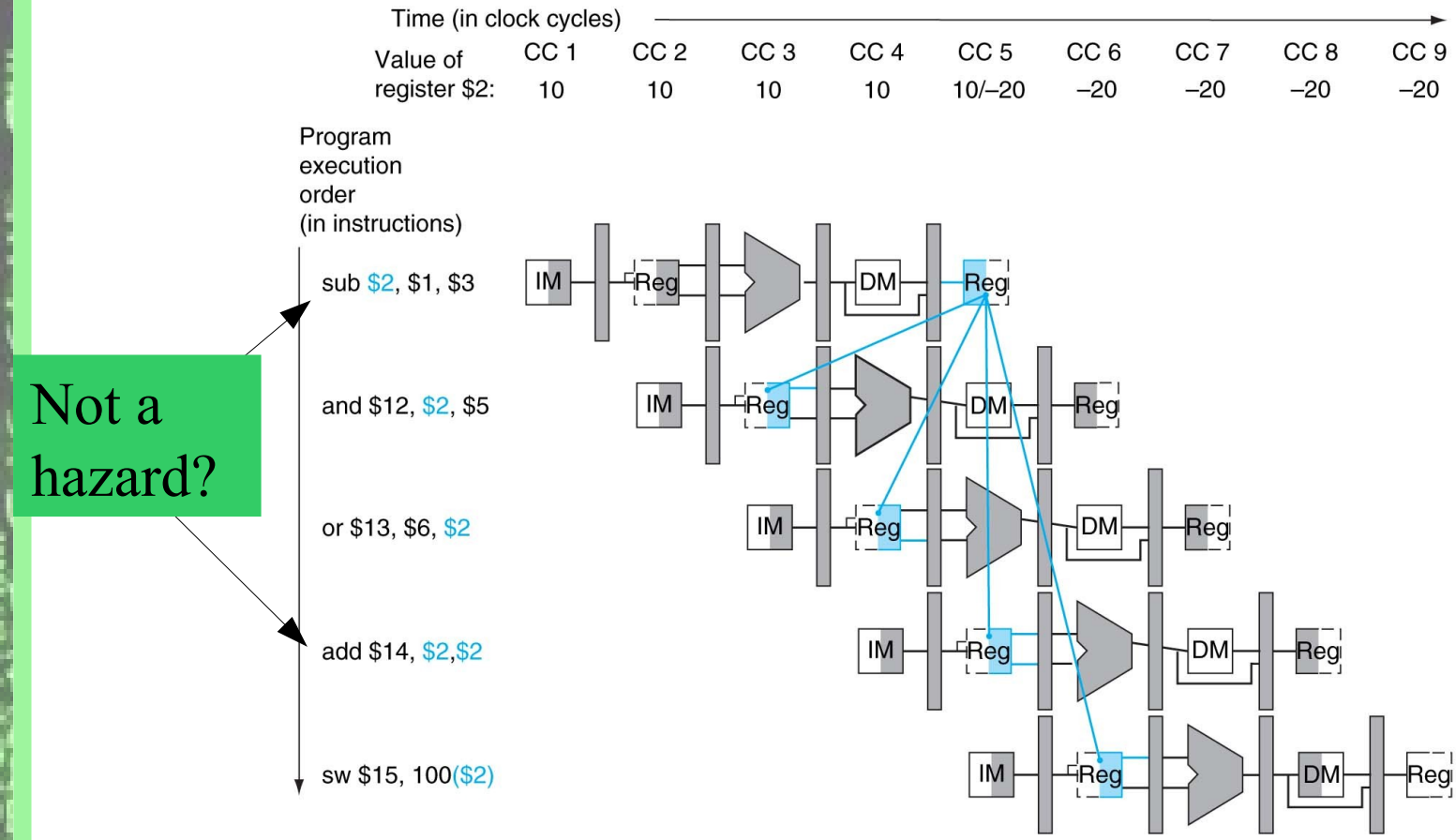
CSE 2021
COMPUTER ORGANIZATION

HUGH CHESSER
CSE B 1012U

W12-W

Data Hazards

Consider the following instruction sequence and the resulting pipeline diagram…



Quiz/Exam Sample Question

Show the contents of the pipeline registers for instructions going through the pipeline (5 cycles after the first instruction begins), identify any data hazards:

```
lw  $10,20($1)
sub $11,$2,$3
and $12,$10,$5
or  $13,$11,$7
and $14,$8,$9
```

