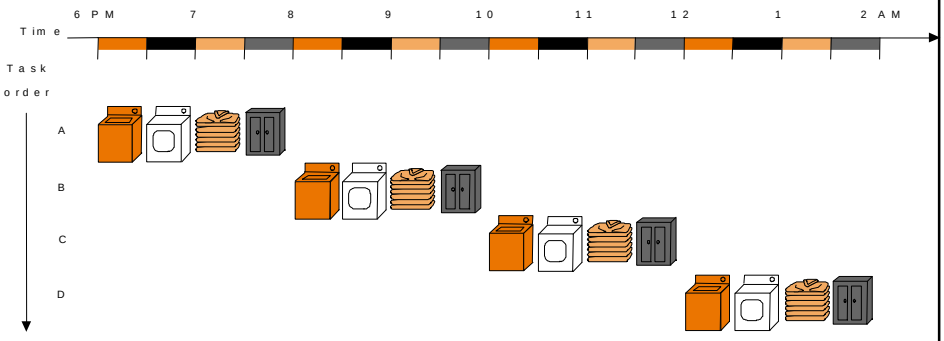
Intro to Pipelining

Chapter 6 P&H

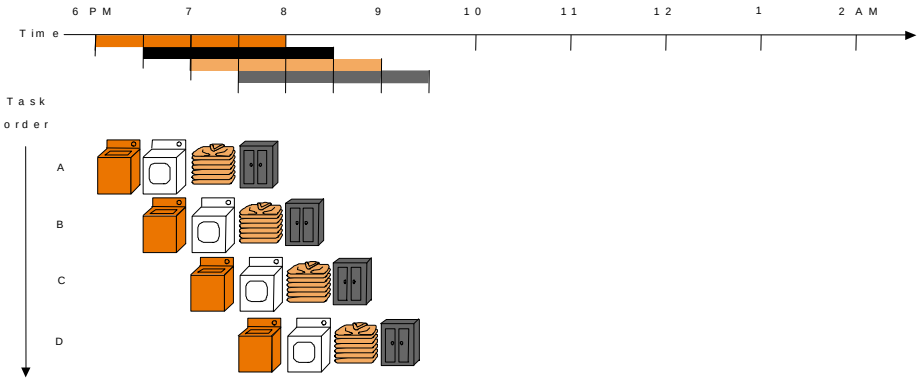
Pipelining

- Multiple instructions are overlapped in execution
- Key to making modern processors fast
- Example – laundry
  - wash load of clothes in machine
  - Dry load of washing in dryer
  - Fold dry load of washing
  - Flatmate puts clothes away

Example – laundry



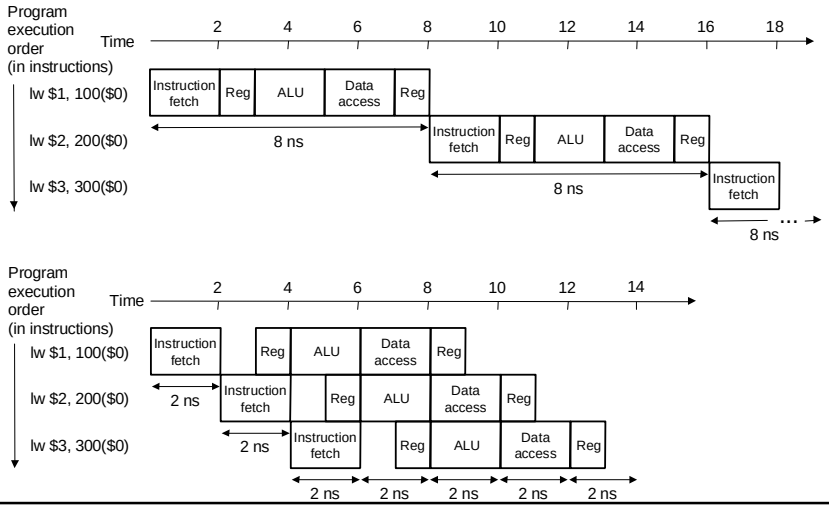
Example – pipelined laundry



Pipelined CPU

- Can apply same principles to CPU Design
- MIPS instructions classically take 5 steps:
  - Fetch instructions from memory
  - Decode instruction and read registers
  - Execute the operation or calculate address
  - Access operand in data memory
  - Write result to a register

