# CSE 2021
# Computer Organization

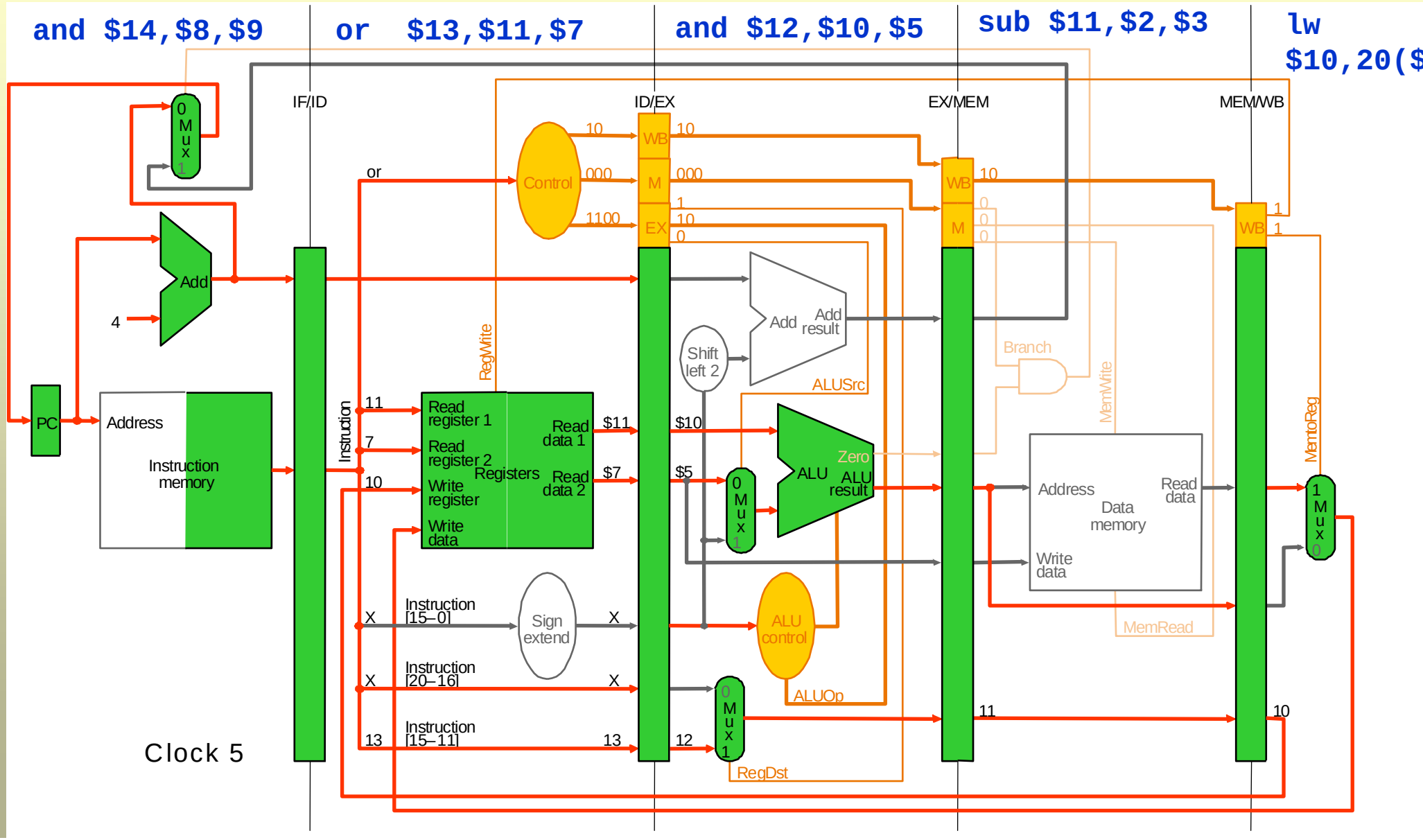
Hugh Chesser, CSEB
1012U

# 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
```

# 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*

# NS1 Example (1)


*Give the logic equation for the NS1 bit for the FSM*

- NS1 is true for states 2($0010_{two}$), 3 ($0011_{two}$), 6 ($0110_{two}$), 7 ($0111_{two}$)
- Referring to the truth table…

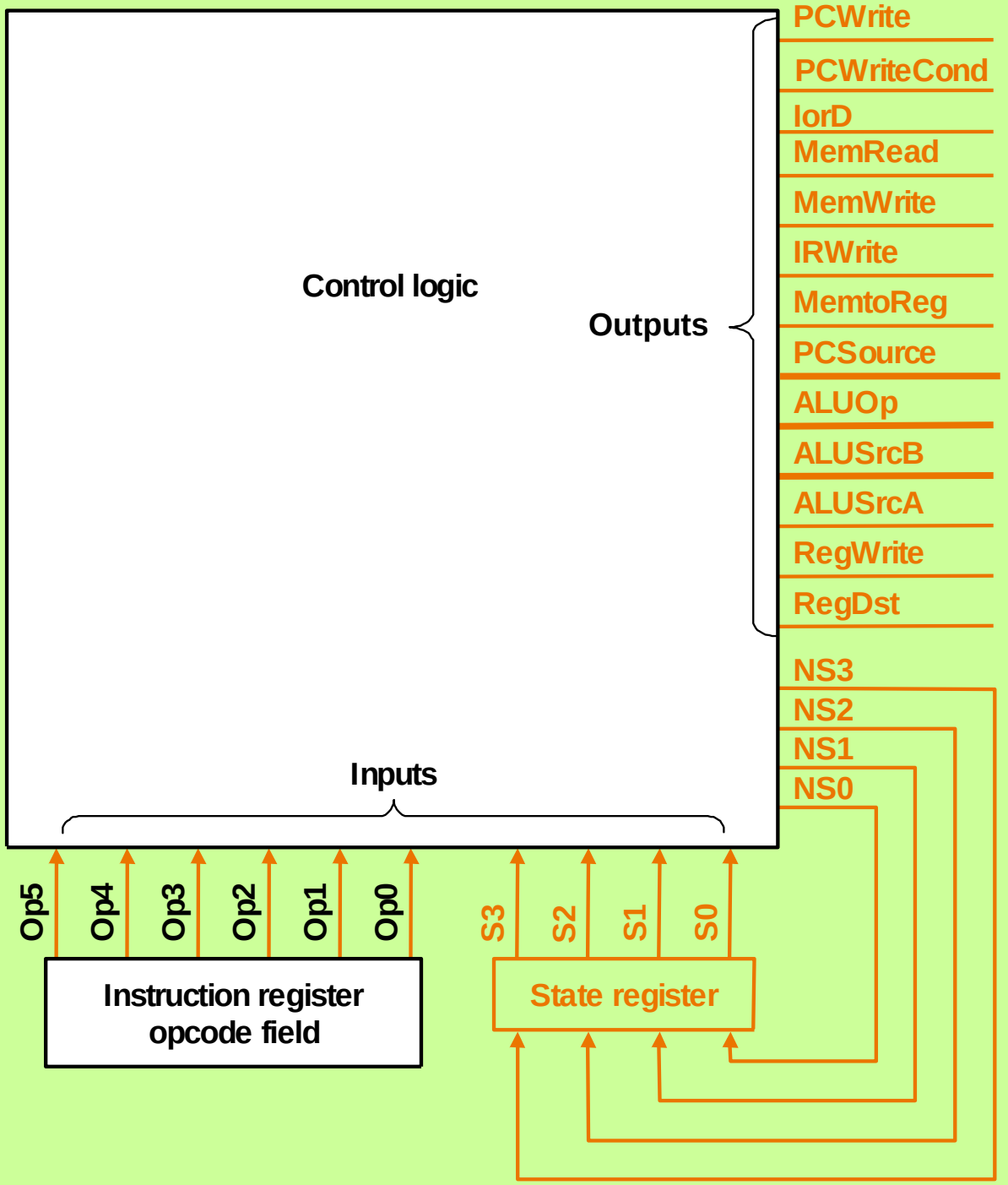
$$\text{NextState2} = \text{state1} \cdot (\text{'lw'+'sw'}) = \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot (Op = 0x23 \text{ OR } 0x2b)$$

| Output | Current states | Op |
|---|---|---|
