Proj-2

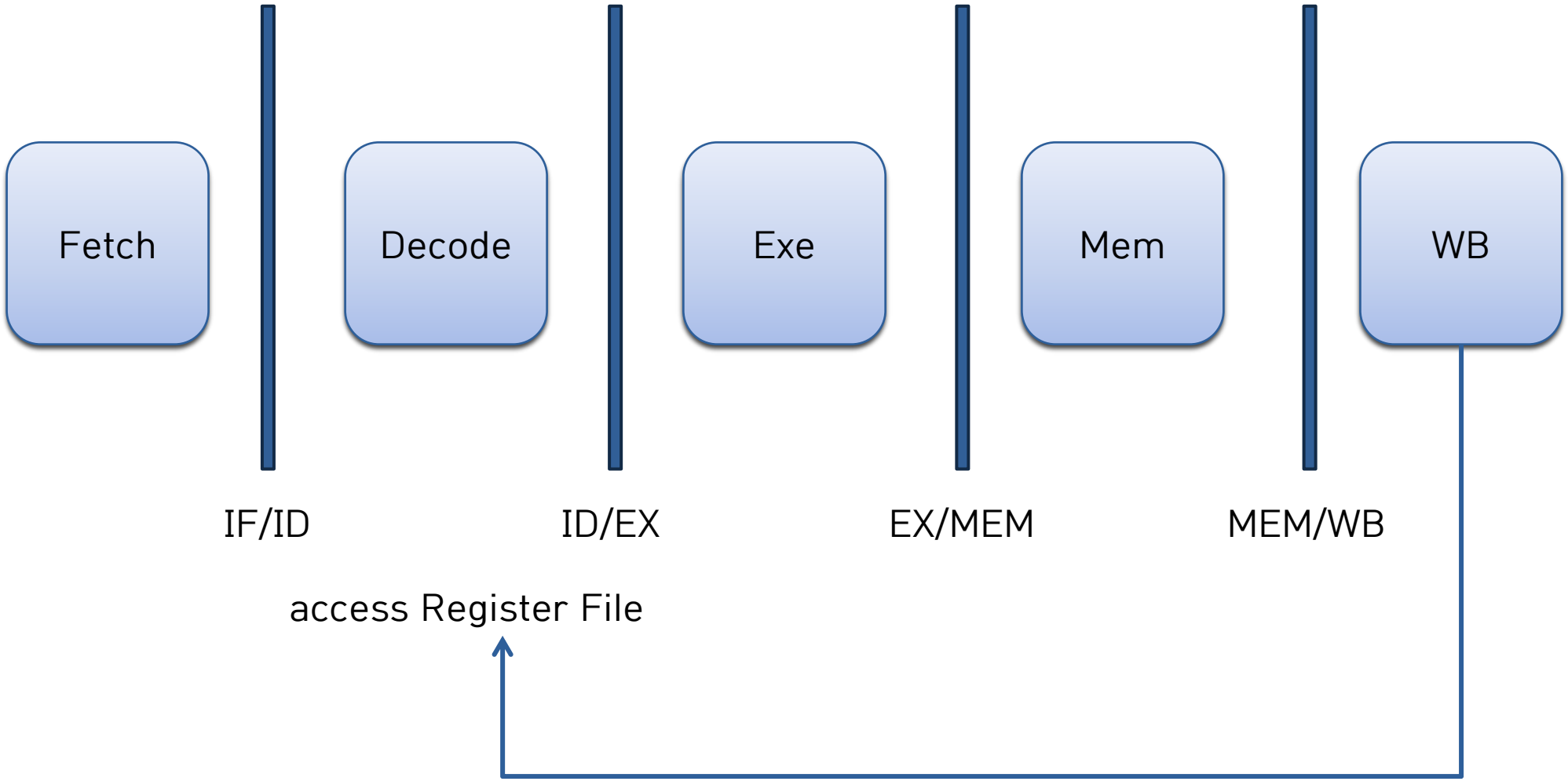
LC-2K Pipeline Simulator

1) No Hazard LC-2K Simulator
2) Data Hazard LC-2K Simulator
3) Branch Hazard LC-2K Simulator

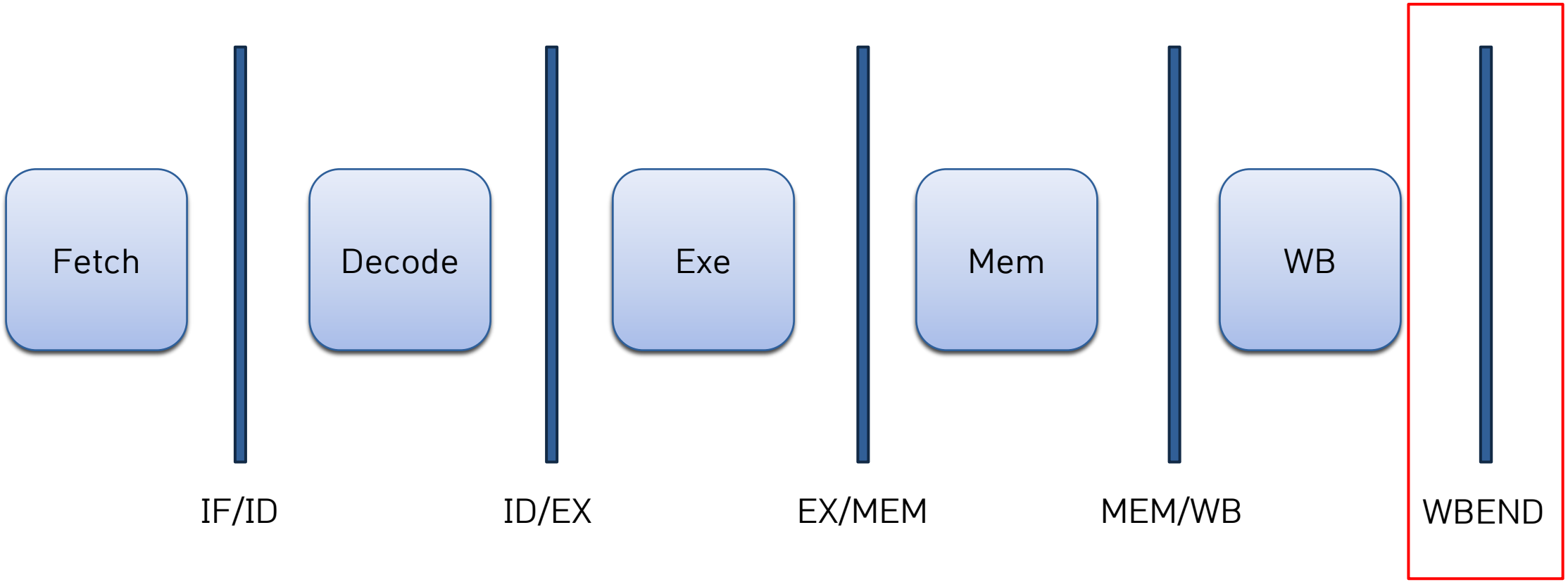
Purpose of Proj2

- Using LC2K ISA , make Pipeline simultor help to understand your pipeline implementation.

