

EE 457 Unit 6b

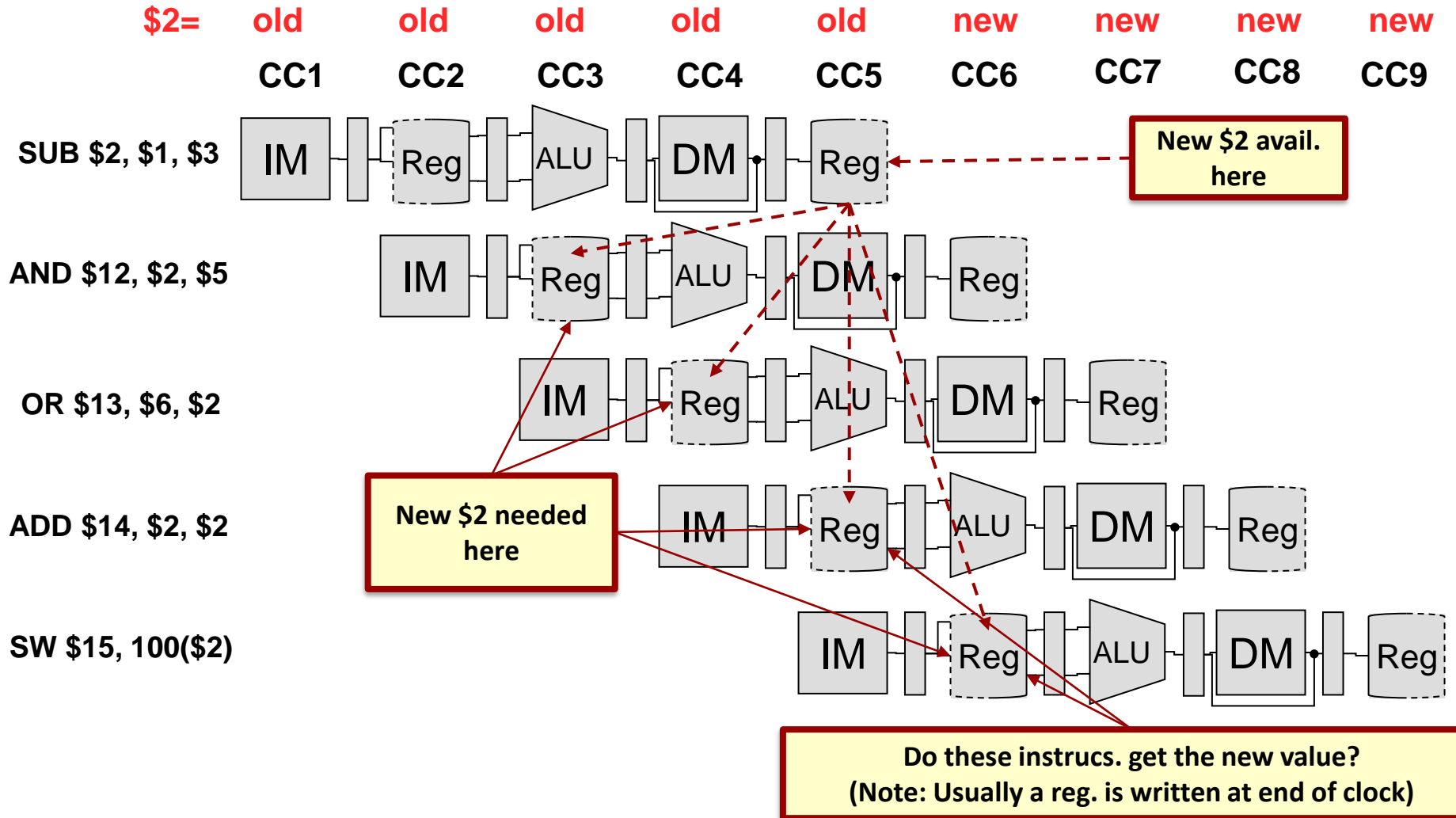
Data Hazards

Data Hazards

- Consider the data dependencies in the following sequence
 - The last four are all dependent on register \$2
- But because of pipelining the instructions **and**, **or**, **add** could read \$2 before the **sub** writes its result
- This is called a **data hazard**, more specifically a **RAW** (Read-After-Write) Hazard
 - If the RAW hazards is not handled, incorrect program execution may result

SUB	\$2	\$1	\$3
AND	\$12	\$2	\$5
OR	\$13	\$6	\$2
ADD	\$14	\$2	\$2
SW	\$15	100	\$2

An Opening Example



- Can the compiler solve this problem w/o hardware help?

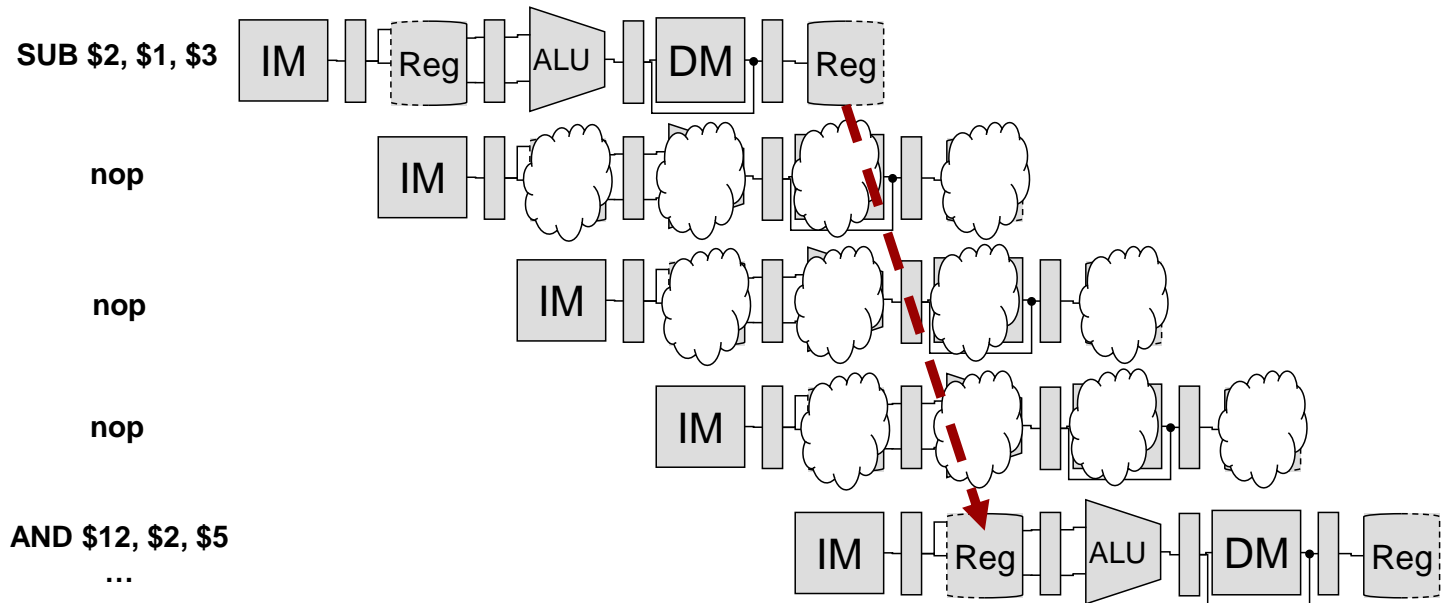
An Opening Example

- The compiler's solution is to insert **nop** (no operation) instructions
- The effect is to push the dependency later in time

```

SUB    $2, $1, $3
nop
nop
nop
AND    $12, $2, $5
OR     $13, $6, $2
ADD    $14, $2, $2
SW     $15, 100($2)
    
```

(Why 3?)



Control for Data Hazards

- Two hardware solutions
 - Stalls
 - Forwarding/bypassing
- Stall Strategy:
 - Detect the hazard and stall the dependent instructions in the pipeline until the hazard is resolved
 - Stalling is achieved by sending bubbles (nops) forward into the pipe and not updating the stalled stage registers

Stalling Strategy

- Since we must be careful not to read a “stale” register value from the register file, we should detect hazards in the ID stage and stall the instruction there!
 - If an instruction stalls, all instructions behind it stall
 - All instructions in front of it are free to continue down the pipe
 - Insert “bubbles” into the subsequent stages (set all control signals to 0 so no incorrect behavior takes place)

LW \$t1,4(\$s0)
 ADD \$t5,\$t1,\$t4

	Fetch	Decode	Exec.	Mem.	WB
C1	LW				
C2	ADD	LW			
C3	i	ADD	LW		
C4	i	ADD	nop	LW	
C5	i	ADD	nop	nop	LW
C6	i	ADD	nop	nop	nop
C7	i+1	i	ADD	nop	nop
C8	i+2	i+1	i	ADD	nop

Using Stalls to Handle
 Dependencies (Data Hazards)

