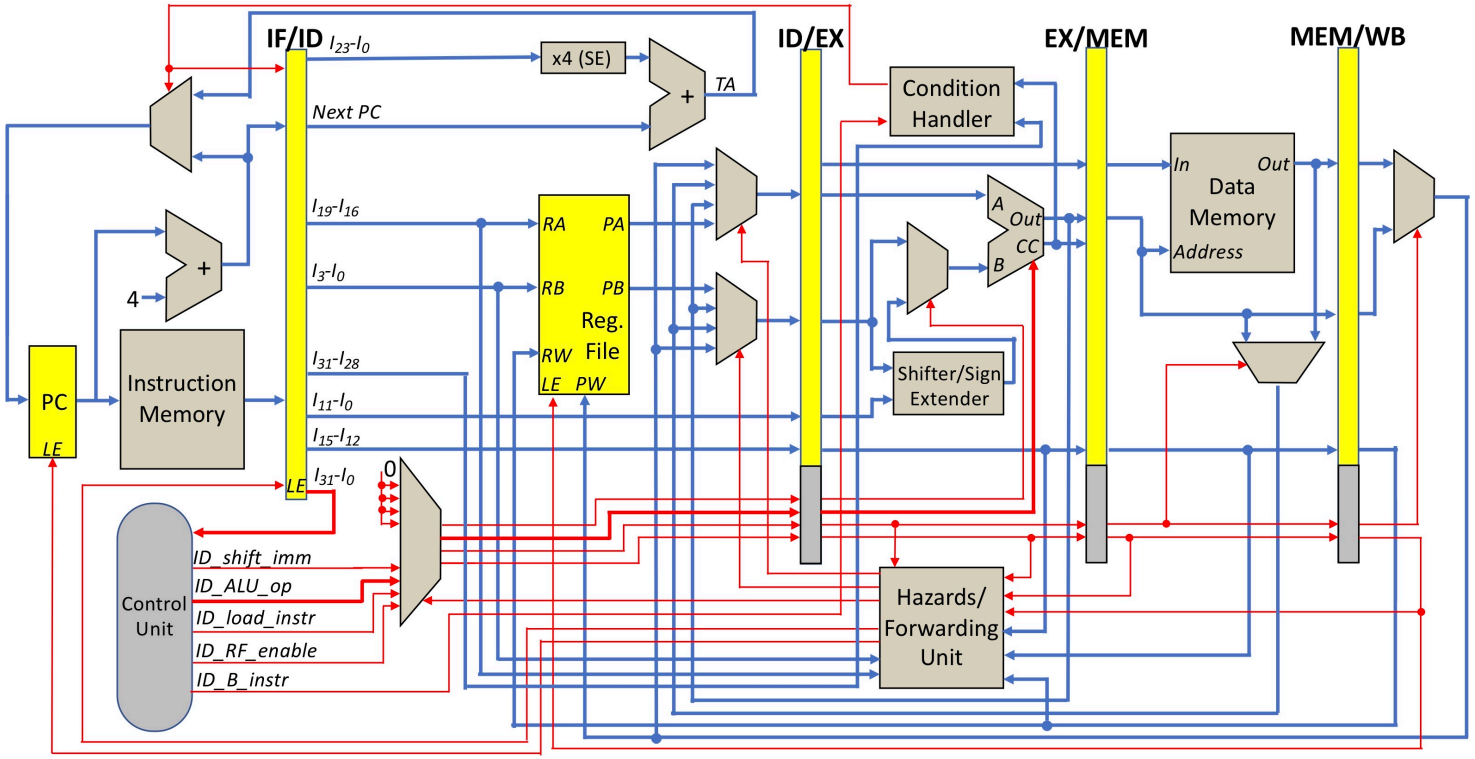
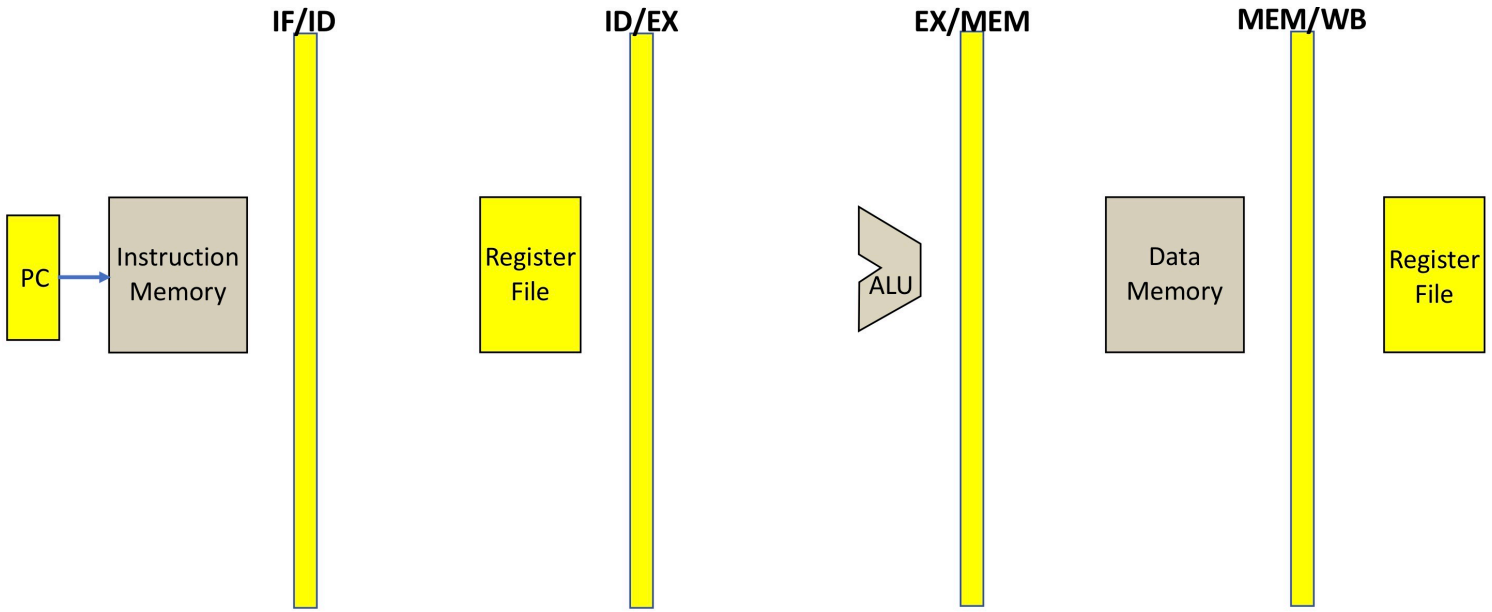
Pipelined Processing Unit



Basic Pipelined Processing Unit



A Warning Note

- The objective of this lesson is to present a methodology for implementing a Pipeline Processing Unit (PPU).
- There will be hidden details omissions that are not strictly relevant for the sake of the understanding of the concepts presented in the lesson.
- However, those details matter when you are implementing a PPU.
- Those details are for the students to discover and deal with during the design an implementation process of the PPU.

Which Pipelined Processing Unit?

- **5-Stage Pipelined Central Processing Unit (PPU)**
- **For subset of ARM architecture (data processing, basic loads/stores, branch, branch and link)**

Based on the pipelined organization model presented on the Computer Organization and Design book by:

Patterson and Hennesy

5-Stage PPU

Functionality of each stage:

Instruction Fetch **(IF)**

Brings the instruction from memory into the CPU.

Instruction decode/register fetch **(ID)**

Decodes the instruction,

Reads the content of the registers specified as operands into temporary registers

Sign-extends the immediate values,

Determines the target address of branch instructions

Execution/effective address **(EX)**

Load/stores – calculates the effective address

Arithmetic/logic – executes the operation

Branches – determines if the branch condition is asserted

5-Stage PPU

Functionality of each stage:

***Memory access* (MEM)**

Load – reads the effective address

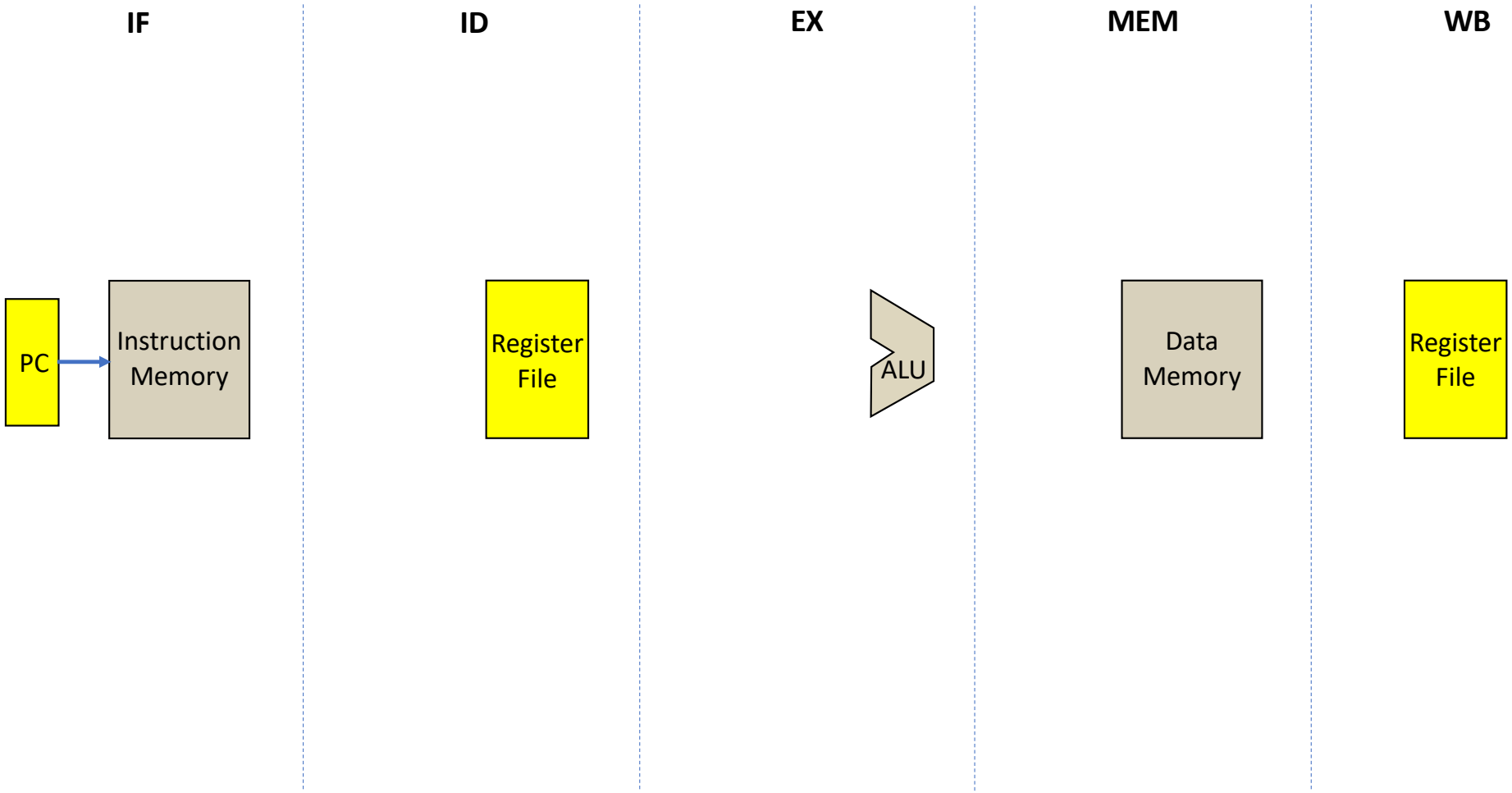
Store – writes the effective address

Any other instruction – no action

***Write-back* (WB)**

Writes the result in the destination register if the instruction has a destination operand

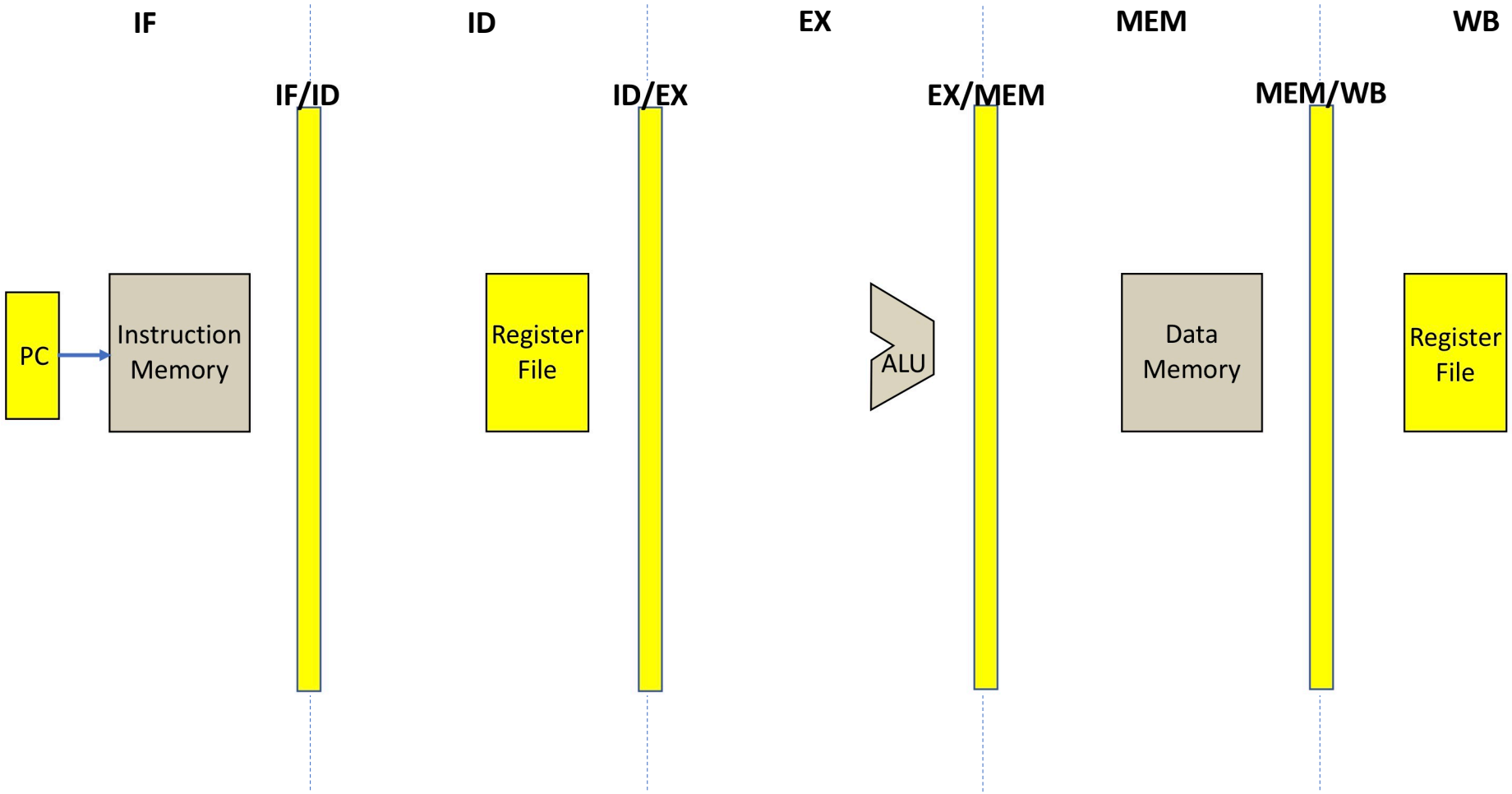
Hardware Components for Each Stage



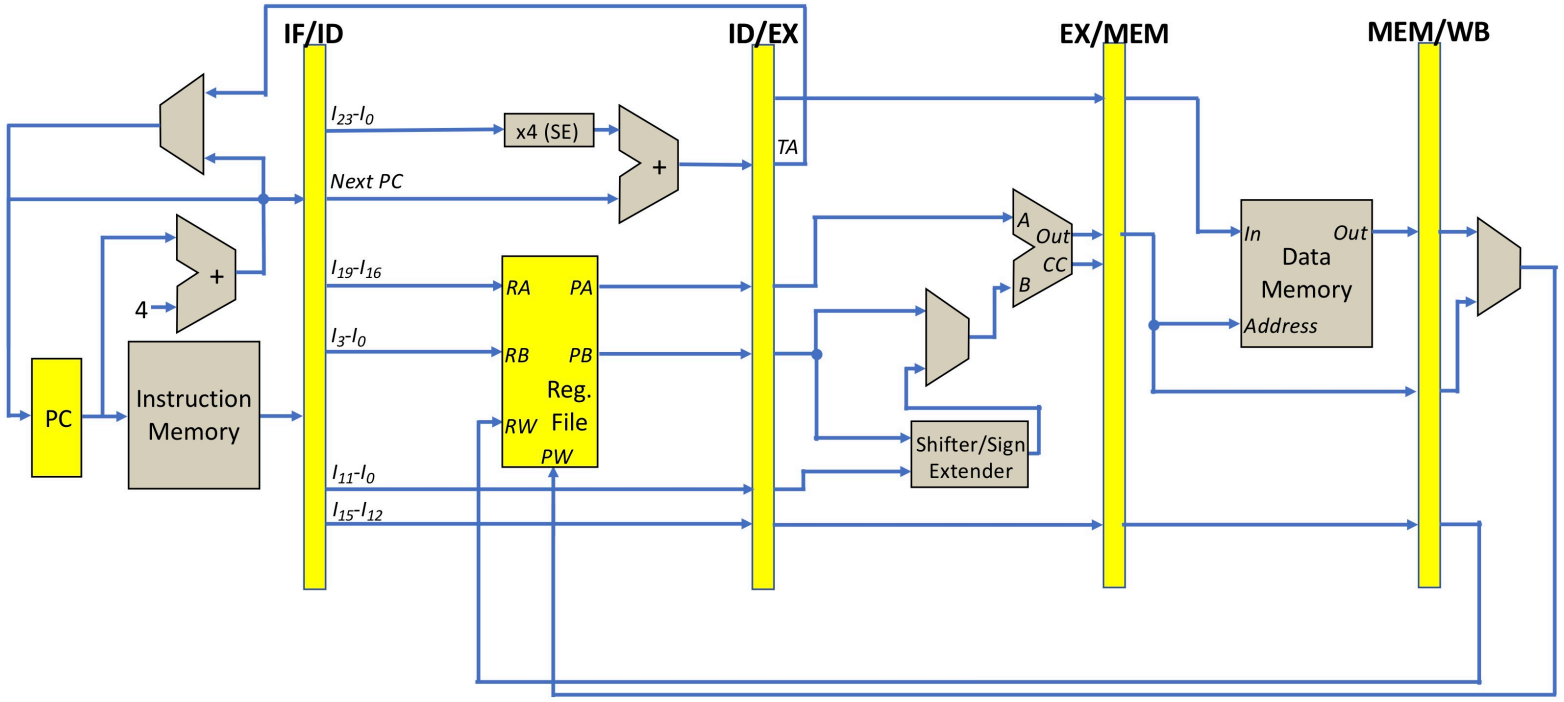
Pipeline Registers

- **Why are they needed?**
  - Information needs to be transferred from one stage to another and held.
  - Information and control signals may be generated on early stages may need to be transferred through middle stages for use on later stages.
- **How big?**
  - They are of different sizes.
  - They size depends on the size of the information needed to be transferred.
- **When are they loaded?**
  - Normally on each clock cycle.

Basic Pipelined Processing Units with Pipeline Registers



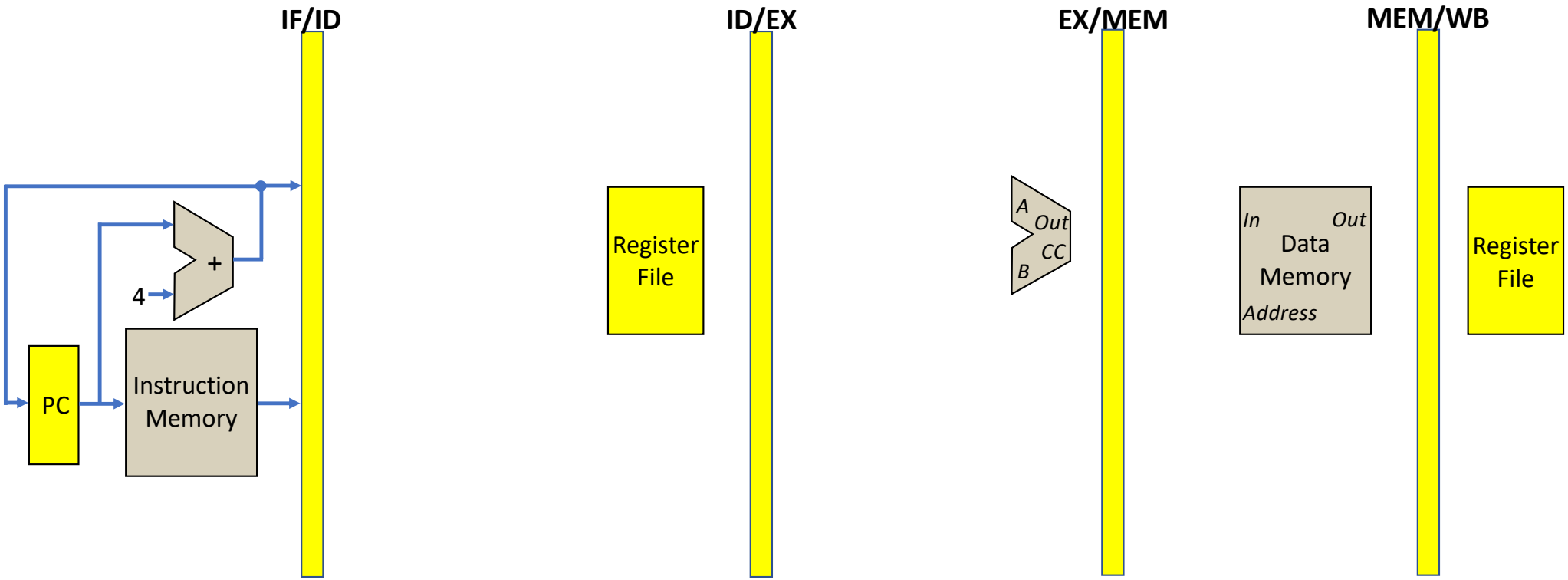
Interconnection of Pipeline Components



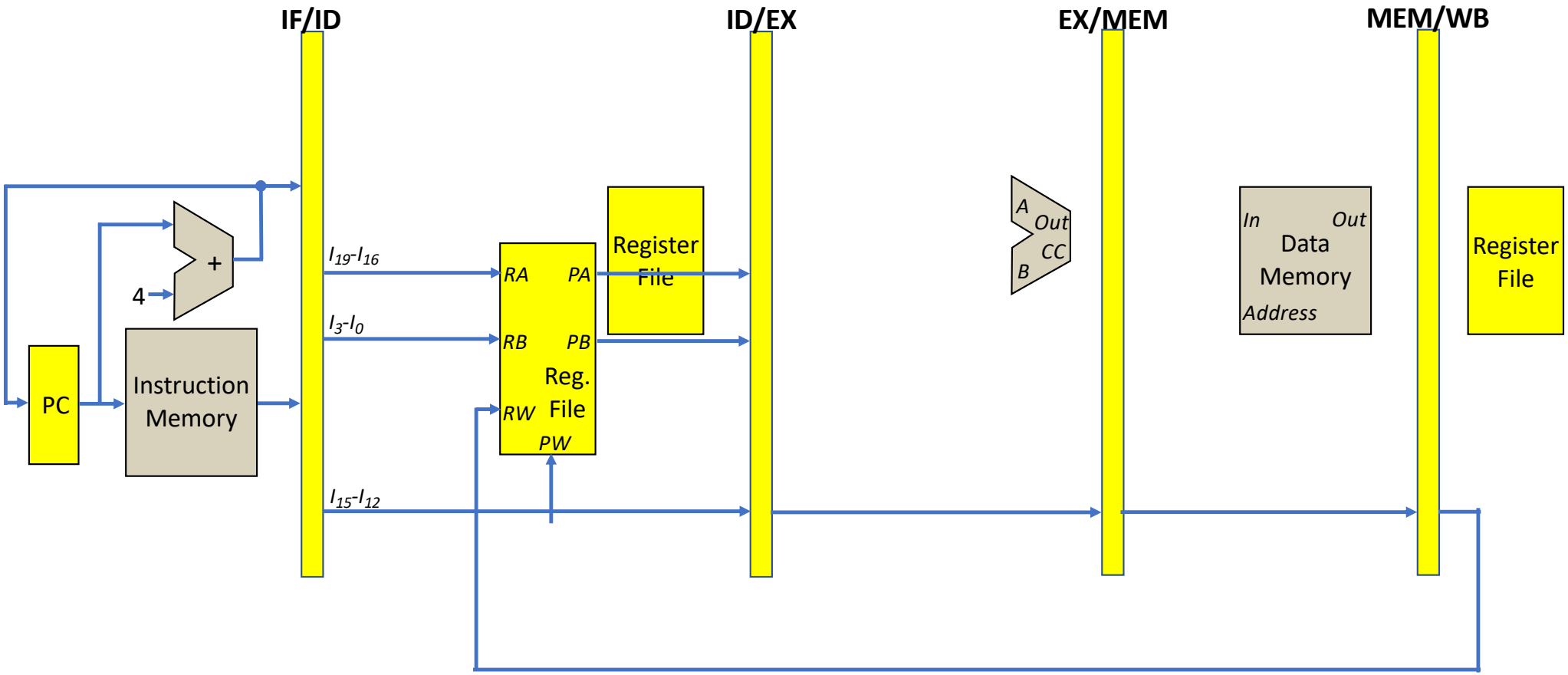
Considerations for the Interconnection of the PPU