| NextState0 | state4 + state5 + state7 + state8 + state9 | |
| NextState1 | state0 | |
| NextState2 | state1 | (Op = 'lw') + (Op = 'sw') |
| NextState3 | state2 | (Op = 'lw') |
| NextState4 | state3 | |
| NextState5 | state2 | (Op = 'sw') |
| NextState6 | state1 | (Op = 'R-type') |
| NextState7 | state6 | |
| NextState8 | state1 | (Op = 'beq') |
| NextState9 | state1 | (Op = 'jmp') |

*Take a couple of minutes to think about the other conditions…*

# Finite State Machine

## Control of Multicycle Datapath (5)



# NS1 Example (2)


*Give the logic equation for the NS1 bit for the FSM*

| Output | Current states | Op |
|---|---|---|
| NextState0 | state4 + state5 + state7 + state8 + state9 | |
| NextState1 | state0 | |
| NextState2 | state1 | (Op = 'lw') + (Op = 'sw |
| NextState3 | state2 | (Op = 'lw') |
| NextState4 | state3 | |
| NextState5 | state2 | (Op = 'sw') |
| NextState6 | state1 | (Op = 'R-type') |
| NextState7 | state6 | |
| NextState8 | state1 | (Op = 'beq') |
| NextState9 | state1 | (Op = 'jmp') |

$$\text{NextState3} = \text{state2} \cdot \text{Op} = \text{'lw'}(23_{\text{hex}}) = \overline{S3} \cdot \overline{S2} \cdot S1 \cdot \overline{S0} \cdot Op5 \cdot \overline{Op4} \cdot \overline{Op3} \cdot \overline{Op2} \cdot Op1 \cdot Op0$$