Detecting Data Hazards

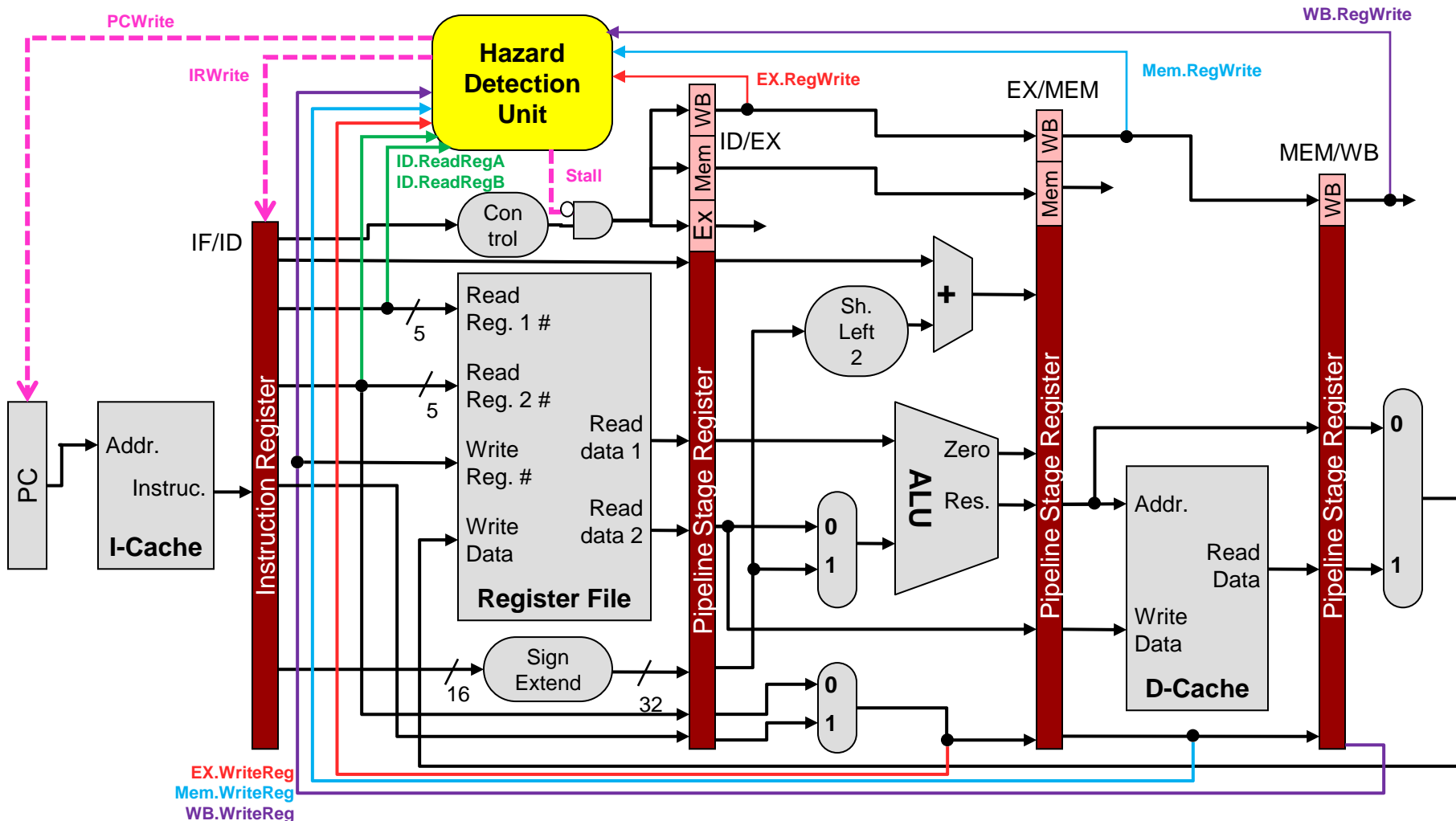
- Need to stall if an instruction in the last 3 stages is going to write a register the currently decoding instruction wants to read (i.e. **READ-AFTER-WRITE**)
- How would we know if an instruction in the pipe is going to write a register than an instruction in ID wants to read?
 - By comparing register ID values!!

Cases for Detecting Data Dependencies

- 1a. ID/**EX.RegWrite** and ID/**EX.WriteRegister** == IF/**ID.ReadRegister1**
- 1b. ID/**EX.RegWrite** and ID/**EX.WriteRegister** == IF/**ID.ReadRegister2**
- 2a. EX/**MEM.RegWrite** and EX/**MEM.WriteRegister** == IF/**ID.ReadRegister1**
- 2b. EX/**MEM.RegWrite** and EX/**MEM.WriteRegister** == IF/**ID.ReadRegister2**
- 3a. MEM/**WB.RegWrite** and MEM/**WB.WriteRegister** == IF/**ID.ReadRegister1**
- 3b. MEM/**WB.RegWrite** and MEM/**WB.WriteRegister** == IF/**ID.ReadRegister2**

Hazard Detection Unit I/O

- Only stall if a Write register in one of the last 3 stages matches one of the read registers in the ID stage



HDU Operation

Hazard	Detection
EX Hazard	ID/ EX RegWrite and ((ID/ EX.WriteRegister = IF/ ID.ReadRegister1) or (ID/ EX.WriteRegister = IF/ ID.ReadRegister2))
MEM Hazard	EX/ MEM RegWrite and ((EX/ MEM.WriteRegister = IF/ ID.ReadRegister1) or (EX/ MEM.WriteRegister = IF/ ID.ReadRegister2))
WB Hazard	MEM/ WB RegWrite and ((MEM/ WB.WriteRegister = IF/ ID.ReadRegister1) or (MEM/ WB.WriteRegister = IF/ ID.ReadRegister2))

HDU Implementation

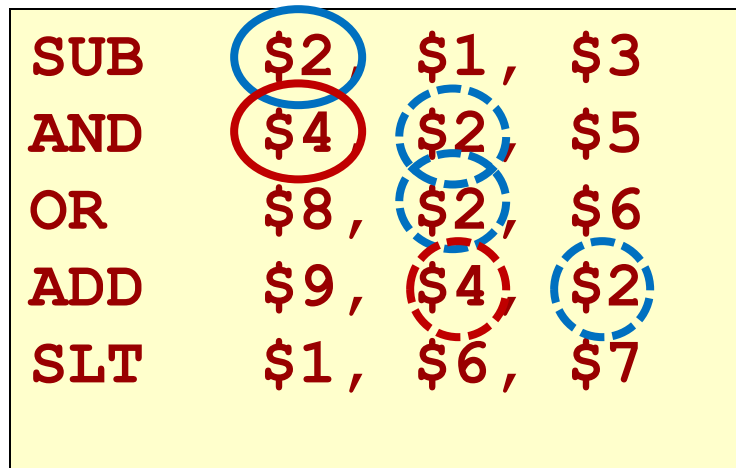
- How long do we stall
 - If the hazard exists in the EX stage, we need to insert 3 bubbles (wait 3 cycle) before restarting the pipeline
 - If the hazard exists in the WB stage we only need to insert 1 bubble (wait 1 cycle)
- So since the delay is time dependent does the HDU require a counter or state machine?
 - No! The producer instruction will keep moving forward and eventually clear The HDU works by simply checking if ANY hazard exists in the forward stages and inserts a bubble into the ID/EX stage register
 - If an EX hazard exists it will take 3 cycle to clear and thus the HDU will detect an EX hazard in one clock, a MEM hazard in the next, and a WB hazard in the third inserting a bubble for each of these cycle = 3 bubbles)

HDU Logic

- Detection logic requires six (6) 5-bit comparators along with some AND and OR gates
- Upon detection, HDU inserts a bubble into the ID/EX stage register
 - Bubble = HW generated NOP = Turn all control signals to zeros

HDU Implementation

- What if two hazards exist at the same time
 - Again, any hazard should cause a bubble
 - The producing instructions will continue to move forward and eventually clear



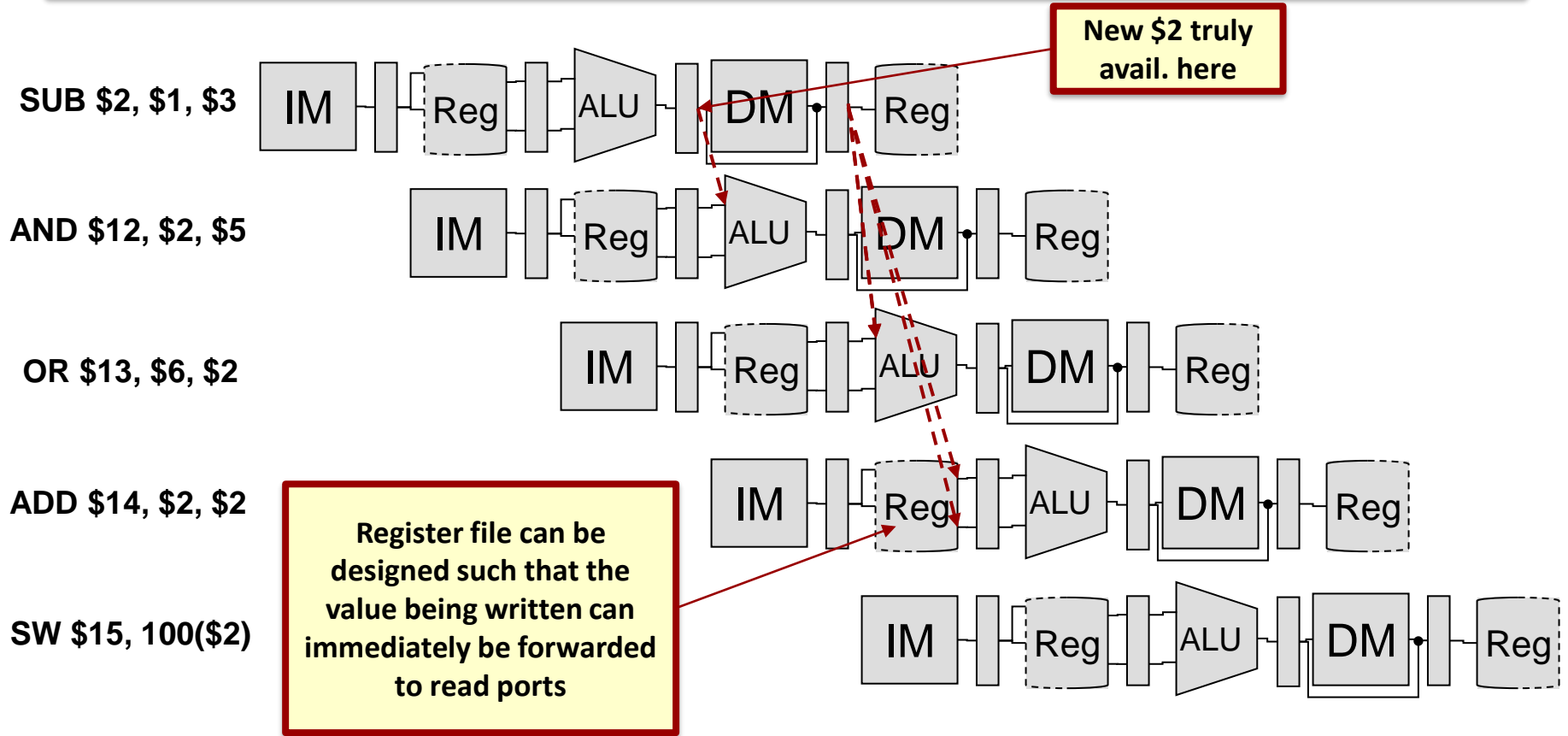
	Fetch	Decode	Exec.	Mem.	WB
C1	SUB				
C2	AND	SUB			
C3	OR	AND	SUB		
C4	OR	AND	nop	SUB	
C5	OR	AND	nop	nop	SUB
C6	OR	AND	nop	nop	nop
C7	ADD	OR	AND	nop	nop
C8	SLT	ADD	OR	AND	nop
C9	SLT	ADD	nop	OR	AND