- **The Instruction Memory and the Data Memory are actually two separate memories that eliminate structural hazards.**
- **The PC is register R15 of the Register File**
- **The PC is incremented in the IF stage.**
- **The Register File on the ID stage and the WB stage are the same unit.**
- **The target address of a branch instruction is calculated on the ID stage.**
- **Some of the instructions will require operand sign extension or shift operation on the EX stage.**
- **If a branch is taken the PC is loaded with the Target Address instead of PC +4.**
- **WB may occur either by data processing instructions or a load instructions.**

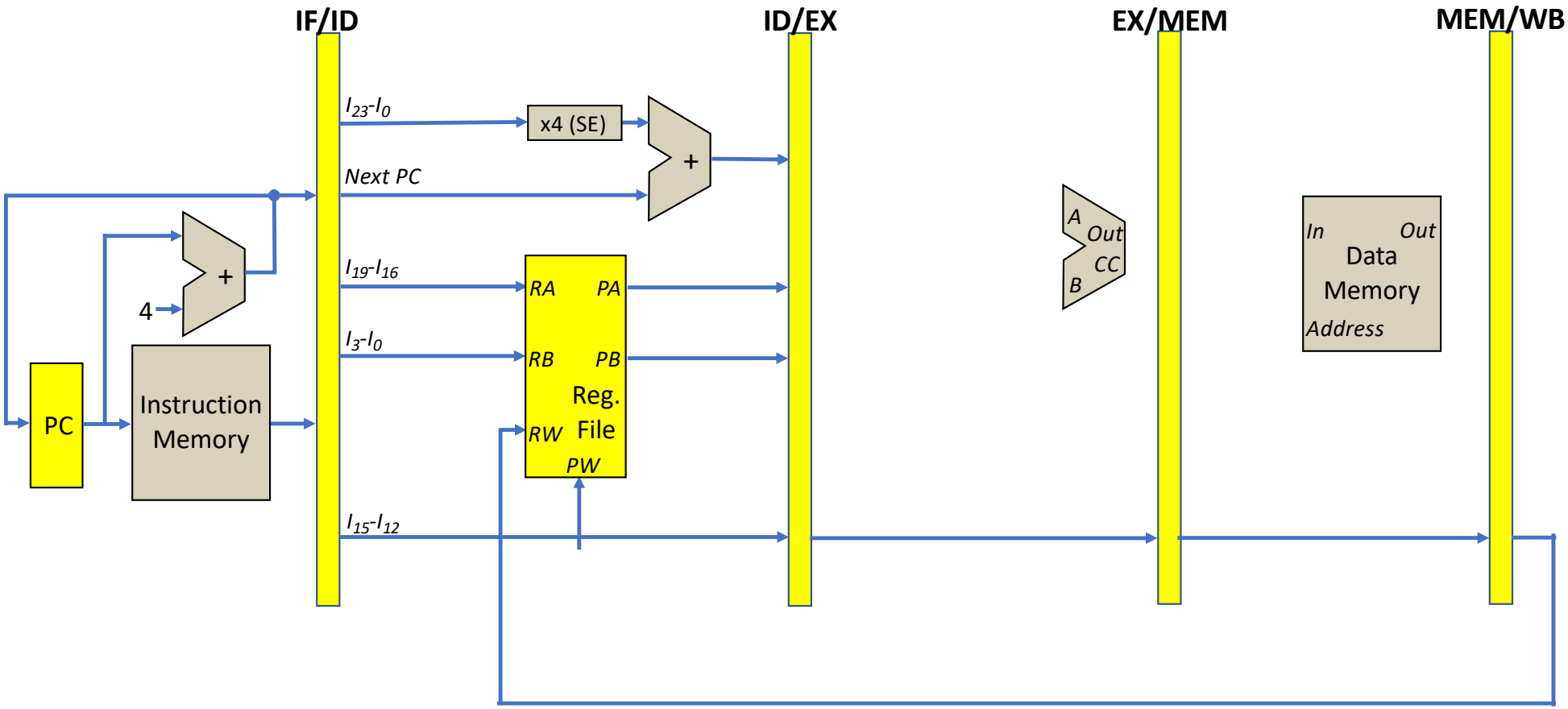
PC is incremented in the IF stage



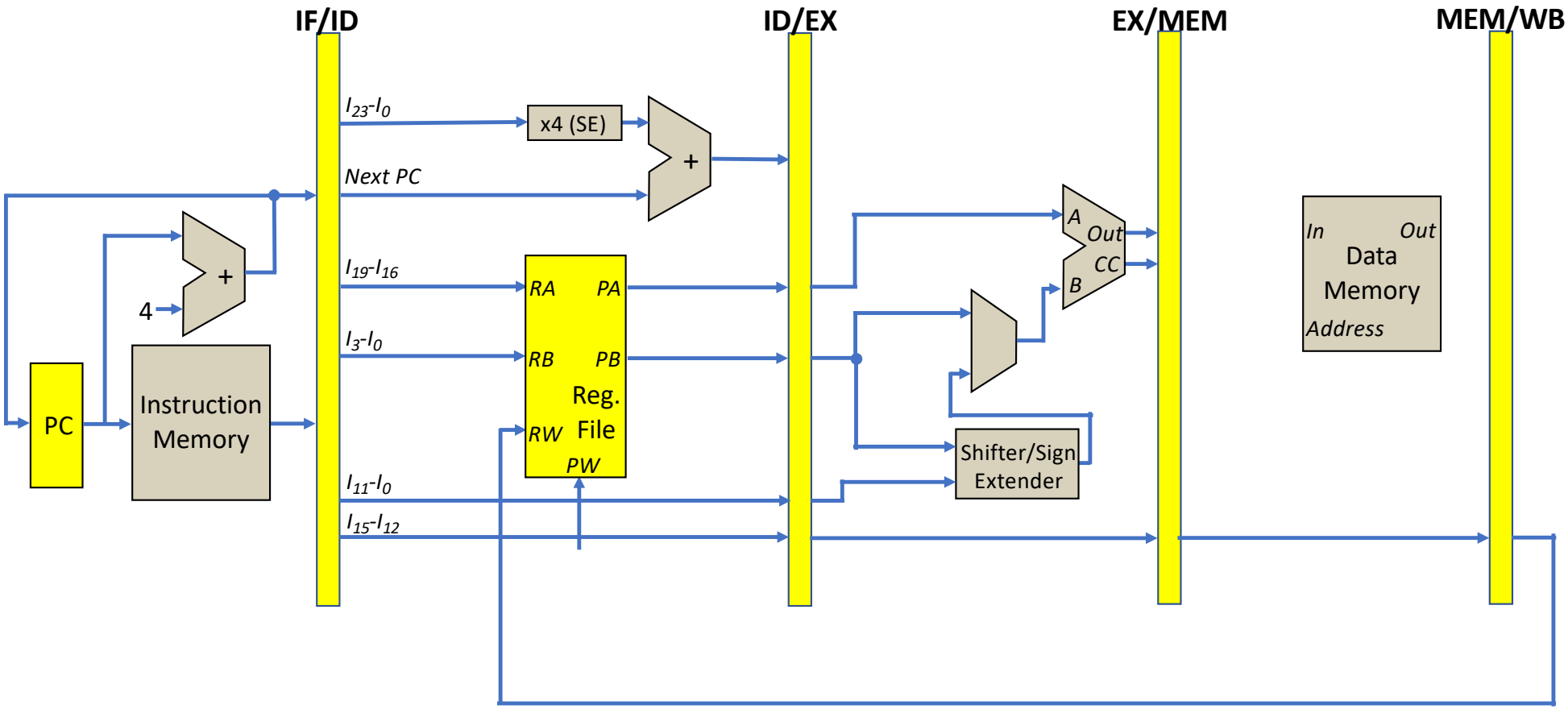
Register File is one unit



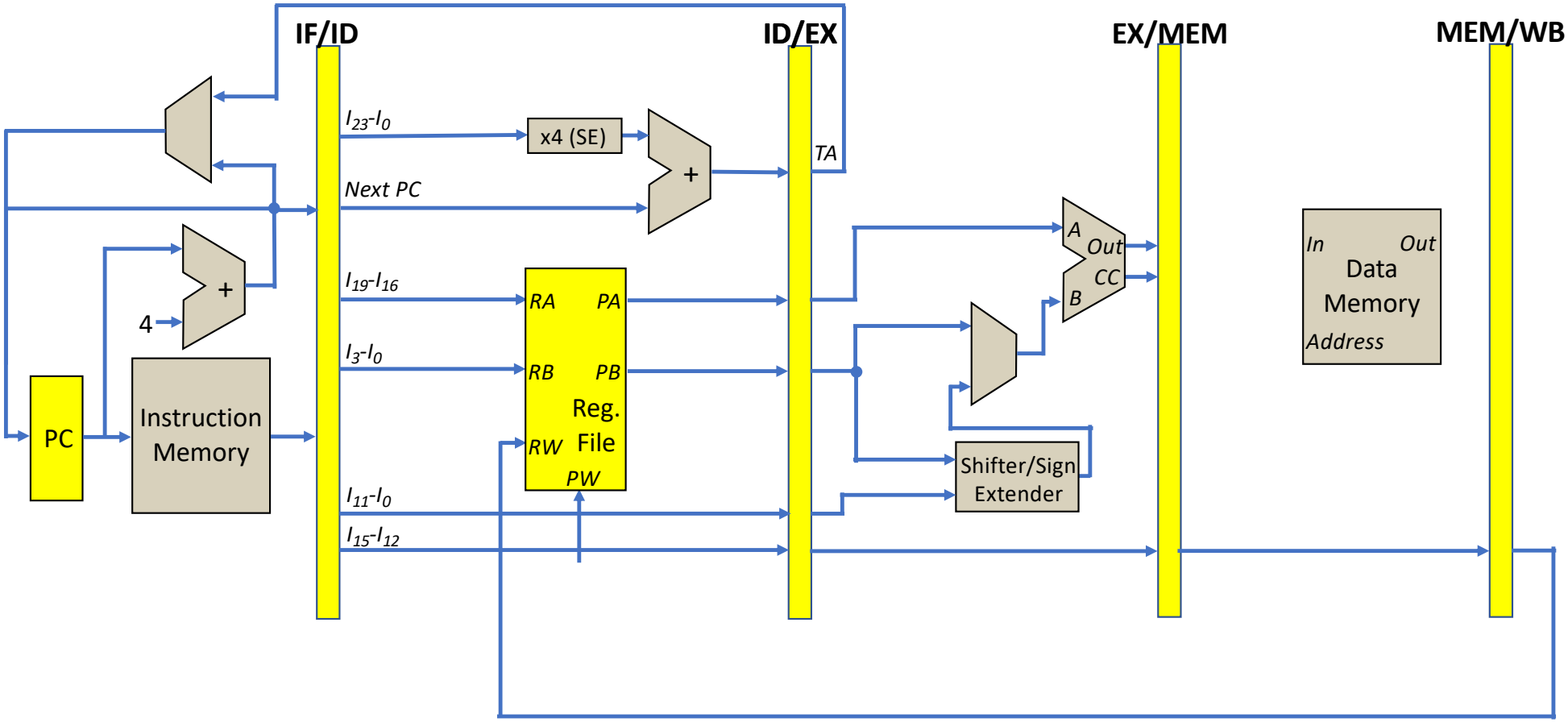
Branch target address calculated on ID stage



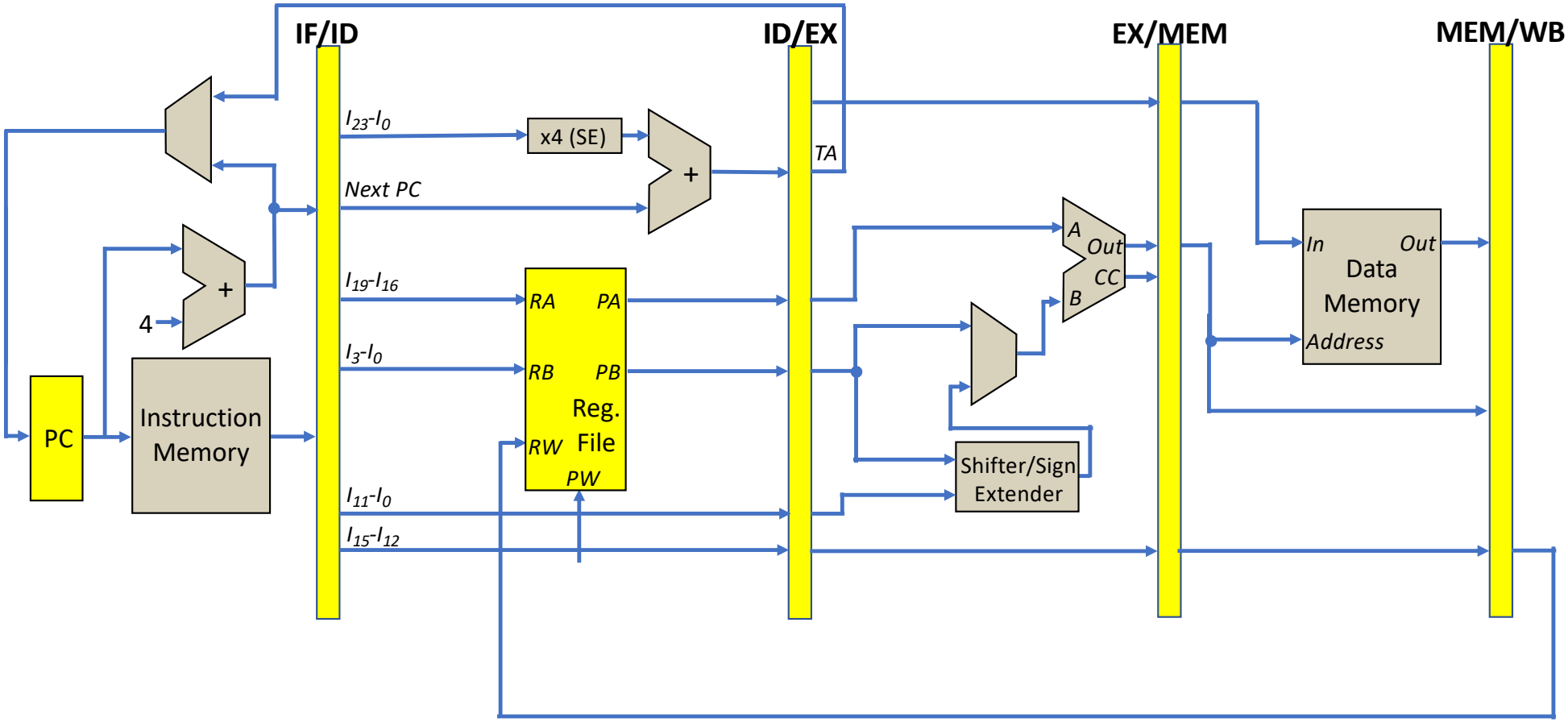
Operand sign extension/shift operation on EX Stage



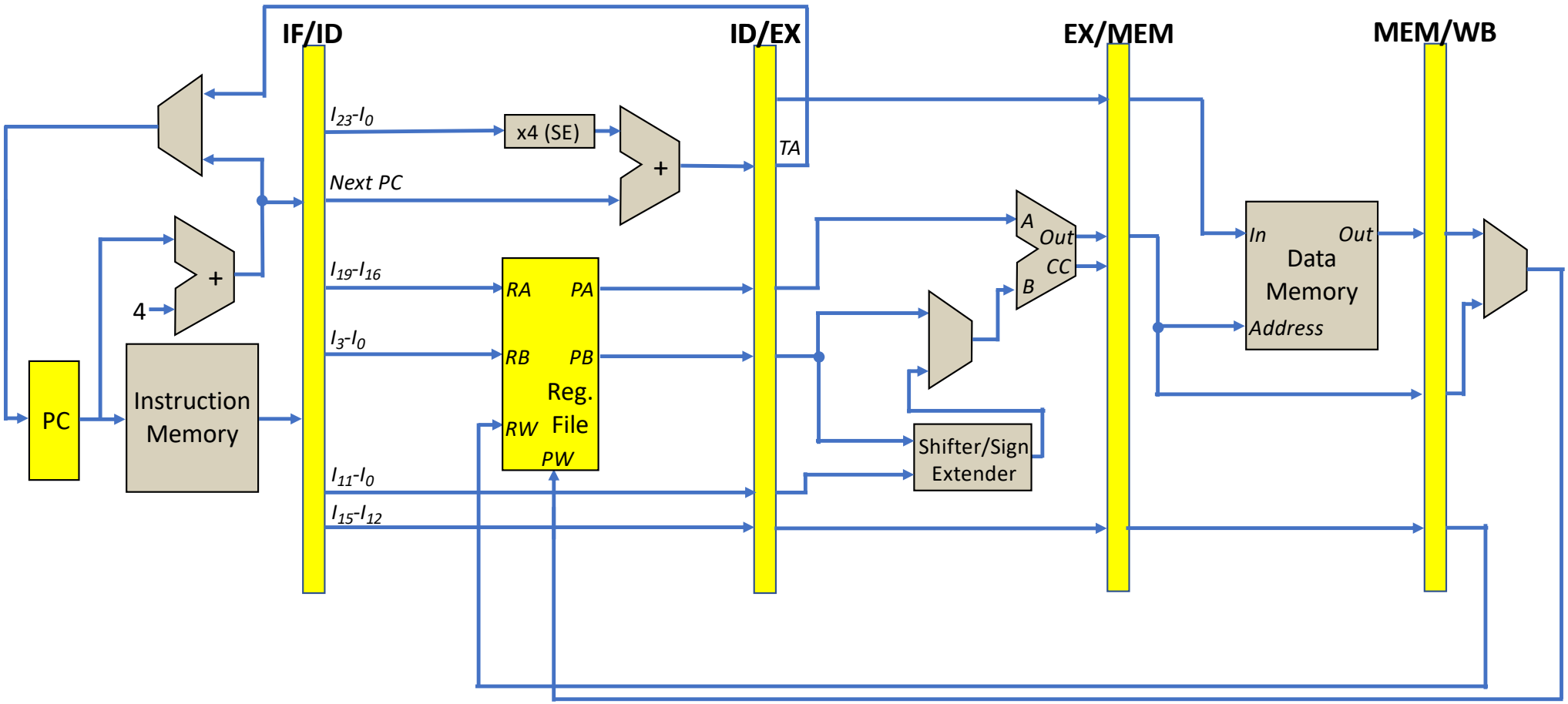
On branch taken load Target Address on PC



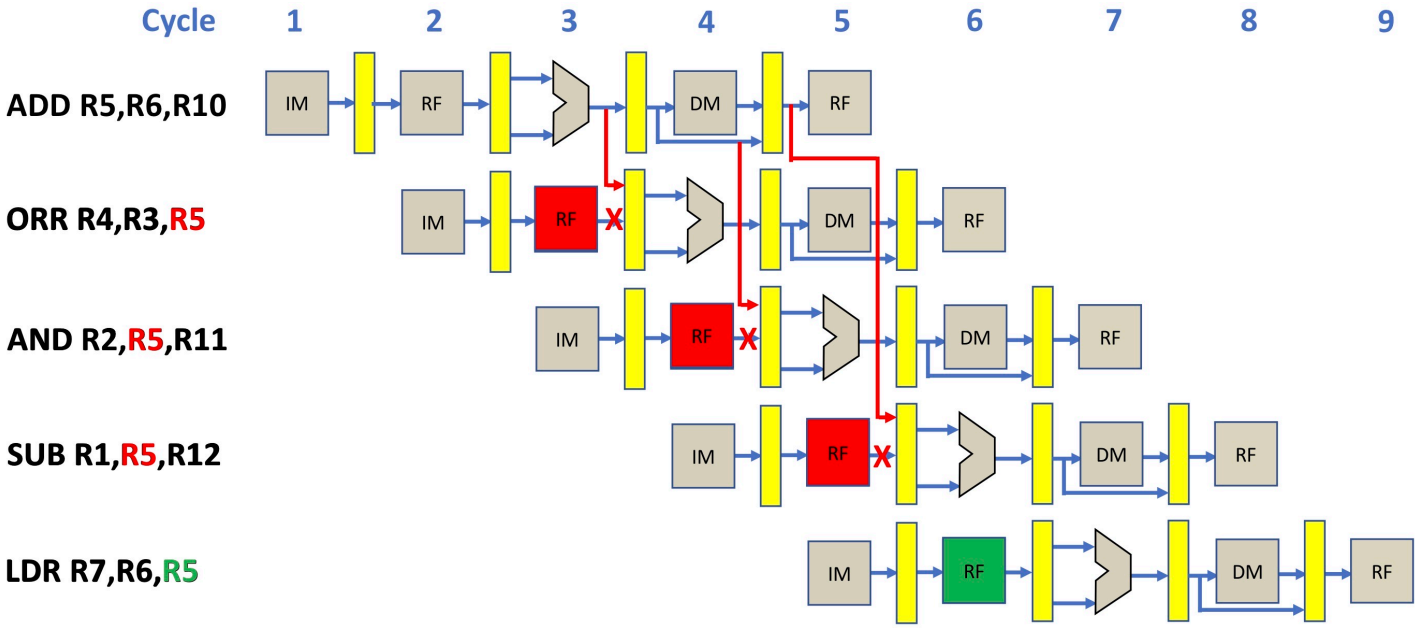
Data Memory Interface



The Write Back Connection



Data Forwarding



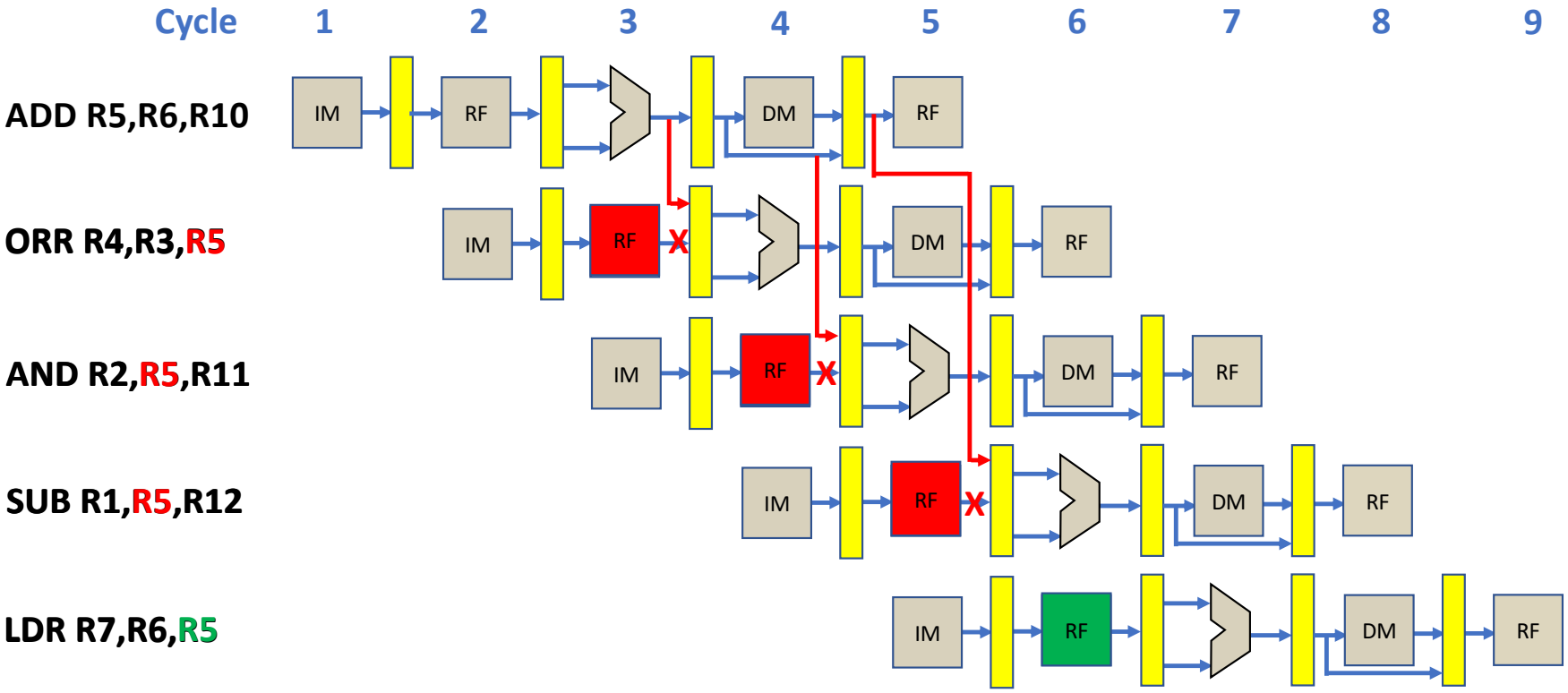
Data Hazard

- **Takes place when an instruction on the ID stage reads a register from the Register File that must be written by an instruction ahead in the pipeline.**
- **If the instruction on the ID stage is allow to proceed, the pipeline will feed an incorrect value for the register to the EX stage.**
- **It can be fixed with a data forwarding mechanism.**