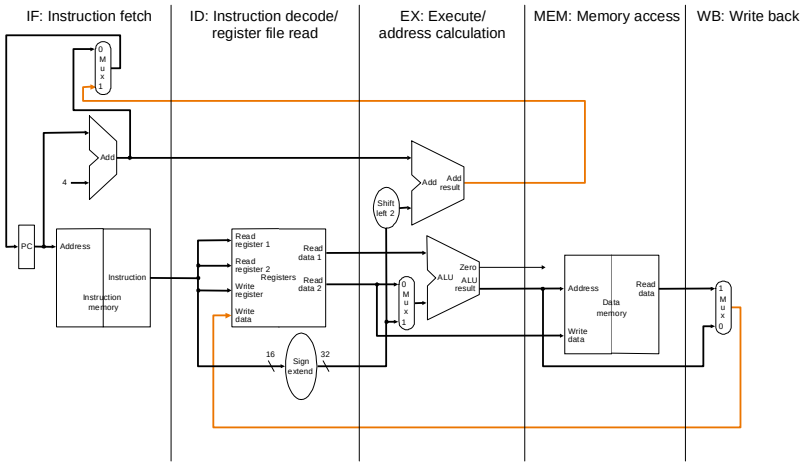
Example MIPS CPU



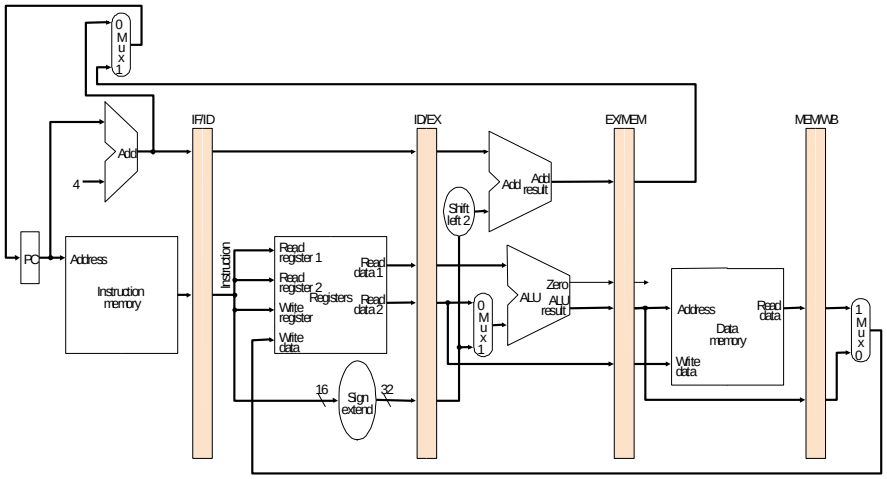
Designing Instruction sets for pipelining

- All MIPS instructions are the same length
- Only a few instruction formats
  - Source register fields in same place
- Memory operands only appear in loads and stores
  - Can calculate address in execute stage
- Operands must be aligned in memory

Single Cycle Datapath



Pipelined version of Datapath



Pipelined version of Datapath

- Pipeline Reg naming conventions
  - Named by two stages separated by that register e.g. ID/EX
- Work trough stages of executing a store instruction
  - *Instruction Fetch*
    - PC + 4 -> IF/ID
    - PC + 4 -> PC
    - Imem[PC] -> IF/ID

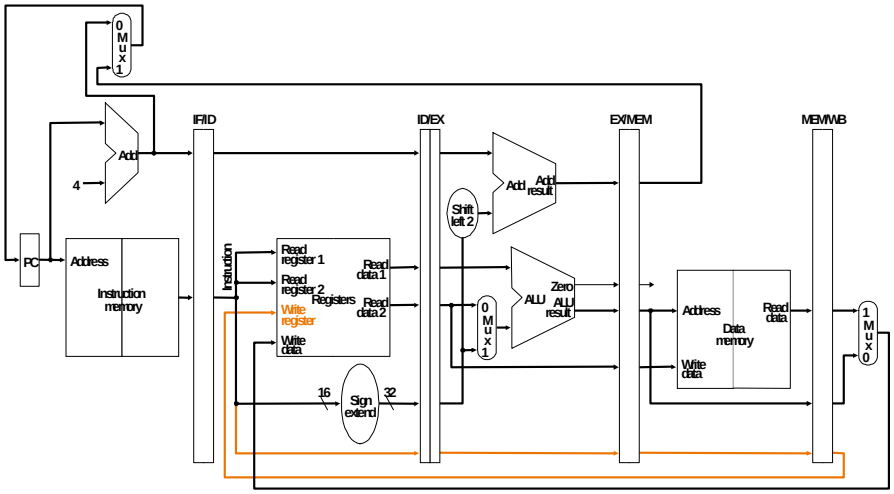
LW instruction

- *Instruction Decode*
  - IF/ID supplies 16 bit Immediate which is signed extended -> ID/EX
  - Two regs are read -> ID/EX
  - PC + 4 from IF/ID -> ID/EX
- *Execute or address calculation*
  - Contents reg and signed extended immediate added -> Ex/Mem
- *Memory Access*
  - Address in EX/Mem used to get data value from data mem -> Mem/WB
- *Write back*
  - Mem/Wb data val -> reg file