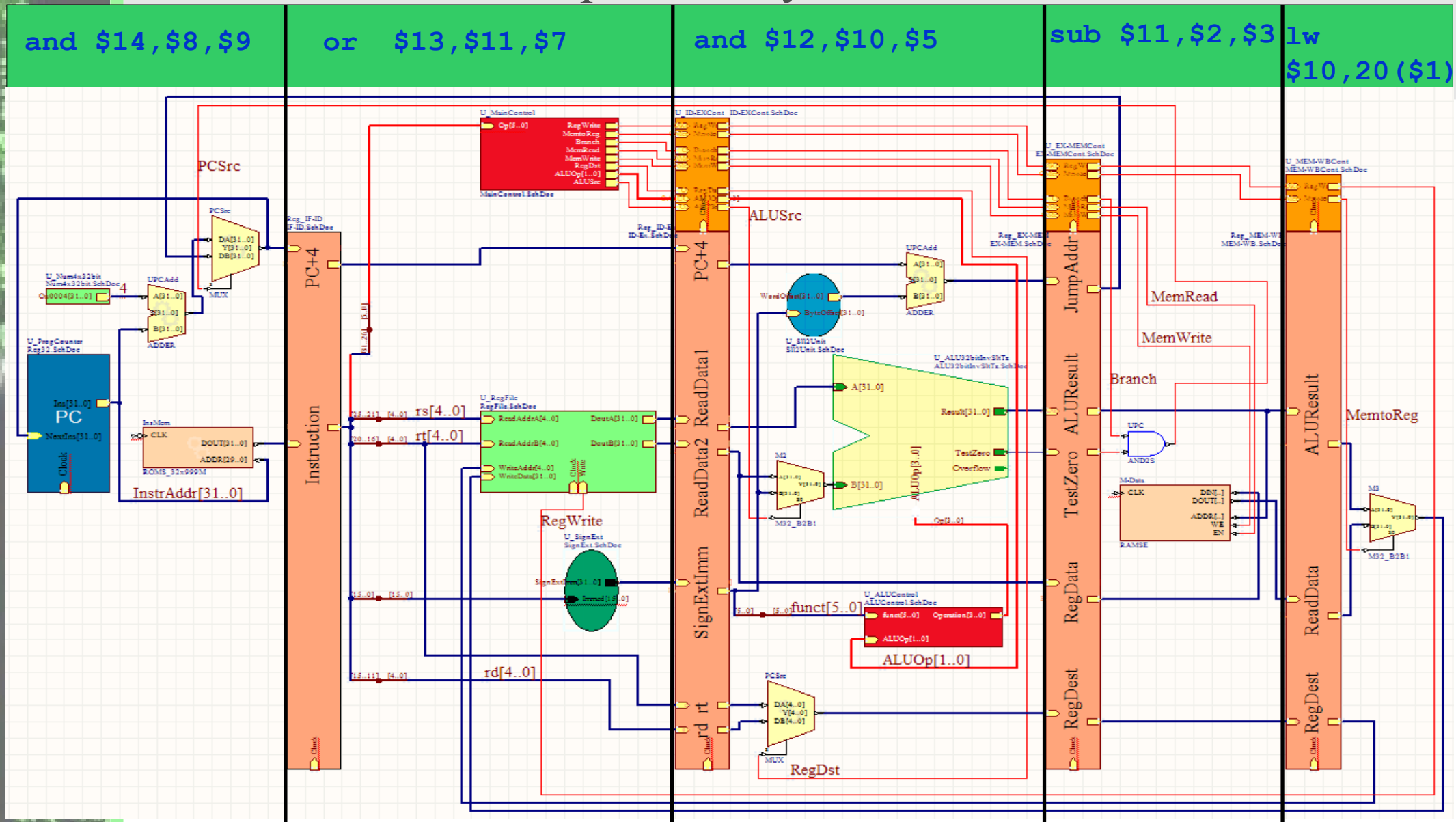
Use the short hand notation from the Green Sheet:

R[reg] – Contents of register file at register 'reg'

M[addr] – Contents of memory at address 'addr'

Operands 'SignExtImm', 'ZeroExtImm' and 'BranchAddr', JumpAddr' as per Green Sheet notes 2 - 5

Pipeline - Cycle # 5



ALU Control Actions

Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Action of Pipeline Control Signals

Signal name	Effect when deasserted (0)	Effect when asserted (1)
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

Control for Pipeline – Arranged by Pipeline Stage

Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Controls same as before for the single or multi-cycle implementations, rearranged according to pipeline stage

Agenda

Topics:

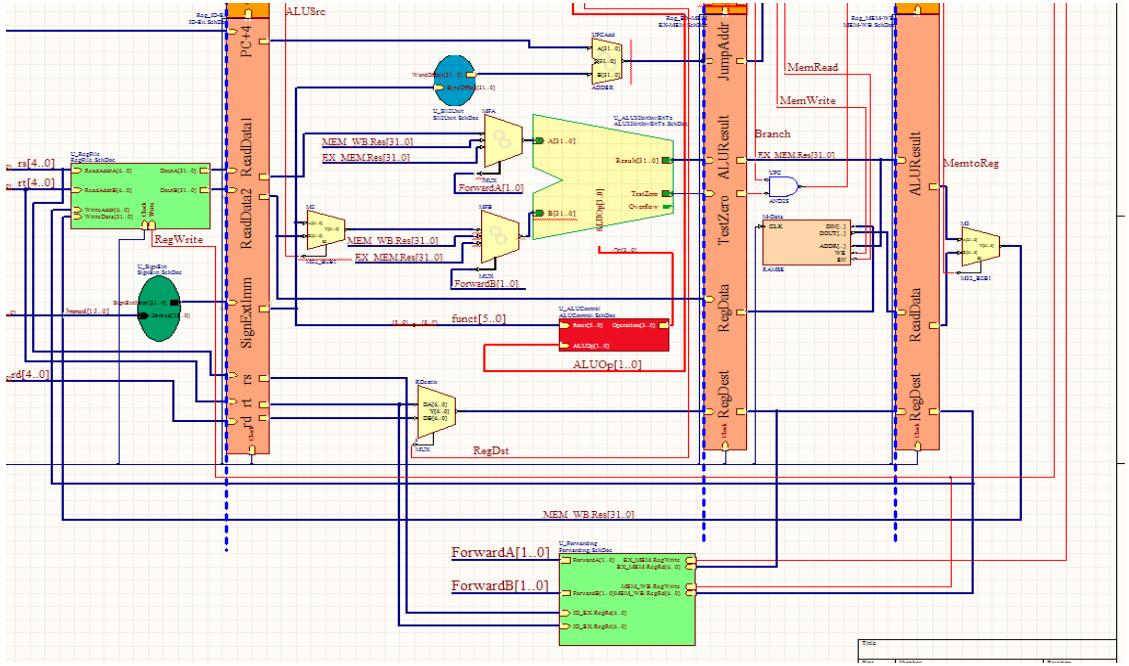
1. Data Hazards – Forwarding – complete
2. Control Hazards

Patterson: 4.7, 4.8

Forwarding from MEM/WB Pipeline Register

Conditions:

```
(MEM/WB.RegWrite &
MEM/WB.RegisterRd ≠ 0 &
MEM/WB.RegisterRd=ID/EX.RegisterRs) ->
ForwardA = 01

(MEM/WB.RegWrite &
MEM/WB.RegisterRd ≠ 0 &
MEM/WB.RegisterRd=ID/EX.RegisterRt) ->
ForwardB = 01
```




```
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
[Note: Only R-type instructions
covered for forwarding,
no I-type, eg - sw $2, 0($13) (Rd = 0)
```

Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

a. R-type instruction

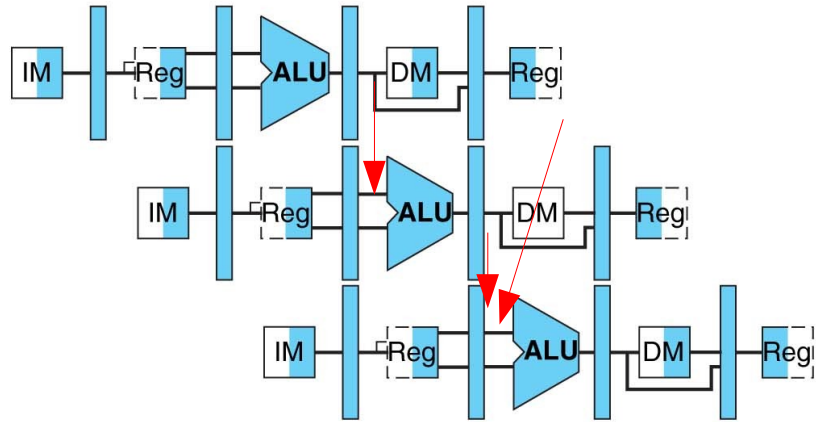
Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

b. Load or store instruction

Additional Condition for WB Forwarding

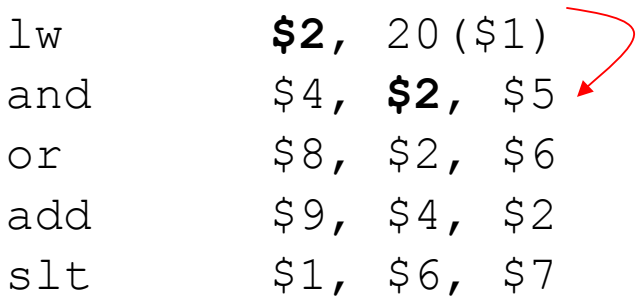
```
(MEM/WB.RegWrite &
MEM/WB.RegisterRd ≠ 0 &
and not(EX/MEM.RegWrite and EX/MEM.RegRd ≠ 0)
and (EX/MEM.RegRd ≠ ID/EX.RegRd)
and MEM/WB.RegisterRd=ID/EX.RegisterRs) ->
ForwardA = 01
```

```
                                    add $1, $1, $2

                                    add $1, $1, $3

(MEM/WB.RegWrite &                  add $1, $1, $4
MEM/WB.RegisterRd ≠ 0 &
and not(EX/MEM.RegWrite and EX/MEM.RegRd ≠ 0)
and (EX/MEM.RegRd ≠ ID/EX.RegRd)
and MEM/WB.RegisterRd=ID/EX.RegisterRt) ->
ForwardB = 01
```