Data Forwarding

- **Although the destination register of an instruction is written just after it leaves the WB stage, its value is generated in the EX and sequentially transferred to the MEM stage and the WB stage.**
- **The value of the destination register can be forwarded to instructions on the ID stage from the EX, MEM or WB stages.**
- **On forwarding, the source operand generated by the Register File (*Rn* or *Rm*) is substituted by the forwarded value.**

Data Forwarding



Considerations for Handling Data Forwarding Cases

- **Requires a comparison of the destination register number of an instruction on the EX, MEM and WB stages with a source register number of an instruction on the ID stage.**
- **The destination register number must be transferred to the EX, MEM and WB stages as the instruction proceeds from stage to stage.**
- **Forwarding could be performed for the Rn or Rm source operands or both.**
- **Forwarding takes place only for instructions with a register destination operand.**
- **A control signal *RF_enable* must be generated by the Control Unit for every instruction in the ID stage. It must be set to 1 if the instruction will write the Register File. Otherwise, it must be set to 0.**
- **The *RF_enable* must be transferred to the EX, MEM and WB stages as the instruction proceeds through those stages.**
- **When the *RF_enable* corresponding to the instruction gets to the WB stage, if it is a 1, it will command the Register File to write the destination register.**

Notation for Signals at Different Stages

Signals will be identified according to the stage they are visible using the following notation:

Stage_Name

Example:

EX_Rd

Signals Needed for Handling Data Forwarding

EX_RF_enable **– the instruction on the EX stage will write the destination register**

MEM_RF_enable **– the instruction on the MEM stage will write the destination register**

WB_RF_enable **– the instruction on the WB stage will write the destination register**

EX_Rd **– number of the Rd register of the instruction on the EX stage**

MEM_Rd **– number of the Rd register of the instruction on the MEM stage**

WB_Rd **– number of the Rd register of the instruction on the WB stage**

ID_Rm **– number of the Rm register of the instruction on the ID stage**

Forwarding for Rm

- **EX forwarding**

  if *EX_RF_enable* & (*ID_Rm* == *EX_Rd*) then forward [*EX_Rd*] to ID stage

- **MEM forwarding**

  if *MEM_RF_enable* & (*ID_Rm* == *MEM_Rd*) then forward [*MEM_Rd*] to ID stage

- **WB forwarding**

  if *WB_RF_enable* & (*ID_Rm* == *WB_Rd*) then forward [*WB_Rd*] to ID

Criteria for Forwarding for Rm

if *EX_RF_enable* & (*ID_Rm* == *EX_Rd*)

then forward [*EX_Rd*] to ID stage

elseif *MEM_RF_enable* & (*ID_Rm* == *MEM_Rd*)

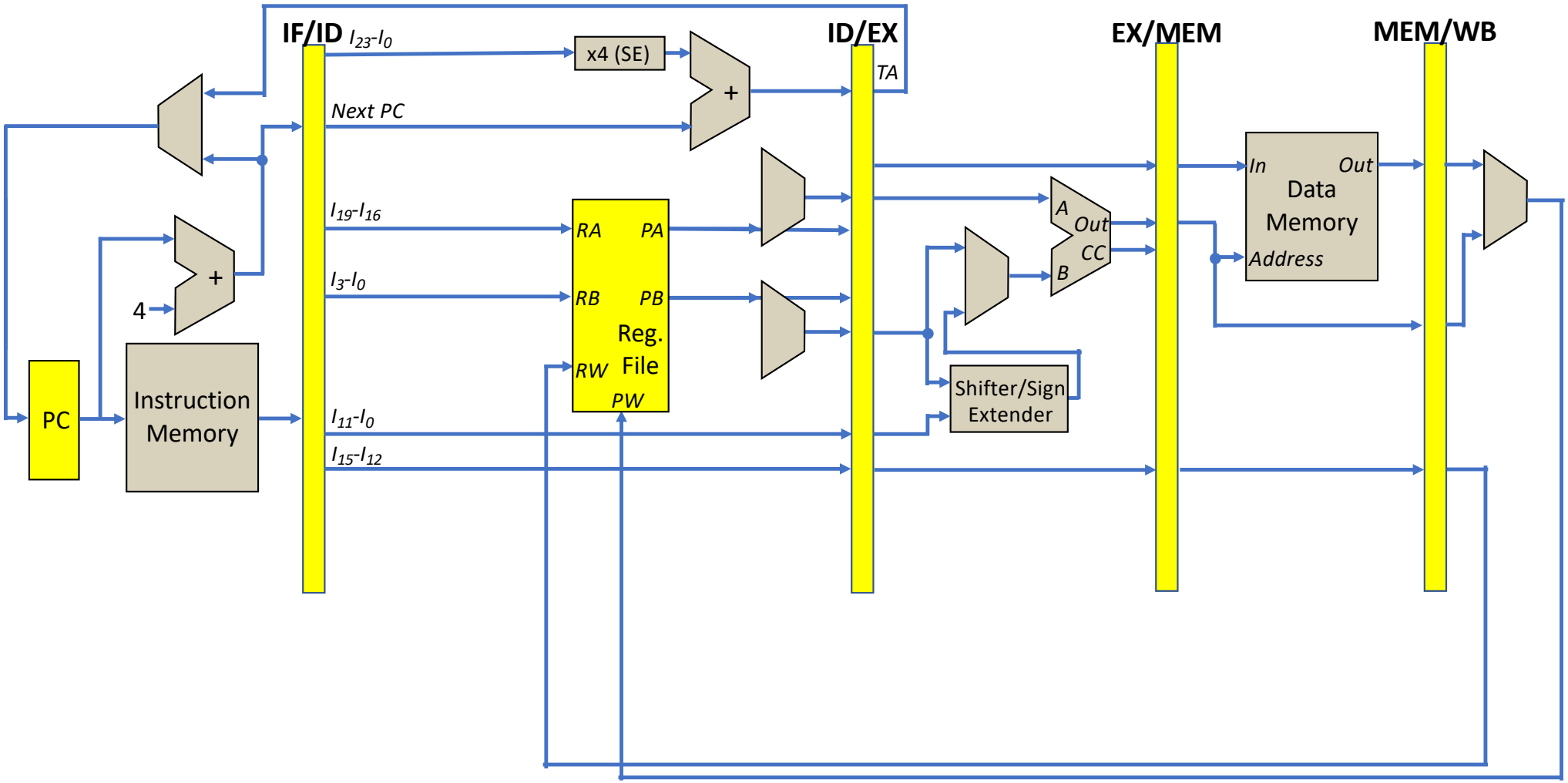
then forward [*MEM_Rd*] to ID stage

elseif *WB_RF_enable* & (*ID_Rm* == *WB_Rd*)

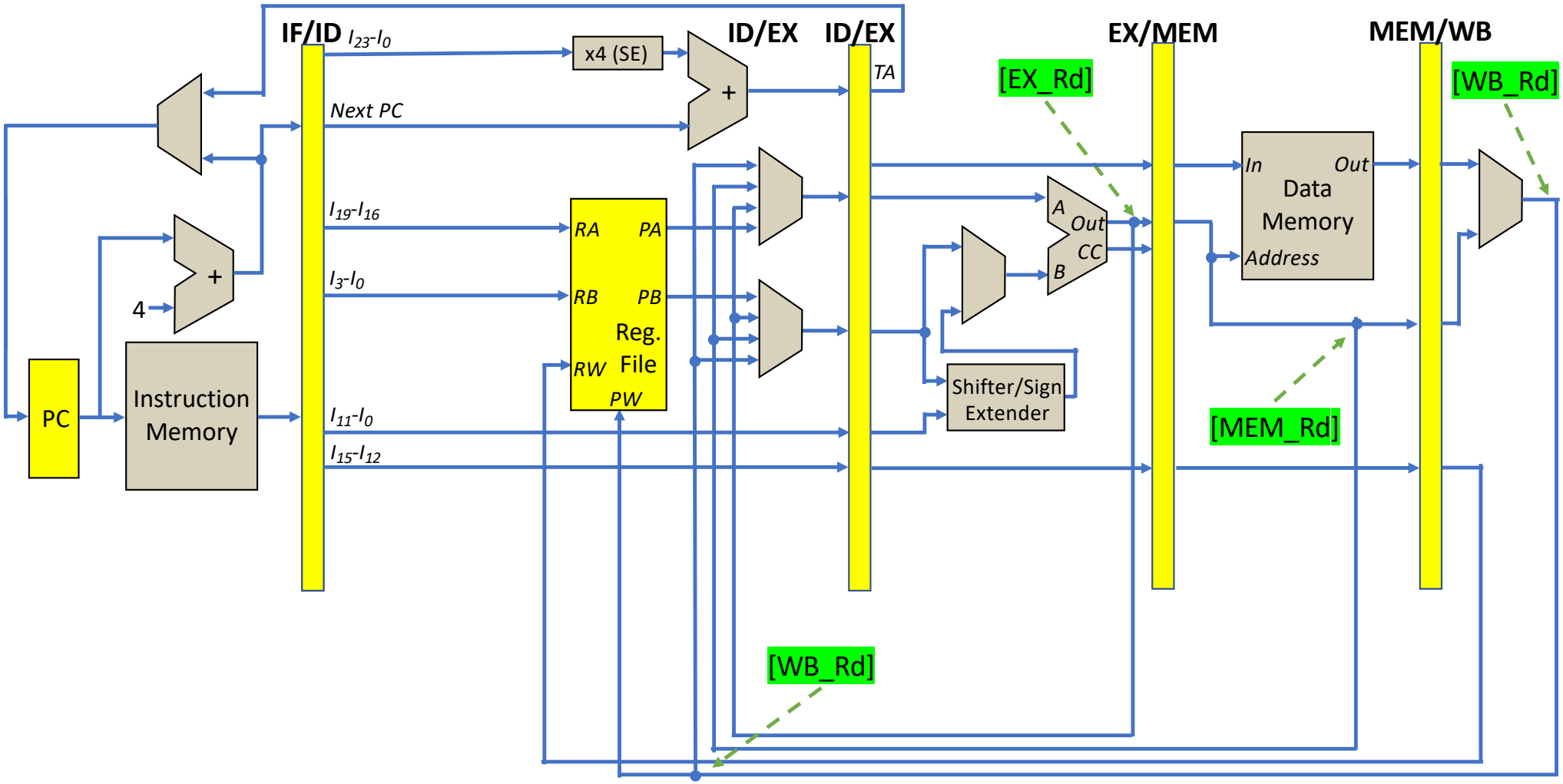
then forward [*WB_Rd*] to ID

(Same applies for Rn)