LW instruction

- Any information that might be needed in a later stage must be passed via the pipeline registers
- Each resource can only be used in a single pipeline stage (otherwise have structural hazard)
- Bug in previous data-path
  - Which instruction provides address for destination register in lw
    - Dest reg numb needs to be passed through pipeline registers

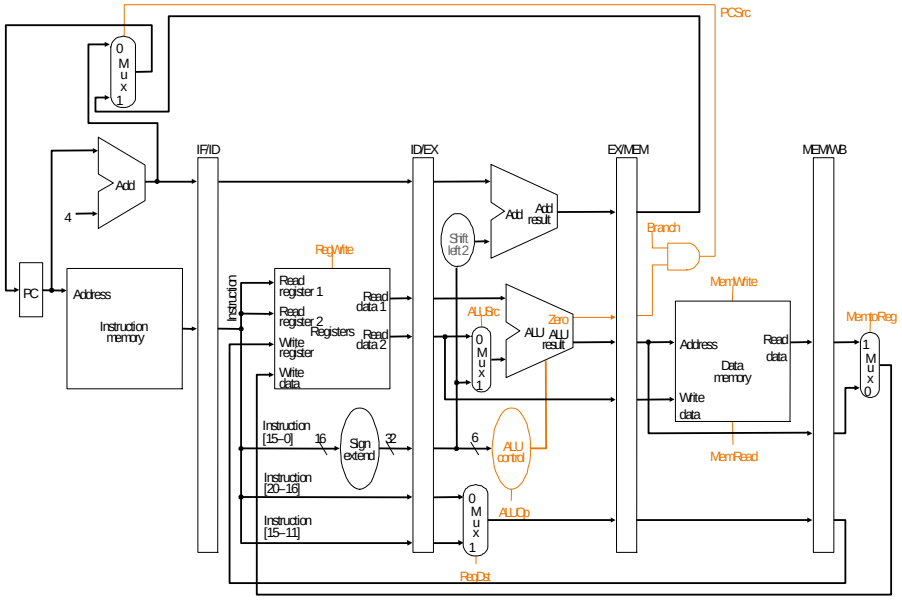
Updated Datapath



Pipelined Control

- Start off by ignoring hazards

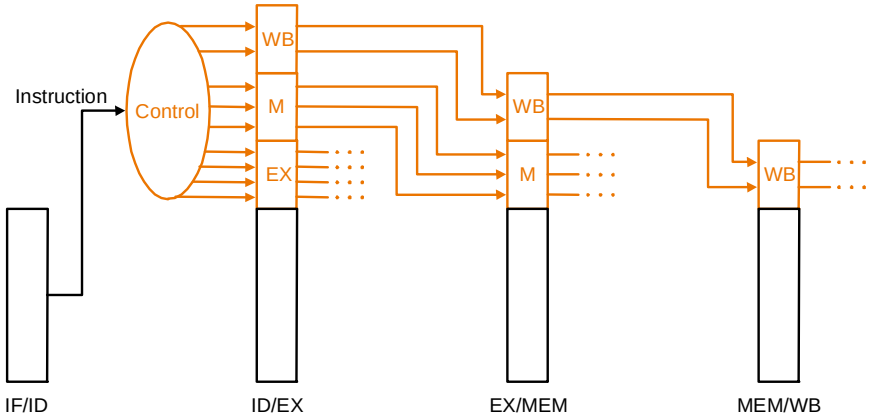
Datapath control signals



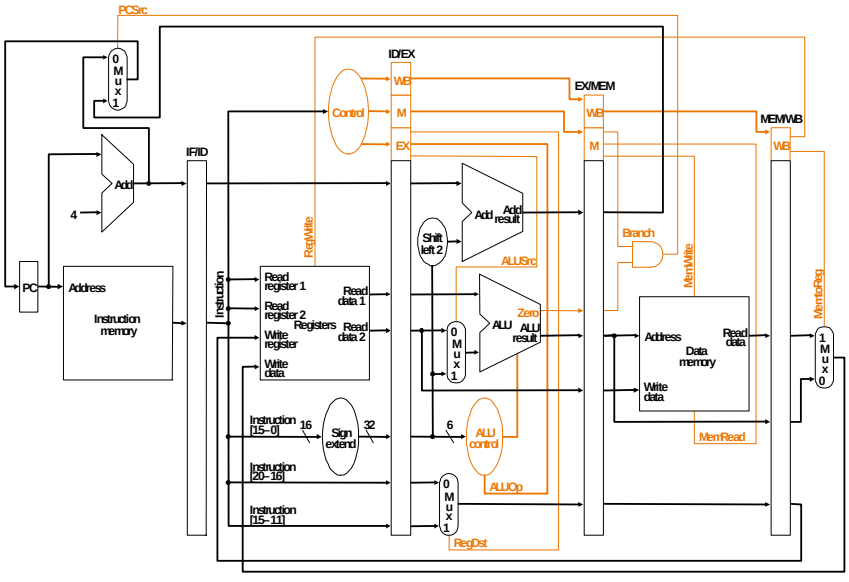
Values on the Control Lines

Instruct	Execution stage				Mem access			Write back	
	RegDst	ALU Op1	ALU OP0	ALU Src	Branch	Mem read	Mem Write	Reg Write	Mem to Reg
Rformat	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

Control Lines for final three stages



Updated Datapath



Pipeline Hazards

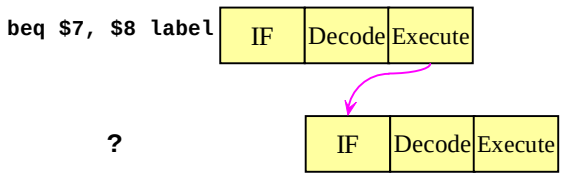
- Situations in pipelining where next instruction cannot execute in following clock cycle

Structural Hazards

- Hardware cannot support combination of instructions we want to execute in same cycle
- E.g. If used a washer/dyer combination or flatmate doing something else
- E.g. MIPS Only a single memory available

Control Hazards

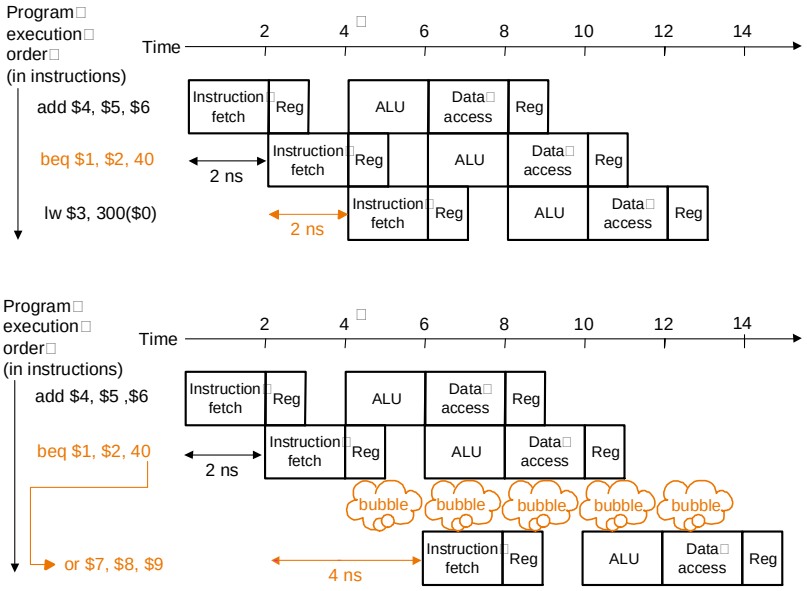
- Need to make a decision based on the results of one instruction while it is still executing
- Example – The branch instruction



Control Hazards

- Can either:
  - Stall
    - Bubble(s) inserted into pipeline
  - Predict
    - Predict outcome of branch
    - Stall if prediction wrong
  - Delay branch
    - Execute instruction after branch regardless of whether branch is taken or not