$$\text{NextState6} = \text{state1} \cdot \text{Op} = \text{'R'}(0_{\text{hex}}) = \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0 \cdot \overline{Op5} \cdot \overline{Op4} \cdot \overline{Op3} \cdot \overline{Op2} \cdot \overline{Op1} \cdot \overline{Op0}$$

$$\text{NextState7} = \text{state6} = \overline{S3} \cdot S2 \cdot S1 \cdot \overline{S0}$$

$$NS1 = NextState2 + NextState3 + NextState6 + NextState7$$

# Agenda


Topics:

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

Patterson: 4.7, 4.8

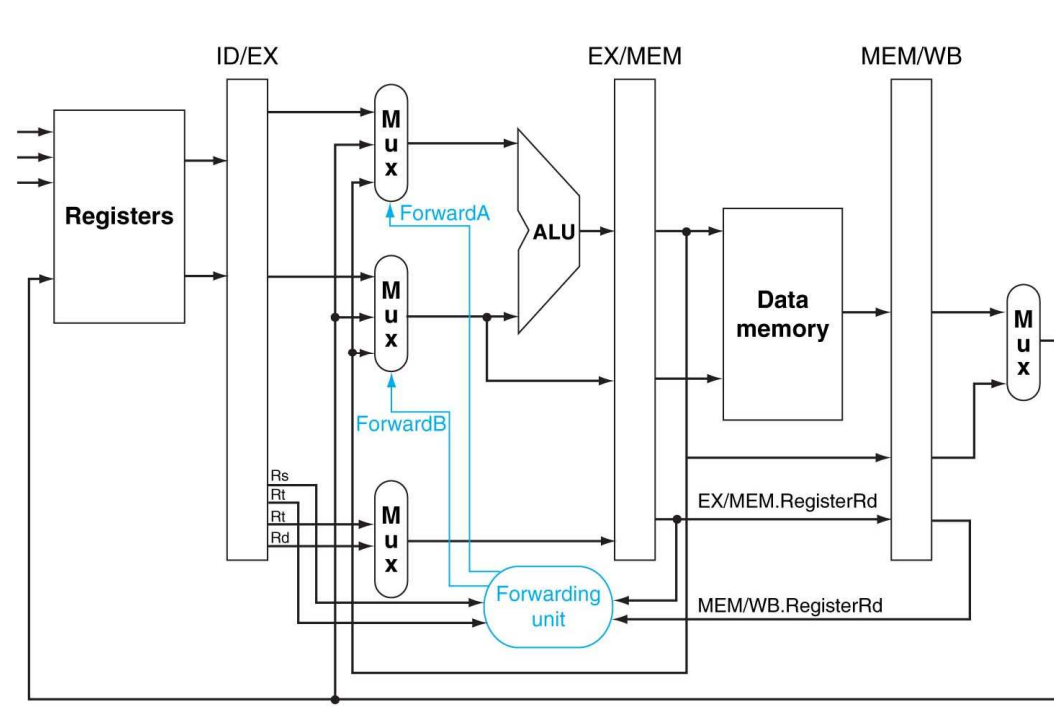
# Forwarding from EX/MEM Pipeline Register

## *Conditions:*

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

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