General Pipeline model



LC-2K Pipeline basic model



Note

- You will not implement the jalr instruction from the LC-2K. Taking out jalr eliminates several dependencies.
- Memory is divided into InstMem and DataMem
- To implement pipeline , run() function will be a loop
- each loop iteration = one cycle
- at first of program you need print current state

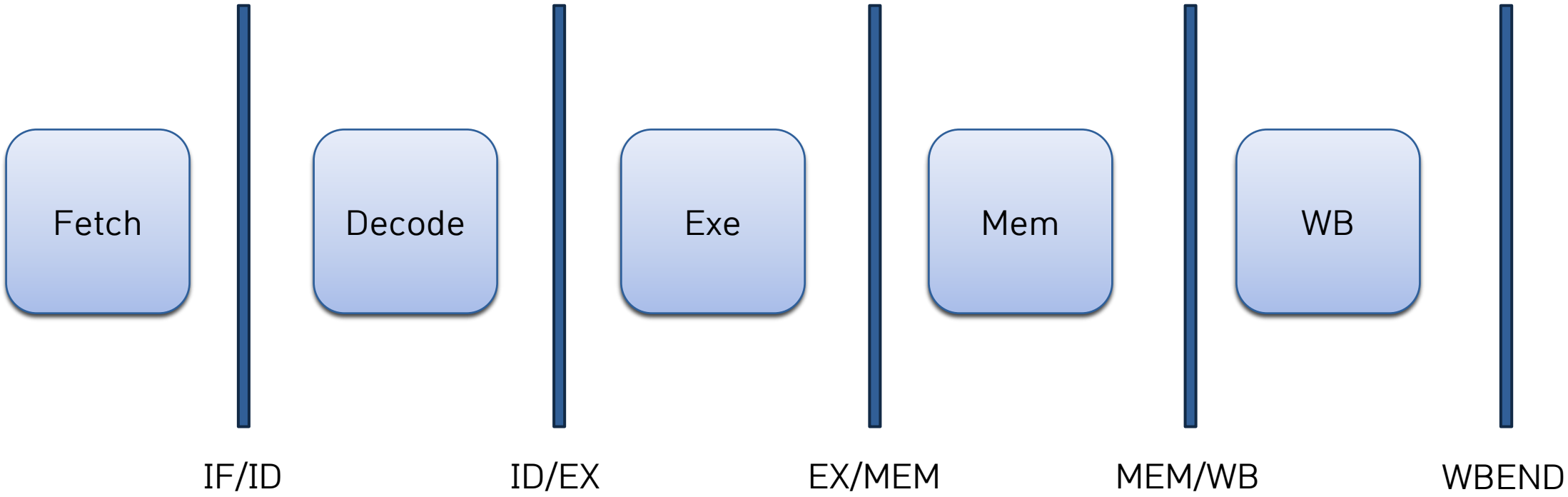
```
While(1)
{
    printState(&state);
    /* check for halt */
    if (opcode(state.MEMWB.instr) == HALT)
    {
        printf("machine halted\n");
        printf("total of %d cycles executed\n", state.cycles);
        exit(0);
    }
    newState = state;
    newState.cycles++;
    IF Stage
    ID Stage
    EX Stage
    MEM Stage
    WB Stage

    state = newState

}
```

Print State

Halt

Save State

LC-2K Pipeline basic model



Simple Pipeline Test

- LW.as

  lw 0 1 data " load data to reg 1 "

  noop

  noop

  noop

  halt

  data .fill 10

Pipeline Register

- IF/ID

->instr = instruction (lw)

->pcPlus1 = state pc +1

```
IFID:
        instruction lw 0 1 5
        pcPlus1 1
```

Pipeline Register

- ID/EX

-> instr = IFID's inst

-> pcPlus1 = state IFID pc

-> readRegA = lw's reg A Value

-> readRegB = lw's reg B Value

-> offset = lw's offset

Pipeline Register

- ID/EX

```
IDEX:
        instruction lw 0 1 5
        pcPlus1 1
        readRegA 0
        readRegB 0
        offset 5
```

current Register 0 ,1 Value is 0 so IDEX regA,B = 0

Pipeline Register

- EX/MEM

-> instr = IDEX's instr

-> branchTarget = IDEX's pc + IDEX's offset

-> readRegB = IDEX's reg B

Pipeline Register

- EX/MEM

-> instr = IDEX's instr

-> branchTarget = IDEX's pc + IDEX's offset

-> readRegB = IDEX's reg B

```
EXMEM:
        instruction lw 0 1 5
        branchTarget 6
        aluResult 5
        readRegB 0
```

Pipeline Register

- MEM/WB

-> instr = EX/MEM's instr

-> writeData = DataMem[EX/MEM offset]

```
MEMWB:
        instruction lw 0 1 5
        writeData 10
```