Branch Prediction

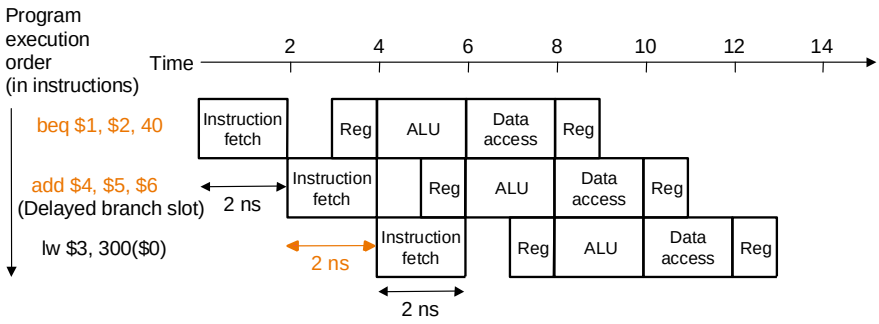


Branch Prediction

- Assume all branches will fail
- Assume backwards branches always succeed while forward branches always fail
- Use a dynamic branch predictor
  - Uses a table to record the outcomes of previous time branch instructions were executed
    - Gets about 90% accuracy
- Longer pipelines amplify problem

Delayed Branches

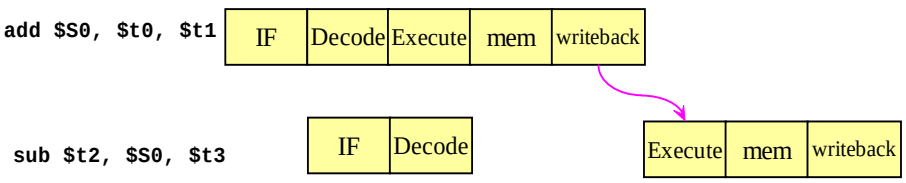
- Used on MIPS R3000 processors



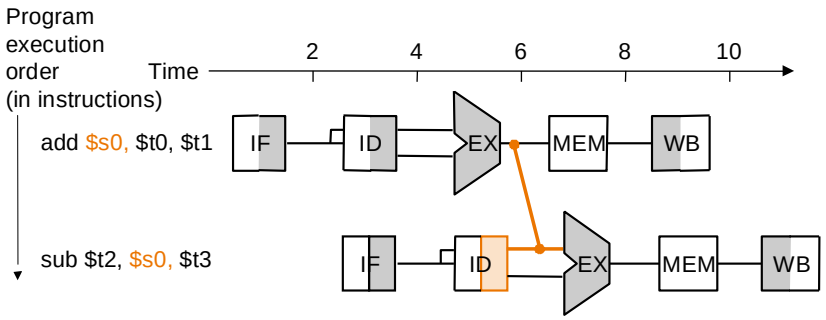
Data Hazards

- Instruction depends on outcome of a previous instruction still in the pipeline
- E.g.

