4
ADD1 $R, $X;	($R) <= ($X) + 1

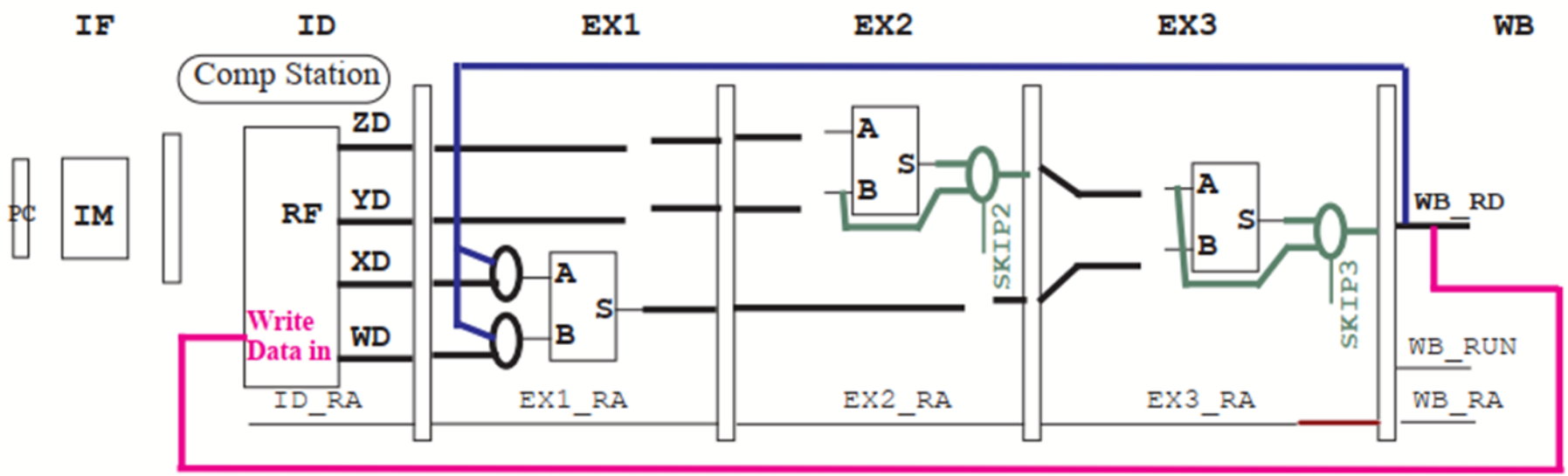


MOV R, W (meaning **R <= W**)
MOV R, Y (meaning **R <= Y**)

MOV R, W (meaning **R <= W**) Your base design + 2 skip-muxes to support a MOV instr



MOV R, Y (meaning **R <= Y**) Your base design + 2 skip-muxes to support a MOV instr



MOV R, W (meaning **R <= W**)
MOV R, Y (meaning **R <= Y**)

MOV does not need to stall at all as it can postpone receiving help in a later stage (as late as EX3).