Pipeline Register

- WBEND

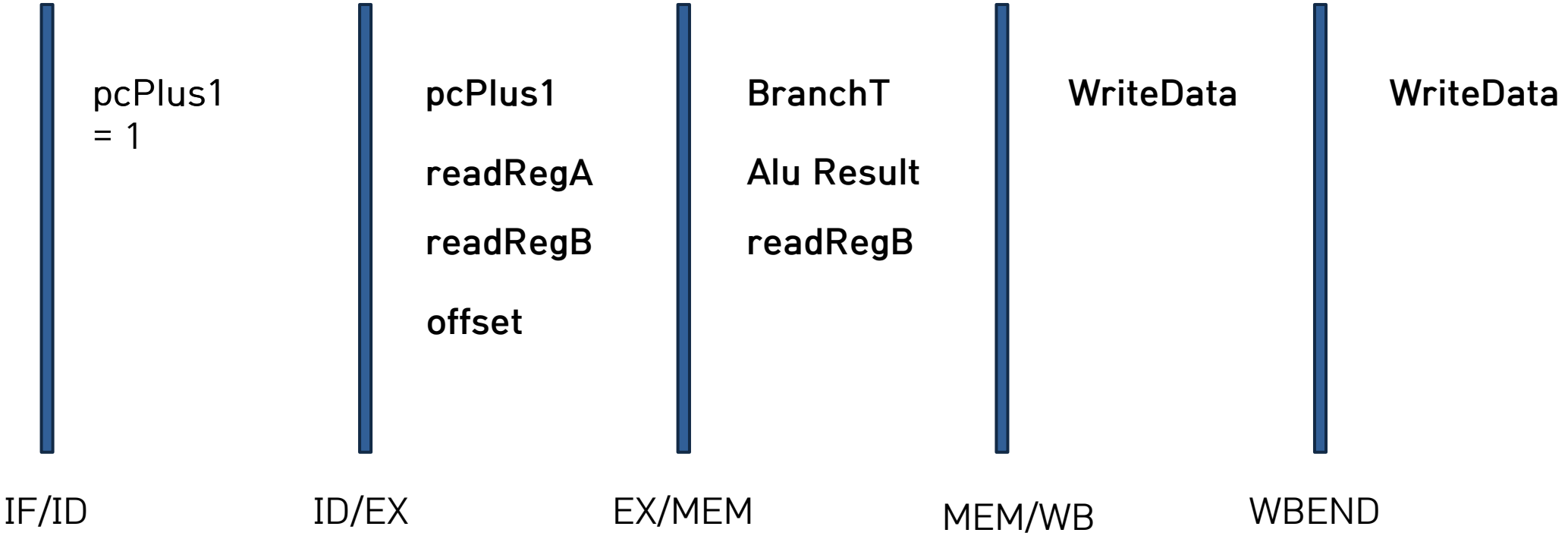
-> instr = MEM/WB's instr

-> WriteData = destinaion register Value

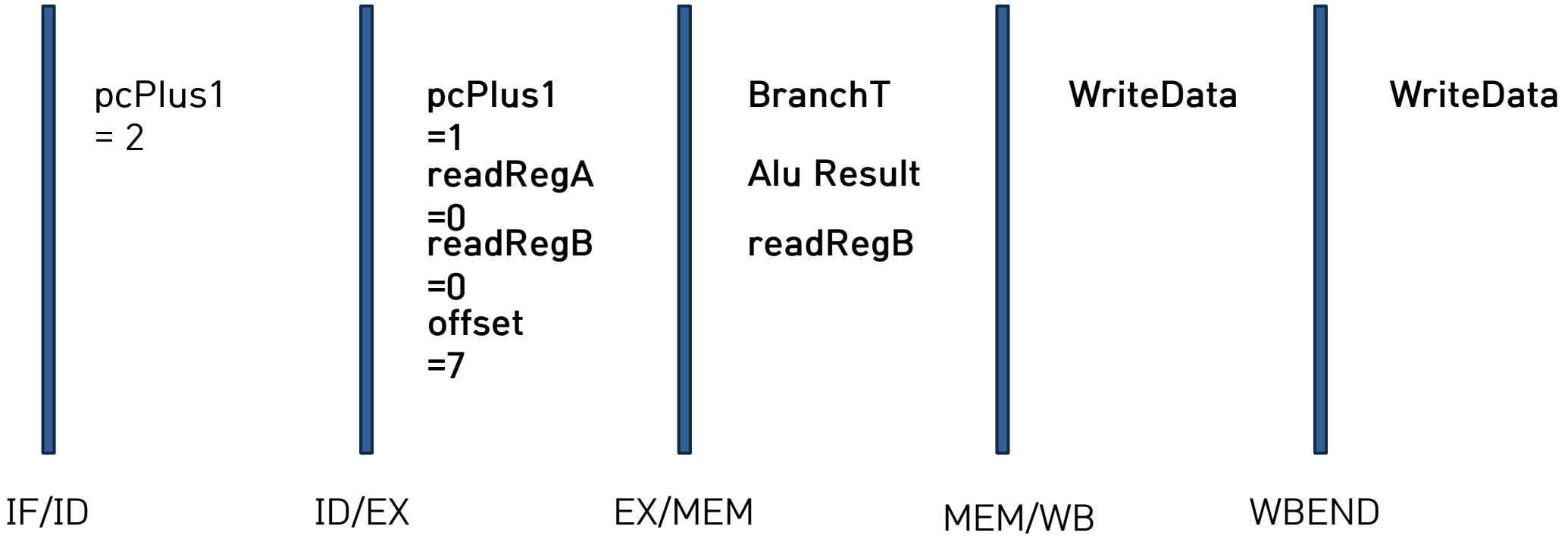
```
    WBEND:
            instruction lw 0 1 5
            writeData 10
```

Lw + Add sample

```
        lw 0 1 data1     "load Data to reg 1 "
        lw 0 2 data2     "load Data to reg 2 "
        noop
        noop
        noop
        add 1 2 3        "add reg1 ,reg2 -> reg3"
        halt
data1   .fill 10
data2   .fill 20
```

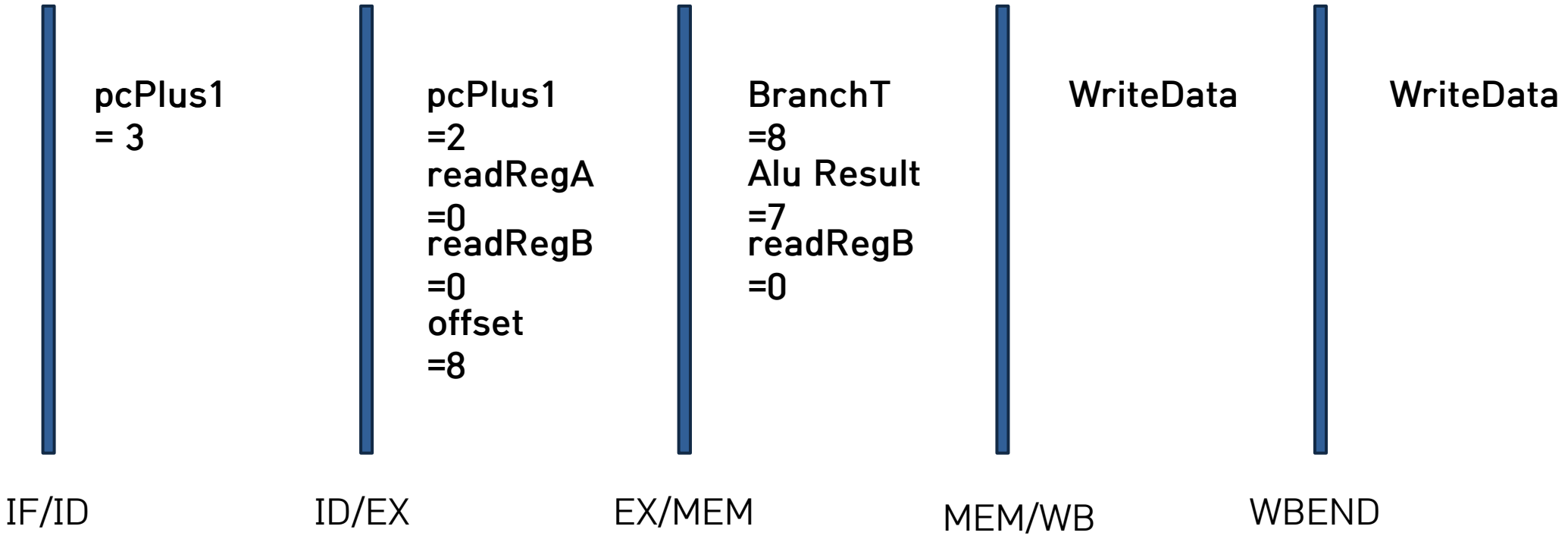

pcPlus1
= 1
pcPlus1
readRegA
readRegB
offset
BranchT
Alu Result
readRegB
WriteData
WriteData
IF/ID
ID/EX
EX/MEM
MEM/WB
WBEND