Register Forwarding/Bypassing

REDUCING DATA HAZARDS

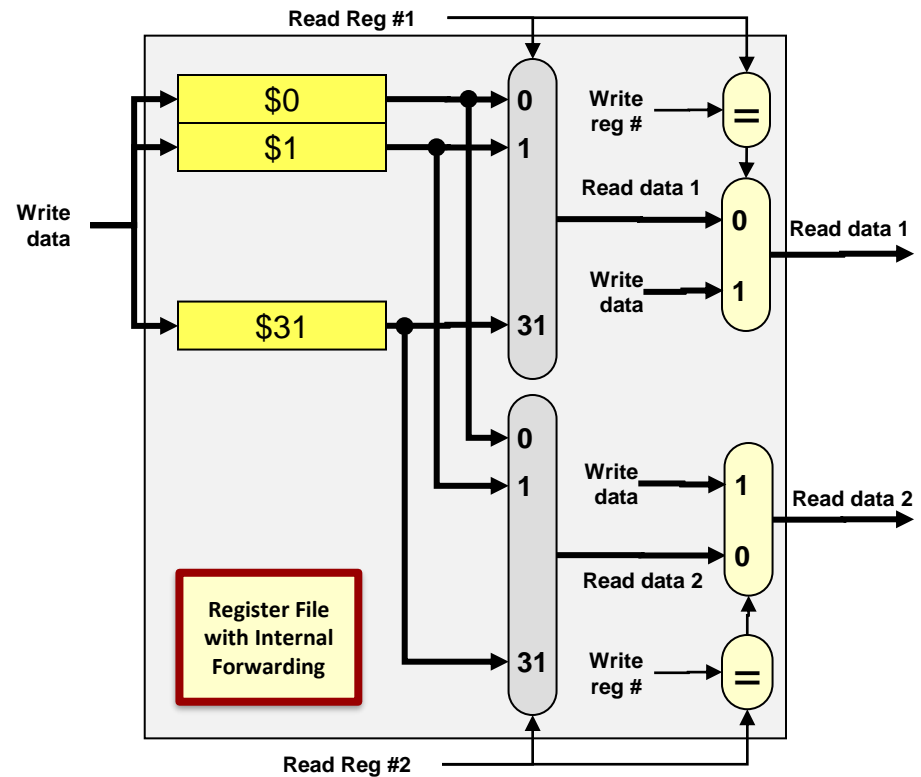
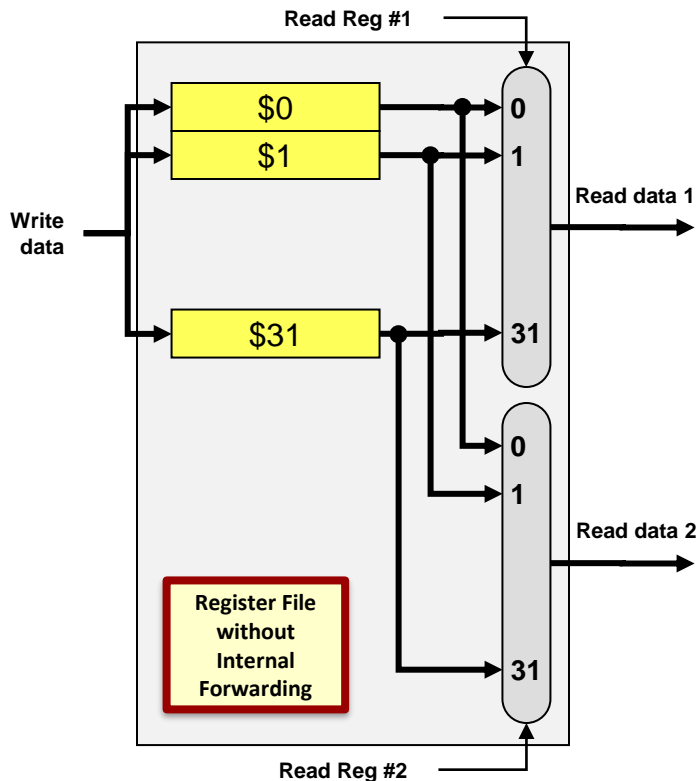
Key Idea

While \$2 is not written until WB stage, the subtraction result is available at the end of the EX stage (beginning of the MEM stage) and can be passed off directly to dependent instructions

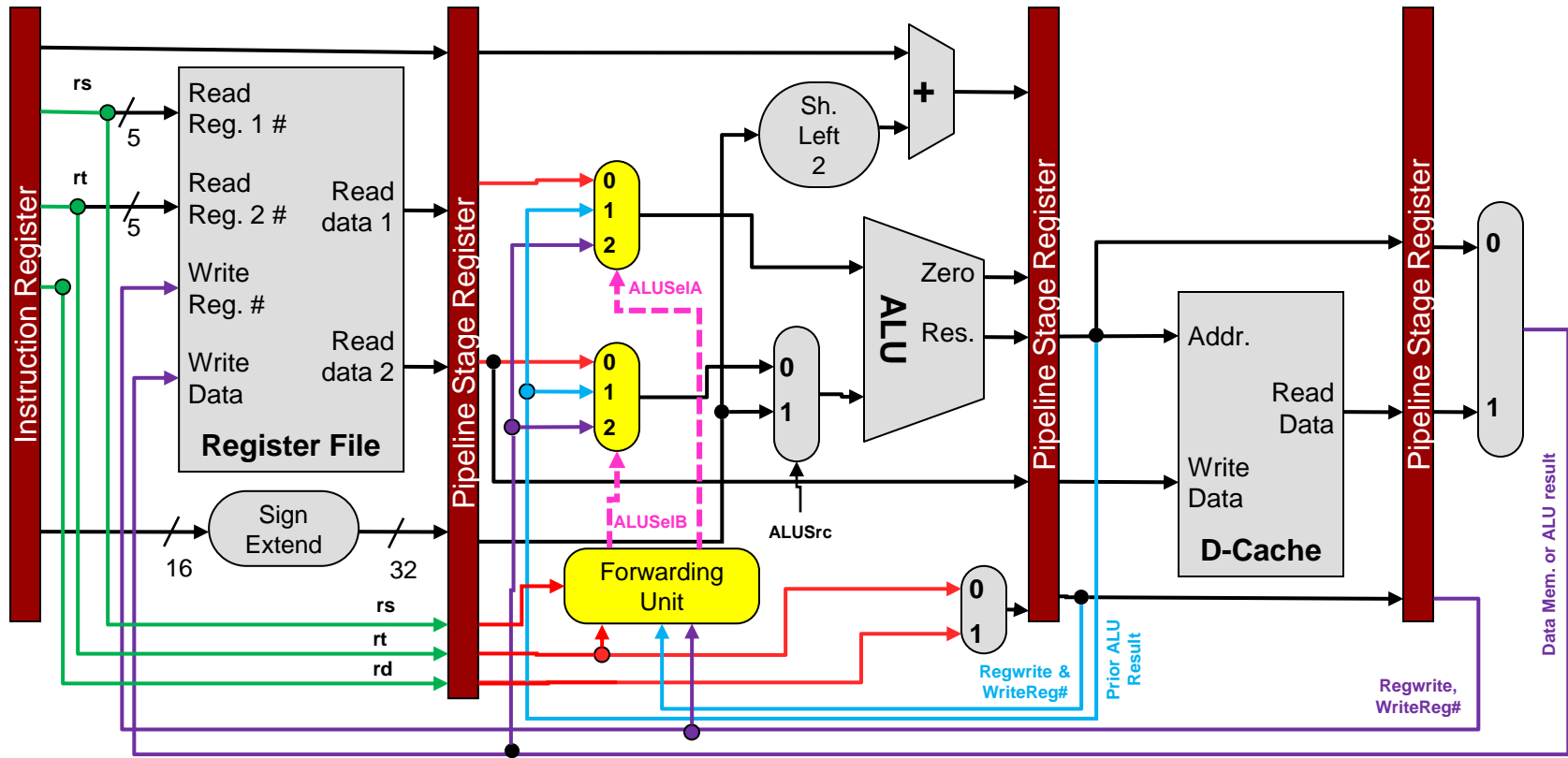


Register File Internal Forwarding

- Internal Forwarding:
 - Value read = Value being written



Forwarding Unit



Mux Control	Source	Explanation
ALUSelA & ALUSelB = 00	ID/EX	The first (if ALUSelA) and/or second (ALUSelB) ALU input comes from the normal ID/EX stage register
ALUSelA & ALUSelB = 01	EX/MEM	The first (if ALUSelA) and/or second (ALUSelB) ALU input comes from the prior ALU result in the EX/MEM stage reg.
ALUSelA & ALUSelB = 10	MEM/WB	The first (if ALUSelA) and/or second (ALUSelB) ALU input comes from the data memory or earlier ALU result