Stalls

Consider the code sequence



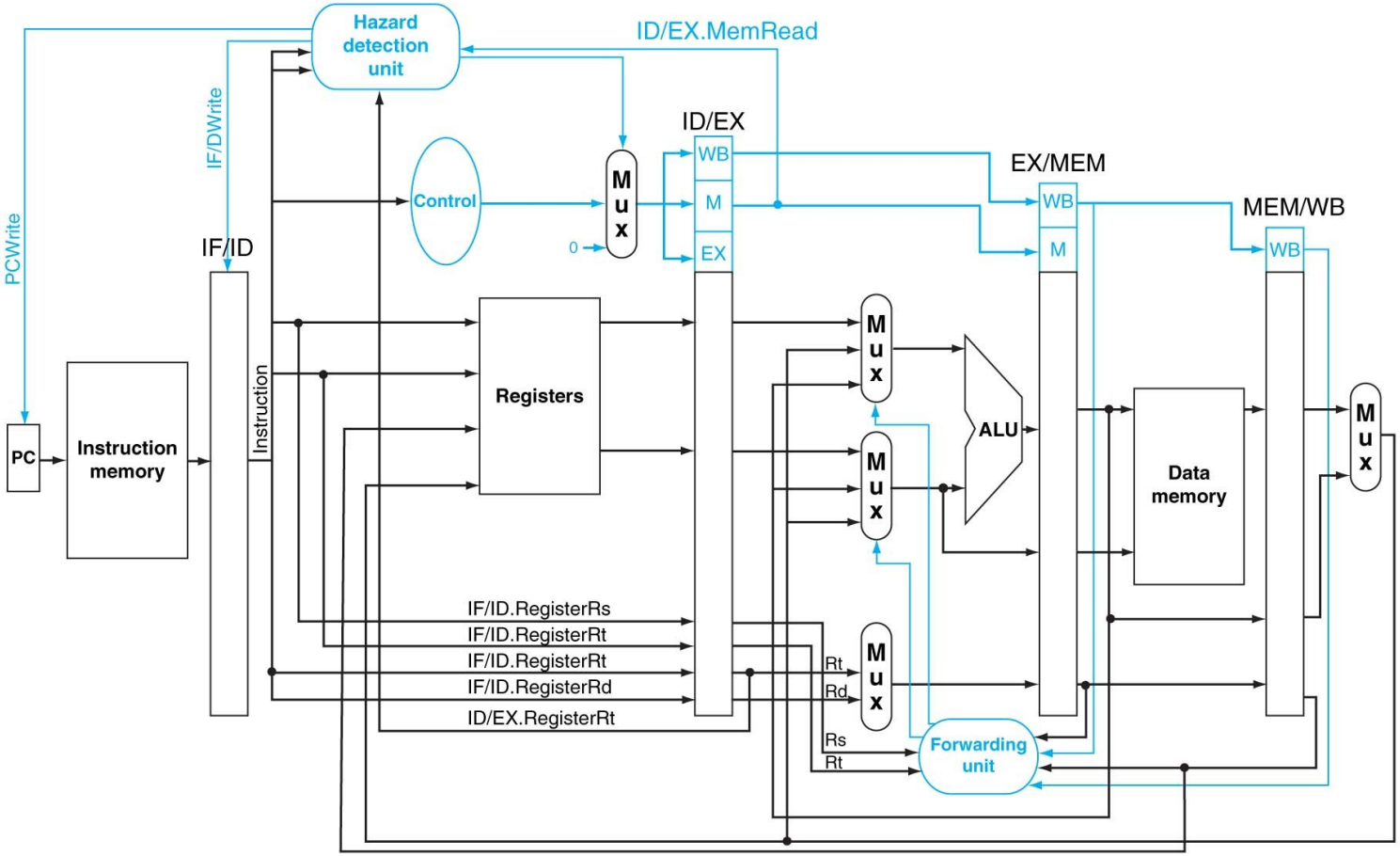
```
lw      $2, 20($1)
and     $4, $2, $5
or      $8, $2, $6
add     $9, $4, $2
slt     $1, $6, $7
```

Hazard condition – identify at ID/EX stage

```
if (ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt)))
    stall the pipeline - 9 controls deasserted, PC not
       incremented
```