*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

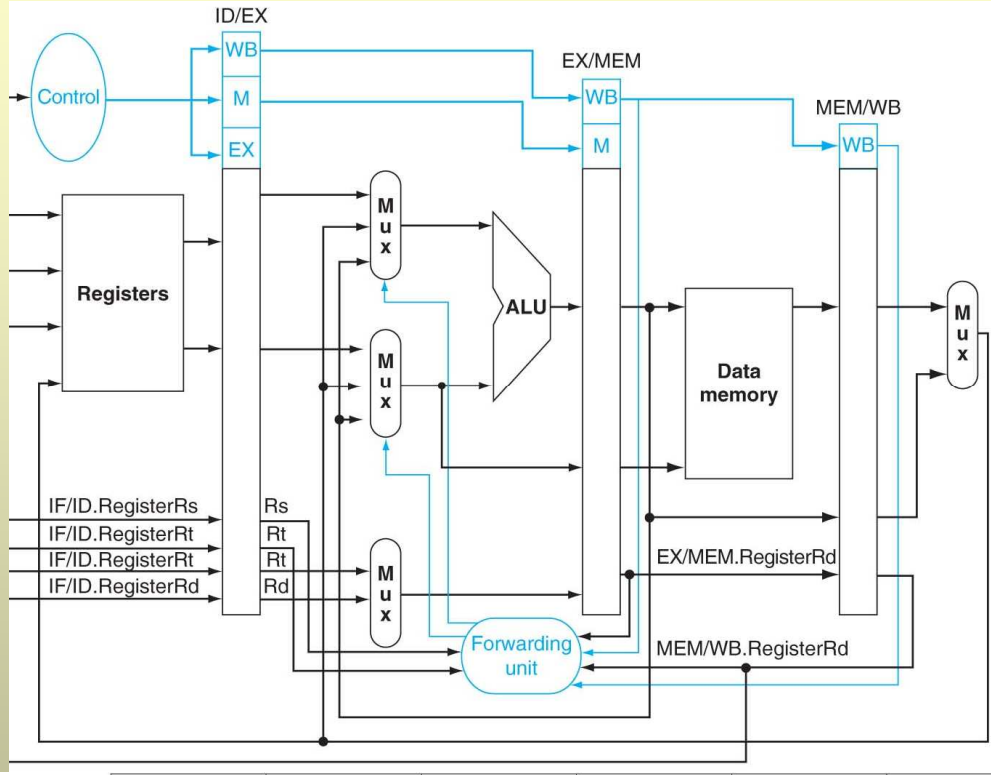
# Forwarding from MEM/WB Pipeline Register


## *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

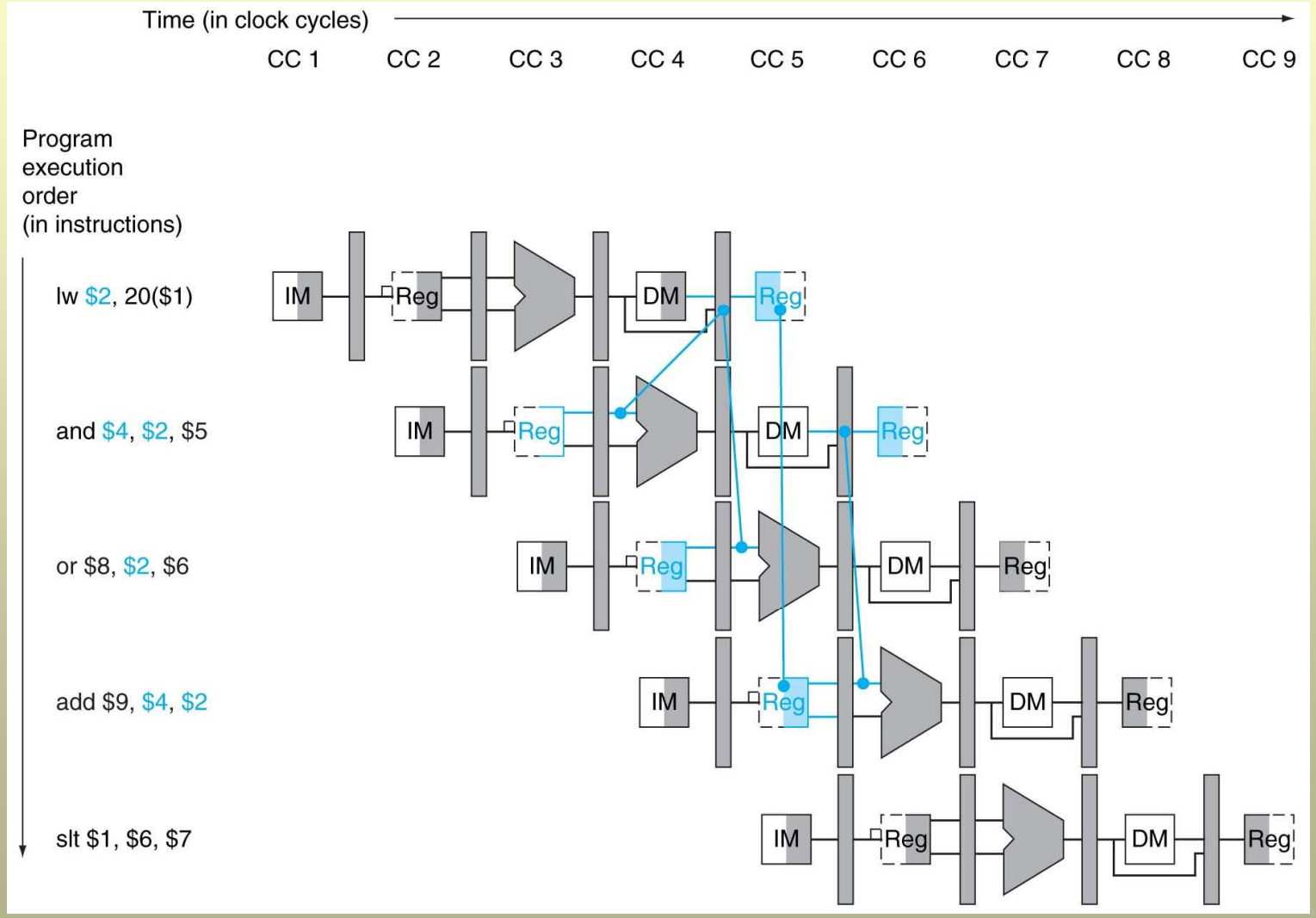
| Field | 35 or 43 | rs | rt | address |
|---|---|---|---|---|
| Bit positions | 31:26 | 25:21 | 20:16 | 15:0 |

b. Load or store instruction

# Mux Truth Table For Forwarding


| Mux control | Source | Explanation |
|---|---|---|
| ForwardA = 00 | ID/EX | The first ALU operand comes from the register file. |
| ForwardA = 10 | EX/MEM | The first ALU operand is forwarded from the prior ALU result. |