Forwarding Unit Addition

- Remove the old HDU in the ID stage
- Add a new Forwarding Unit (FU) in the EX stage
 - Like HDU it services dependent instructions
 - Compares write register ID's in later stages to read register ID's in earlier stages

Forwarding Unit vs. HDU

- Since the HDU stalled instructions in the ID stage it needed to compare 2 source ID's with 3 destination ID's
- Because we let instructions fetch stale register values and just replace them in the EX (or MEM) stage, the forwarding Unit compares 2 source ID's with 2 destination ID's
- HDU had 6 comparators while the FU requires 4

ReadReg1 Forwarding

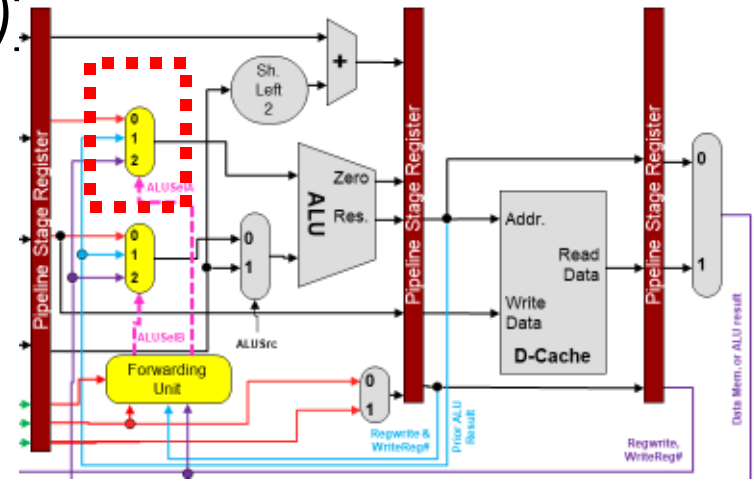
- ALUSelA mux

If (*MEM.RegWrite*
 and (*MEM.WriteReg* \neq 0)
 and (*MEM.WriteReg* = *EX.ReadReg1*))
 then ALUSelA = 01

Else if (*WB.RegWrite*
 and (*WB.WriteReg* \neq 0)
 and (*WB.WriteReg* = *EX.ReadReg1*))
 then ALUSelA = 10

Else // RegFile value is latest
 ALUSelA = 00

If both, MEM and WB stage contain an instruction producing the value needed by the EX stage, Mem stage should prevail since it has the latest producer



ReadReg2 Forwarding

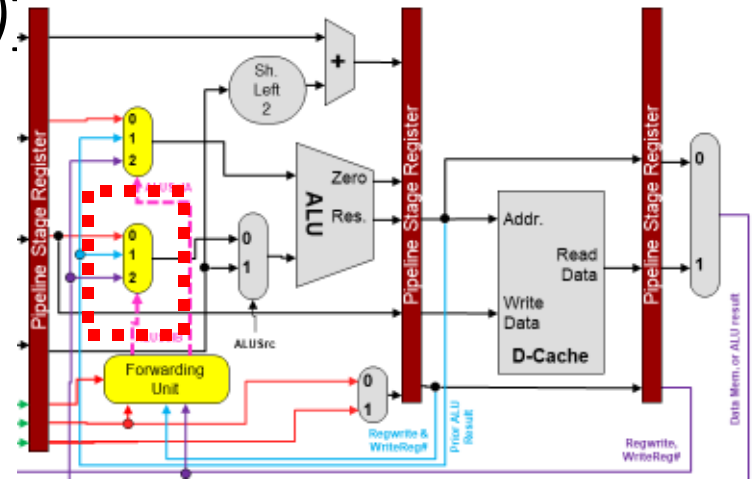
- ALUSelB mux

If (*MEM.RegWrite*
 and (*MEM.WriteReg* \neq 0)
 and (*MEM.WriteReg* = *EX.ReadReg2*))
 then ALUSelB = 01

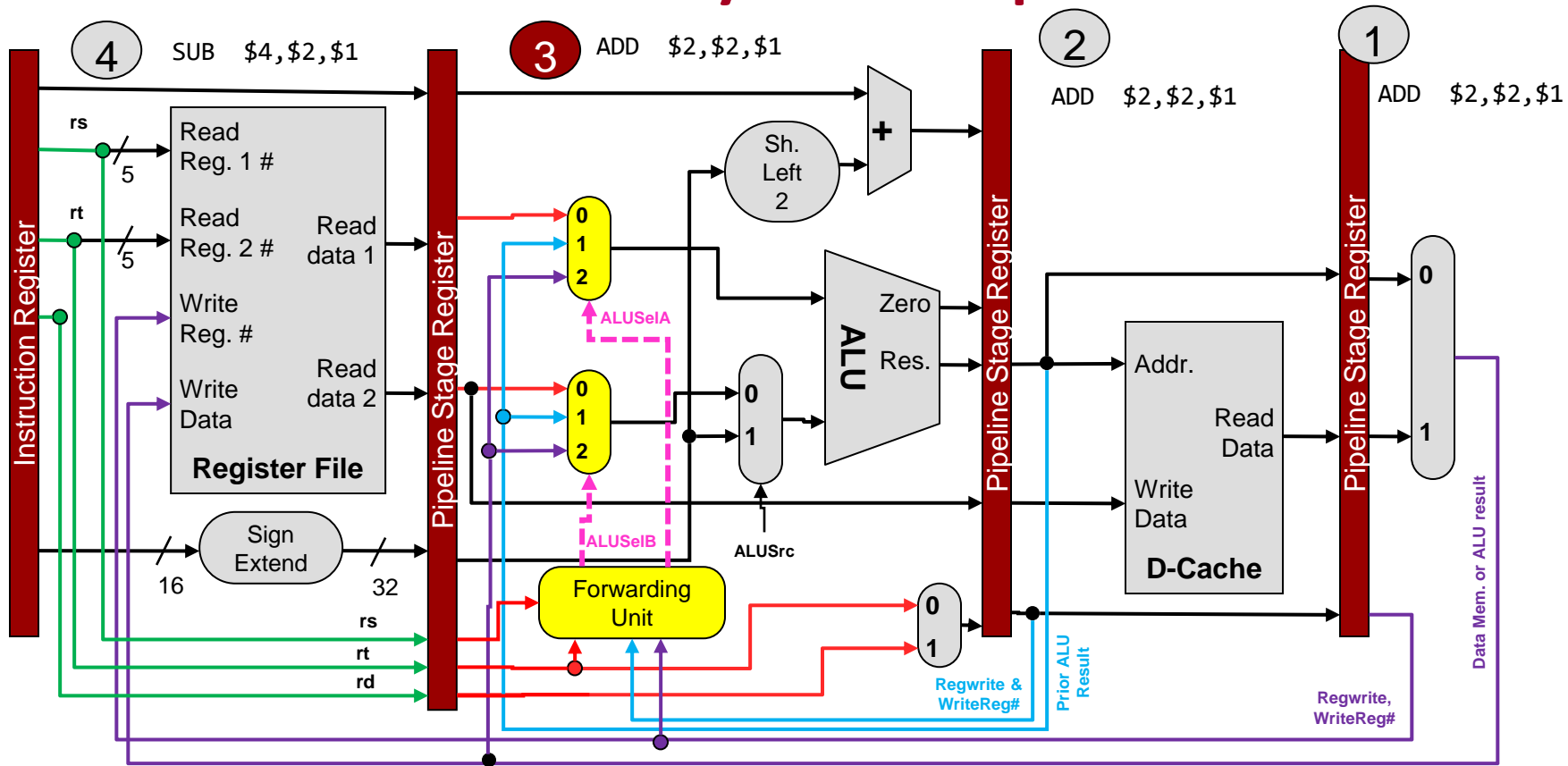
Else if (*WB.RegWrite*
 and (*WB.WriteReg* \neq 0)
 and (*WB.WriteReg* = *EX.ReadReg2*))
 then ALUSelB = 10

Else // RegFile value is latest
 ALUSelB = 00

If both, MEM and WB stage contain an instruction producing the value needed by the EX stage, Mem stage should prevail since it has the latest producer