Hazard Detection Unit

By detecting the hazard at the IF/ID phase, PC and Control lines can be altered



Control Hazards

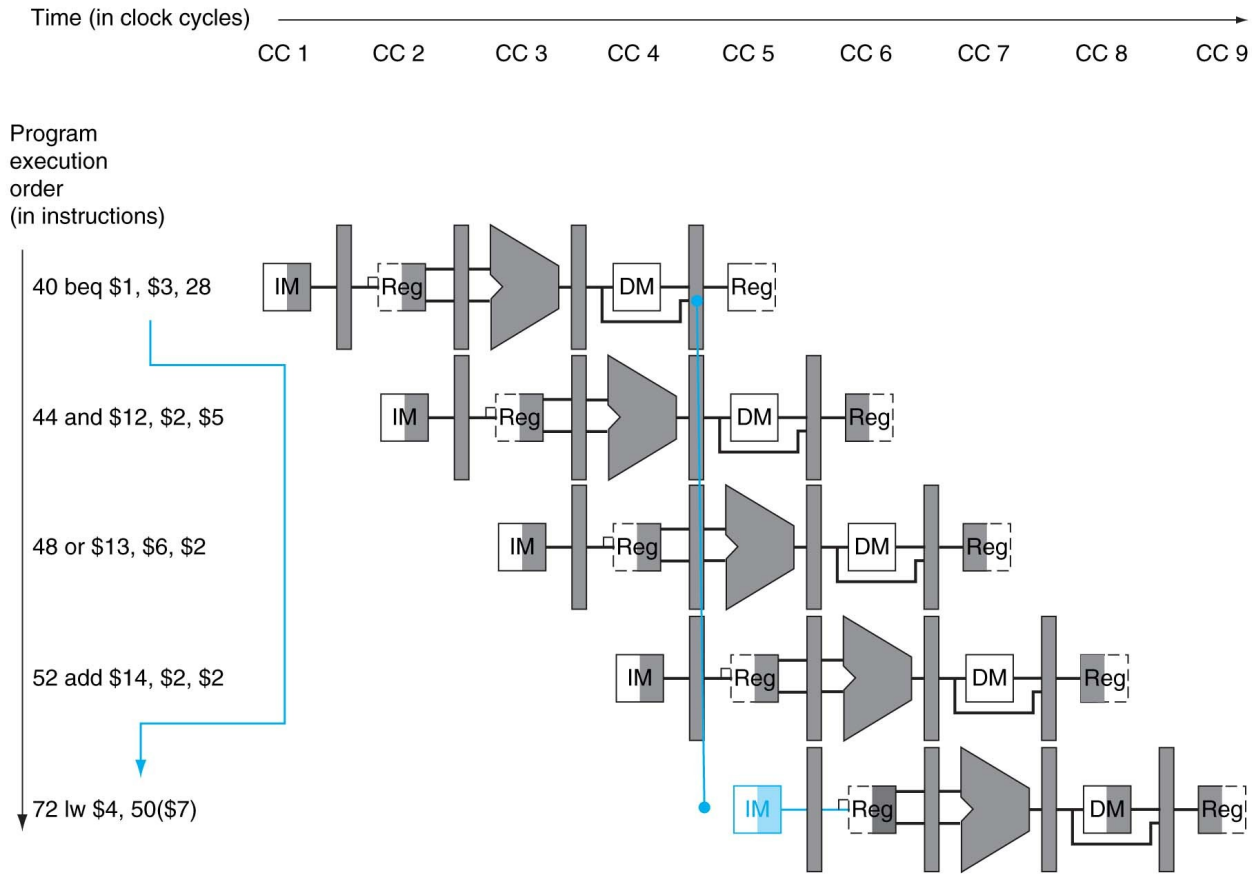
Three methods to minimize stalls

- Always assume branch not taken – must “flush” results if the branch IS taken
- Reduce branch delays – separate branch adder to calculate branch target address, move execution (test) earlier (“equality unit” – XOR)
- Dynamic branch prediction – remember if the branch was taken the last time the branch instruction was executed – branch history table

Assume Not Taken

Proceed with instructions as if branch not there

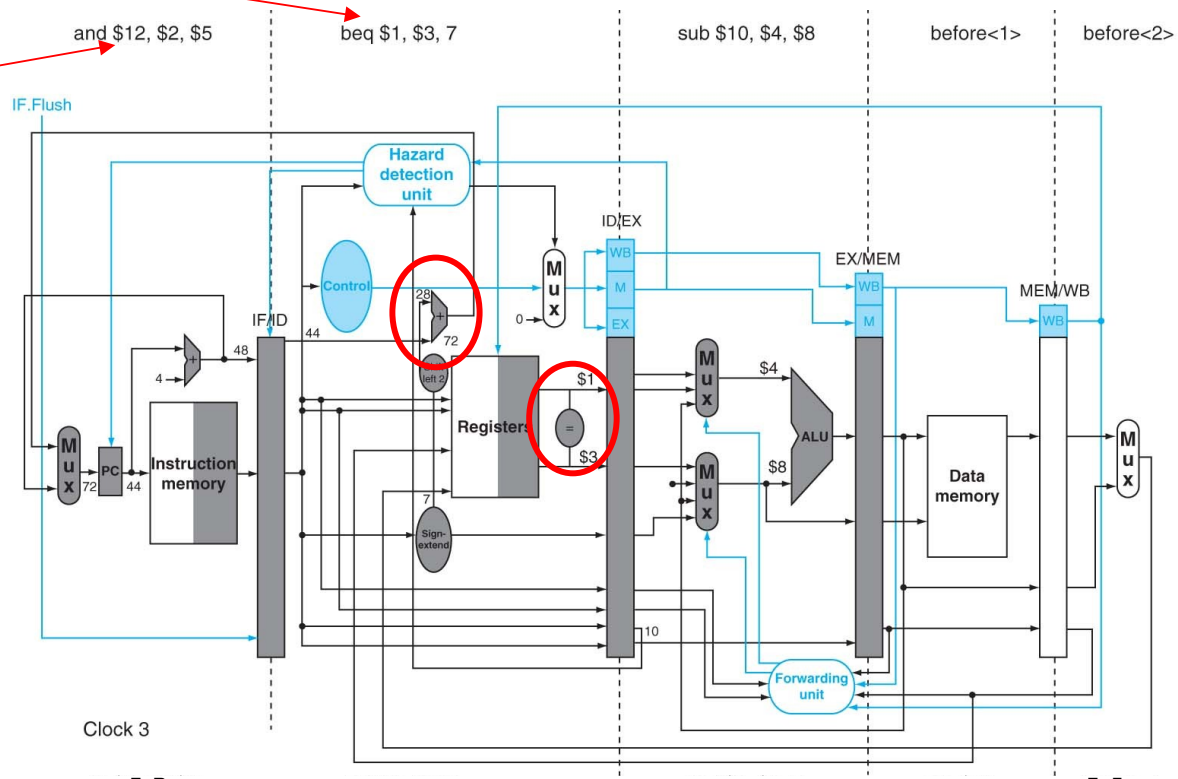
- "Flush" instructions in the pipeline if taken
- Effectively this creates a stall if branch is taken