lw 0 1 7

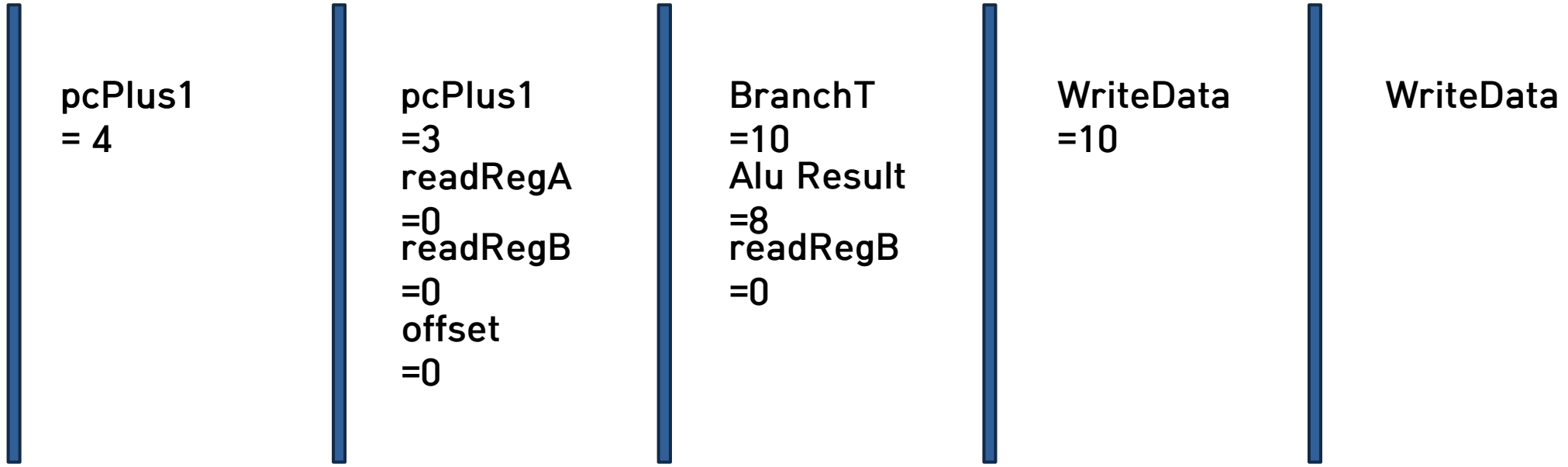

pcPlus1
= 2
pcPlus1
=1
readRegA
=0
readRegB
=0
offset
=7
BranchT
Alu Result
readRegB
WriteData
WriteData
IF/ID
ID/EX
EX/MEM
MEM/WB
WBEND