EX Priority Example

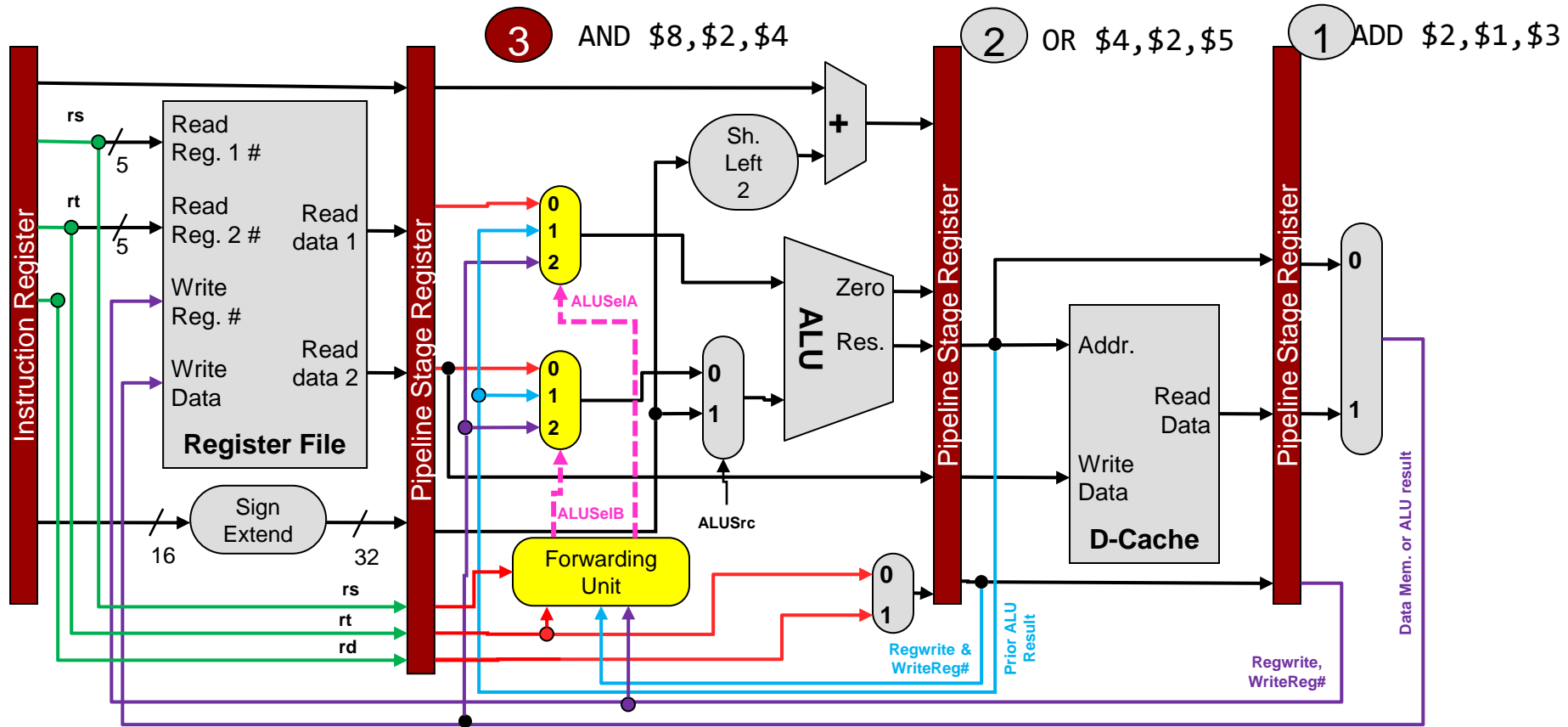


	Instruction	Explanation (Assume init value of \$2 = 0x03 and \$1 = 0x01)
1	ADD \$2,\$2,\$1	New \$2 should equal 0x04
2	ADD \$2,\$2,\$1	New \$2 should equal 0x05
3	ADD \$2,\$2,\$1	New \$2 should equal 0x06
4	SUB \$4,\$2,\$1	...

Who should help instruction 3? *Instruc. 2 or 1*

Instruc 2 (the latest producer)

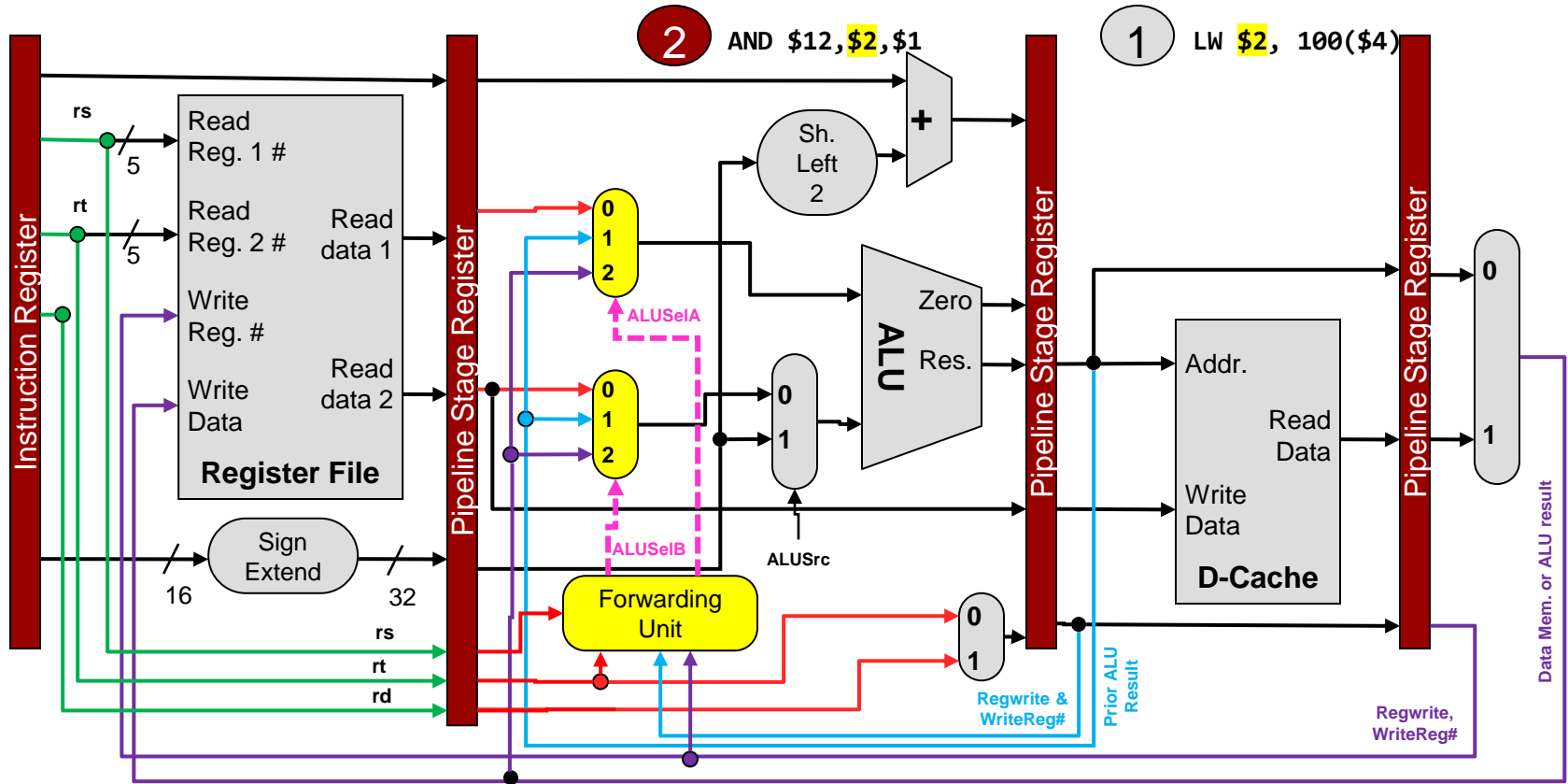
Different Forward Sources



	Instruction	
1	ADD \$2 , \$1, \$3	
2	OR \$4, \$2 , \$5	
3	AND \$8, \$2 , \$4	

Who should help instruction 3?
Both Instruc 1 (for \$2) and Instruc 2 (for \$4)
(the latest producer of each dependent register)

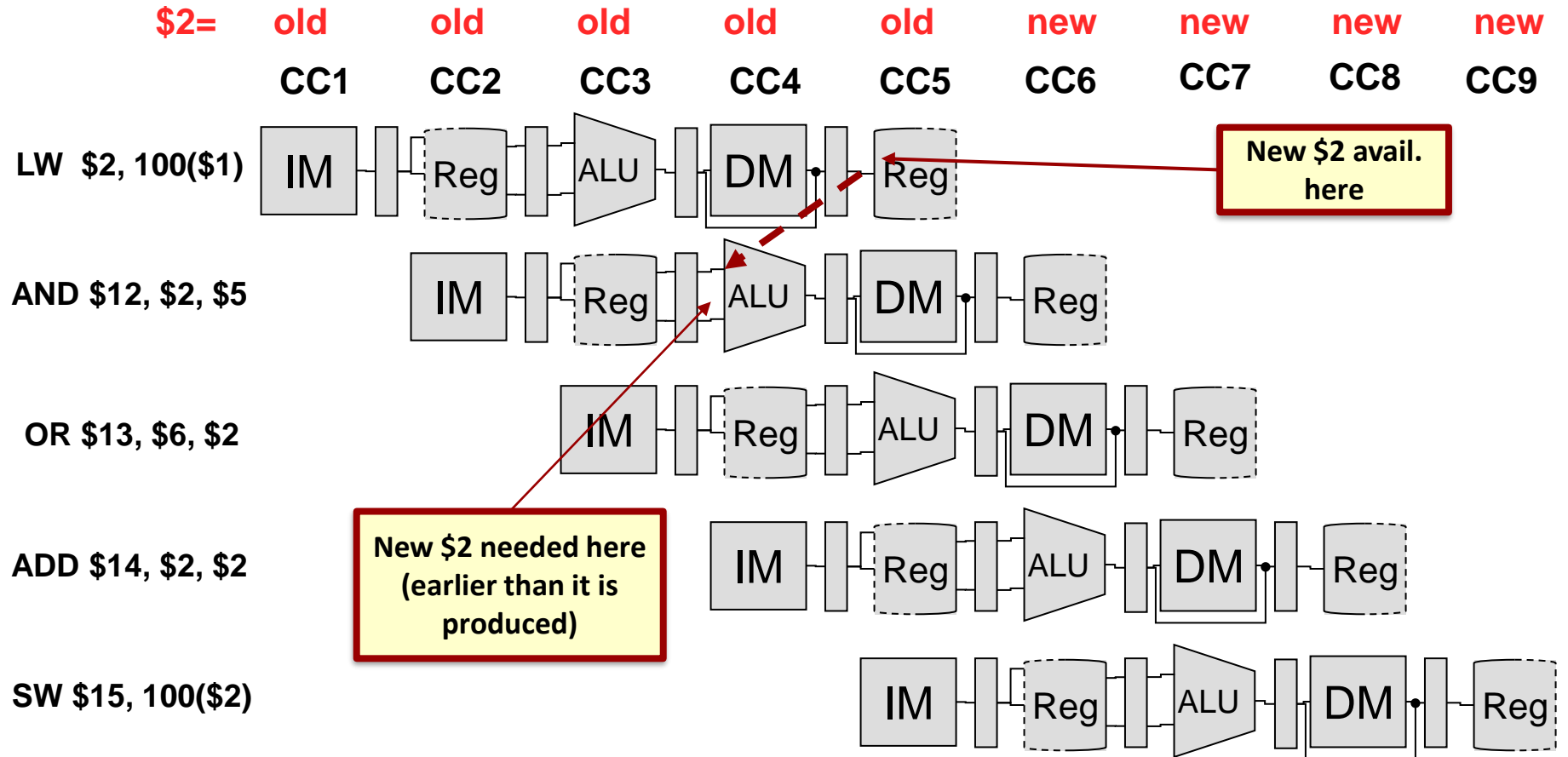
Don't Declare Success Yet



	Instruction	
1	LW \$2 , 100(\$4)	
2	AND \$12, \$2 , \$1	
3	SUB \$8, \$2, \$4	

Is the new value of register \$2 available for forwarding when 'AND' needs it?
NO!!