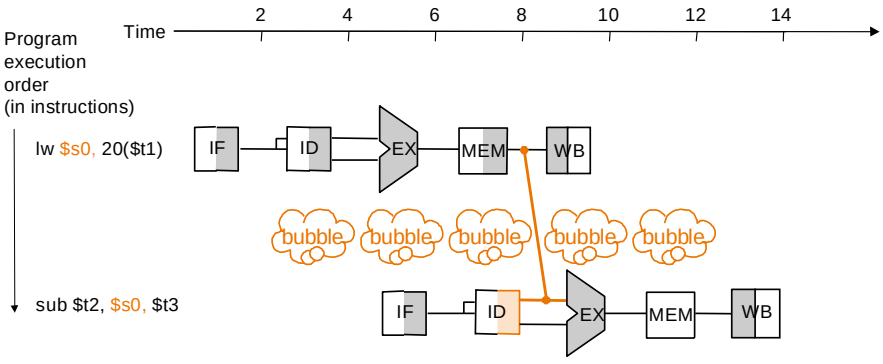
```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```



Forwarding



Forwarding

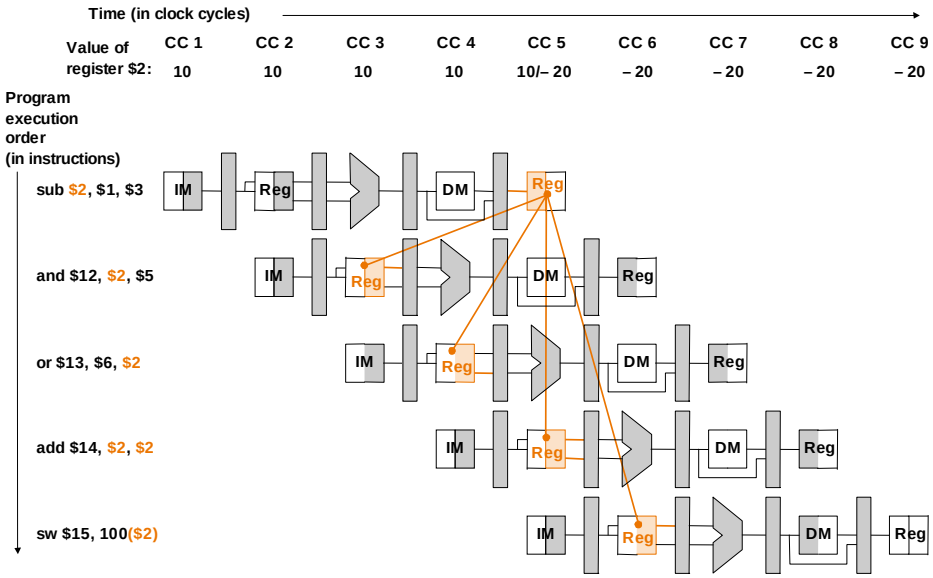


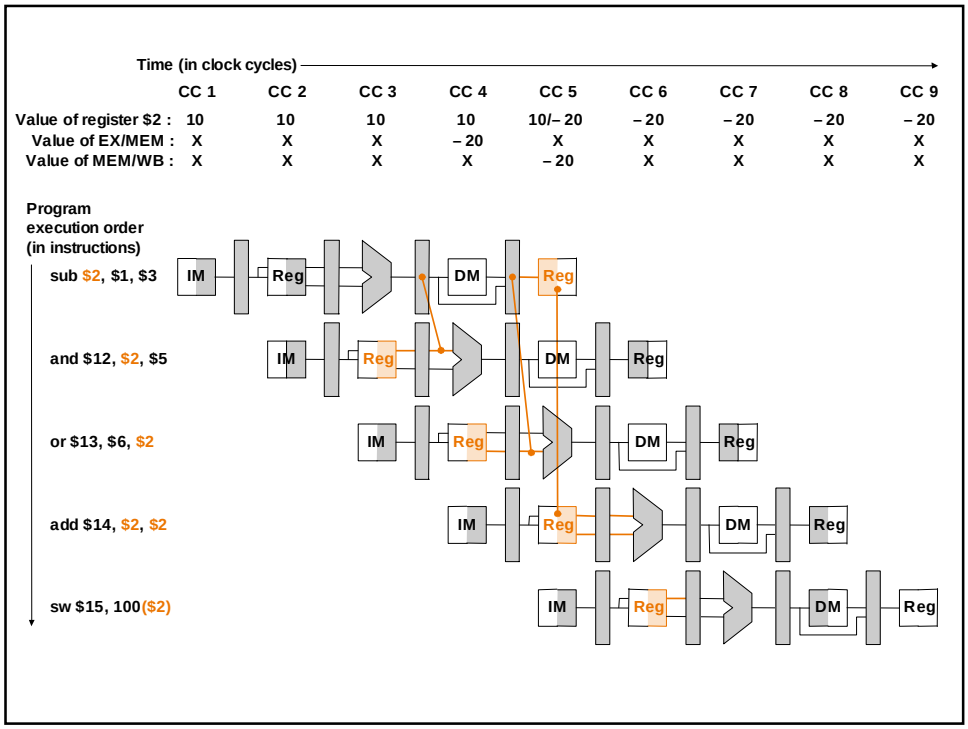
Data Hazards and Forwarding

- Consider execution on following sequence of instructions:

```
Sub   $2,  $1, $3
And   $12, $2, $5
Or    $13, $6, $2
Add   $14, $2, $2
Sw    $15, 100($2)
```

Pipelined Dependencies



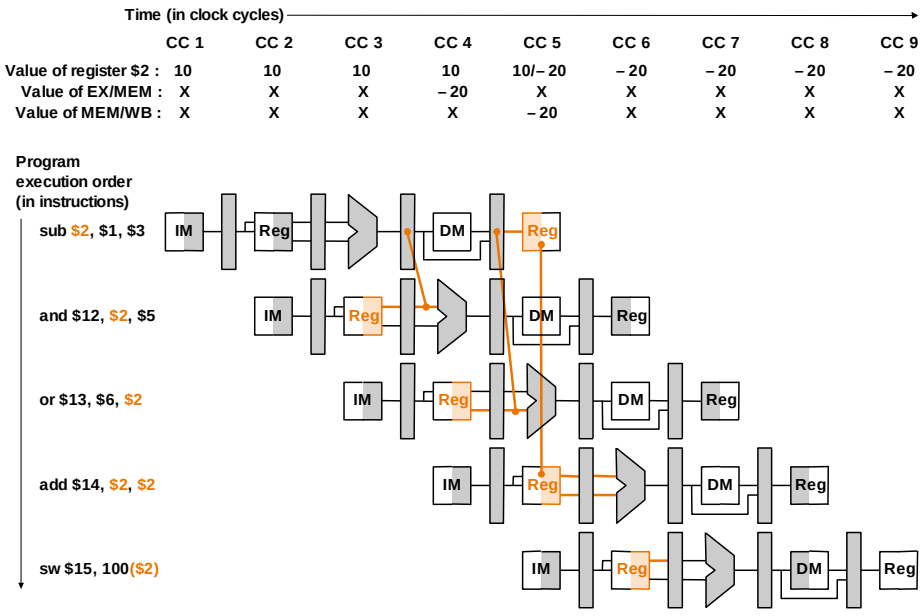


Hazard notation

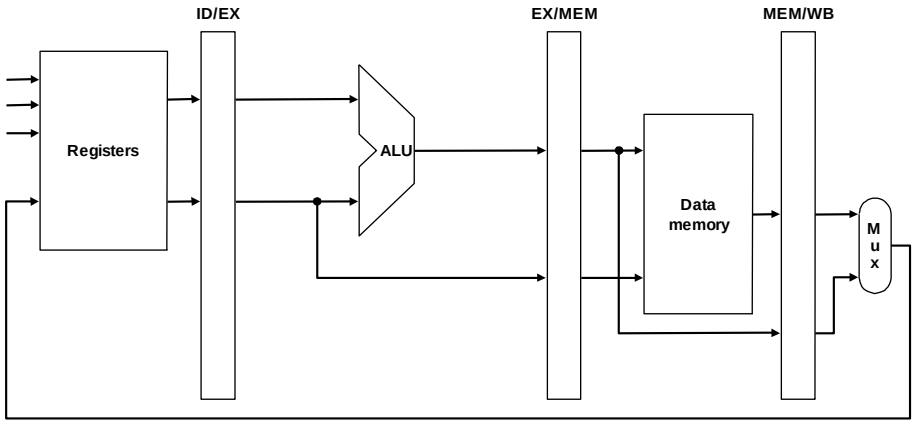
- IN our previous example the two pairs of hazard conditions are
  - **1a.** `EX/Mem.RegisterRd = ID/EX.RegisterRs`
  - **1b.** `EX/Mem.RegisterRd = ID/EX.RegisterRt`
  - **2a.** `Mem/WB.RegisterRd = ID/EX.RegisterRs`
  - **2b.** `Mem/WB.RegisterRd = ID/EX.RegisterRs`
- Hazard on $2 between the sub and and instructions is:
  - `EX/Mem.RegisterRd = ID/EX.RegisterRs = $2`

Hazards

- Above notation will do forwarding unnecessarily as some instruction do not write registers