lw 0 2 8

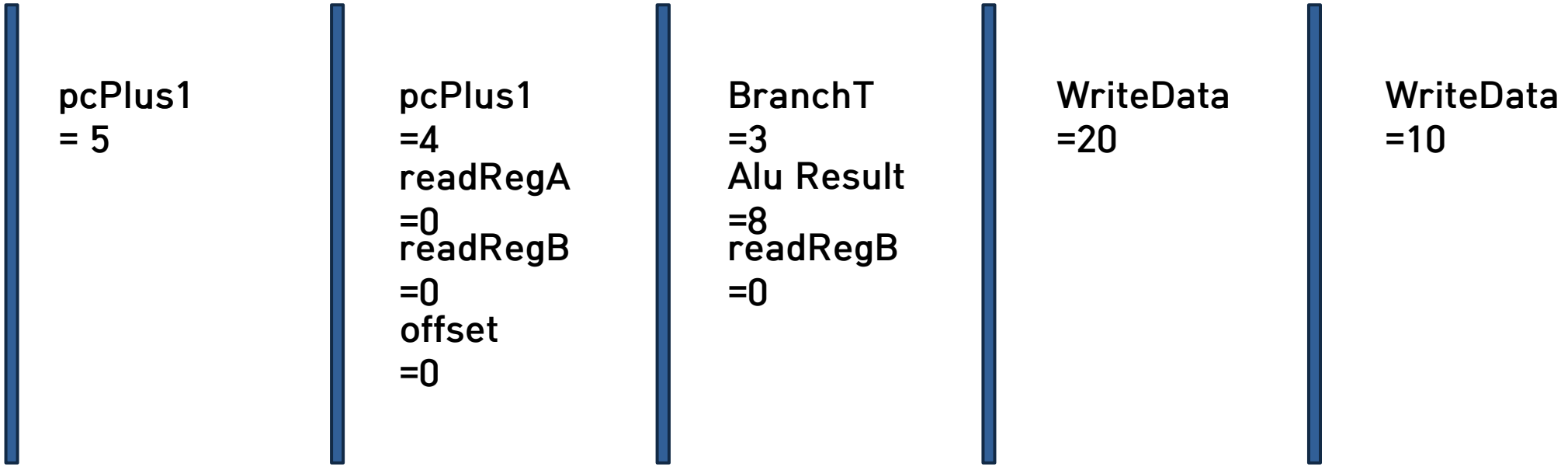
lw 0 1 7



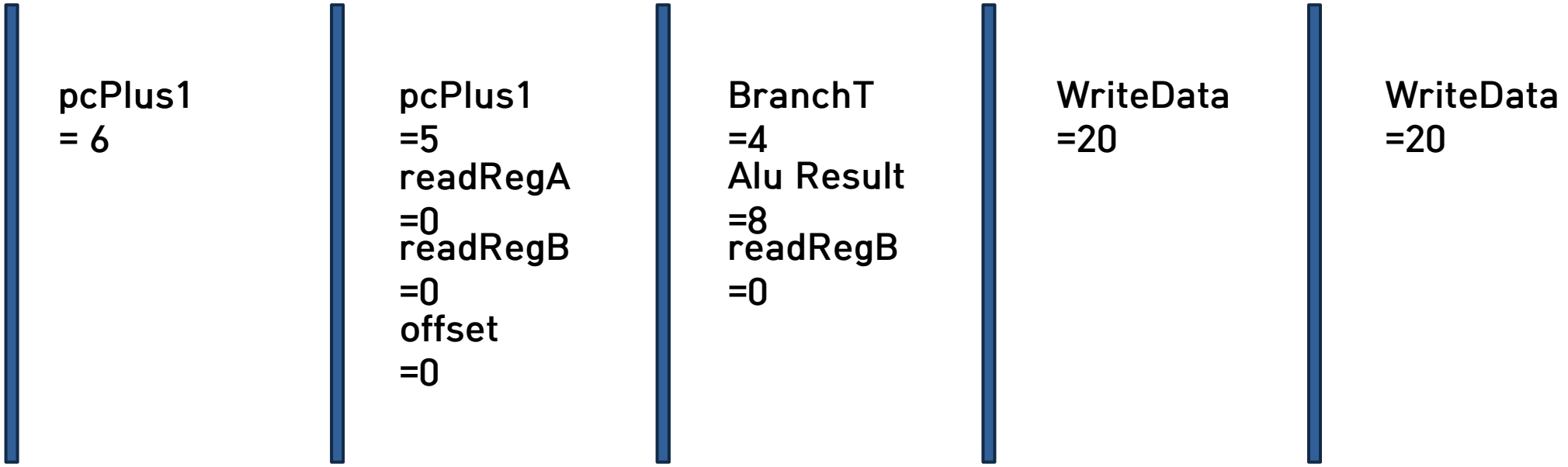
Noop lw 0 2 8 lw 0 1 7



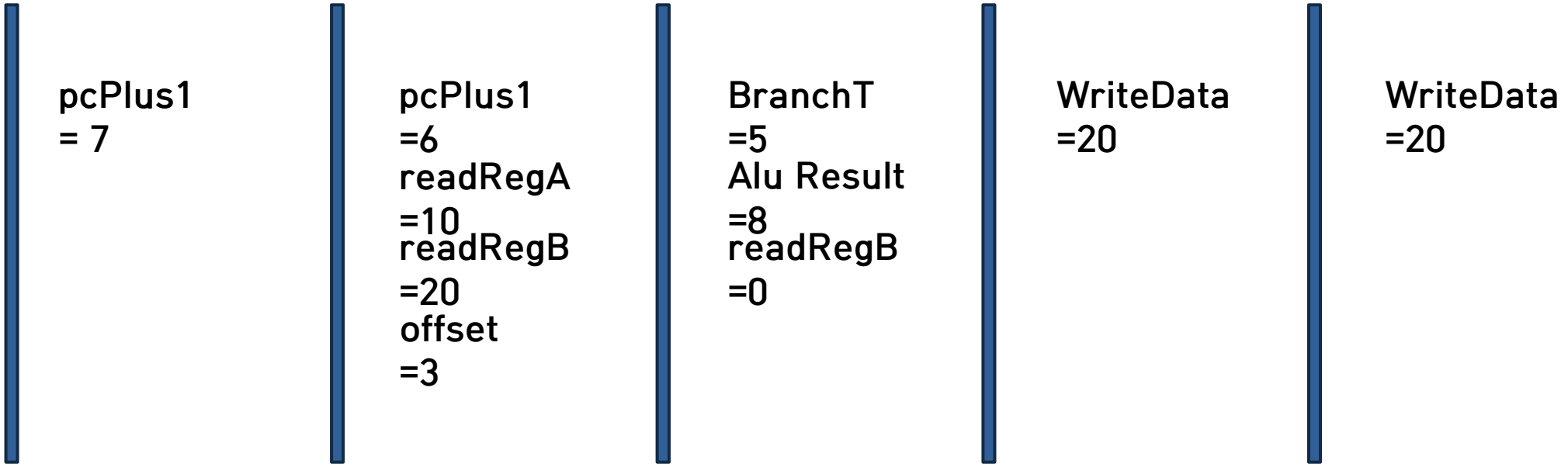
Noop Noop lw 0 2 8 lw 0 1 7



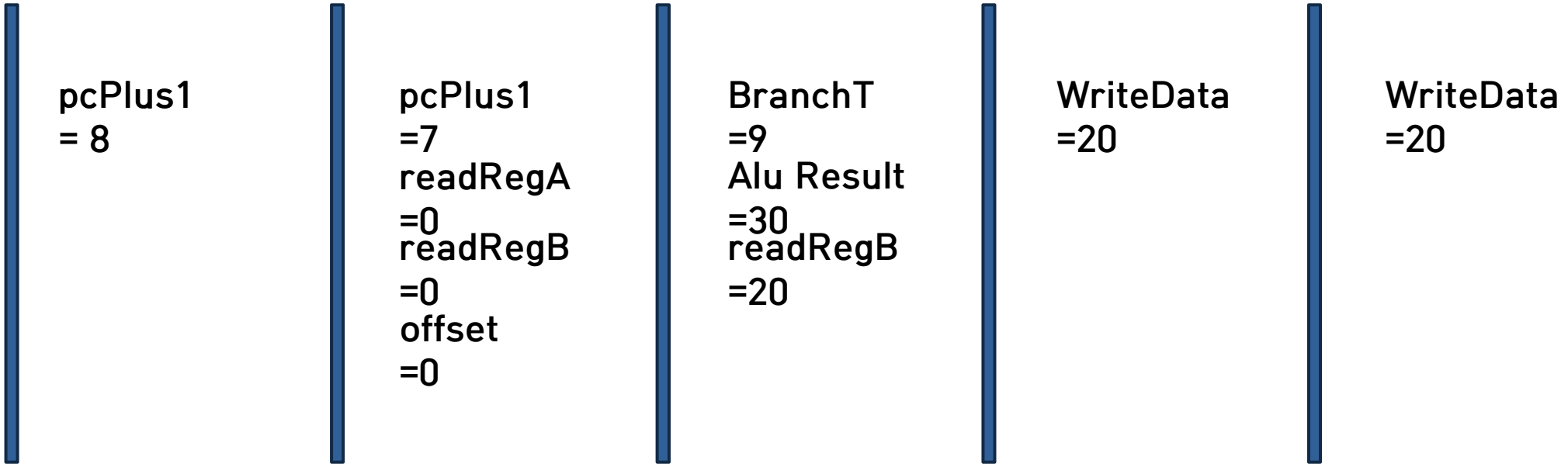
Noop Noop Noop lw 0 2 8 lw 0 1 7



add 1 2 3 | Noop | Noop | Noop | lw 0 2 8



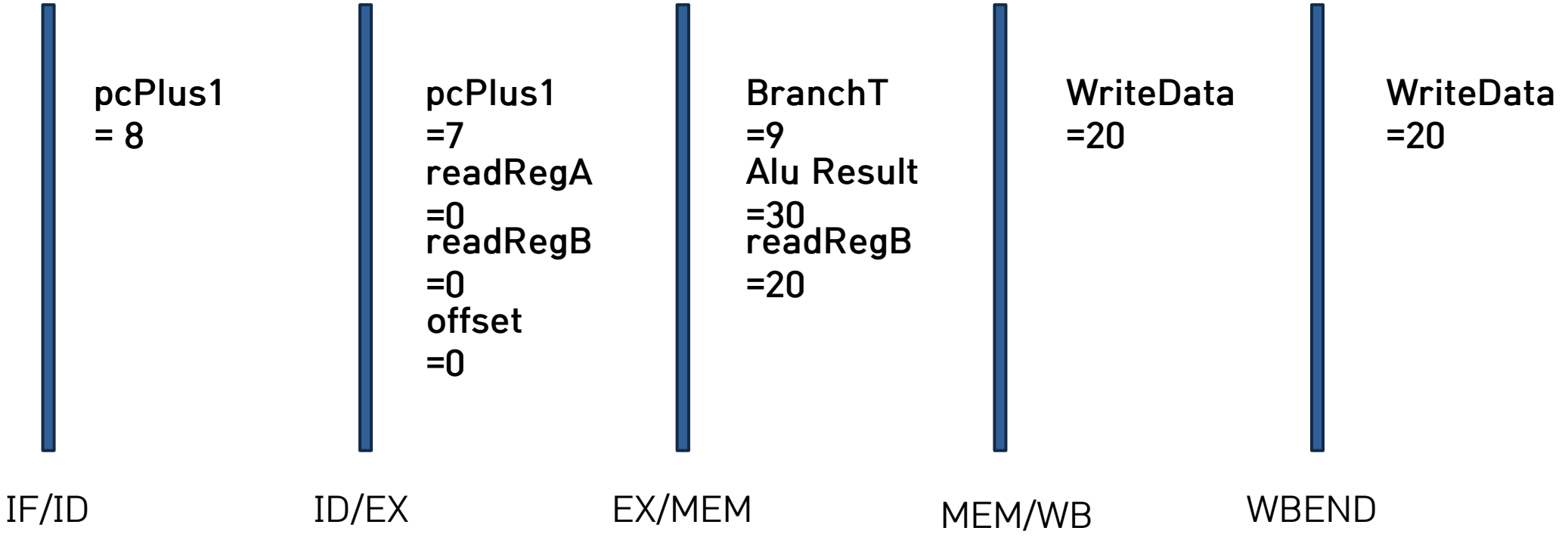
halt add 1 2 3 Noop Noop Noop



add 0 0 10 halt add 1 2 3 Noop Noop