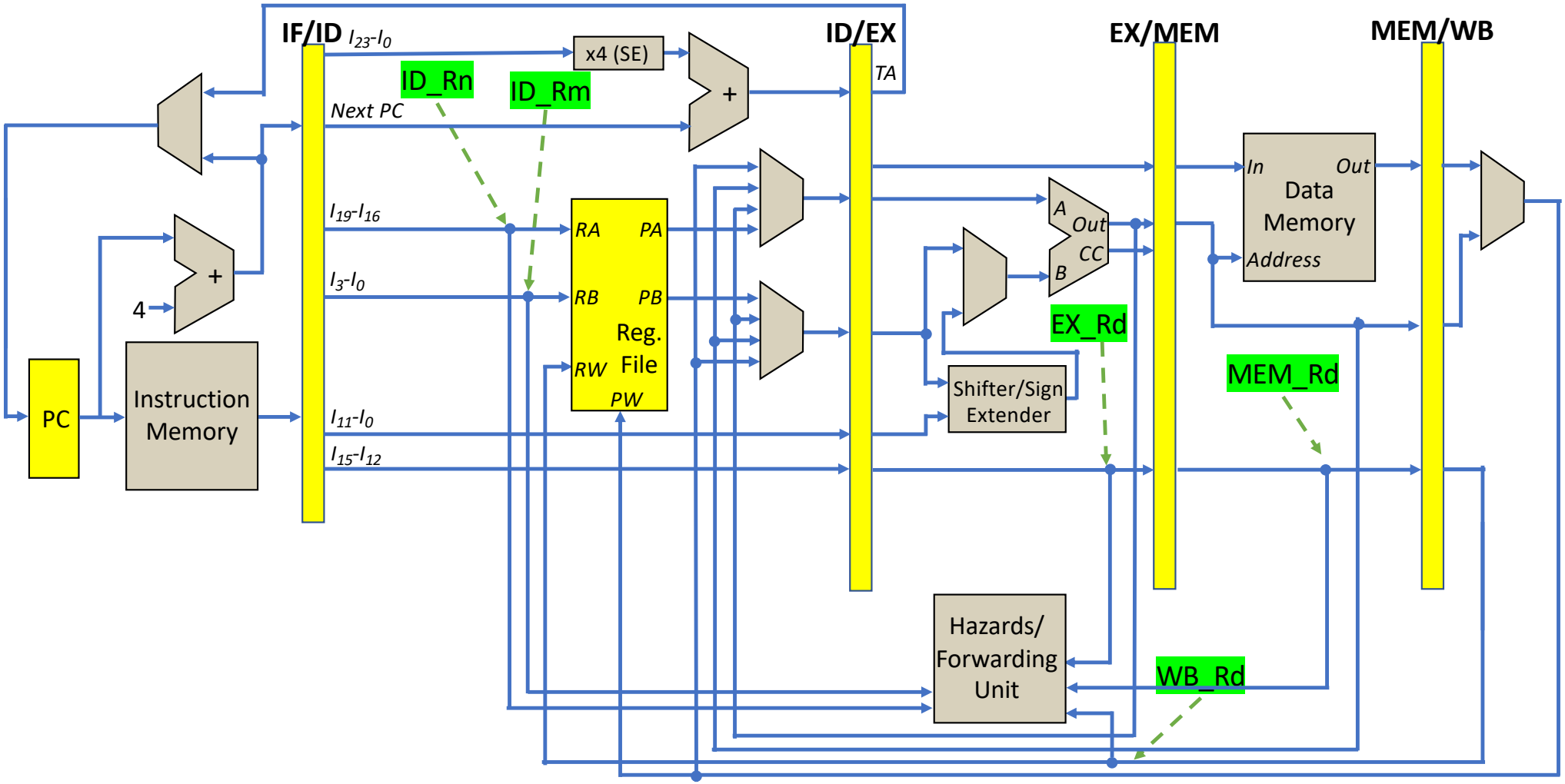
Adding Multiplexers on ID Stage



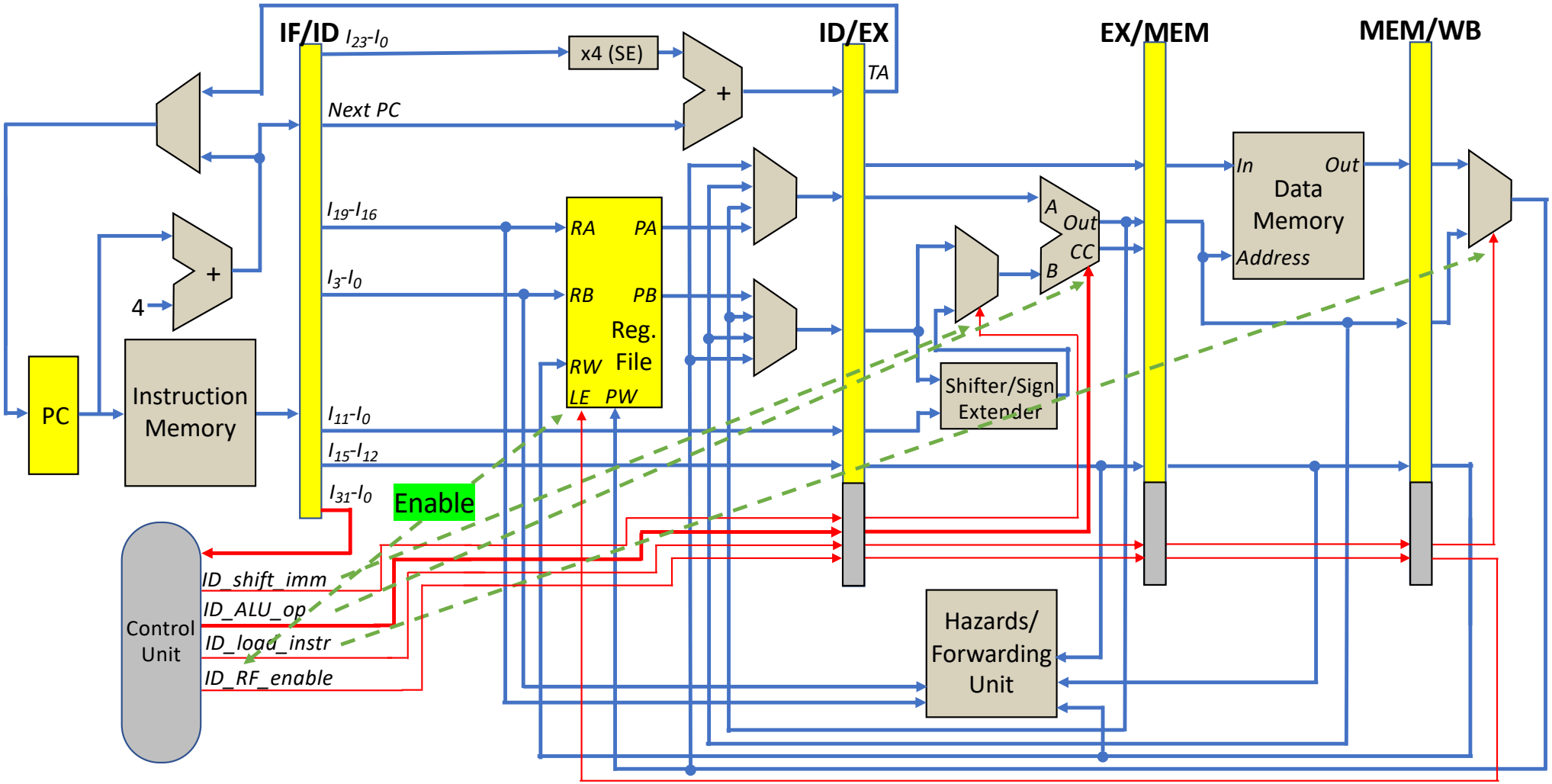
Connecting Forwarding Operands



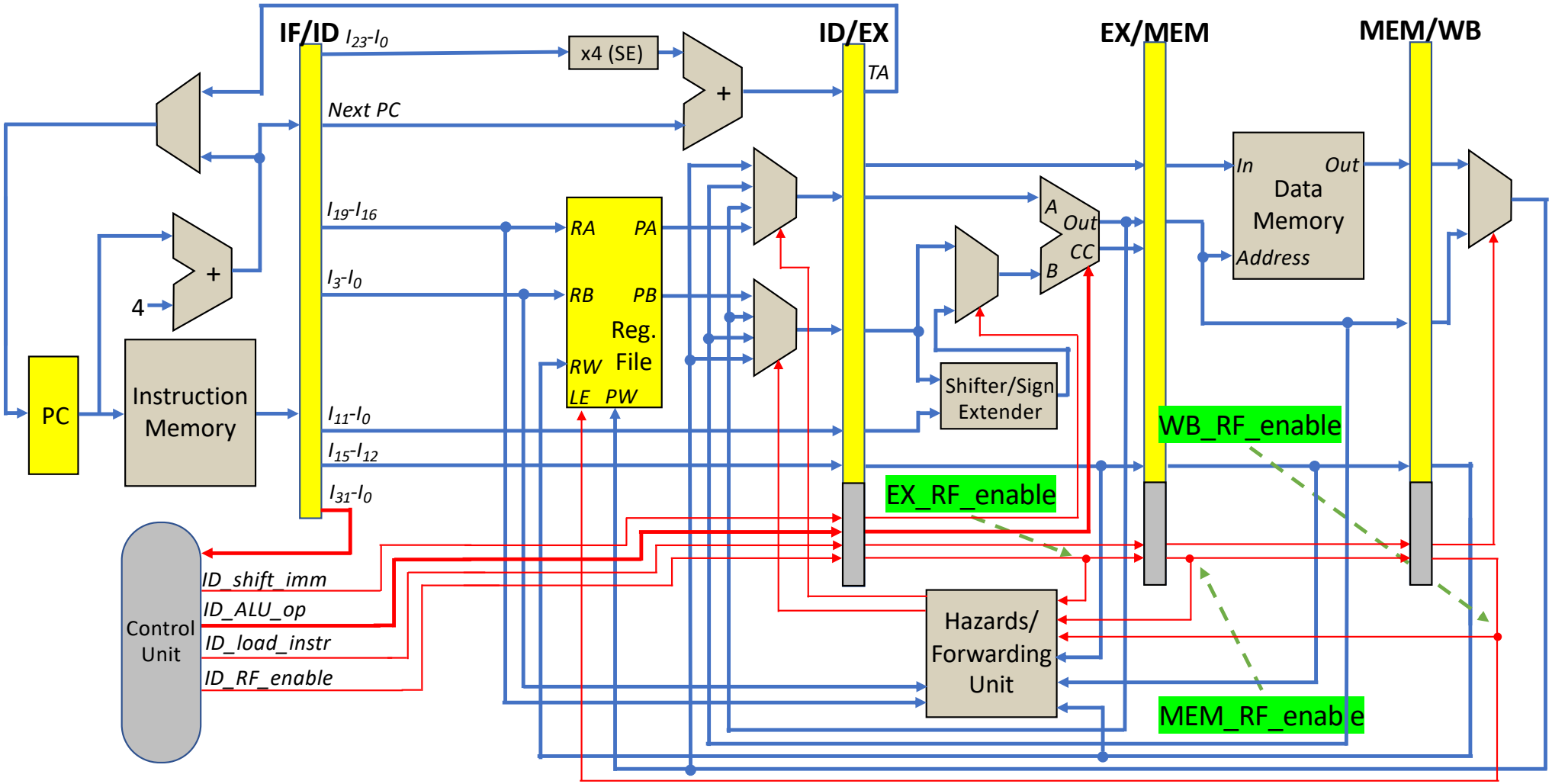
Adding Hazards/Forwarding Unit



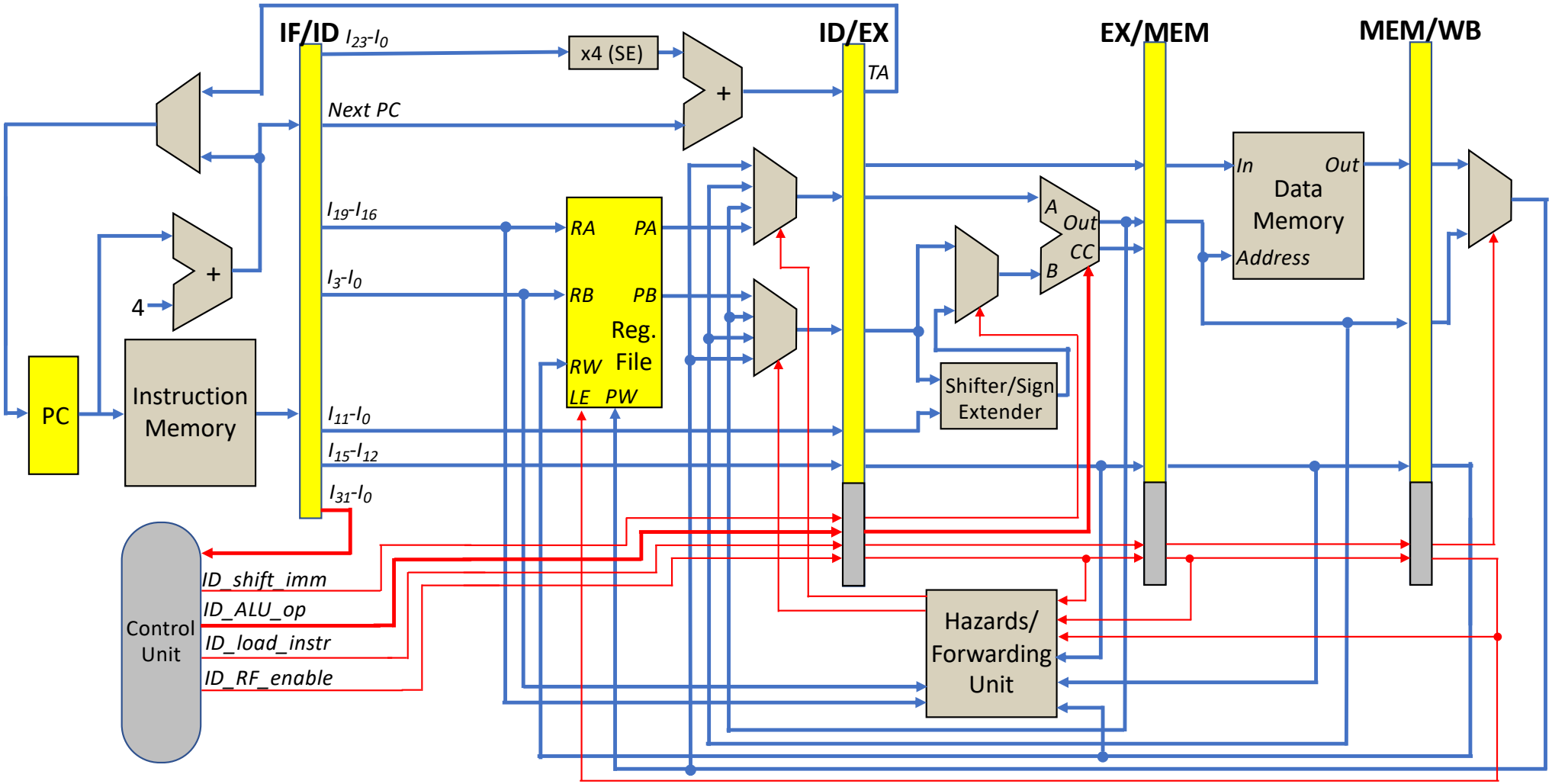
Adding Control Unit



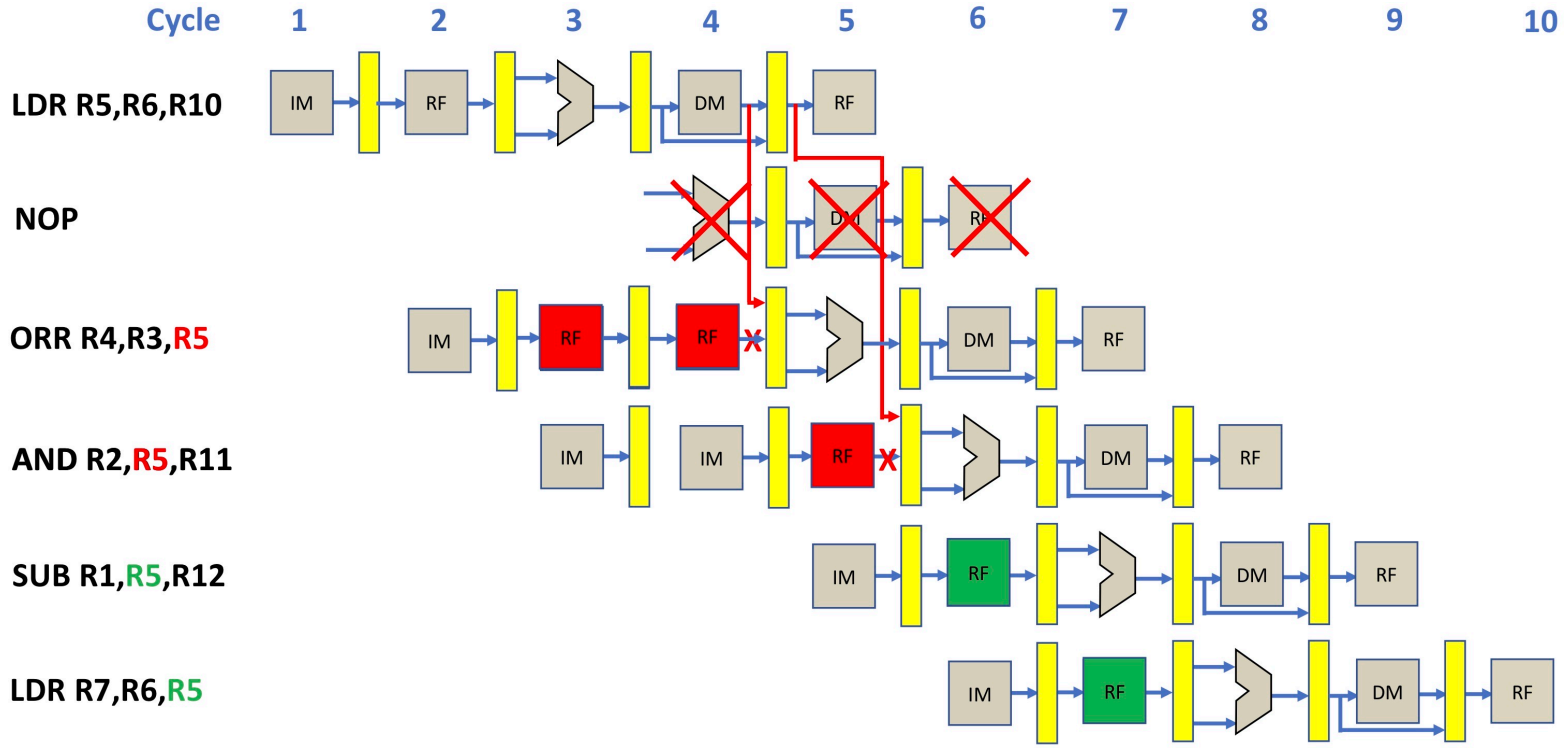
Adding Forwarding Control Signals



Adding Forwarding Control Signals



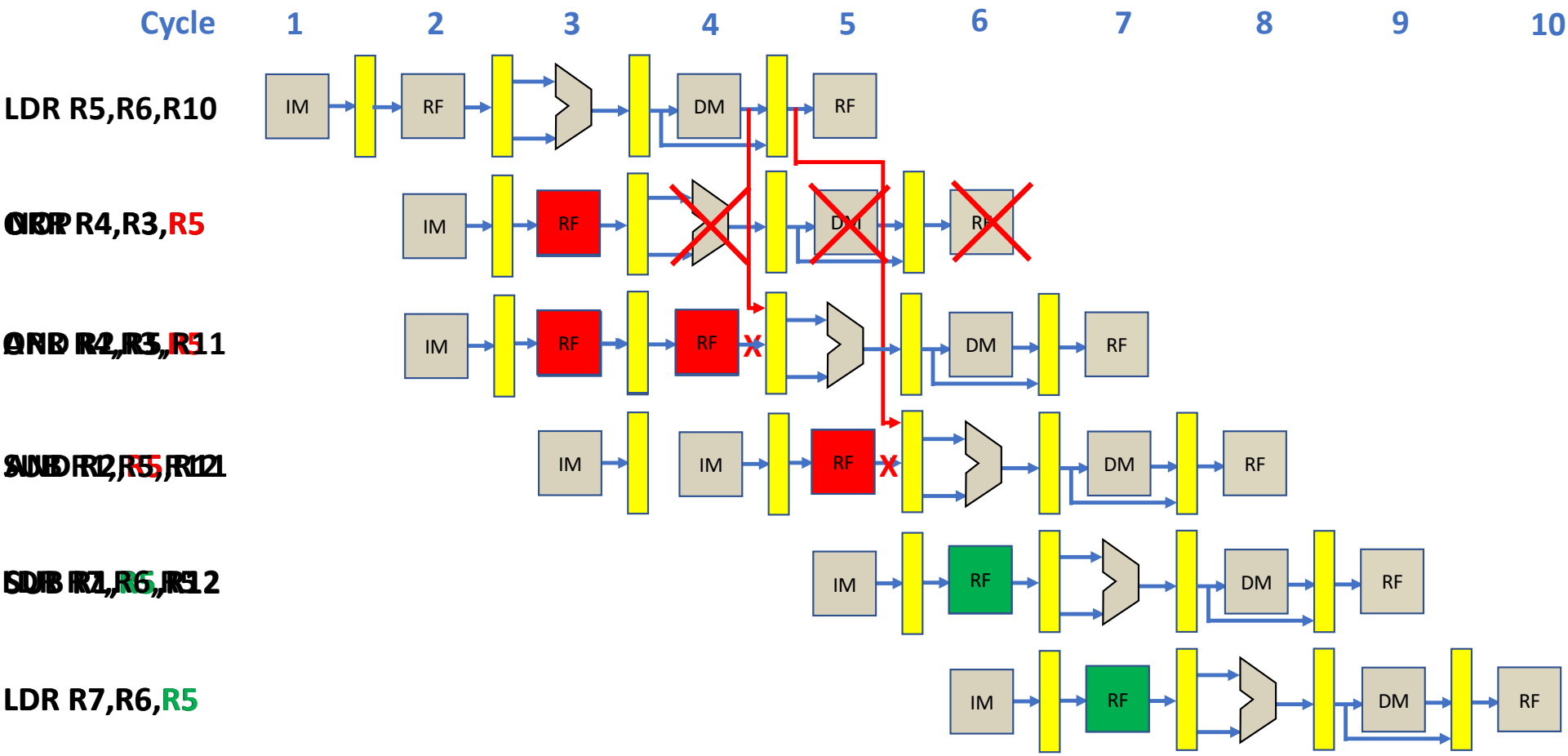
Data Hazard by Load Instruction



Data Hazard by Load Instruction

- When a source register of an instruction immediately following a load instruction is the same as the destination register of the load instruction a data hazard occurs.
- Since the destination operand is generated in the MEM stage instead of the EX stage, when the instruction following the load instruction is in the ID stage the load instruction is in the EX stage and thus, the destination operand is not available to be forwarded to the ID stage.
- If the instruction on the ID stage is allowed to proceed, it will use an incorrect value for the source register.
- When this type of hazard is detected the instructions in the ID and IF must be prevented from advancing to the next stage, and a nop operation must be advanced to the EX stage, creating sort of a bubble in the pipeline).

Data Hazard by Load Instruction



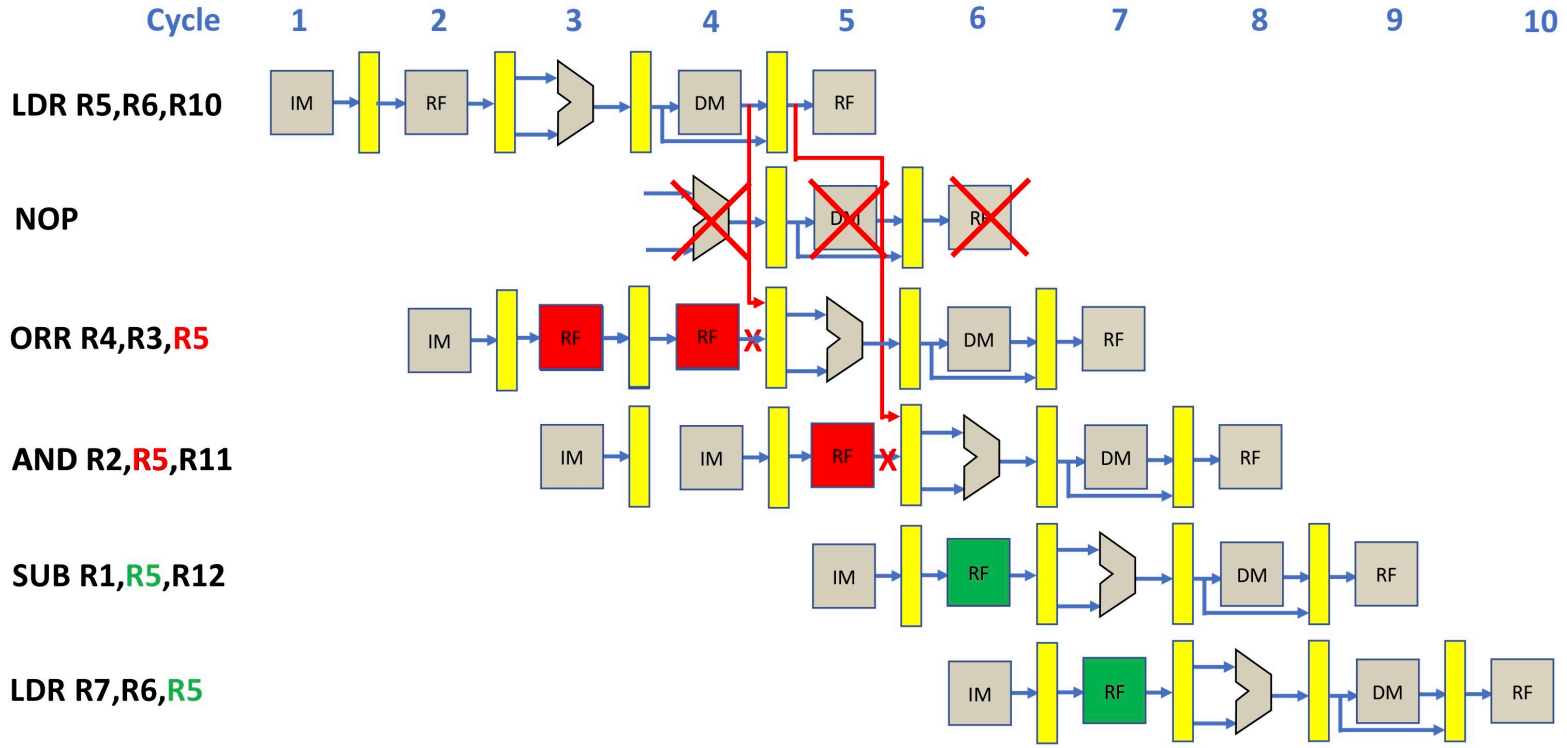
Considerations for Detecting and Handling the Load Hazard

- If a signal *EX_load_instr* is used to identify an instruction in the EX stage as a load instruction, then the load hazard can be detected as follows:

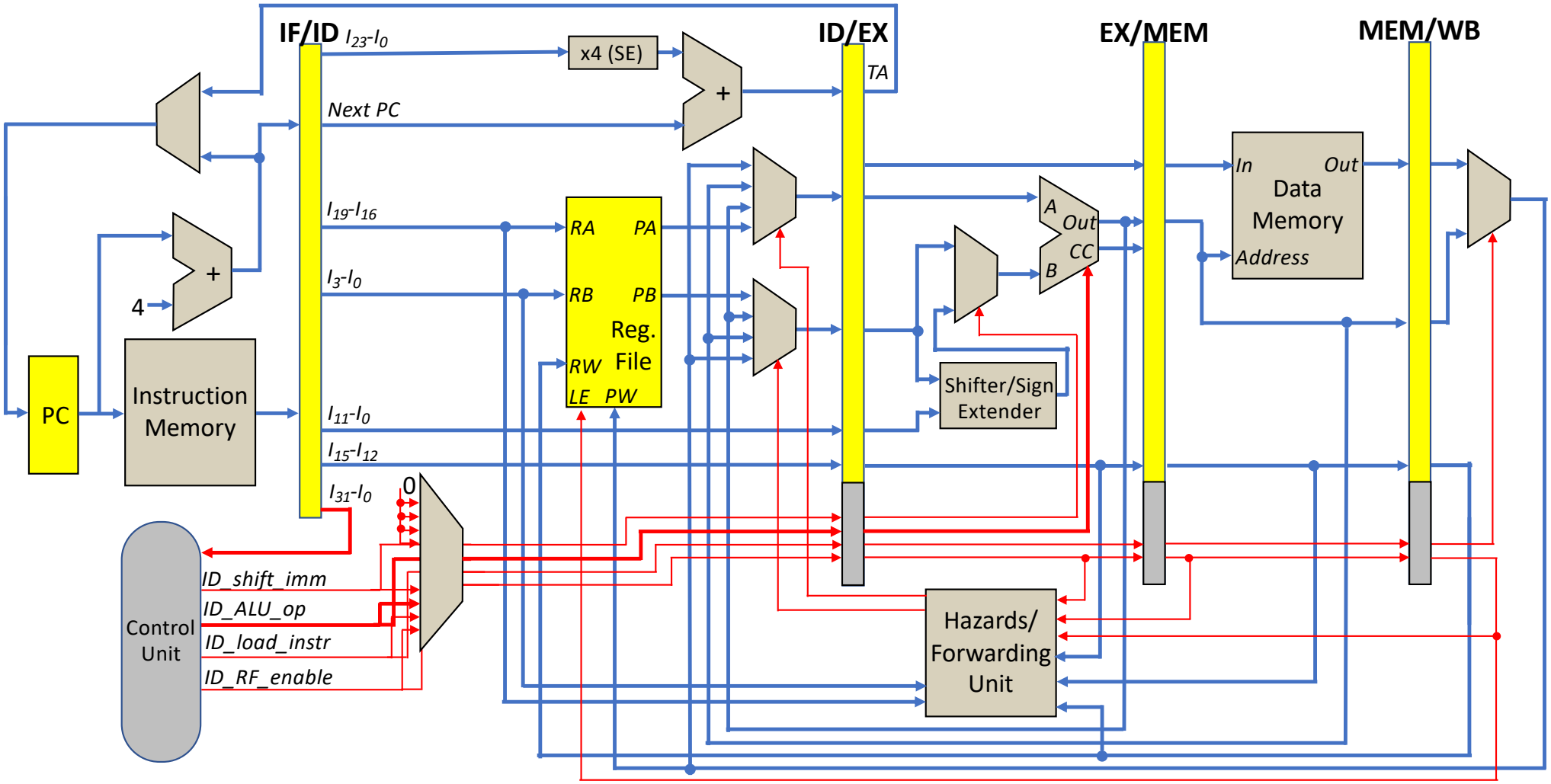
  if *EX_load_instr* & ((*ID_Rn* == *EX_Rd*) or (*ID_Rm* == *EX_Rd*)) then hazard asserted

- When a hazard is asserted the Hazard/Forwarding Unit must take the following actions:
  - Forward control signals corresponding to a *nop* instruction to the EX stage.
  - Disable the IF/ID pipeline register from loading
  - Disable the load enable of the Program Counter
- When the load instruction reaches the MEM stage the destination operand must be forwarded from the output of the Data Memory output. A *MEM_load_instr* signal will do the selection.

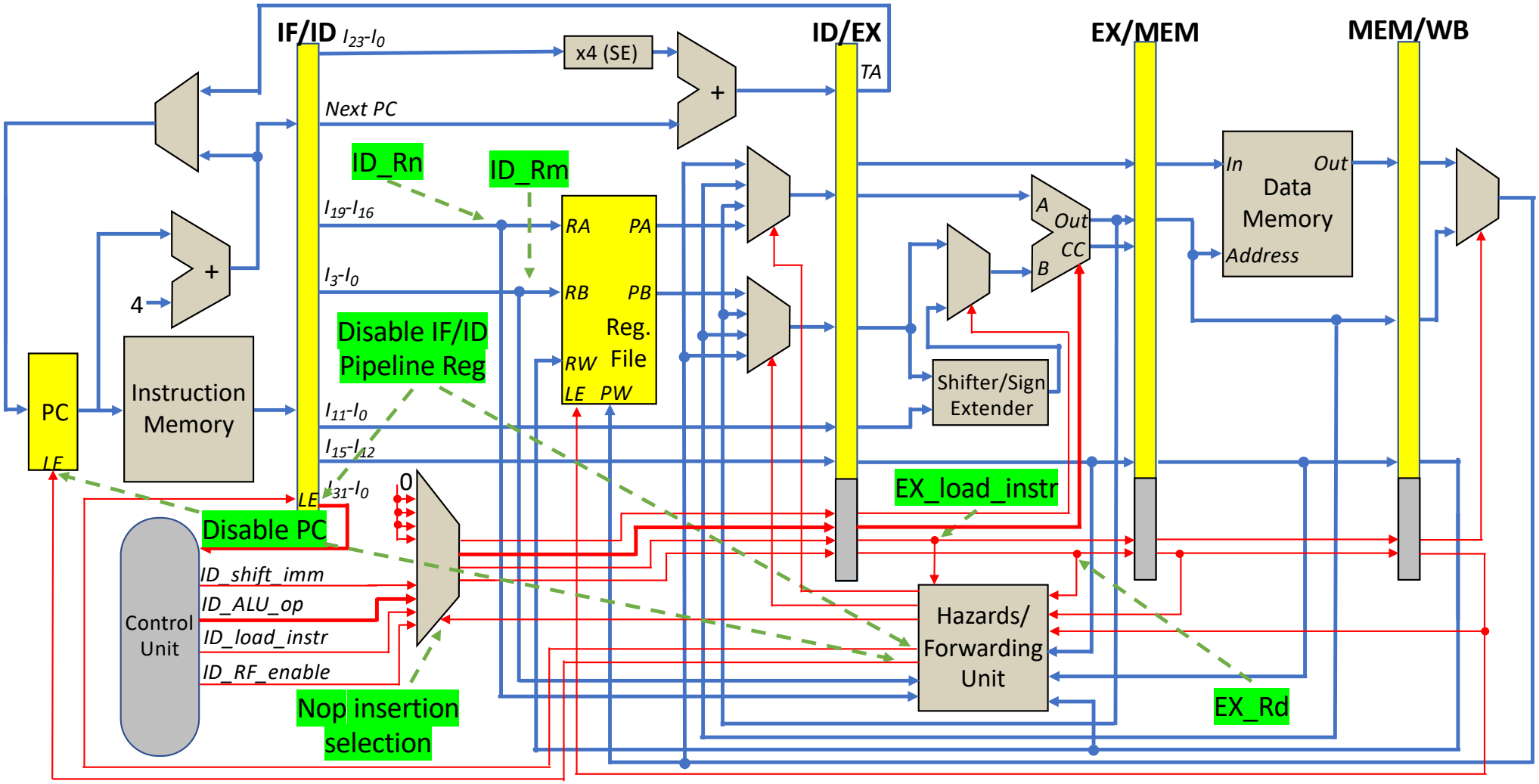
Handling Load Hazards



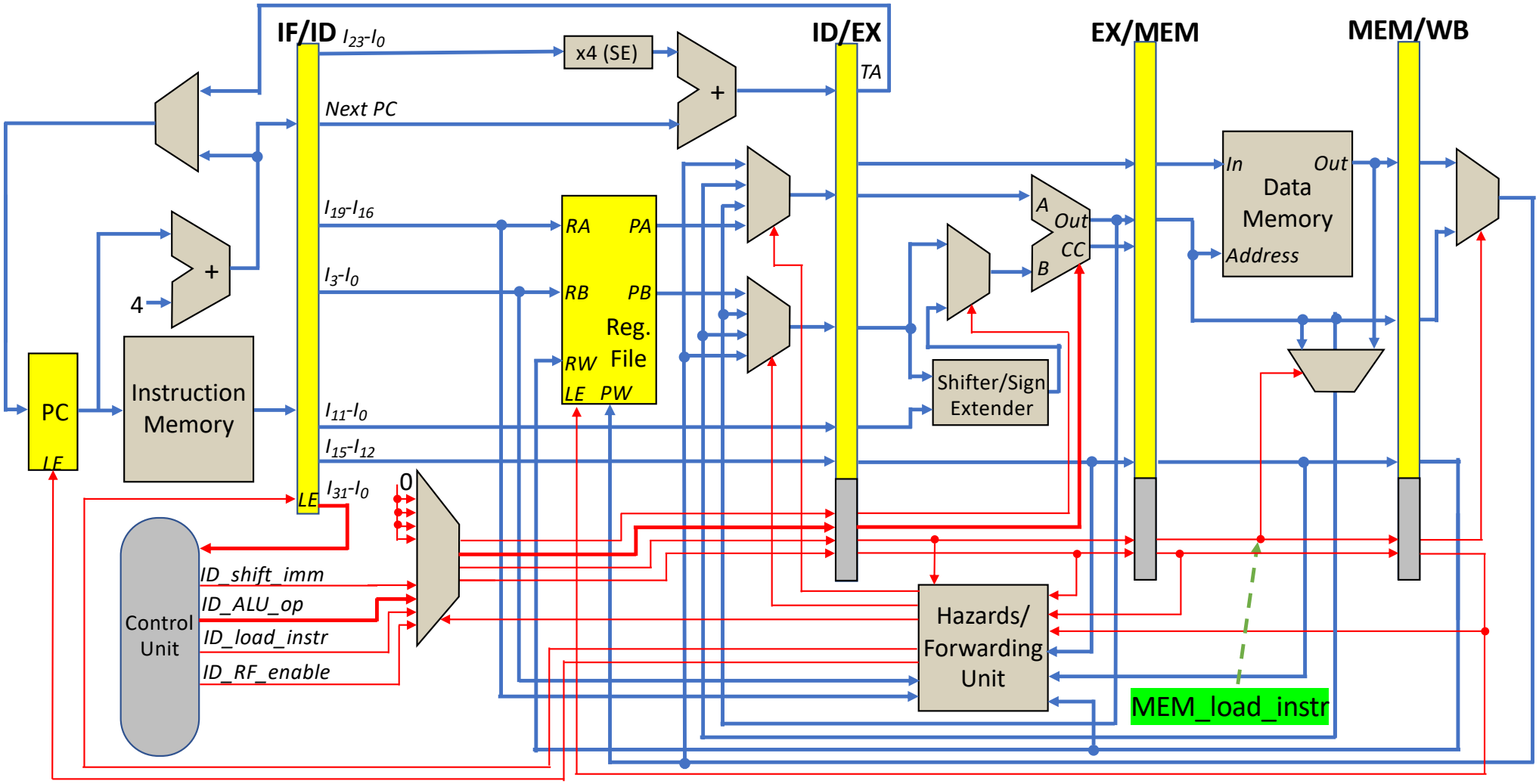
Hardware for Feeding a NOP to the EX Stage



Detecting Data Hazard by Load Instruction



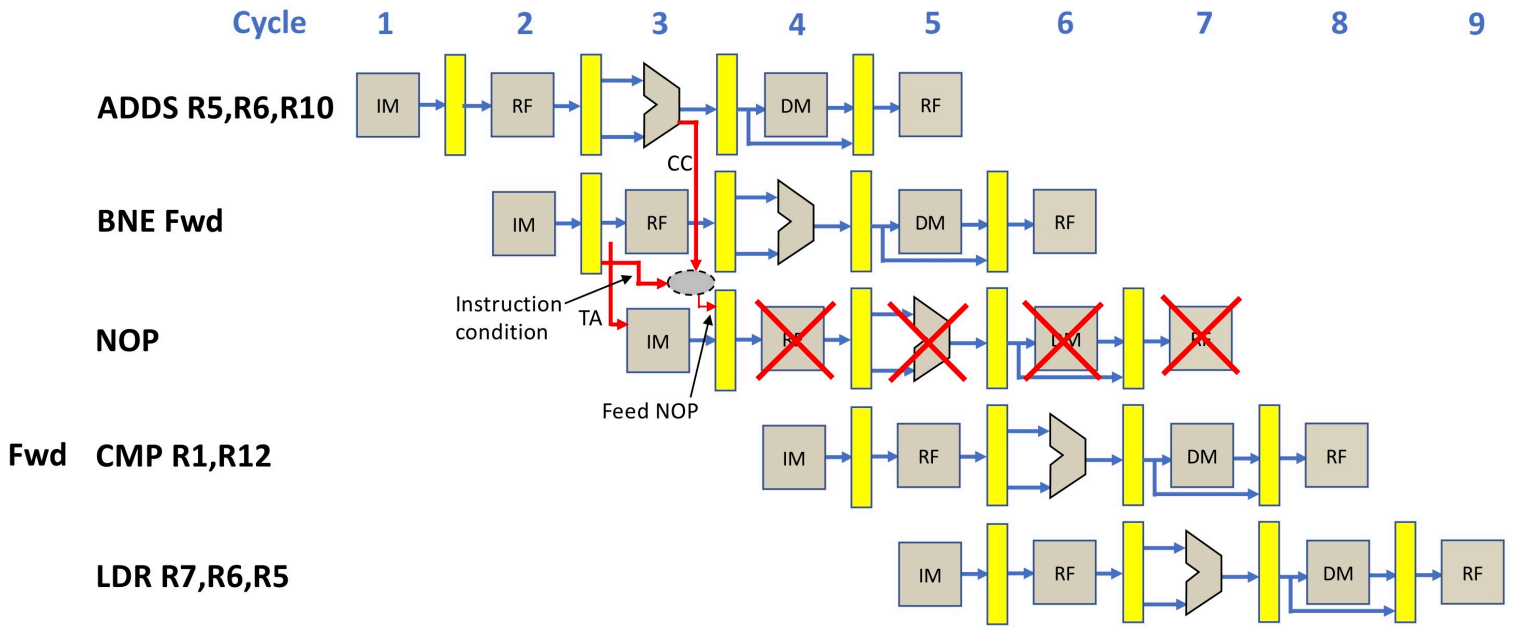
Selecting Forwarding Operand from Memory Output



A Warning Note

- The hardware described thus far is capable of handling all cases of data hazard except one, the case of involving a store instruction.
- A store instruction is the only instruction for which register Rd is a source operand instead of a destination operand.
- For all practical purposes a store instruction has three source operands Rn, Rm and Rd)
- A data hazard between a store instruction on the ID stage and another instruction on the EX, MEM or WB may occur for any of its three source registers.
- Handling this case is left as an exercise for the students.