  - Solution: can check WB control to check if register writes back
- What about $0
  - Do not forward as should always be zero

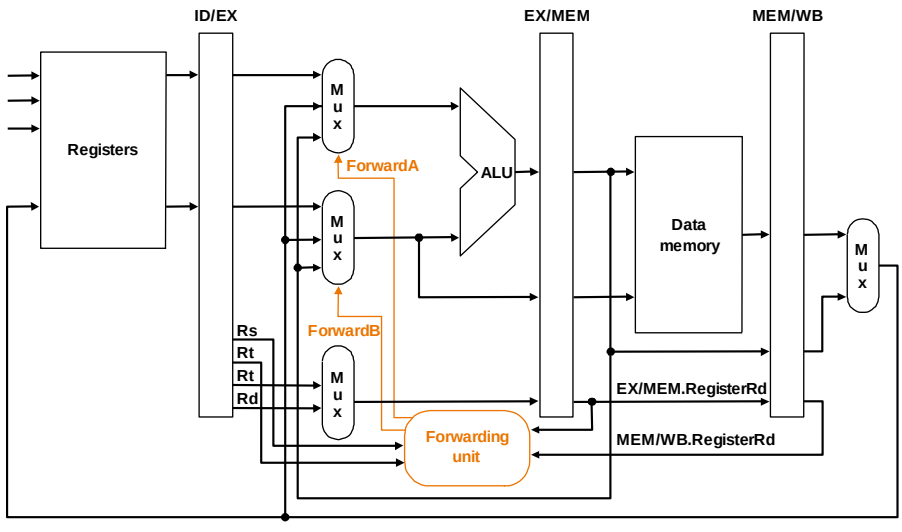


No Forwarding



a. No forwarding

Forwarding



b. With forwarding

EX Hazard

- If (EX/Mem.RegWrite) and
  (EX/Mem.RegRd != 0) and
  (EM/Mem.RegRd = ID/EX.RegRs) then
  ForwardA = 10
- If (EX/Mem.RegWrite) and
  (EX/Mem.RegRd != 0) and
  (EM/Mem.RegRd = ID/EX.RegRt) then
  ForwardB = 10

Mem Hazard

- If (Mem/WB.RegWrite) and
  (Mem/WB.RegRd != 0) and
  (Mem/WB.RegRd = ID/EX.RegRs) then
  ForwardA = 01

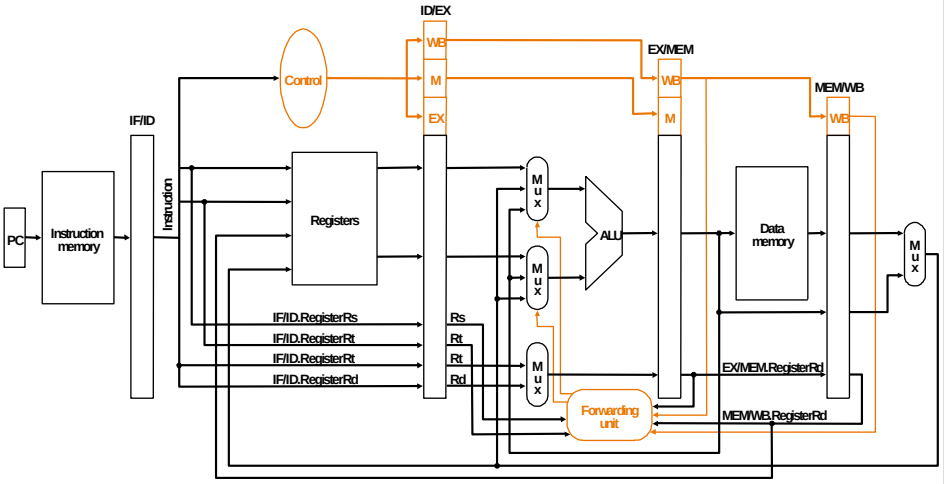
  If (Mem/WB.RegWrite) and
  (Mem/WB.RegRd != 0) and
  (Mem/WB.RegRd = ID/EX.RegRt) then
  ForwardB = 01

What about:

```
Add $1, $1, $2
Add $1, $1, $3
Add $1, $1, $4
```

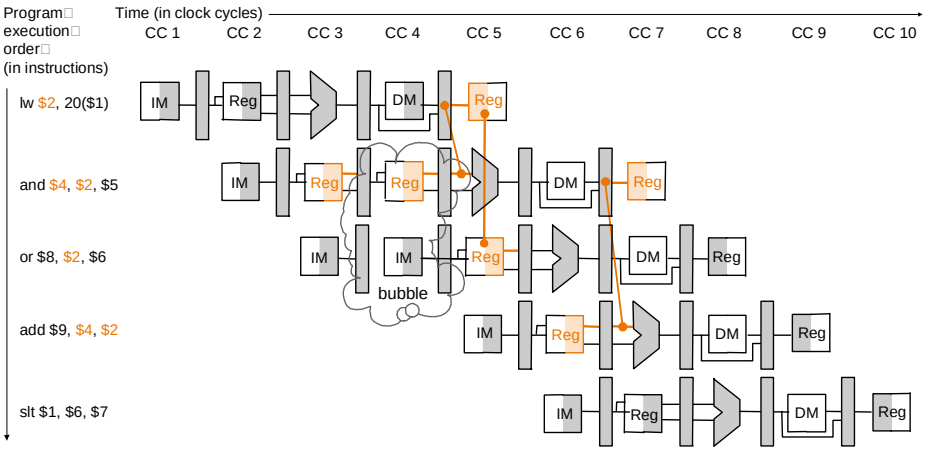
- If (Mem/WB.RegWrite) and
  (Mem/WB.RegRd != 0) and
  (EX/Mem.RegRd != ID/EX.RegRs) and
  (Mem/WB.RegRd = ID/EX.RegRs) then
    ForwardA = 01
  If (Mem/WB.RegWrite) and
  (Mem/WB.RegRd != 0) and
  (EX/Mem.RegRd != ID/EX.RegRt) and
  (Mem/WB.RegRd = ID/EX.RegRt) then
    ForwardB = 01

Updated datapath to resolve data hazards



Data Hazards and Stalls

- Forwarding will not work where an instruction tries to read a register following a load instruction that writes the same register



Stalls

- Hazard detection unit required at the ID stage
  - `If (ID/EX.MemRead) and`
    `(ID/EX.RegRt = IF/ID.RegRs) or`
    `(ID/EX.RegRt = IF/ID.RegRt then`
    `stall the pipeline`