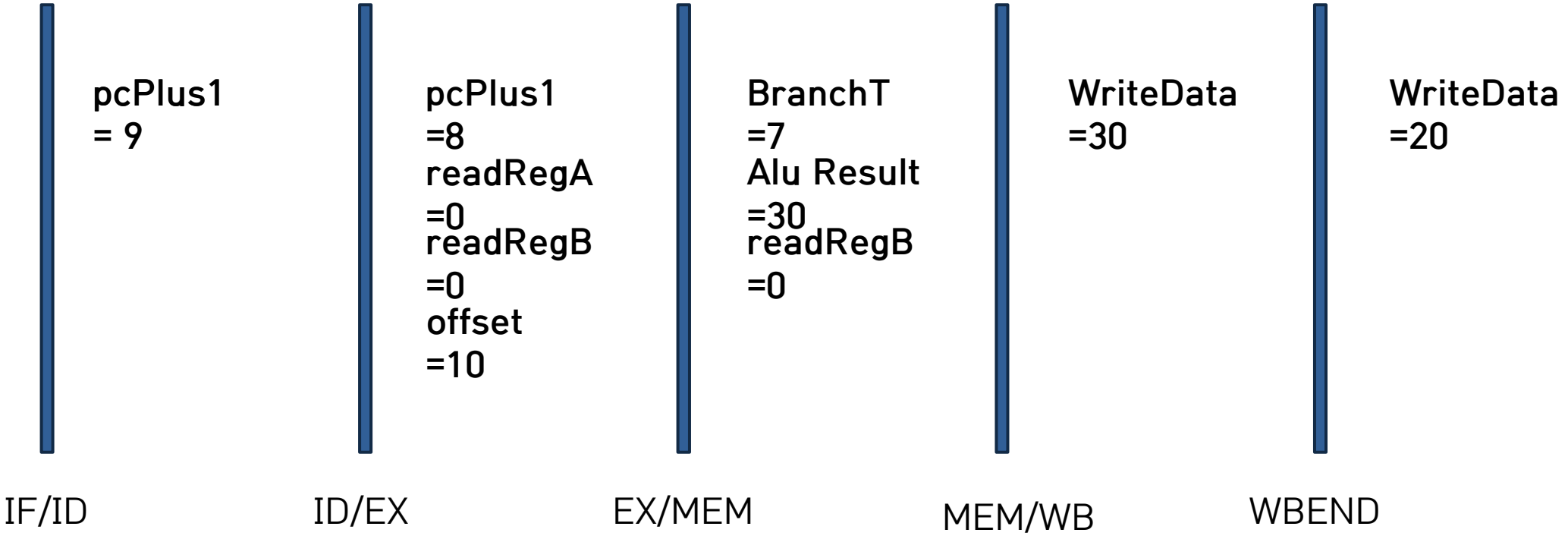
Lw + Add sample

```
        lw 0 1 data1     "load Data to reg 1 "
        lw 0 2 data2     "load Data to reg 2 "
        noop
        noop
        noop
        add 1 2 3        "add reg1 ,reg2 -> reg3"
        halt
data1   .fill 10
data2   .fill 20
```

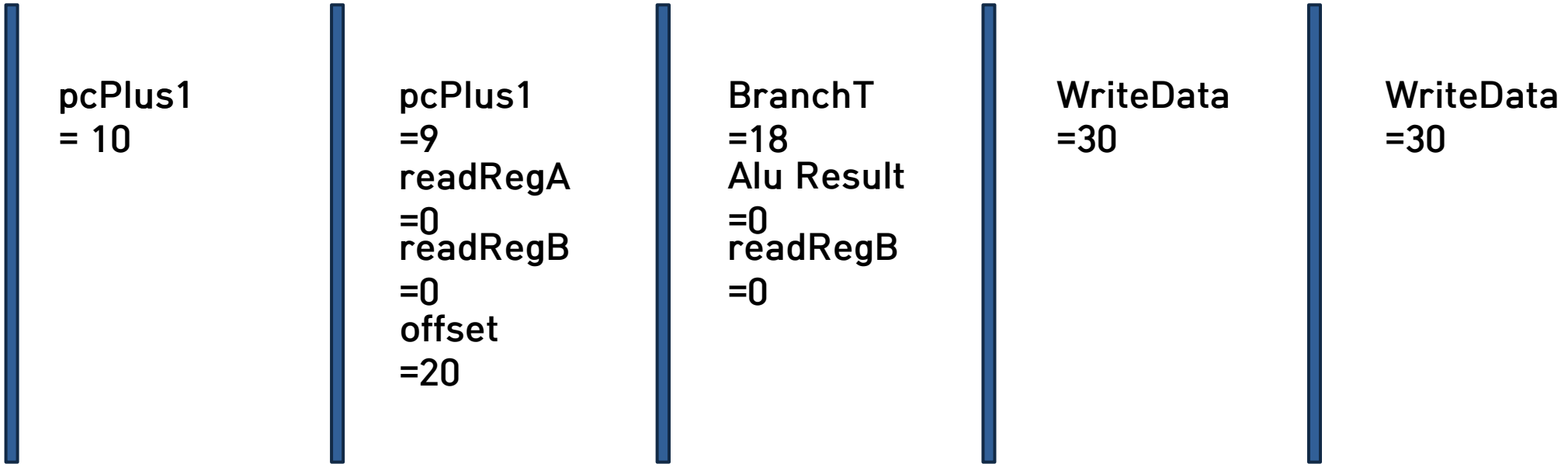


add 0 0 10 | halt | add 1 2 3 | Noop | Noop

This Instruction is same as [data .fill 10]