Assume Branch Not Taken – Minimize Stall

```
36 sub $10, $4, $8

40 beq $1, $3, 7

44 and $12, $2, $5

…

72 lw  $4, 50($7)
```

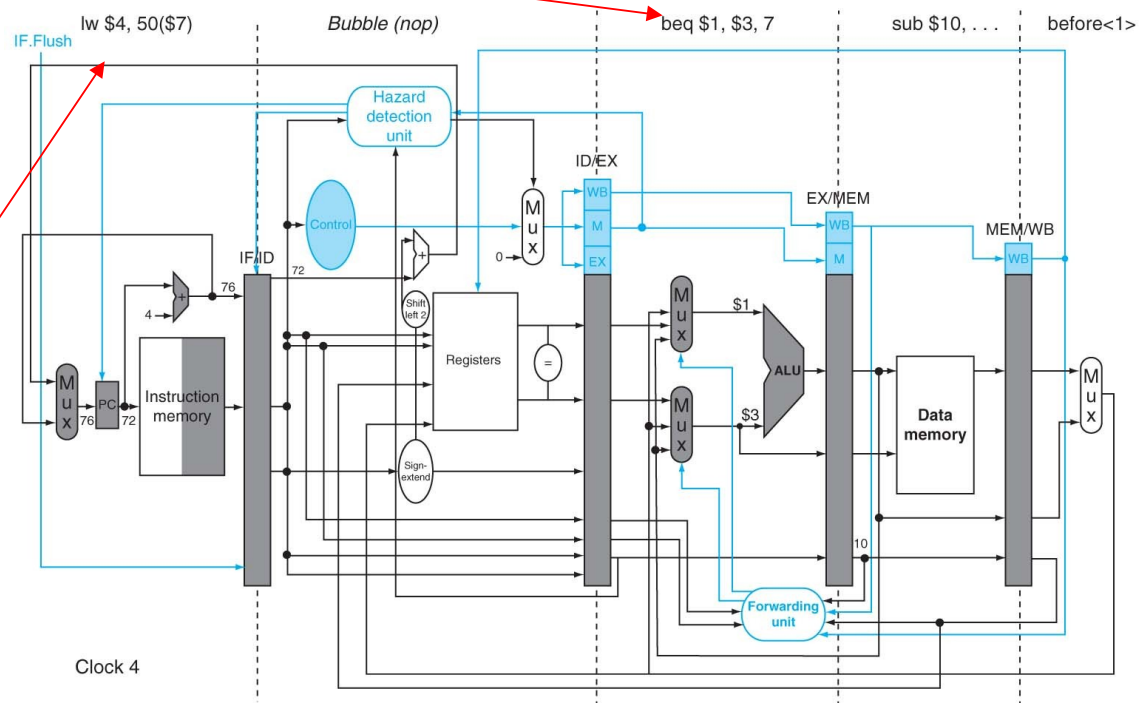
$$PC \leftarrow PC + 4 + BranchAddr = 40 + 4 + 7 \cdot 4 = 72$$