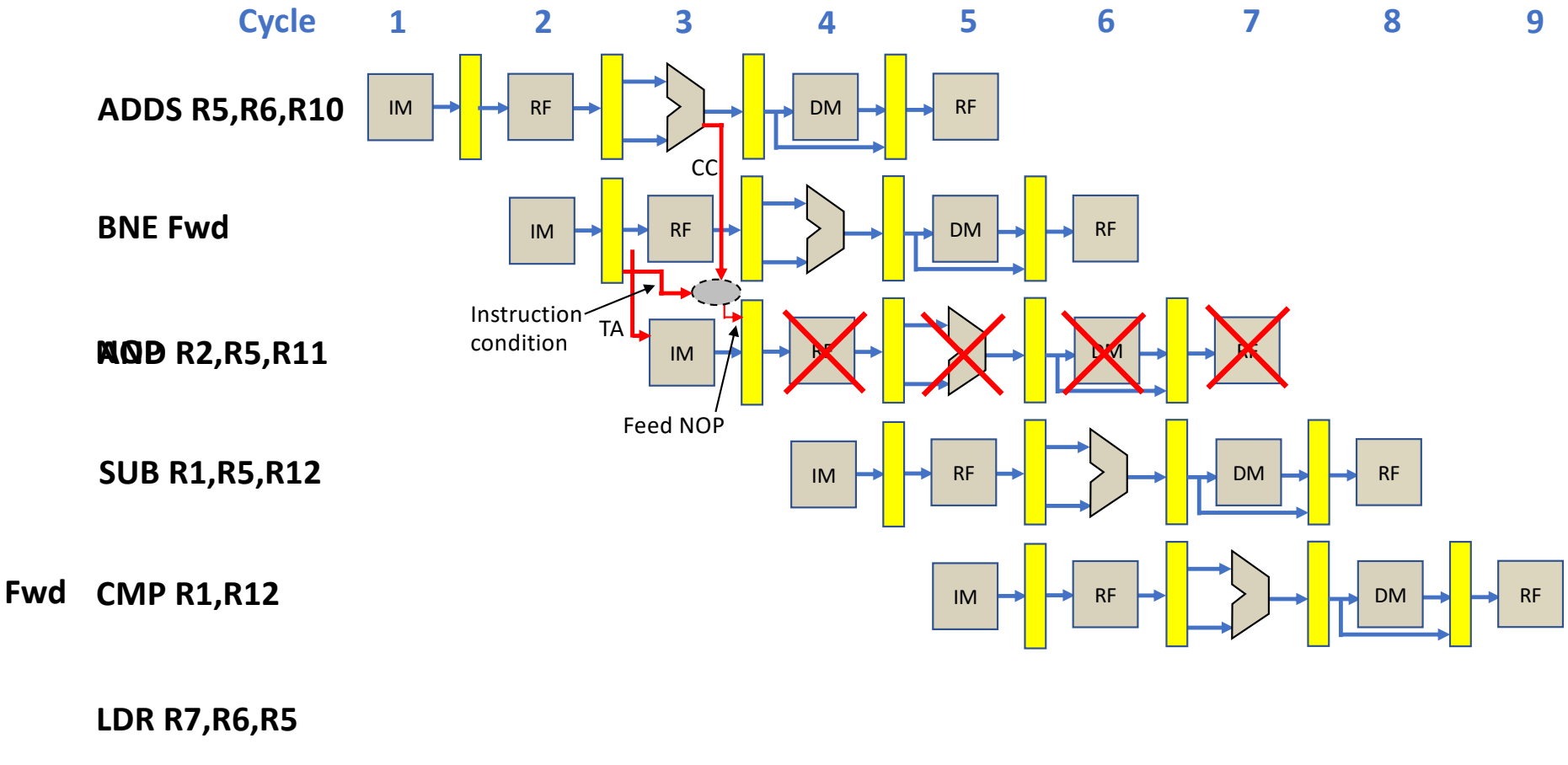
Control Hazards



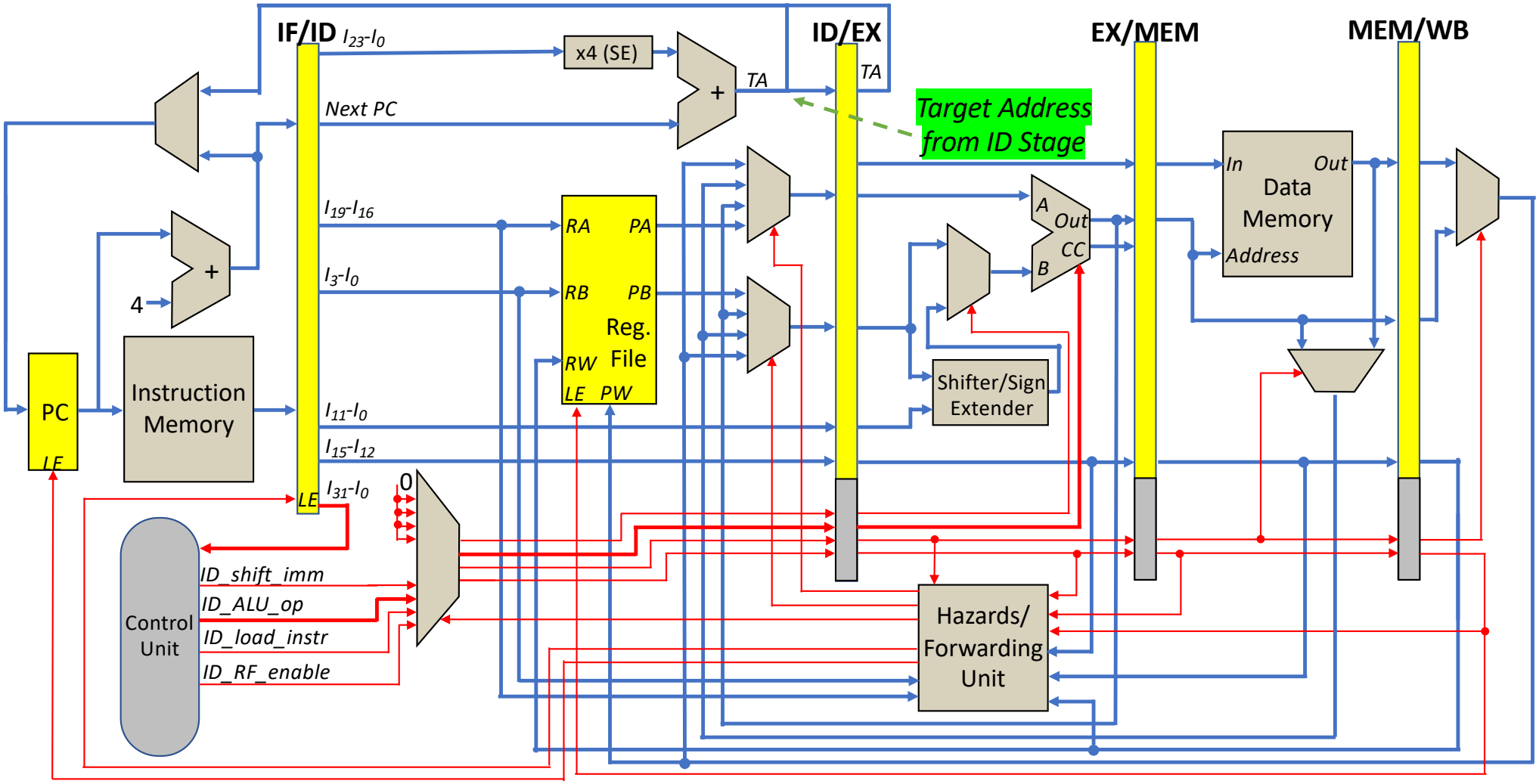
Control Hazards

- **For the pipeline unit developed so far conditional branches are resolved in the EX stage what requires the cancelation of the two instructions following the branch is the branch is taken.**
- **However, when a conditional branch is in the ID stage and the instruction in the EX stage is allowed to modify the condition codes, these values can be forwarded to the ID stage so that the branch can be resolved in that stage.**
- **Since the Target Address is also calculated in the ID stage it can be made available to the IF stage to accomplish the branch.**
- **Resolving the branch in the ID stage requires only the cancellation of the instruction following the branch. Thus, the number of idle cycles caused by a branch taken is reduced from two to one.**

Control Hazard



Sourcing Target Address From ID stage



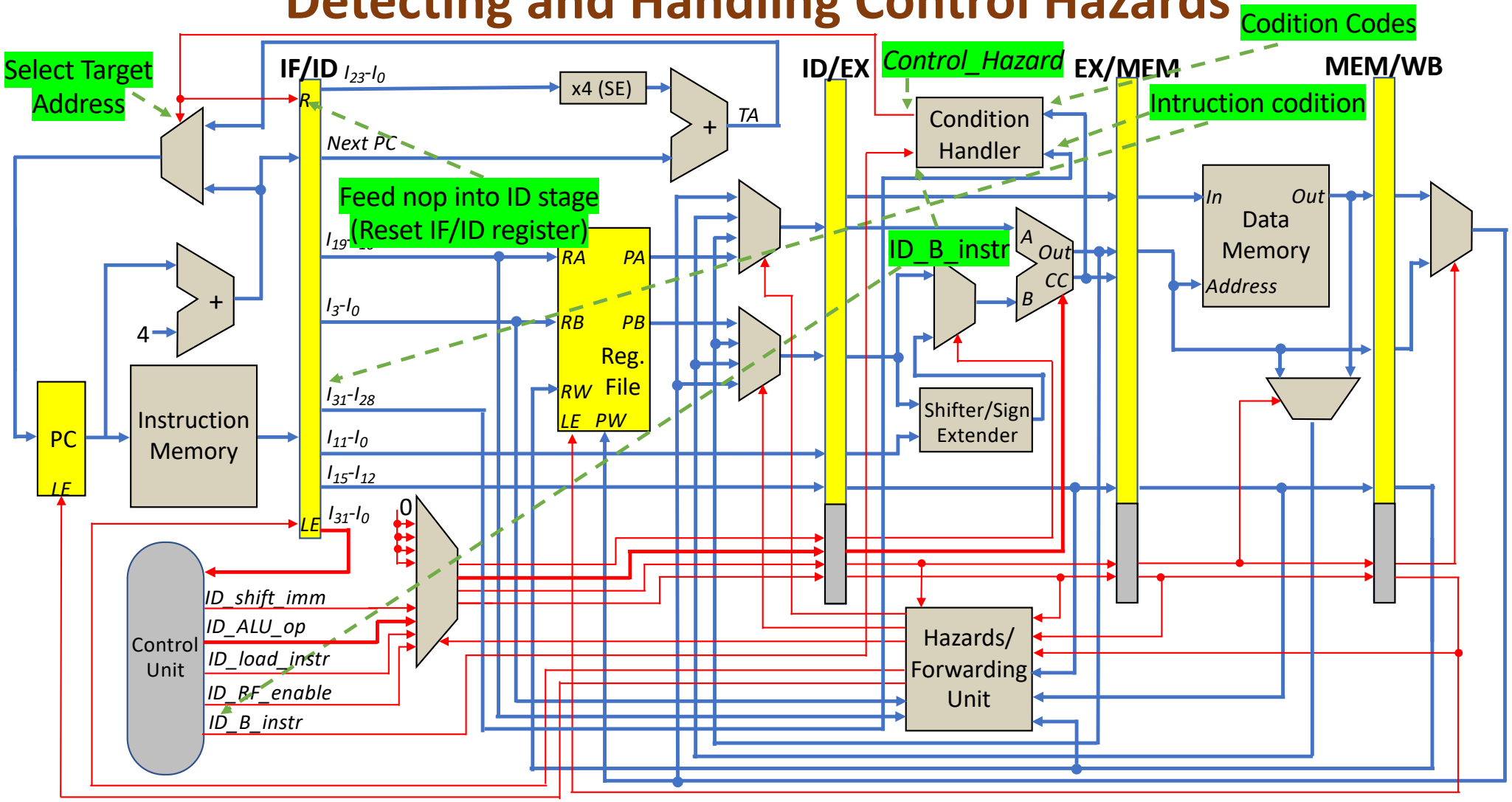
Considerations for Detecting and Handling Control Hazards

- **Let's assume that the instruction in the EX stage is allowed to modify the condition codes.**
- **Let's assume that a *Cond_true* signal can be generated as a function of the Instruction Condition Code (the most significant four bits of the instruction) of the instruction on the ID stage and the Condition Codes generated by the ALU on the EX stage.**
- **If a signal *ID_B_instr* is used to identify a branch instruction in the ID stage, then the control hazard can be detected as follows:**

  if *ID_B_instr* & *Cond_true* then Control Hazard asserted

- **When a control hazard is asserted, the following actions must take place:**
  - **Target Address loaded into PC**
  - **Feed NOP into ID stage (Basically reset IF/ID pipeline register)**

Detecting and Handling Control Hazards



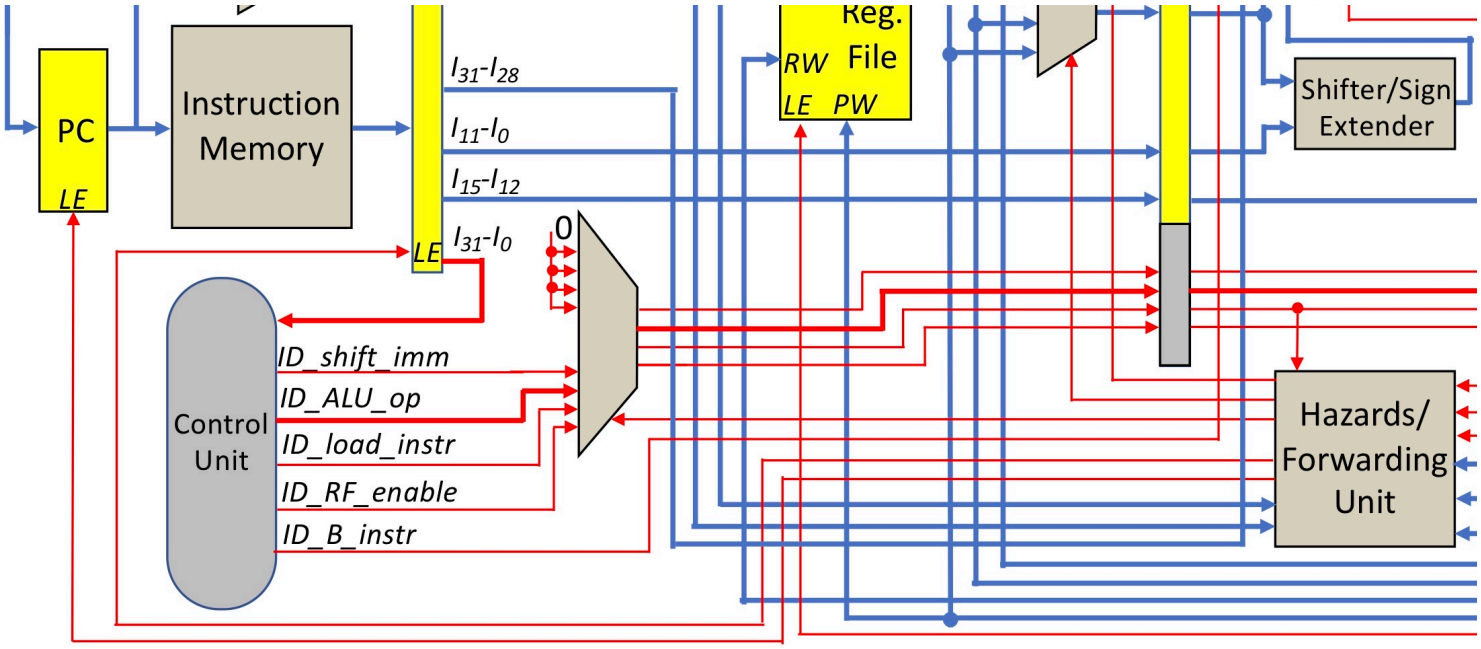
A Warning Note

- The hardware described so far is capable of handling control hazards as long as the instruction following the branch instructions is allowed to modify the condition codes.
- However, this is not always the case since ARM provides a mechanism for disabling a data processing instruction from changing the condition codes.
- If the instruction following the branch does not modify the condition codes, the hardware described so far will need to be modified since it will be using the wrong values of the condition codes to determine if the condition is asserted.
- Handling this case is left as an exercise for the students.

Another Warning Note

- The hardware described so far is capable of handling branch and branch and link instructions with any instruction condition code, and other instructions with a 1110 (Always) instruction condition code.
- If a non branch instruction with an instruction condition code other than 1110 gets to the EX stage it may end up wrongly executed because the hardware does not test the instruction condition to determine if it must be executed or converted to a Nop instruction.
- Handling this case is left as an exercise for the students.

Secondary Control Signals



Control Signals

- **The control signals needed to implement instructions on a PPU are dependent on the architecture to be implemented.**
- **There are basically two types of control signals:**

  Primary Control Signals

  - **generated by the Control Unit**
  - **instruction dependent**

  Secondary Control Signals

  - **generated by the logic boxes**
  - **program dependent**

Primary Control Signals

B_instr : 1 indicates that a branch instruction is in the stage

load_instr : 1 indicates that a load instruction is in the stage

RF_enable : 1 indicates that the instruction in the stage will write the Register File on the WB stage

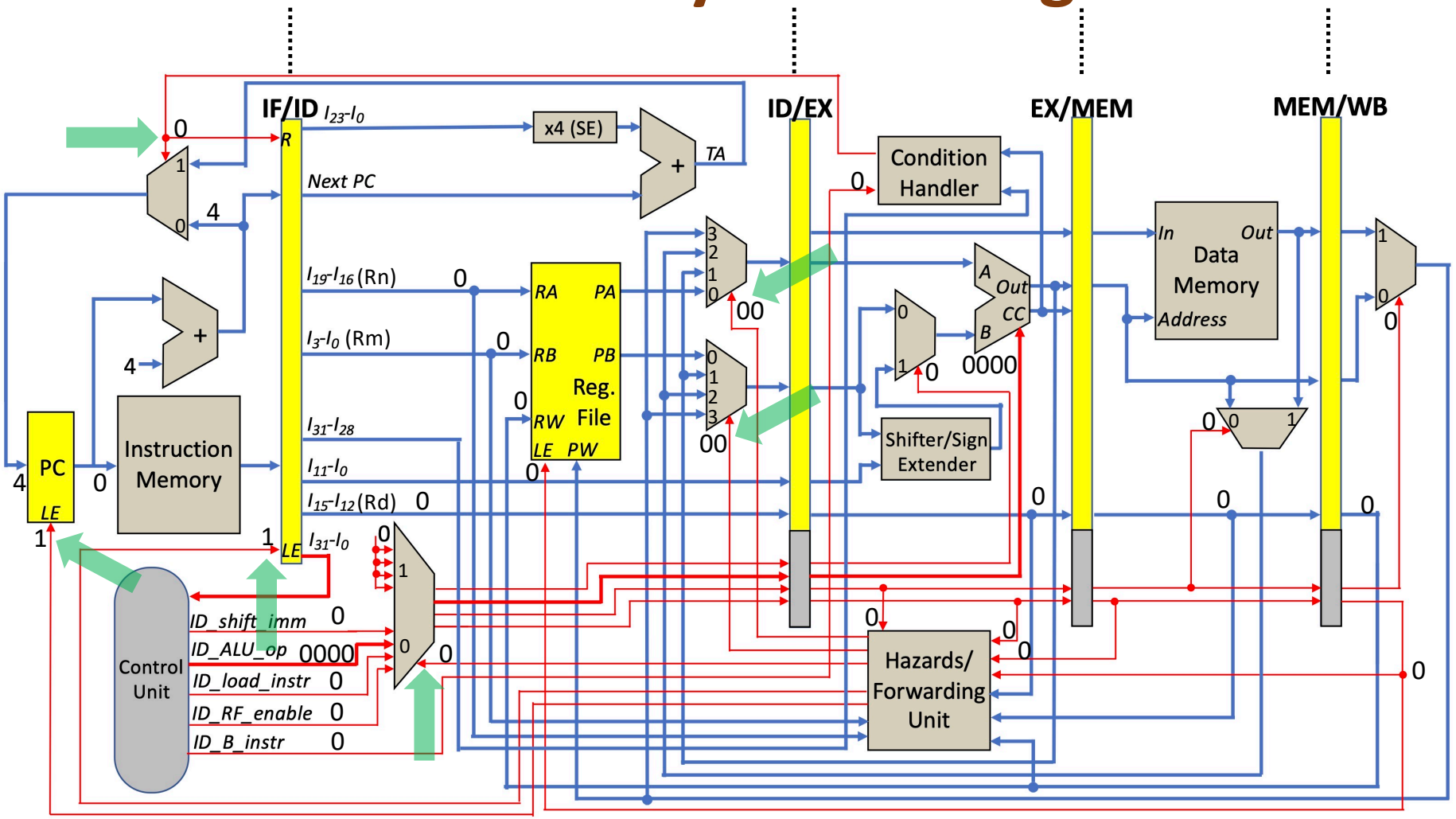
shift_imm : 1 indicates that the second source operand of the instruction in the stage is an immediate or shifted operand

ID_ALU_op: four-bit code that tells the ALU what operation to perform. For the sake of simplicity, it is assumed that the function that corresponds to each code is the same as the opcode field of the data processing instructions (bits I_{24}-I_{21})

Coding Primary Control Signals

	B_instr	*load_instr*	*RF_enable*	*shift_imm*	*ALU_op*
AND R0,R1,#0	0	0	1	1	0000
OR R1,R0,#40	0	0	1	1	1100
ADD R5,R0,R0	0	0	1	0	0100
LDRB R3, [R1,R0]	0	1	1	0	0100
SUBS R3,R3,#1	0	0	1	1	0010
BNE -3	1	0	0	0	XXXX
LDRB R2, [R1, #2]	0	1	1	1	0100

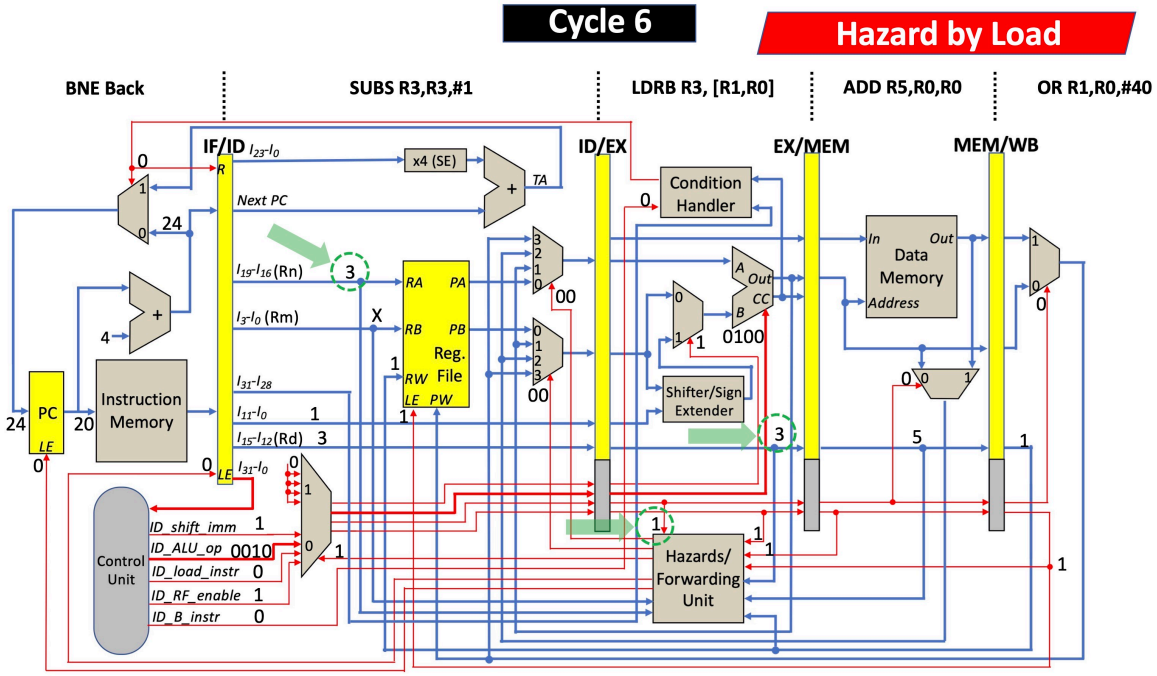
Secondary Control Signals



A Warning Note

- In order to develop a PPU hardware that can fully support the execution of ARM code, additional control signals will certainly be needed.
- Those signals will emerge as each instruction is encoded and every architectural requirement is implemented.
- Identifying the additional control signals needed is left as an exercise for the students.