add 0 0 20 | add 0 0 10 | halt 0 0 0 | add 1 2 3 | Noop



add 0 0 0

add 0 0 20

add 0 0 10

halt 0 0 0

add 1 2 3

Detect hazard

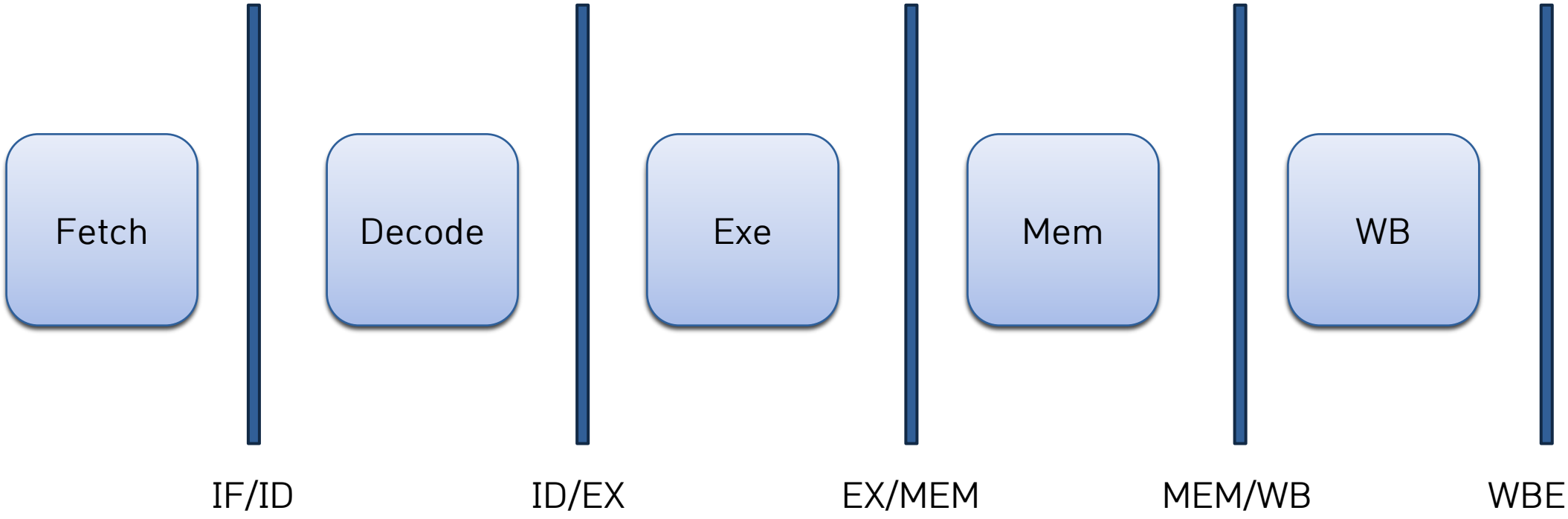
- - Data Hazard
- - Branch Hazard

Data Hazard

- lw 0 1 data
  add 1 1 2
  ...

-> reg1 data will be set WB stage ,
but add Inst Use reg1 befor WB stage

Data Hazard



Cycle

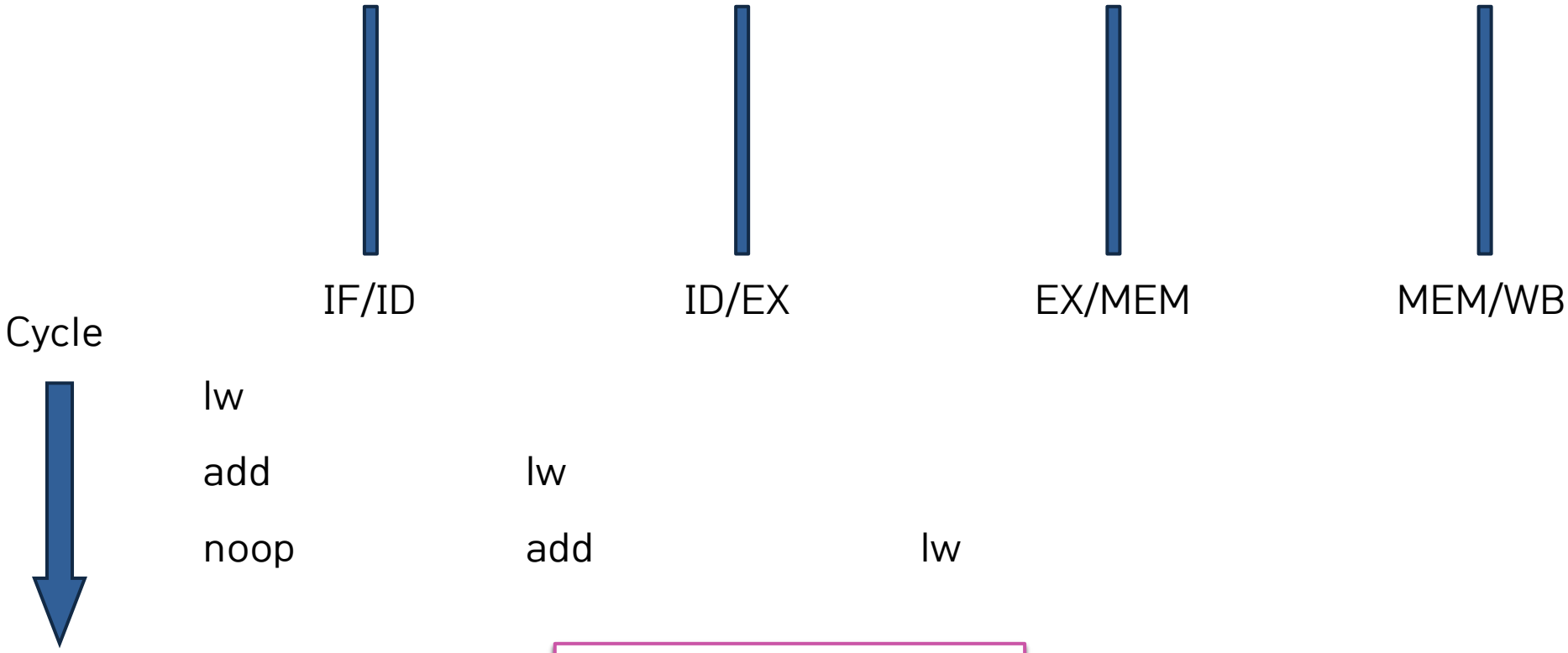
add 1 1 2 lw 0 1 Data

add 1 1 2 lw 0 1 Data

Reg 1 Value is not Data .