Understanding the Problem



You cannot forward data "back" in time. In these time space diagrams, forwarding must be "forward" in time

- What can we do to solve this problem? Stall!!

Back to the HDU

- Re-introduce the HDU to handle the case of a LW immediately followed by a dependent instruction
- EX Hazard:

If (ID/EX.RegWrite and ID/EX.MemRead = 1 and
 (ID/EX.WriteRegRt = IF/ID.ReadReg1
 or
 ID/EX.WriteRegRt = IF/ID.ReadReg2))




Then

Stall the Pipeline

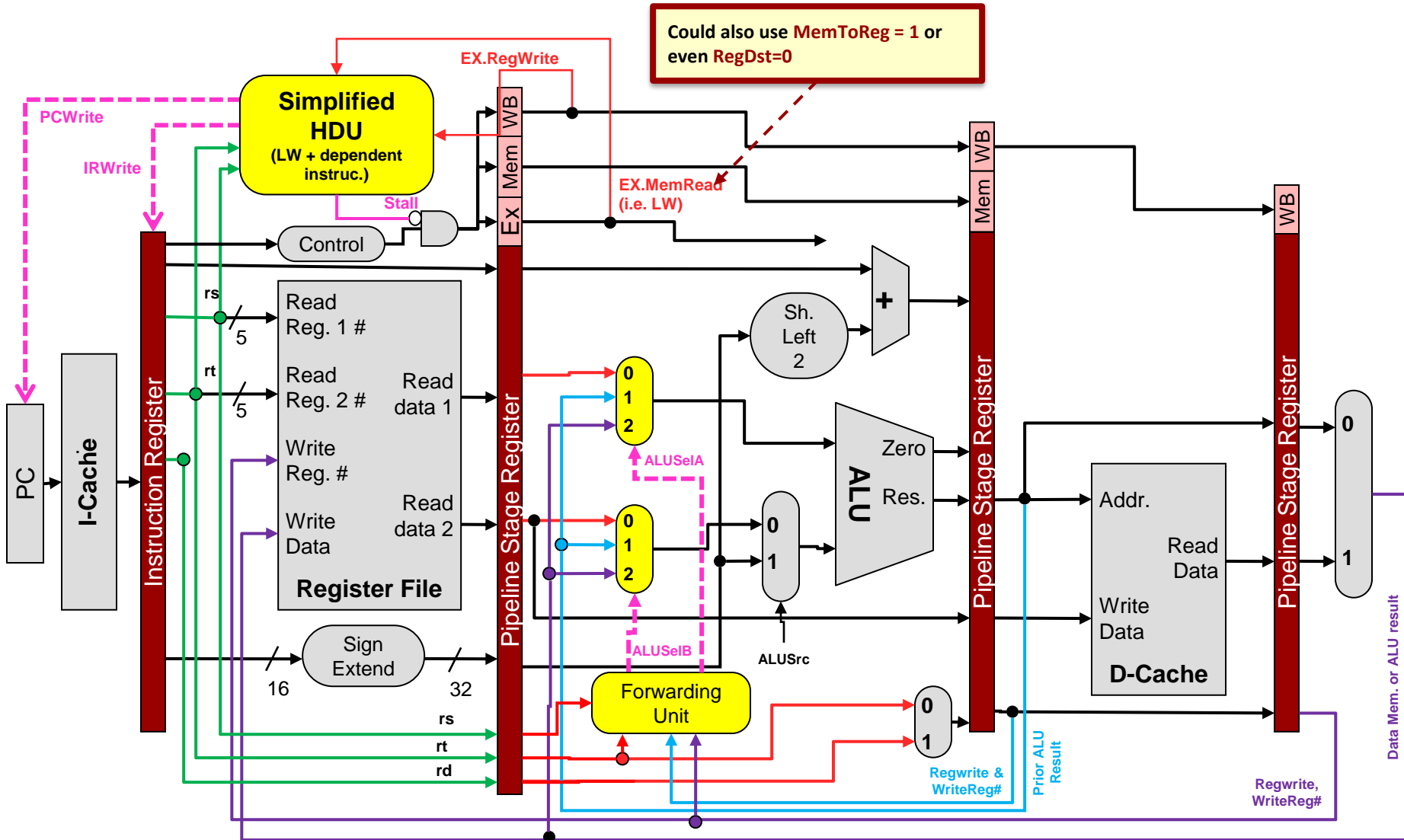
```

LW  $2, 100($1)
AND $12, $2, $5
OR  $13, $6, $2
ADD $14, $2, $2
SW  $15, 100($2)
```

Note: We use **MemRead = 1** to indicate the instruction in ID/EX is an LW. We could also use **MemToReg = 1** or even **RegDst=0**

	Fetch	Decode	Exec.	Mem.	WB
C1	LW				
C2	AND	LW			
C3	OR	AND	LW		
C4	OR	AND		LW	
C5	ADD	OR	AND		LW
C6	SW	ADD	OR	AND	
C7	...	SW	ADD	OR	AND

Back to the HDU



One More Consideration

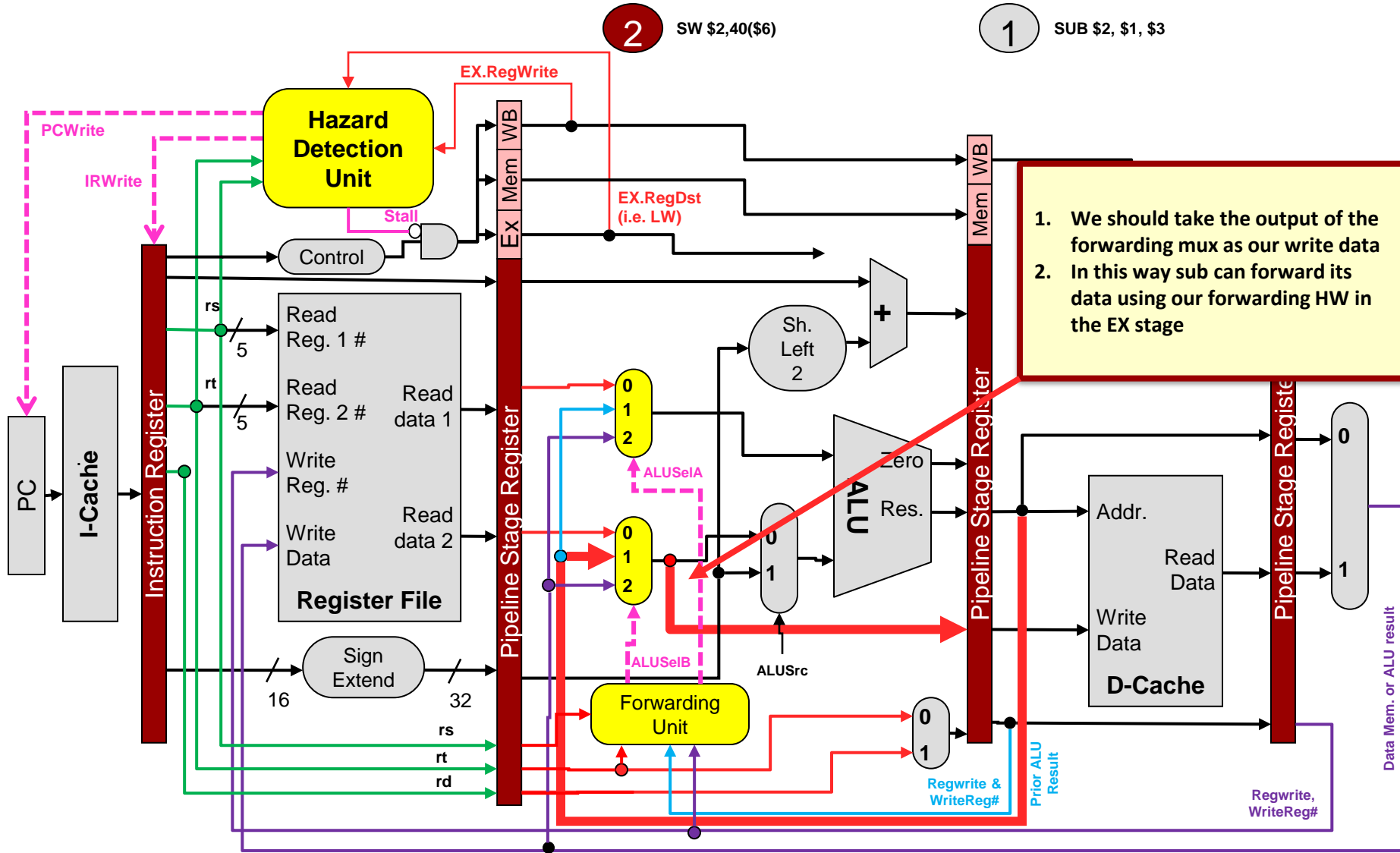
- Consider the sequence shown to the right
- Is there a dependency?
 - Yes, SW needs the new value of \$2 to write to memory
- Do we have the forwarding paths to handle this dependency?
 - At first glance no, because it may seem we need to forward from WB back to MEM
 - But we can actually forward earlier from MEM back to EX and use our current forwarding muxes