| ForwardA = 01 | MEM/WB | The first ALU operand is forwarded from data memory or an earlier ALU result. |
| ForwardB = 00 | ID/EX | The second ALU operand comes from the register file. |
| ForwardB = 10 | EX/MEM | The second ALU operand is forwarded from the prior ALU result. |
| ForwardB = 01 | MEM/WB | The second ALU operand is forwarded from data memory or an earlier ALU result. |

# Pipeline Diagram



# Stalls


*Consider the code sequence*

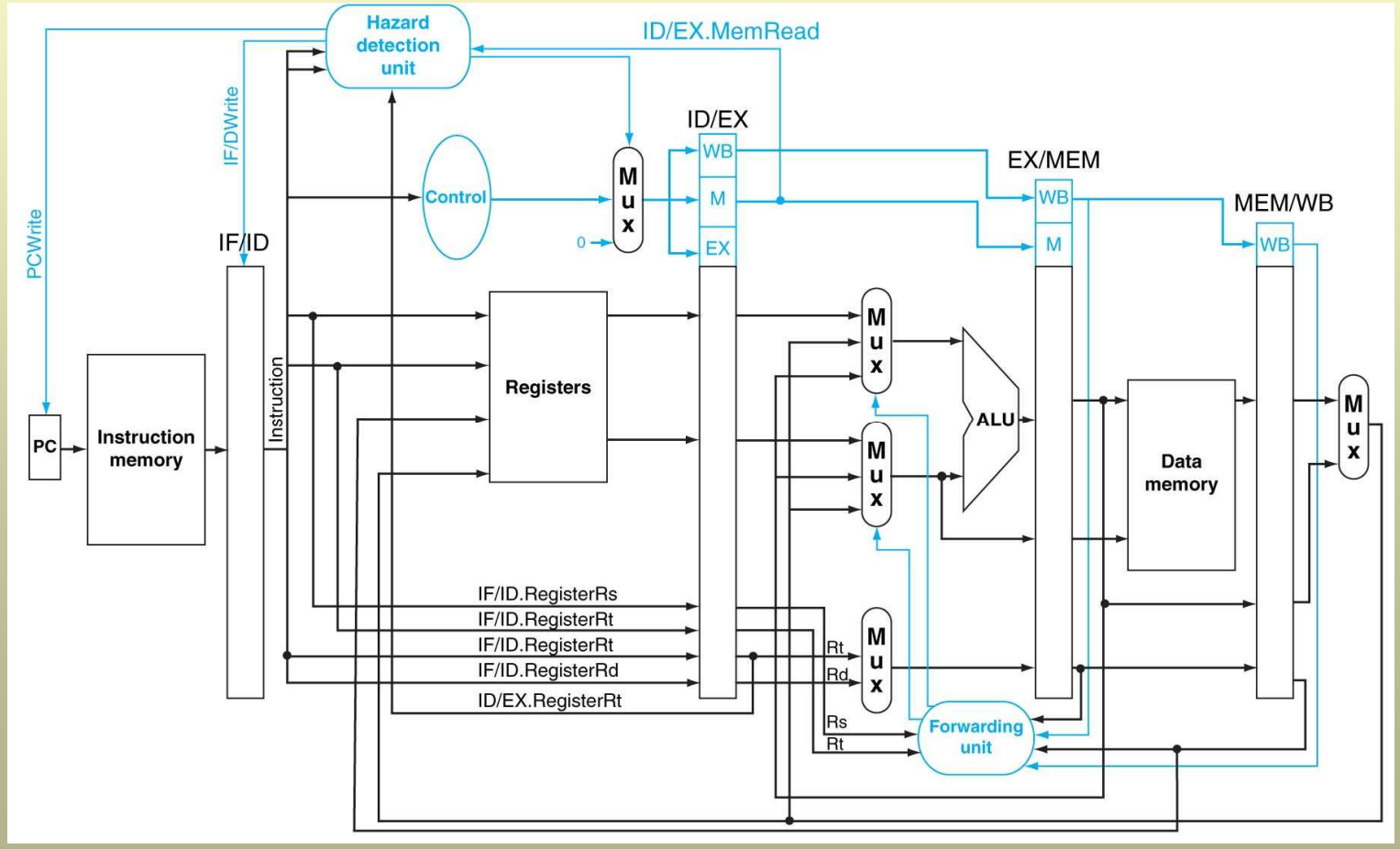
```
lw      $2, 20($1)
and     $4, $2, $5
or      $8, $2, $6
add     $9, $4, $2
slt     $1, $6, $7
```

*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