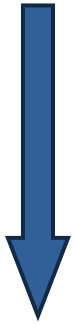
How To Forwading?

Data Hazard


IF/ID
ID/EX
EX/MEM
MEM/WB
Cycle
lw
add
noop
lw
add
lw
ID/EX RS or RT ==
EX/MEM Reg Dest

Data Hazard

Cycle



lw			
add	lw		
noop	add	lw	
noop	add	stall (noop)	lw
noop	noop	add	stall(noop)

Data Hazard

```
IFID:
        instruction add 1 1 2
        pcPlus1 2
IDEX:
        instruction lw 0 1 3
        pcPlus1 1
        readRegA 0
        readRegB 0
        offset 3
EXMEM:
        instruction noop 0 0 0
        branchTarget 0
        aluResult 0
        readRegB 0
```

Data Hazard

```
reg[ 7 ] 0
IFID:
        instruction add 1 1 2
        pcPlus1 2
IDEX:
        instruction noop 0 0 0
        pcPlus1 0
        readRegA 0
        readRegB 0
        offset 0
EXMEM:
        instruction lw 0 1 3
        branchTarget 4
        aluResult 3
        readRegB 0
```

Stall One Cycle

Branch Hazard

- Branch determines flow of control

-> Fetching next instruction depends on branch outcome

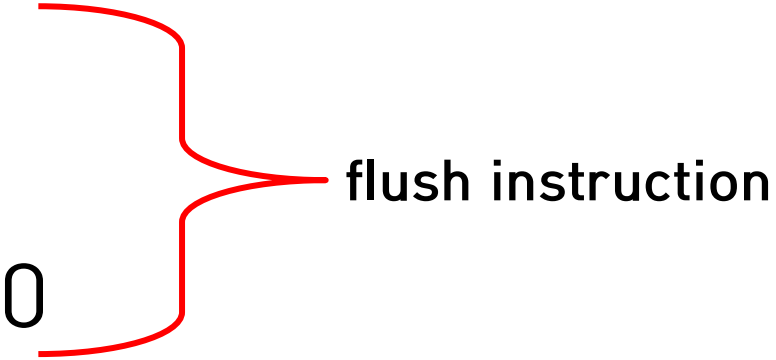
-> Pipeline can’t always fetch correct instruction

-> In MIPS , Fetch instruction after branch, with no delay

Branch Hazard

- beq 1 2 32
  add 1 1 2
  nor 2 2 3
  sw 5 2 Mem0
  lw 0 1 Data
  ...

IF reg 1 === reg 2 -> branch

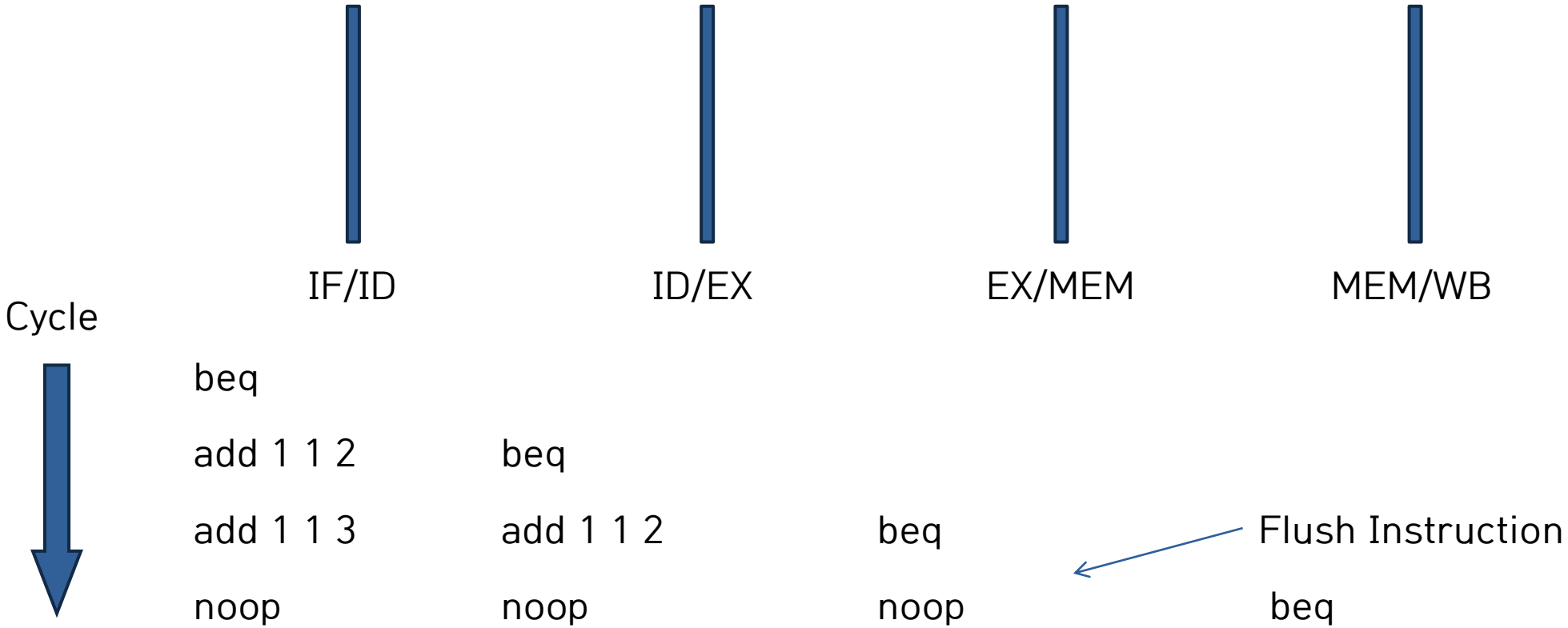


Use branch-not-taken to resolve most branch hazards,
and decide whether or not to branch in the MEM stage.
Beq Instruction condition is determined "*Mem stage*" so we need flush 3 instruction

Branch Hazard

```
        lw 0 1 data1
        noop
        noop
        noop
        beq 0 0 3
        add 1 1 2
        add 1 1 3
        add 1 1 4
        halt
data1   .fill 1
```

Branch Hazard



Full Credit

- In your program add data/branch hazard resolving logic
- Submit 5 test case (2 of them -> hazard Code)
- And write a simple comment in your test case, e.g) why do you add 3 noop between lw and add or why two instruction has a hazard etc...

Full Credit

- Do NOT modify and move printState()
- Do NOT use any printf().
- Initialize all values correctly (when program start , PC and other register value must be 0)

- Submit – Simulator.c + 5 test cases

Question

- designer9406@gmail.com