Pipelined Processing Unit Simulation



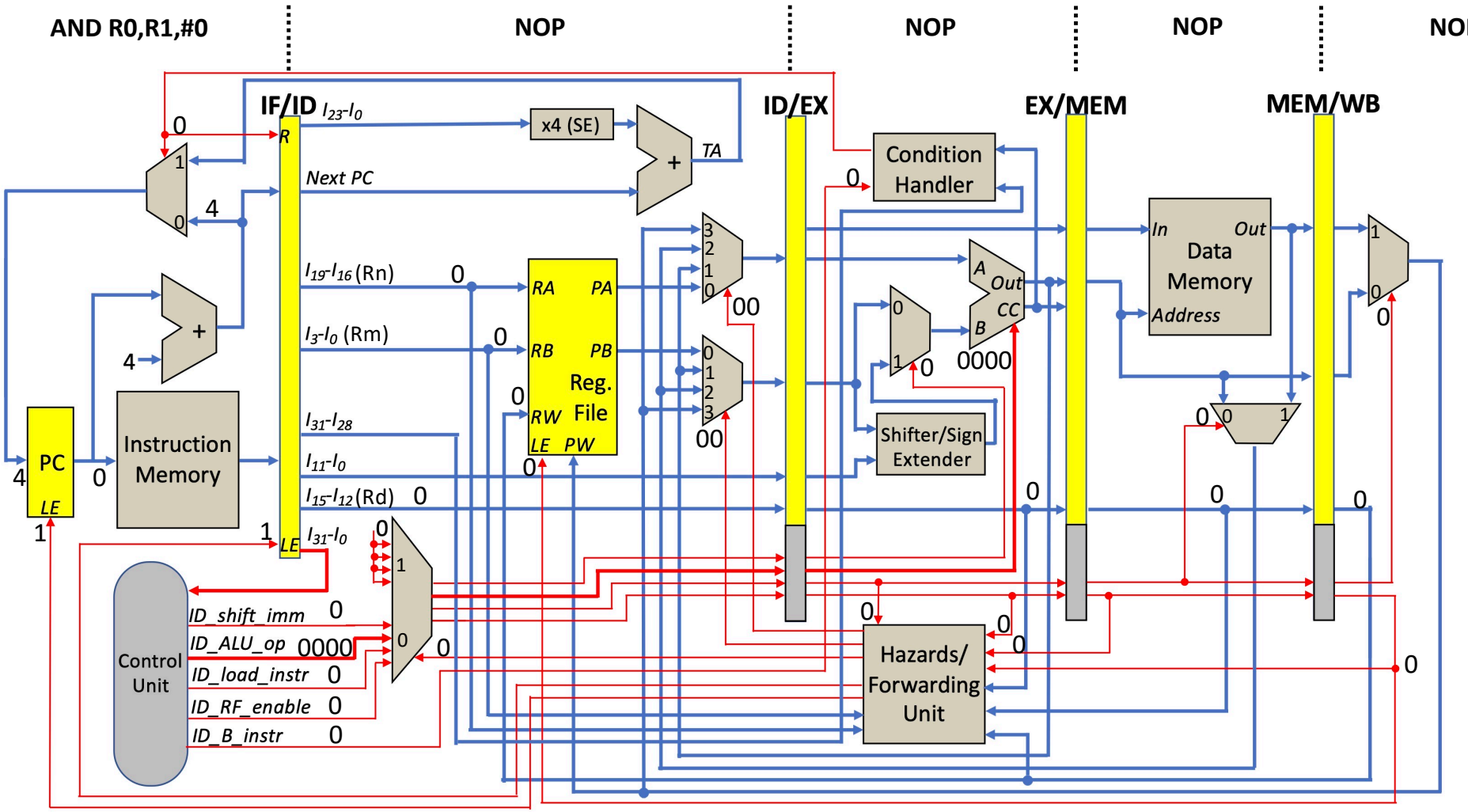
Details of Simulation

- **Will focus on control signals.**
- **The PC, Register File and the four pipeline registers are synchronized with the rising edge of a clock signal.**
- **Will run the following piece of ARM code:**

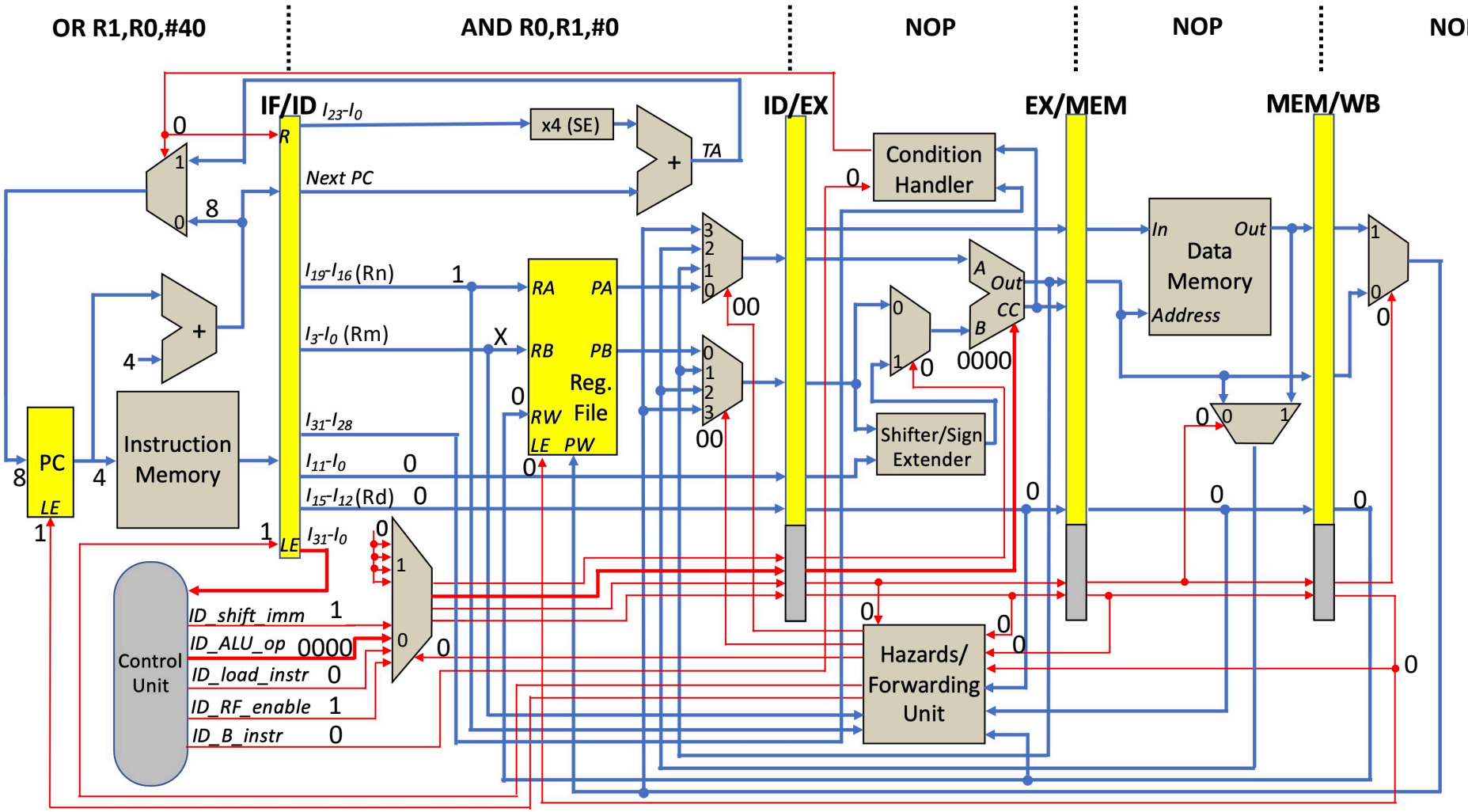
```
        AND R0,R1,#0
        OR R1,R0,#40
Back    ADD R5,R0,R0
        LDRB R3, [R1,R0]
        SUBS R3,R3,#1
        BNE Back
        LDRB R2, [R1, #2]
```

- **The program begins at memory location 0 after a reset that have placed a NOP on the ID, EX , MEM and WB stages.**
- **BNE will branch the first time it is executed.**

Cycle 1

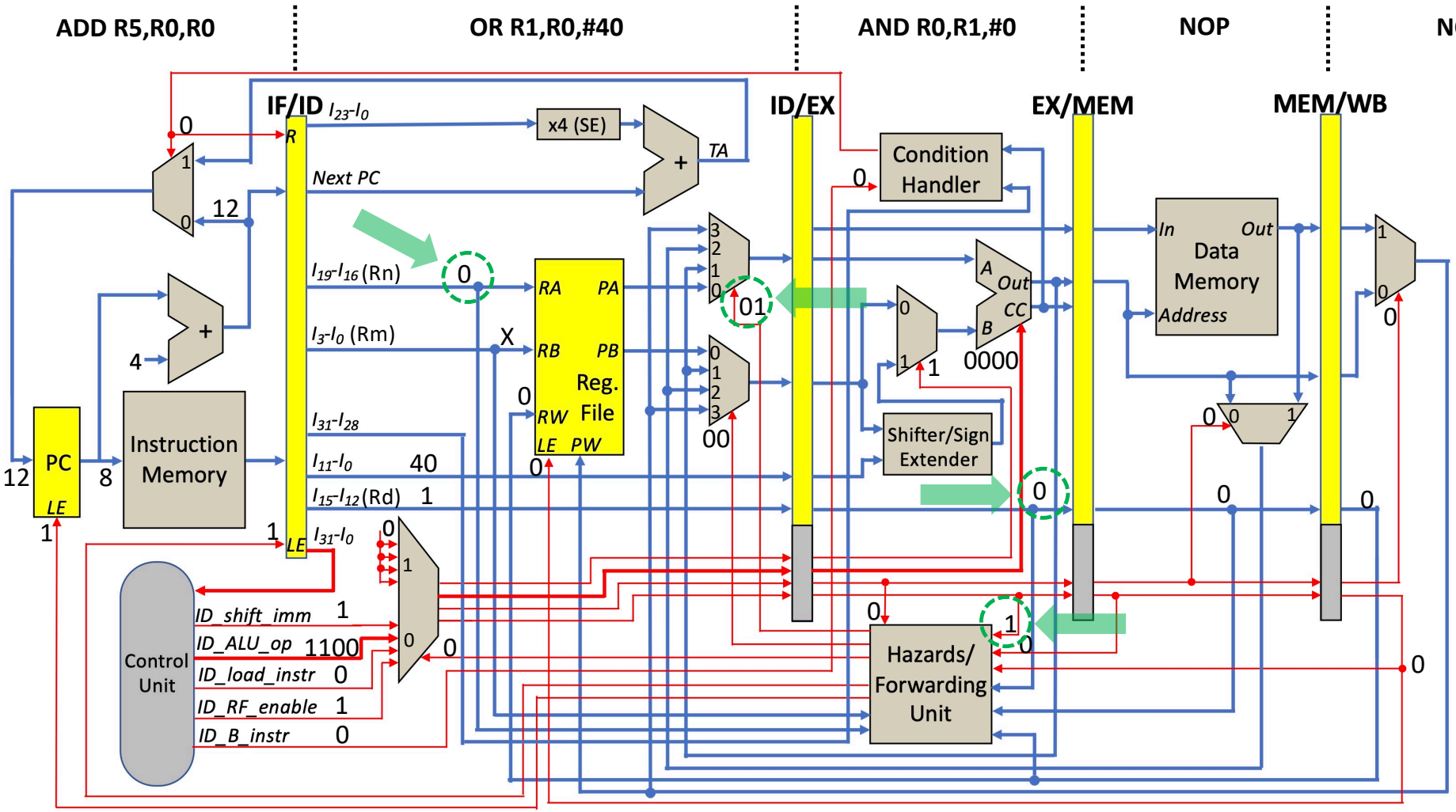

AND R0,R1,#0
NOP
NOP
NOP
NOP
IF/ID
ID/EX
EX/MEM
MEM/WB
PC
LE
Instruction Memory
Next PC
x4 (SE)
TA
Reg. File
RA
RB
RW
LE
PA
PB
PW
Condition Handler
Shifter/Sign Extender
Data Memory
In
Out
Address
Hazards/ Forwarding Unit
Control Unit
ID_shift_imm
ID_ALU_op
ID_load_instr
ID_RF_enable
ID_B_instr

Cycle 2


OR R1,R0,#40
AND R0,R1,#0
NOP
NOP
NOP
IF/ID
ID/EX
EX/MEM
MEM/WB
I_{23}-I_0
x4 (SE)
TA
Next PC
I_{19}-I_{16} (Rn)
I_3-I_0 (Rm)
I_{31}-I_{28}
I_{11}-I_0
I_{15}-I_{12} (Rd)
I_{31}-I_0
RA
PA
RB
PB
RW
PW
LE
Reg. File
PC
Instruction Memory
Condition Handler
Shifter/Sign Extender
A
B
Out
CC
0000
In
Out
Data Memory
Address
Control Unit
ID_shift_imm
ID_ALU_op
ID_load_instr
ID_RF_enable
ID_B_instr
Hazards/ Forwarding Unit