Branch adder and decision logic added to minimize branch stall – branch decided during ID phase

Branch to be taken – Flush current pipeline instructions

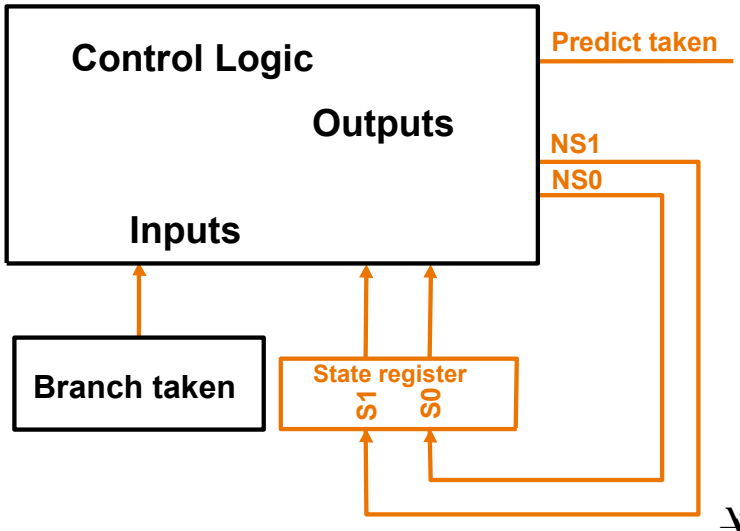
```
36 sub $10, $4, $8

40 beq $1, $3, 7

44 and $12, $2, $5

…

72 lw  $4, 50($7)
```

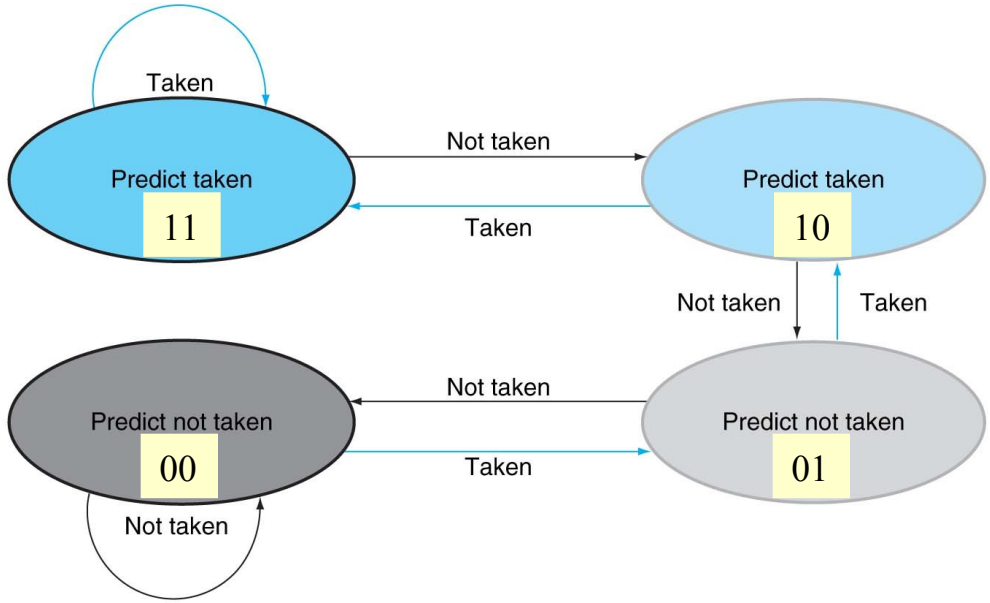


Dynamic Branch Prediction

Prediction of branches while the program is executing

- A portion of memory is utilized which indicates whether or not the branch was taken last time the instruction was executed
- FSM implementation is…





Finite State Machine

Control of Branch Prediction

Branch Taken	*State*	*Next State*	*Predict taken*
0	*00*	*00*	*0*
1	*00*	*01*	*0*
0	*01*	*00*	*0*
1	*01*	*10*	*0*
0	*10*	*01*	*1*
1	*10*	*11*	*1*
0	*11*	*10*	*1*
1	*11*	*11*	*1*