SUB \$2, \$1, \$3
 SW \$2, 40(\$6)

	Fetch	Decode	Exec.	Mem.	WB
C1	SUB				
C2	SW	SUB			
C3	i	SW	SUB		
C4	i+1	i	SW	SUB	
C5	i+2	i+1	i	SW	SUB

	Fetch	Decode	Exec.	Mem.	WB
C1	SUB				
C2	SW	SUB			
C3	i	SW	SUB		
C4	i+1	i	SW	SUB	
C5	i+2	i+1	i	SW	SUB

Dealing with Memory Dependency



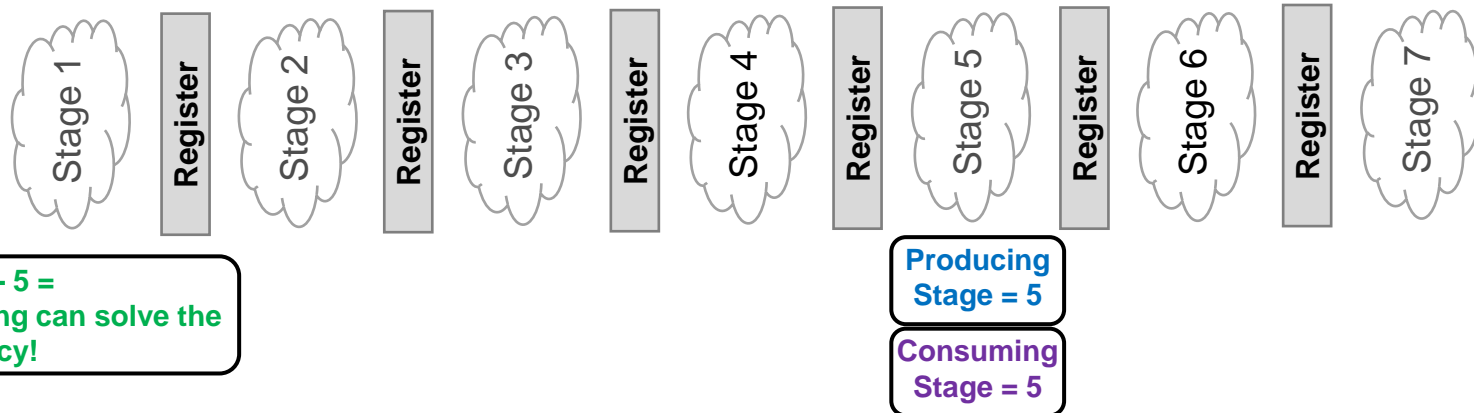
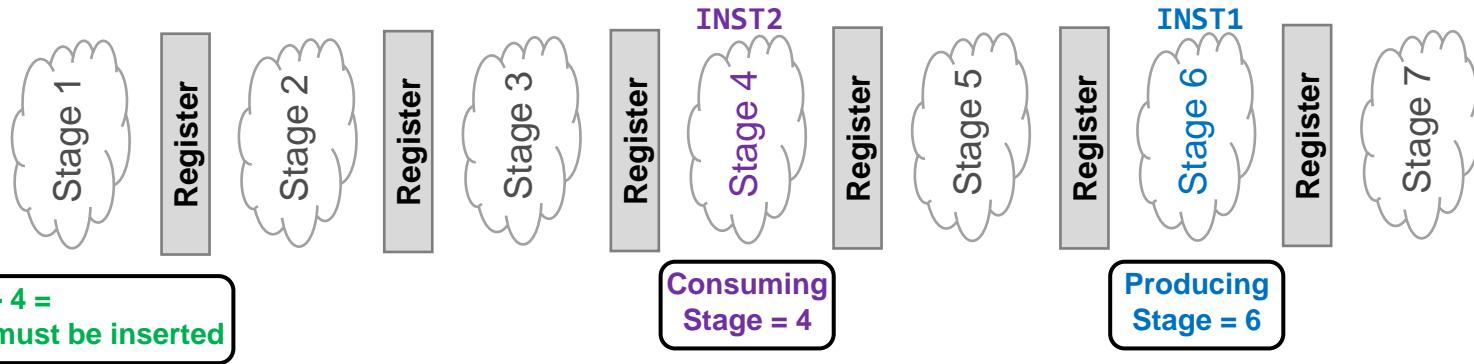
1. We should take the output of the forwarding mux as our write data
2. In this way sub can forward its data using our forwarding HW in the EX stage

Calculating Stall Cycles

- To find the number of bubbles (stall cycles) that the HDU will need to insert:
 - # stall cycles = Producing Stage Depth – Consuming Stage Depth

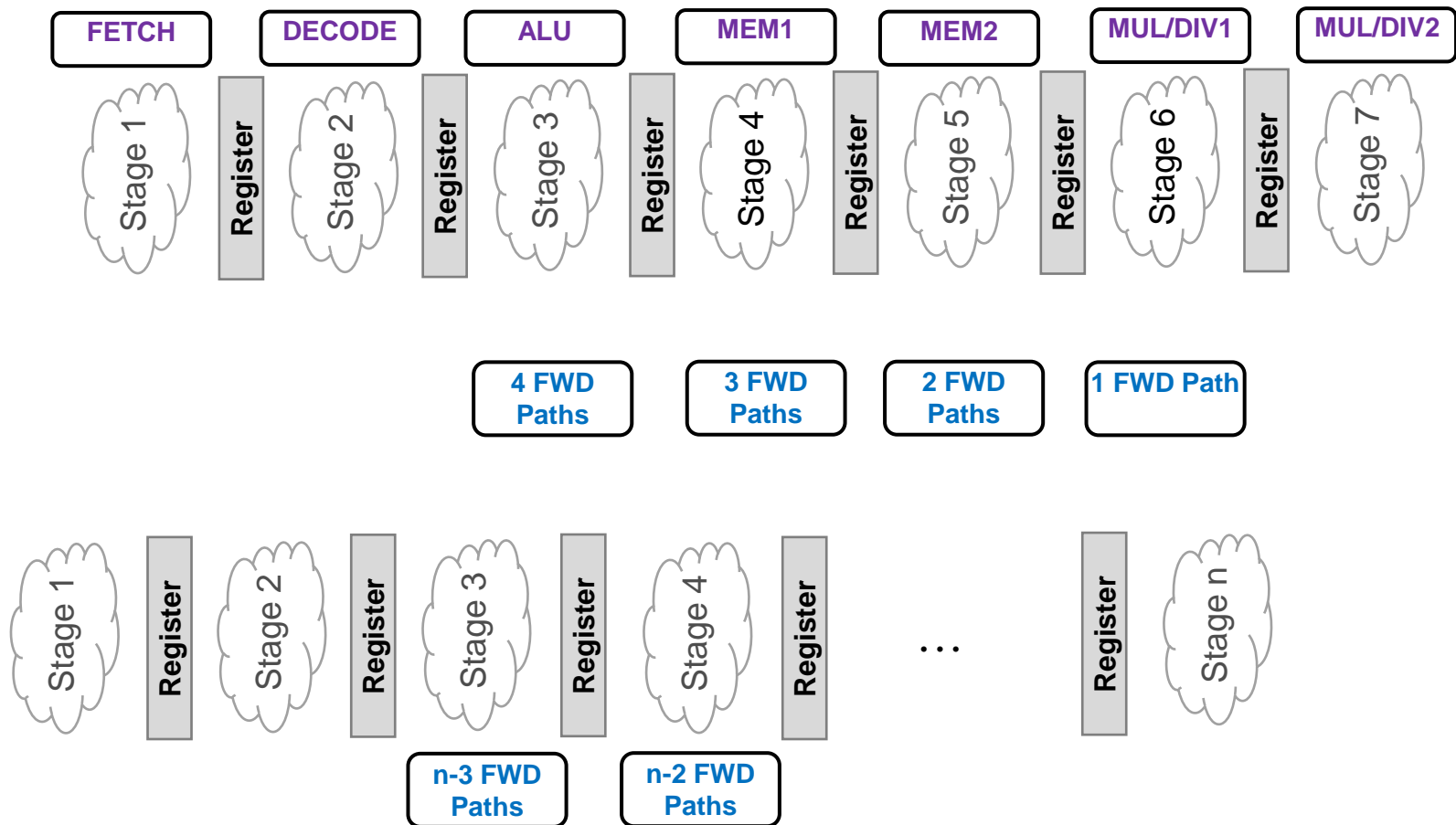
```

INST1 $2, x, x # Producer
INST2 x, $2, x # Consumer
    
```