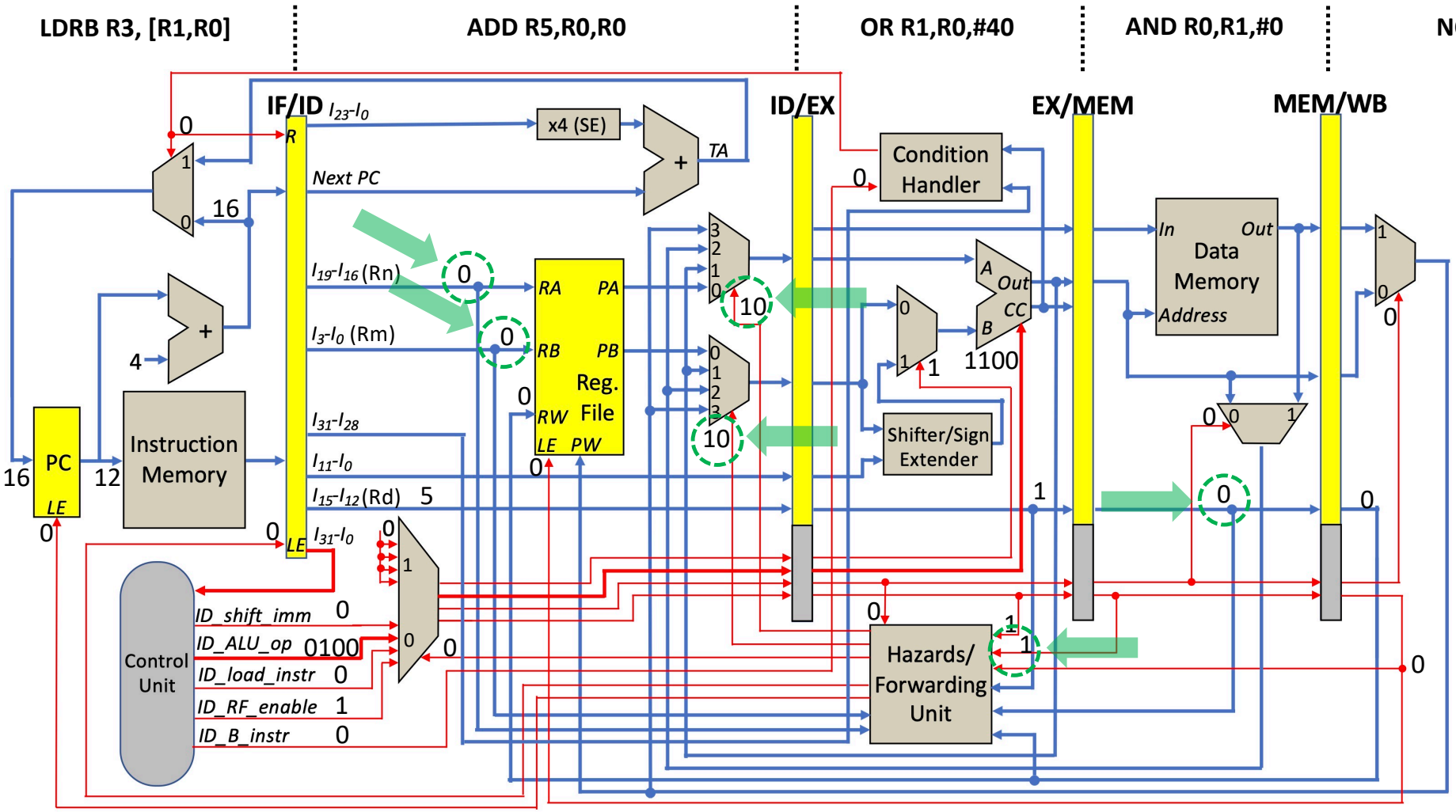
Cycle 3

Forwarding R0 from EX



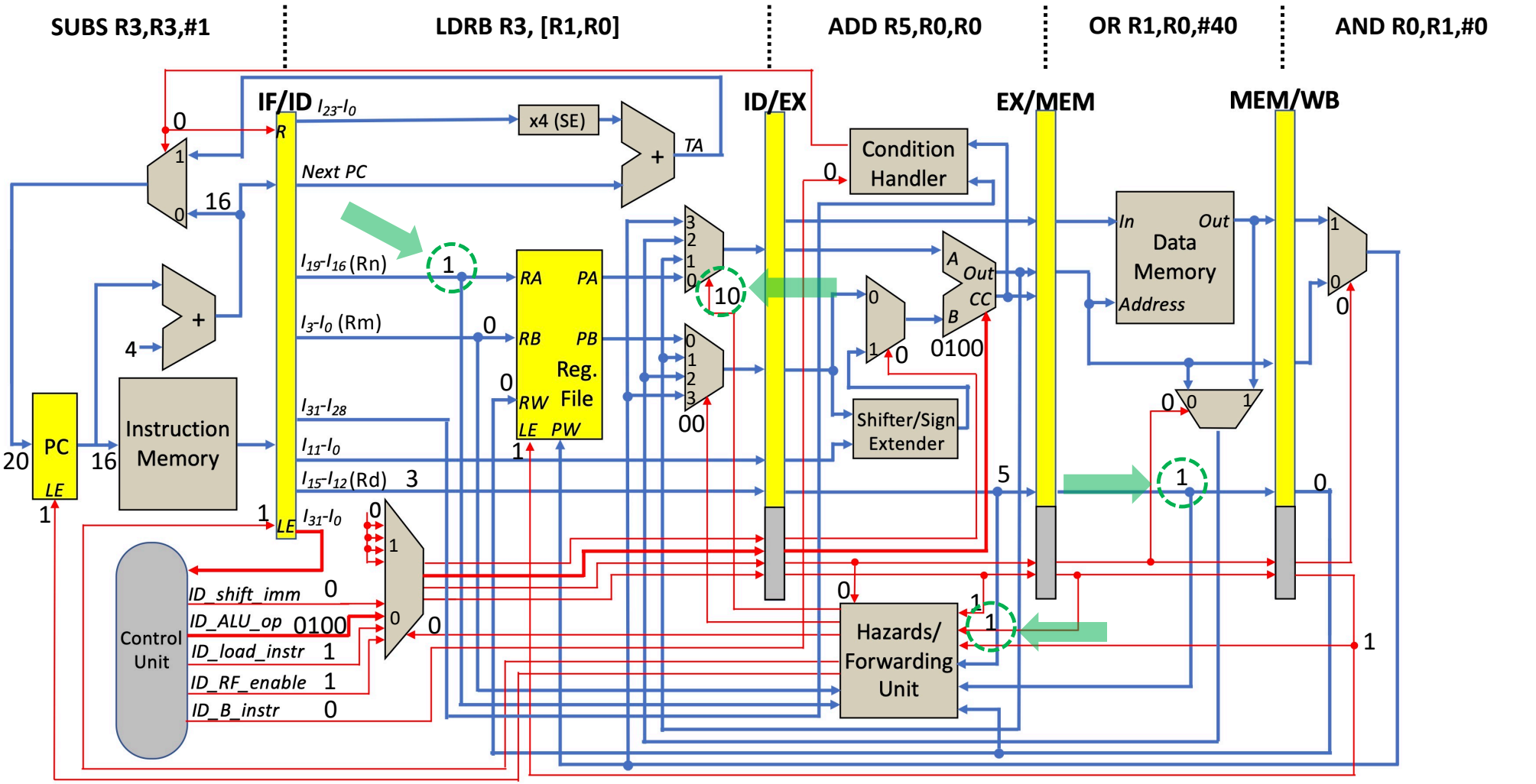
Cycle 4

Forwarding R0 from MEM



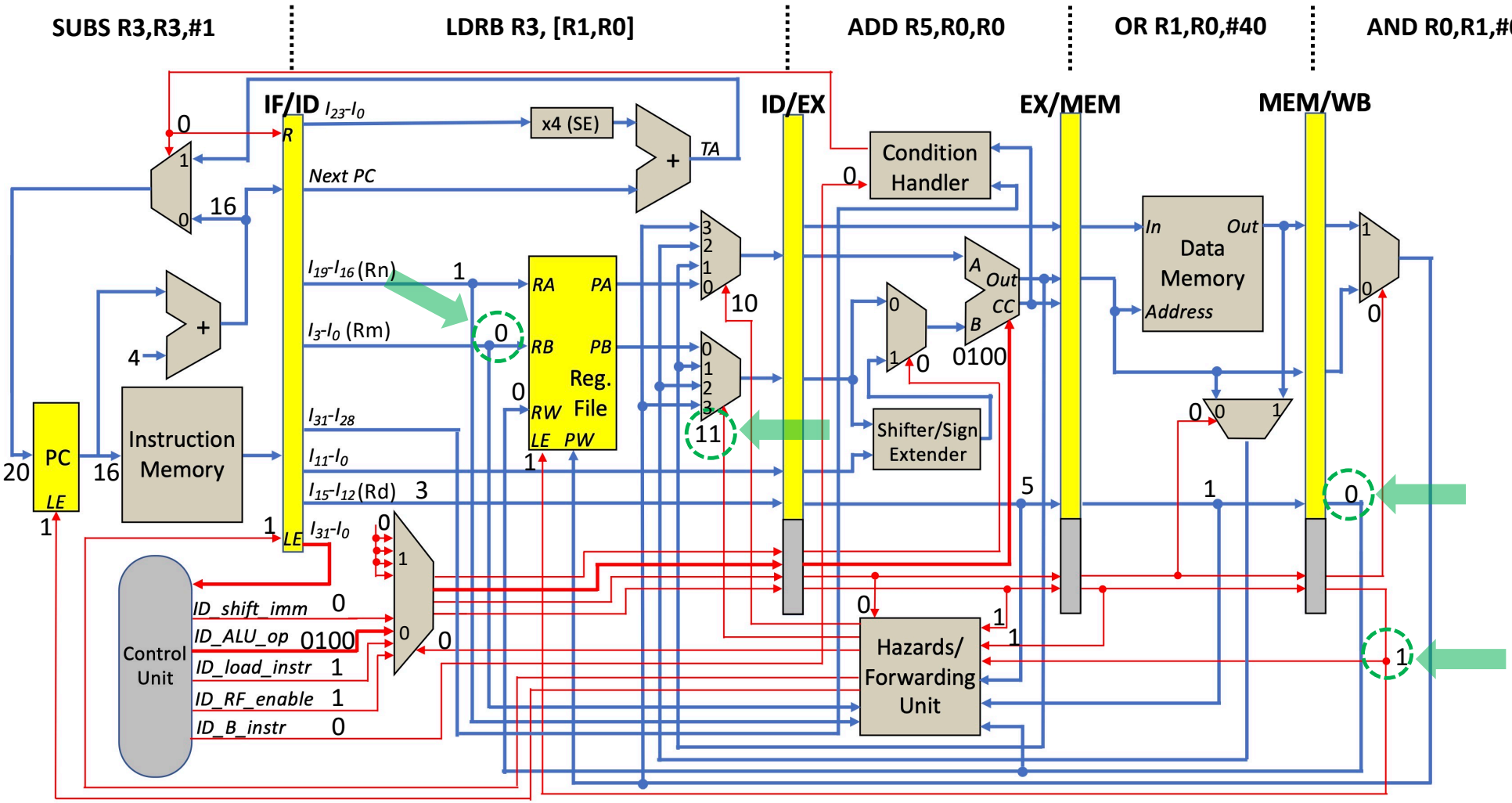
Cycle 5

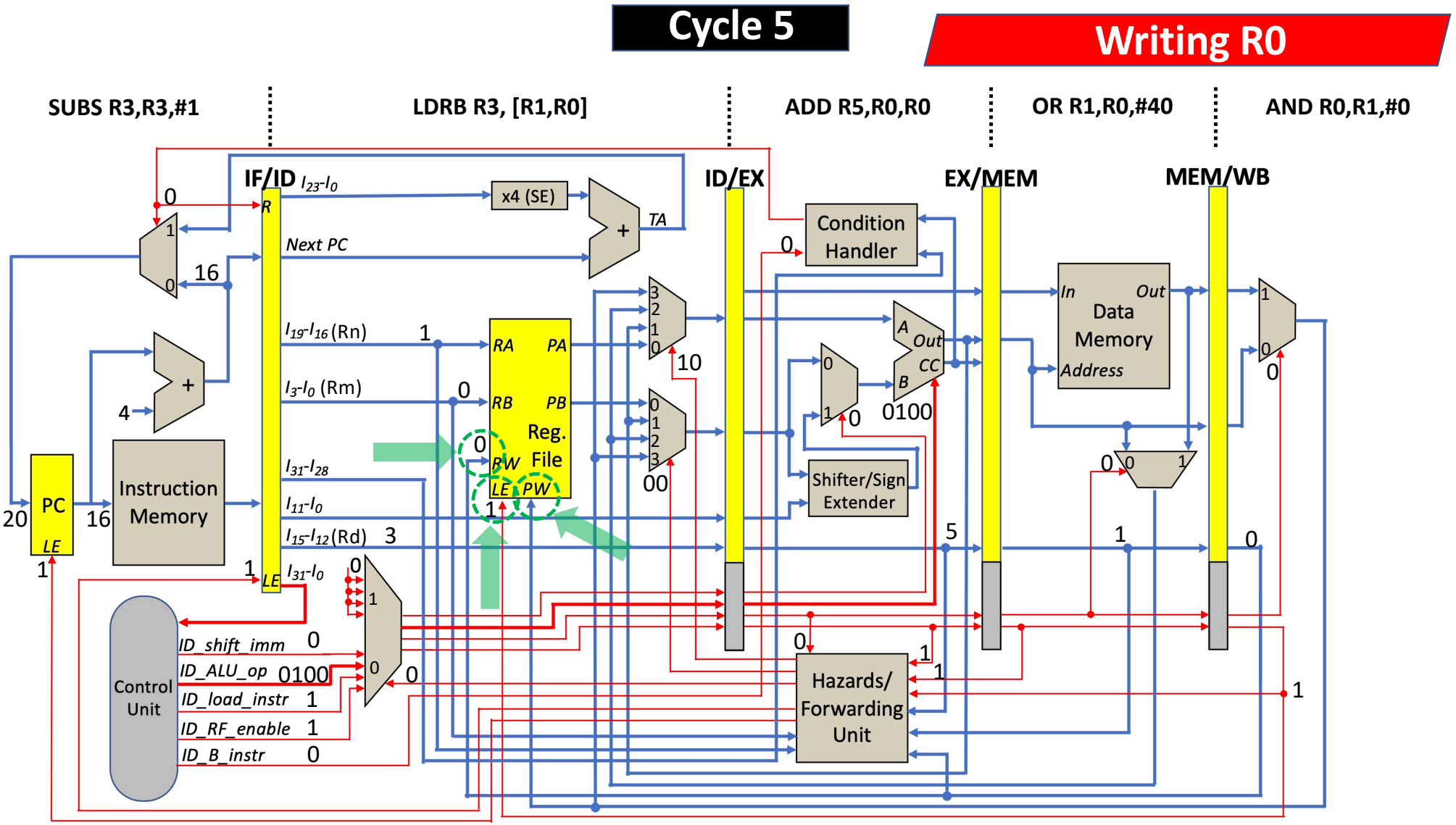
Forwarding R1 from MEM



Cycle 5

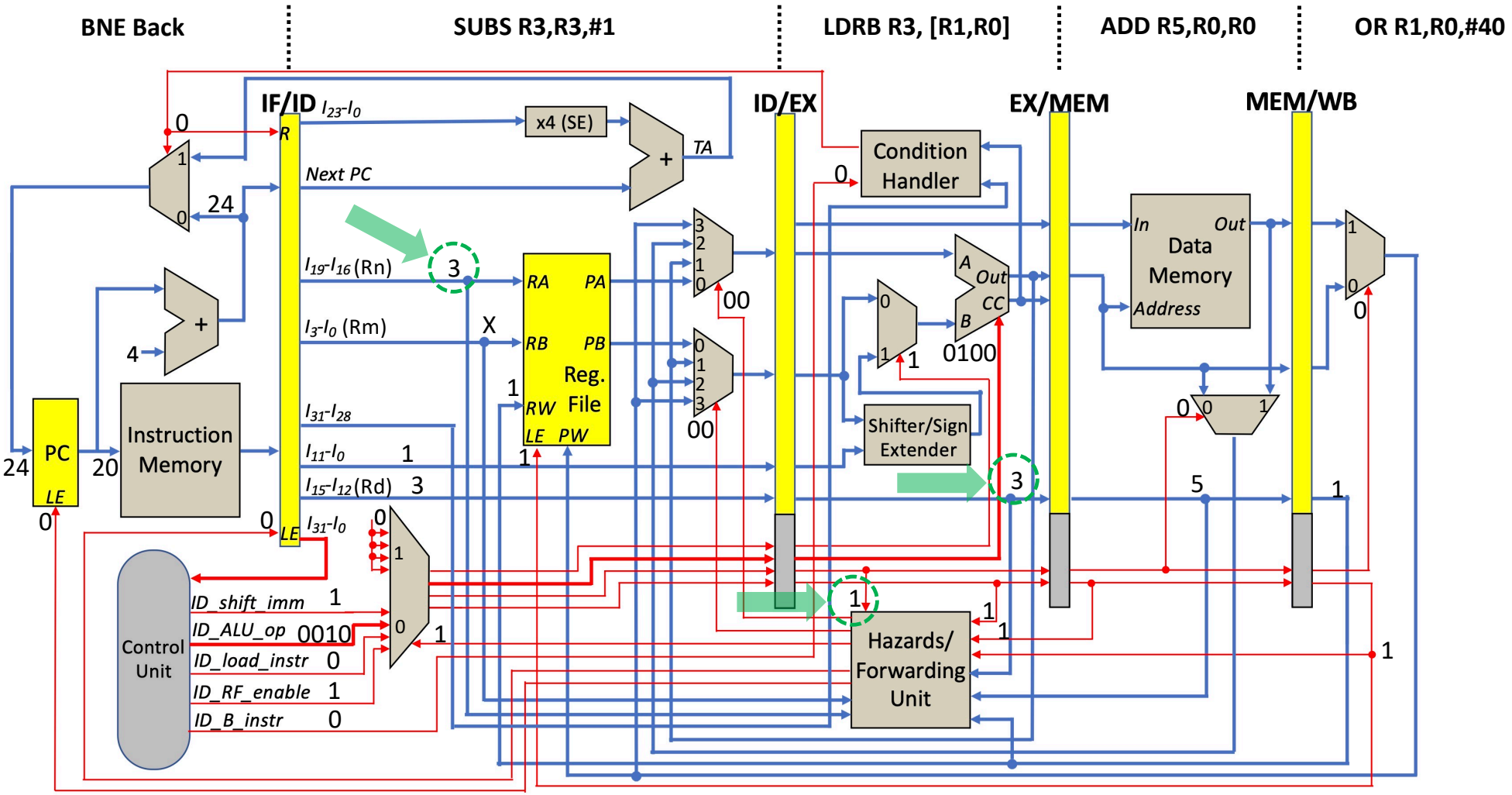
Forwarding R0 from WB

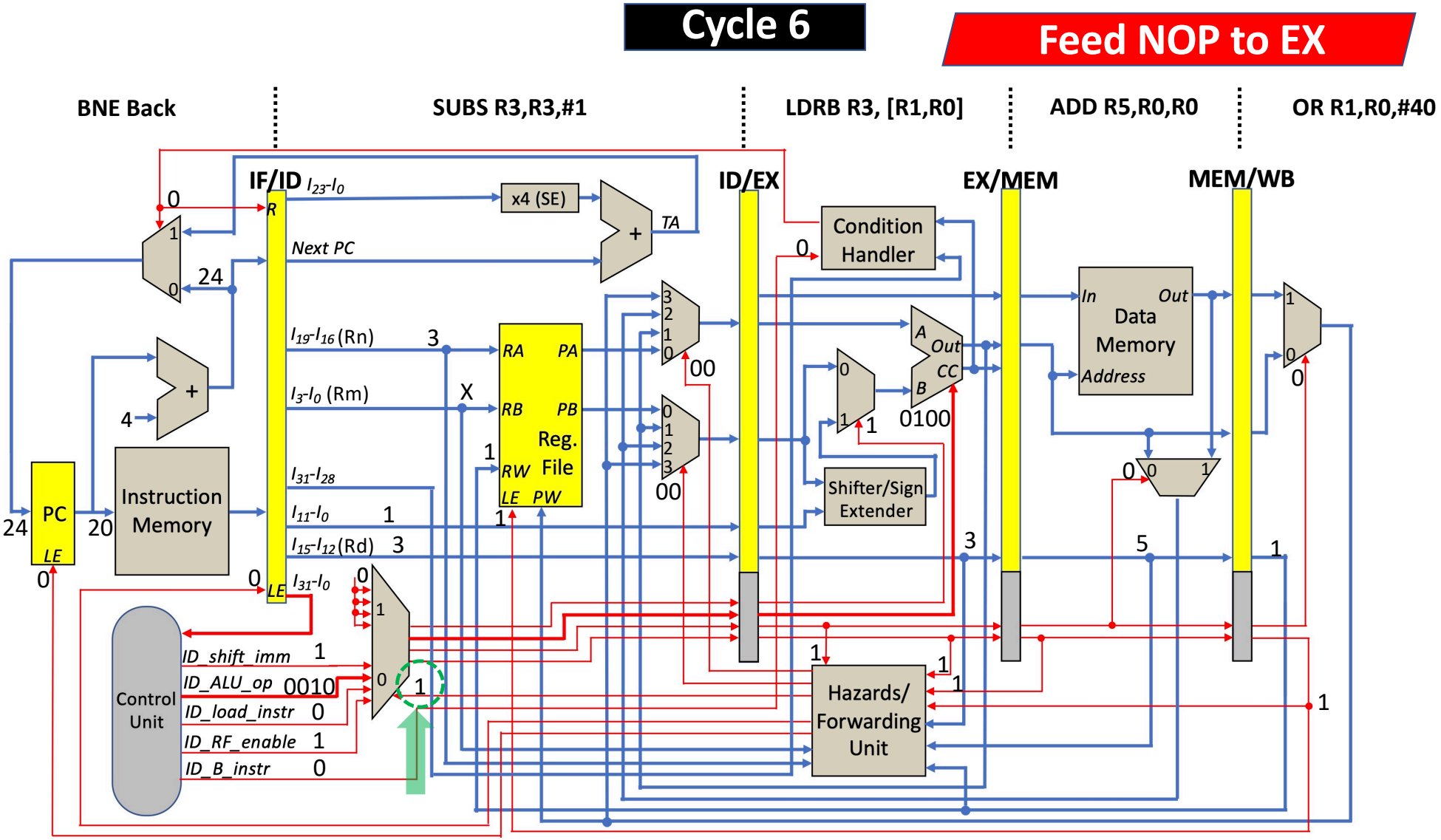


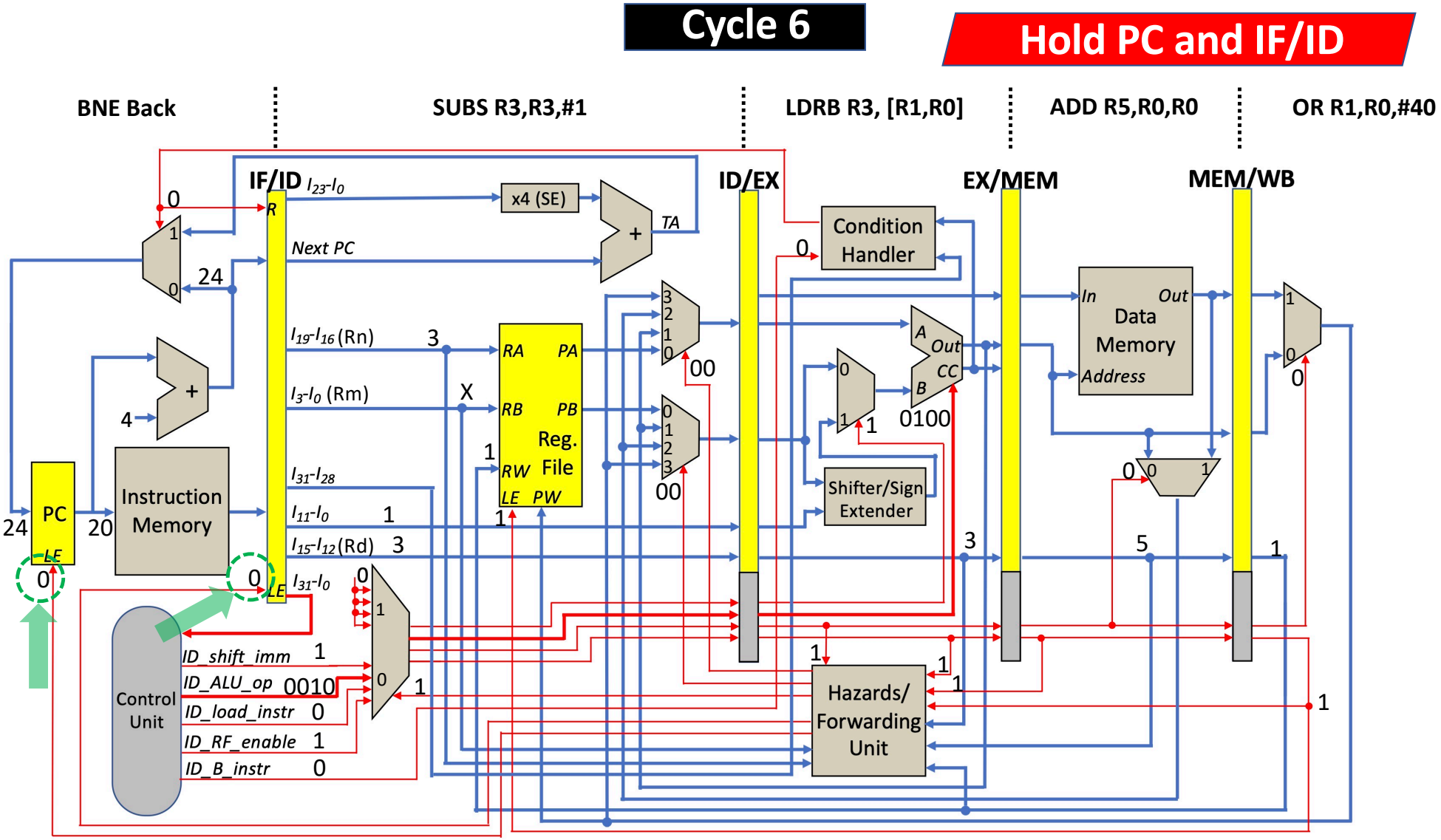

Cycle 5
Writing R0
SUBS R3,R3,#1
LDRB R3, [R1,R0]
ADD R5,R0,R0
OR R1,R0,#40
AND R0,R1,#0
IF/ID I_{23}-I_0
ID/EX
EX/MEM
MEM/WB
x4 (SE)
TA
Next PC
Condition Handler
I_{19}-I_{16} (Rn)
I_3-I_0 (Rm)
I_{31}-I_{28}
I_{11}-I_0
I_{15}-I_{12} (Rd)
I_{31}-I_0
RA
PA
RB
PB
RW
PW
Reg. File
Shifter/Sign Extender
A
Out
CC
B
In
Out
Data Memory
Address
PC
LE
Instruction Memory
Control Unit
ID_shift_imm
ID_ALU_op
ID_load_instr
ID_RF_enable
ID_B_instr
Hazards/ Forwarding Unit

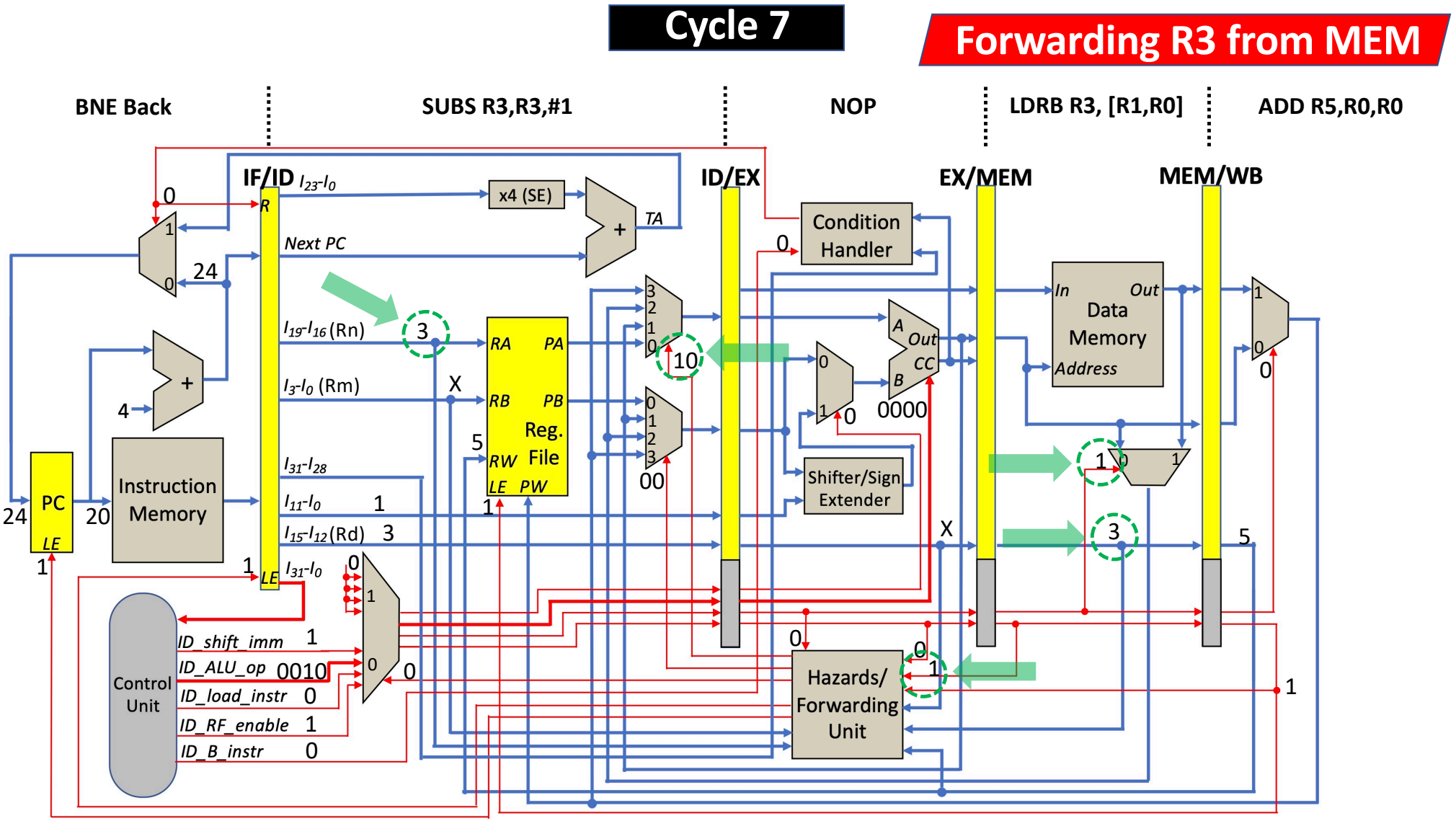
Cycle 6

Hazard by Load



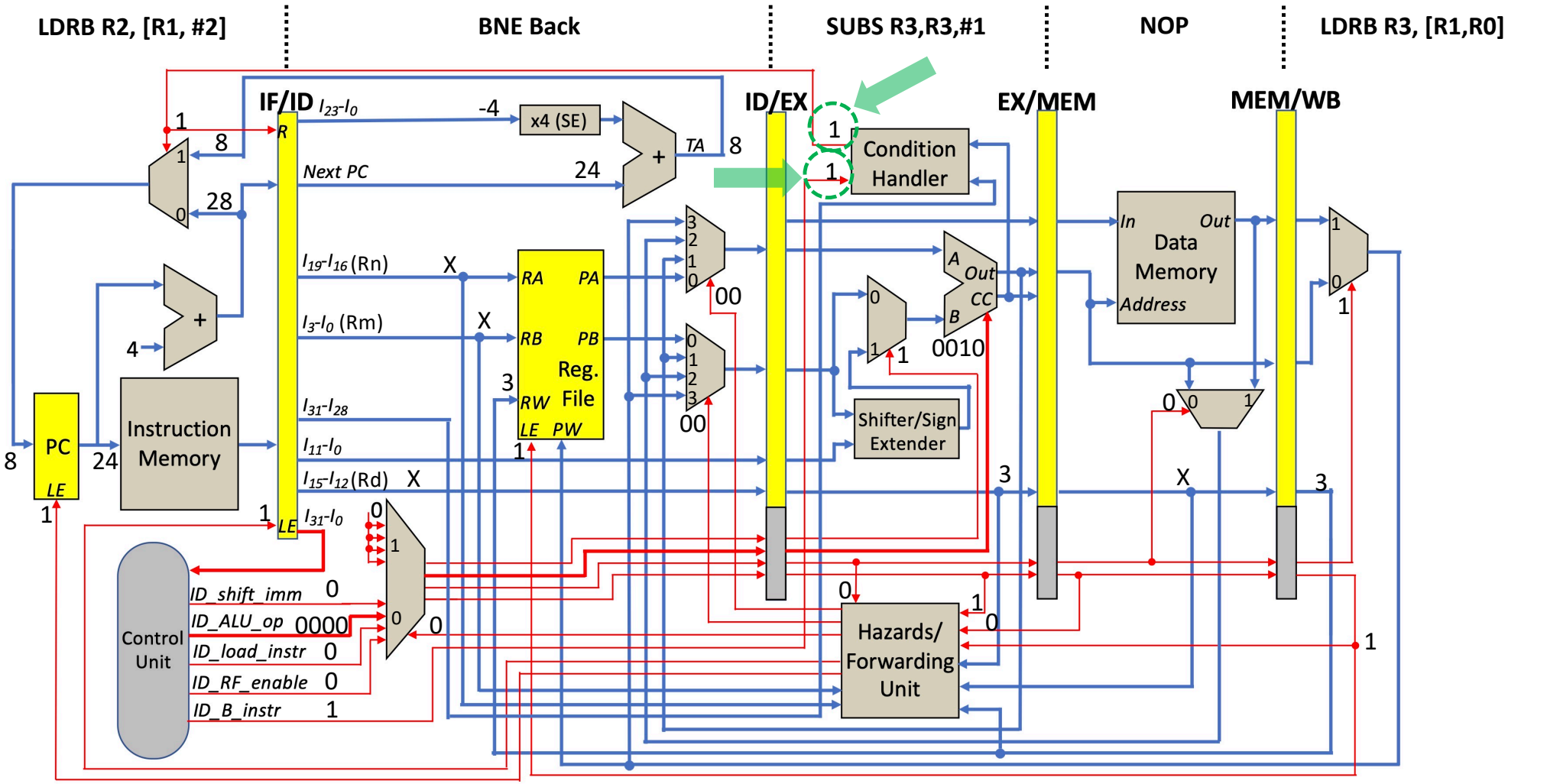

Cycle 6
Feed NOP to EX
BNE Back
SUBS R3,R3,#1
LDRB R3, [R1,R0]
ADD R5,R0,R0
OR R1,R0,#40
IF/ID I_{23}-I_0
ID/EX
EX/MEM
MEM/WB
x4 (SE)
TA
Next PC
Condition Handler
I_{19}-I_{16} (Rn)
I_3-I_0 (Rm)
RA PA
RB PB
Reg. File
RW
LE PW
I_{31}-I_{28}
I_{11}-I_0
I_{15}-I_{12} (Rd)
I_{31}-I_0
PC
LE
Instruction Memory
Shifter/Sign Extender
A Out
B CC
0100
In Out
Data Memory
Address
Control Unit
ID_shift_imm
ID_ALU_op
ID_load_instr
ID_RF_enable
ID_B_instr
0010
Hazards/ Forwarding Unit


Cycle 6
Hold PC and IF/ID
BNE Back
SUBS R3,R3,#1
LDRB R3, [R1,R0]
ADD R5,R0,R0
OR R1,R0,#40
IF/ID I_{23}-I_0
ID/EX
EX/MEM
MEM/WB
x4 (SE)
TA
Next PC
Condition Handler
I_{19}-I_{16} (Rn)
I_3-I_0 (Rm)
RA PA
RB PB
Reg. File
RW
LE PW
Data Memory
In Out
Address
A Out
B CC
0100
PC
LE
Instruction Memory
I_{31}-I_{28}
I_{11}-I_0
I_{15}-I_{12} (Rd)
I_{31}-I_0
Shifter/Sign Extender
Hazards/ Forwarding Unit
Control Unit
ID_shift_imm
ID_ALU_op
ID_load_instr
ID_RF_enable
ID_B_instr


Cycle 7
Forwarding R3 from MEM
BNE Back
SUBS R3,R3,#1
NOP
LDRB R3, [R1,R0]
ADD R5,R0,R0
IF/ID
ID/EX
EX/MEM
MEM/WB
PC
LE
Instruction Memory
x4 (SE)
Next PC
TA
Condition Handler
Reg. File
RA
RB
RW
LE
PA
PB
PW
Shifter/Sign Extender
Data Memory
In
Out
Address
Control Unit
ID_shift_imm
ID_ALU_op
ID_load_instr
ID_RF_enable
ID_B_instr
Hazards/ Forwarding Unit

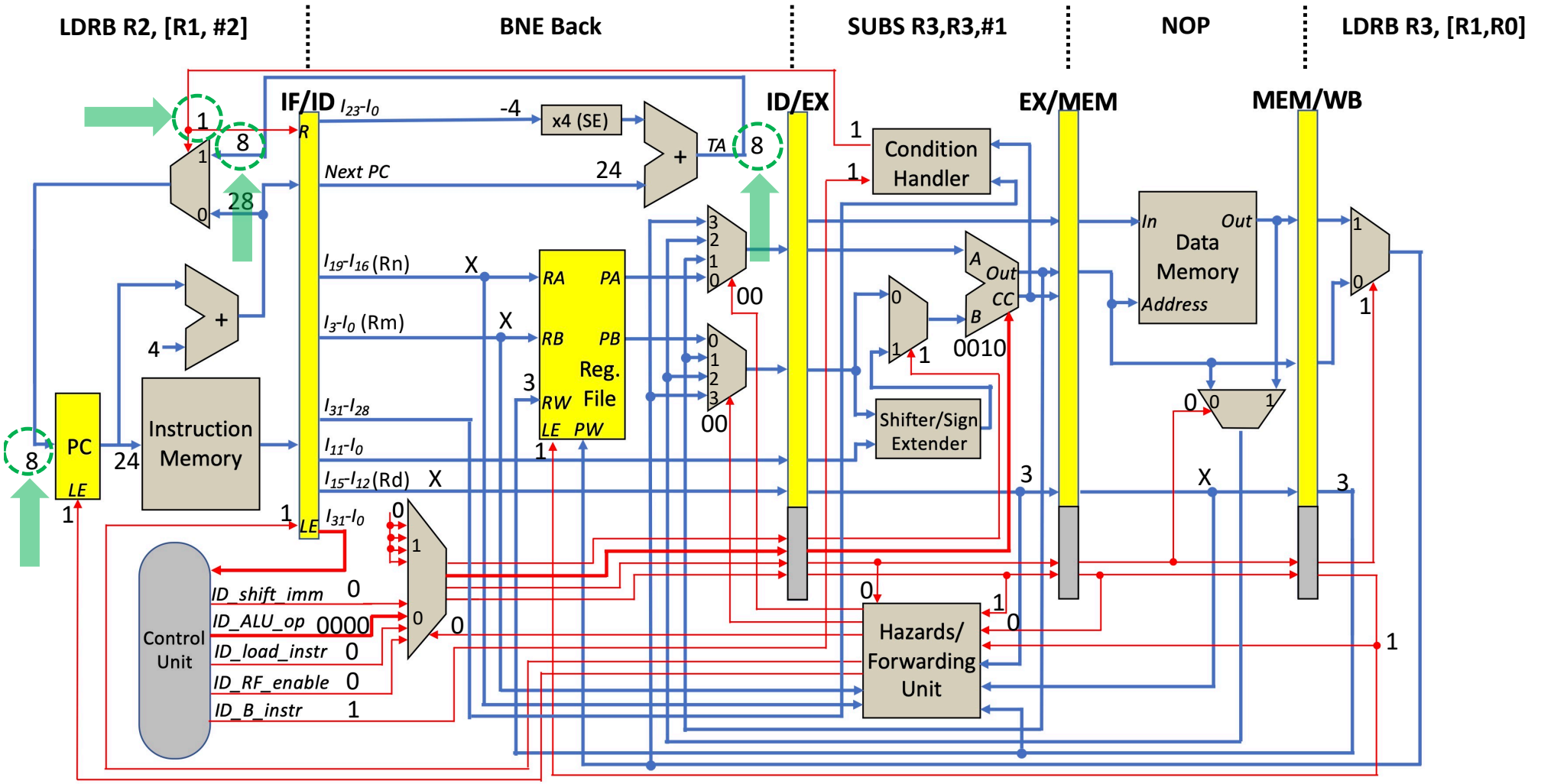
Cycle 8

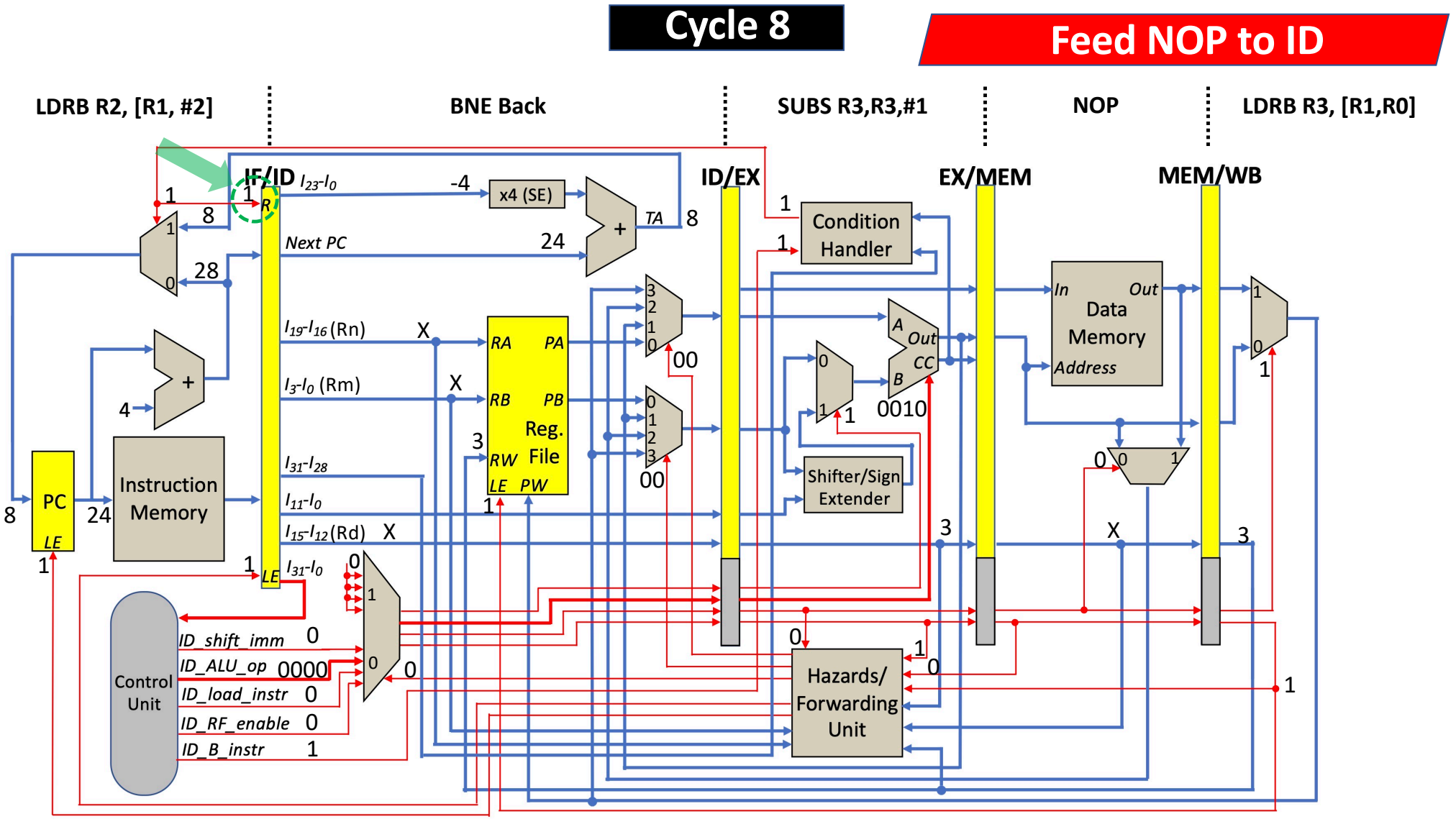
Control Hazard

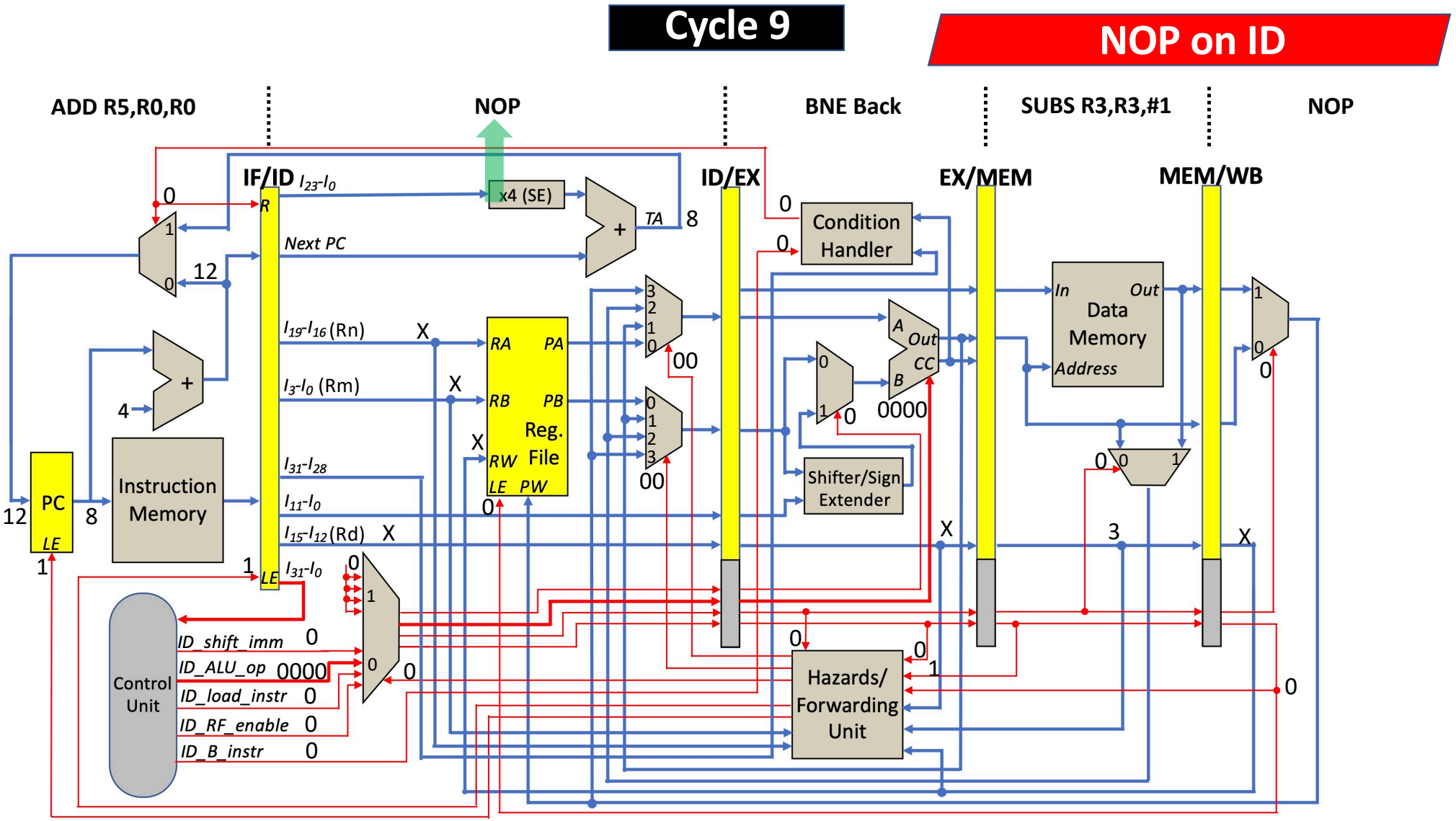


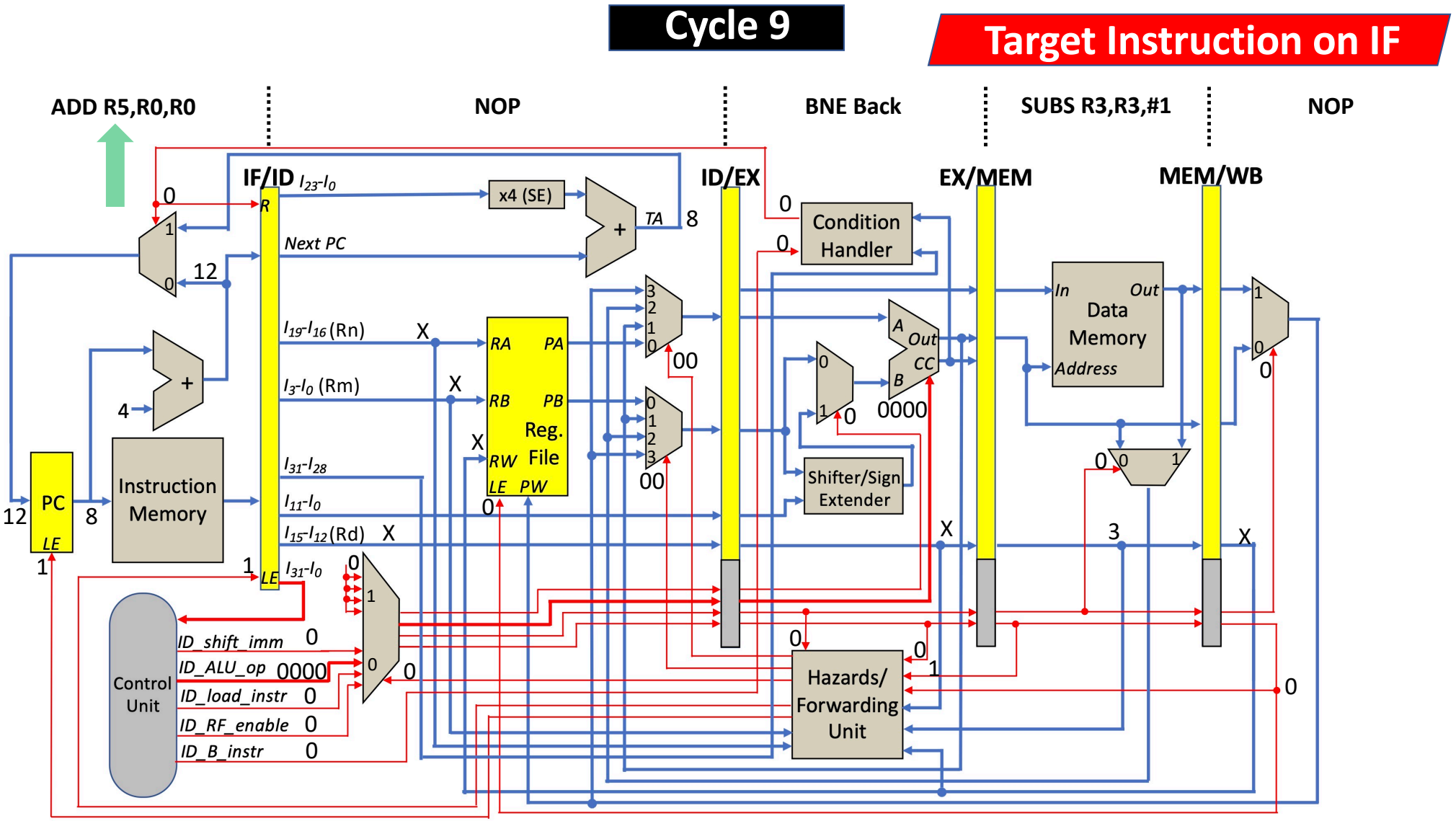
Cycle 8

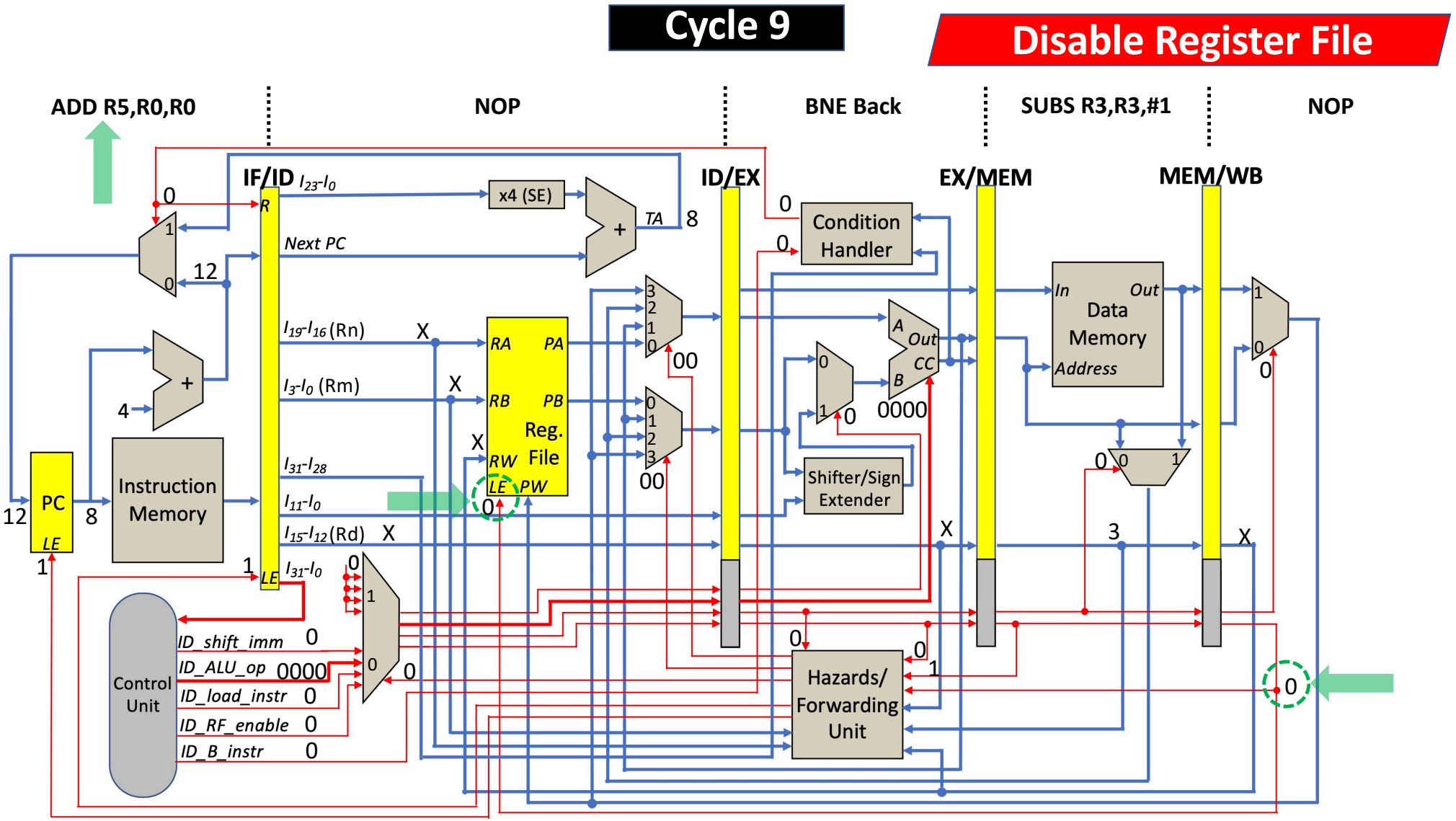
Load PC with Target Address




Cycle 8
Feed NOP to ID
LDRB R2, [R1, #2]
BNE Back
SUBS R3,R3,#1
NOP
LDRB R3, [R1,R0]
IF/ID
ID/EX
EX/MEM
MEM/WB
PC
LE
Instruction Memory
Next PC
x4 (SE)
-4
TA
Reg. File
RA
RB
RW
LE
PA
PB
PW
Condition Handler
Shifter/Sign Extender
Data Memory
In
Out
Address
A
B
Out
CC
0010
Hazards/ Forwarding Unit
Control Unit
ID_shift_imm
ID_ALU_op
ID_load_instr
ID_RF_enable
ID_B_instr
0000


Cycle 9
NOP on ID
ADD R5,R0,R0
NOP
BNE Back
SUBS R3,R3,#1
NOP
IF/ID
ID/EX
EX/MEM
MEM/WB
PC
LE
Instruction Memory
Next PC
x4 (SE)
TA
Condition Handler
Reg. File
RA
PA
RB
PB
RW
PW
Shifter/Sign Extender
Data Memory
In
Out
Address
Control Unit
ID_shift_imm
ID_ALU_op
ID_load_instr
ID_RF_enable
ID_B_instr
Hazards/ Forwarding Unit


Cycle 9
Target Instruction on IF
ADD R5,R0,R0
NOP
BNE Back
SUBS R3,R3,#1
NOP
IF/ID
ID/EX
EX/MEM
MEM/WB
PC
LE
Instruction Memory
Next PC
x4 (SE)
TA
Condition Handler
Reg. File
RA
RB
RW
LE
PA
PB
PW
Shifter/Sign Extender
Data Memory
In
Out
Address
Hazards/ Forwarding Unit
Control Unit
ID_shift_imm
ID_ALU_op
ID_load_instr
ID_RF_enable
ID_B_instr


Cycle 9
Disable Register File
ADD R5,R0,R0
NOP
BNE Back
SUBS R3,R3,#1
NOP
IF/ID
ID/EX
EX/MEM
MEM/WB
x4 (SE)
Next PC
TA
Condition Handler
Reg. File
RA
RB
RW
PA
PB
PW
LE
Instruction Memory
PC
Shifter/Sign Extender
Data Memory
In
Out
Address
A
B
CC
Control Unit
ID_shift_imm
ID_ALU_op
ID_load_instr
ID_RF_enable
ID_B_instr
Hazards/ Forwarding Unit