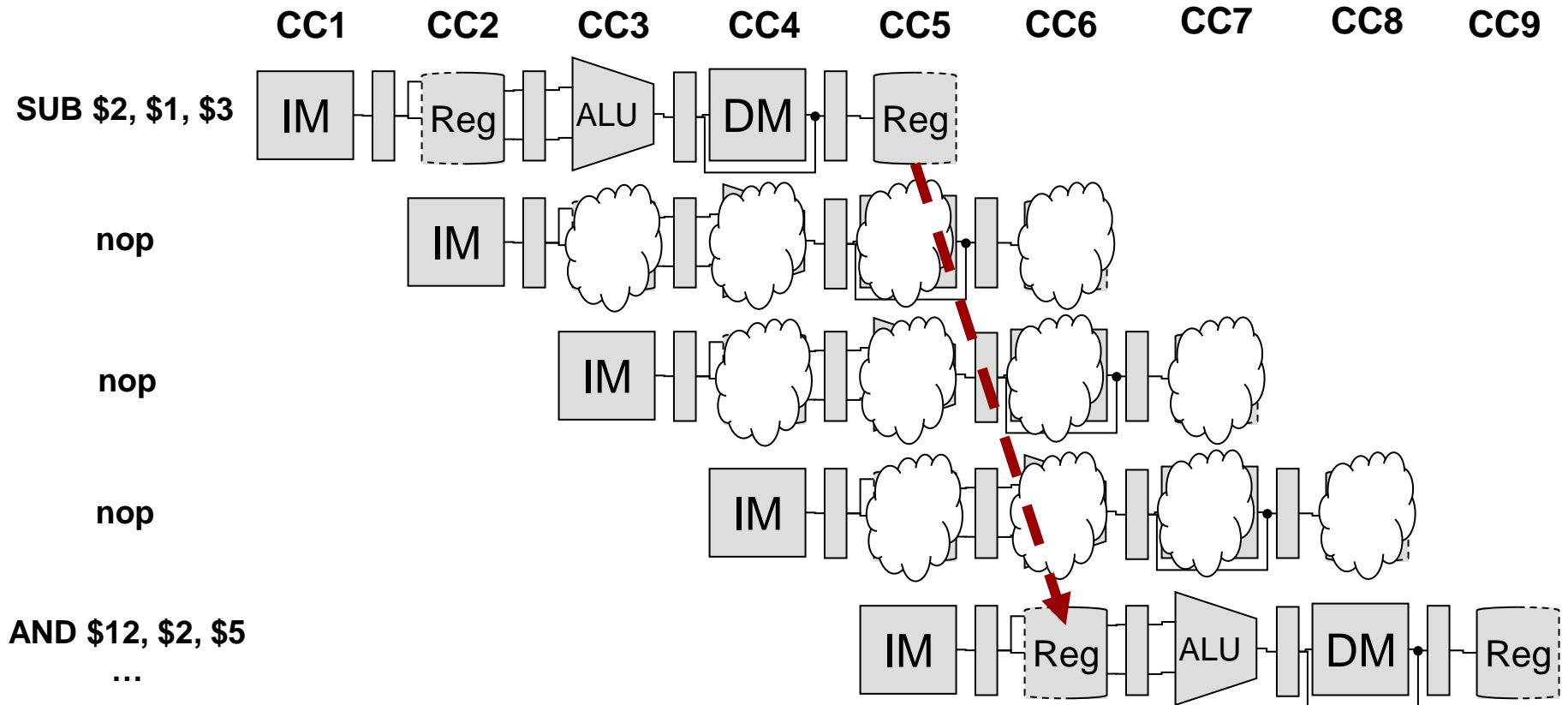
Forwarding Unit Complexity

- Consider how many muxes and pathways must be added to support forwarding in the worst case? For n stages, forward logic complexity = $O(n^2)$



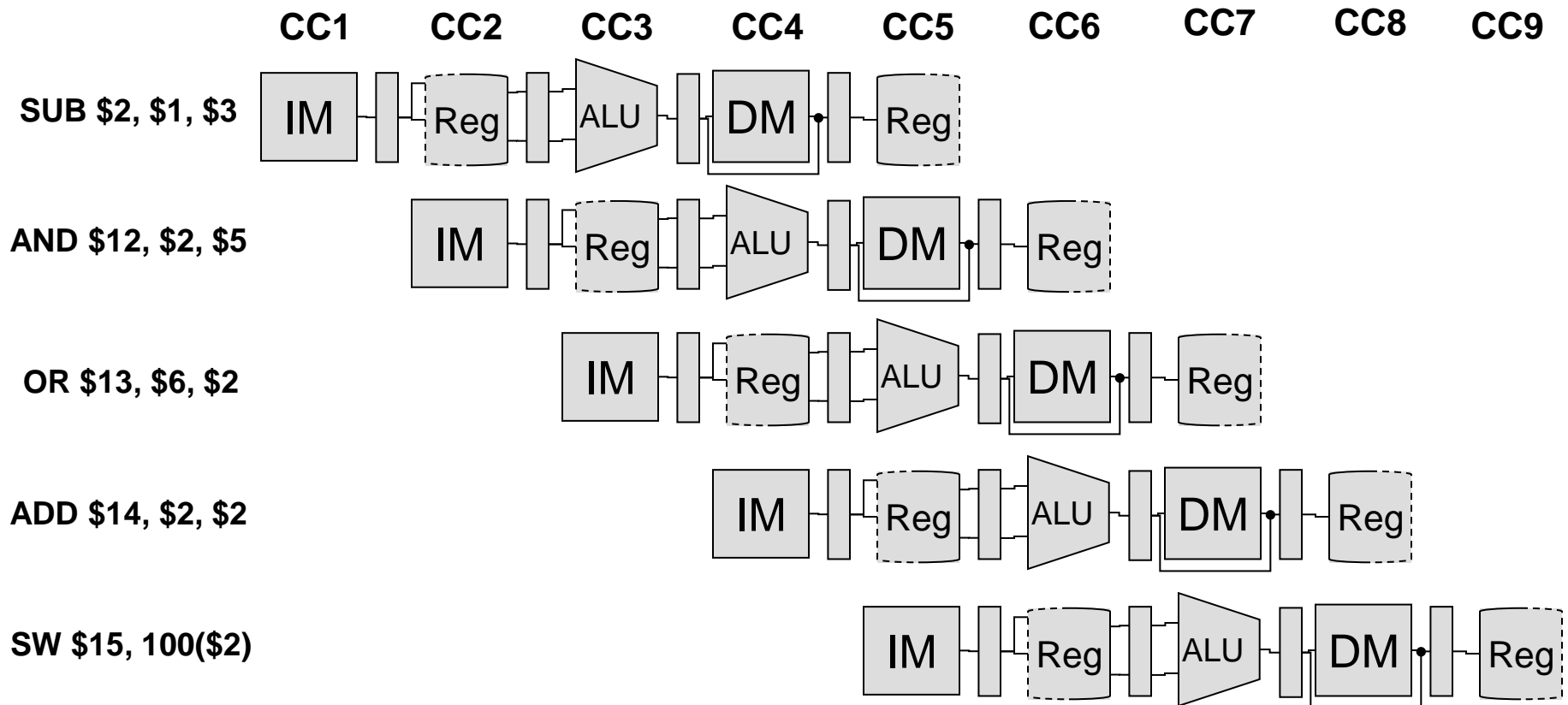
IMAGES

An Opening Example (nops)



An Opening Example

- ds



Hazards

- EX Hazard
 - HDU: Hazard occurs if data dependence between ID and EX stages
 - FU: Between EX and MEM stage
- MEM Hazard
 - HDU: Hazard occurs if data dependence between ID and MEM stages
 - FU: Between EX and WB stages
- Idea: Hazard is named based on who produces the data the *dependent instruction* needs

Book may refer to this as an EX hazard since data being forwarded from the MEM stage was produced in the EX stage (but, due to our datapath, is not forwarded until it reaches the MEM stage).

Hazard Definitions

- MEM Hazard

If [*MEM.RegWrite*
 and (*MEM.WriteReg* \neq 0)
 and (*MEM.WriteReg* = *EX.ReadReg1*)]

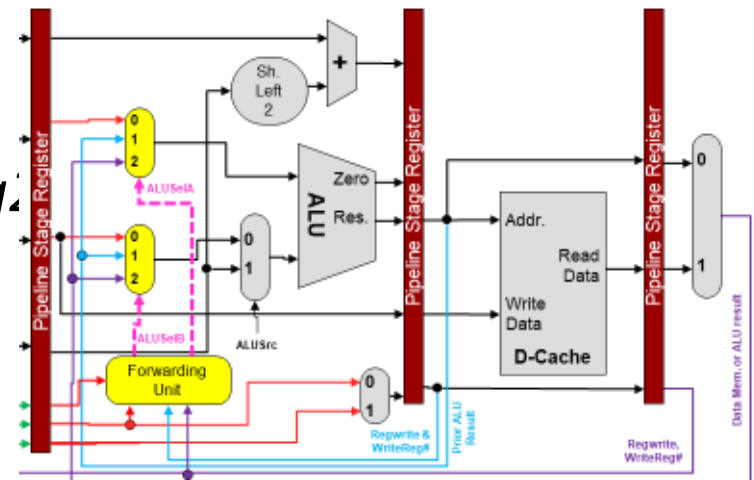
Then EX1 = True

If (*EX1* = True) then ALUSelA = 01

If [*MEM.RegWrite*
 and (*MEM.WriteReg* \neq 0)
 and (*MEM.WriteReg* = *EX.ReadReg2*)]

Then EX2 = True

If (*EX2* = True) then ALUSelB = 01



Hazard Definitions

- MEM Hazard

[*MEM/WB.RegWrite*
and (*MEM/WB.WriteReg* \neq 0)
and (*MEM/WB.WriteReg* = *EX.ReadReg1*)]
and (*EX1* \neq *True*)

Then *ALUSelA* = 10

If [*MEM/WB.RegWrite*
and (*MEM/WB.WriteReg* \neq 0)
and (*MEM/WB.WriteReg* = *ID/EX.ReadReg2*)
and (*EX2* \neq *True*)

Then *ALUSelB* = 10

An EX Hazard should prevail over a MEM hazard since the EX hazard has the latest data

Hazards

- EX Hazard
 - HDU: Hazard occurs if data dependence between ID and EX stages
 - FU: Between EX and MEM stage
- MEM Hazard
 - HDU: Hazard occurs if data dependence between ID and MEM stages
 - FU: Between EX and WB stages
- Idea: Hazard is named based on who _____ the data the *dependent instruction* needs