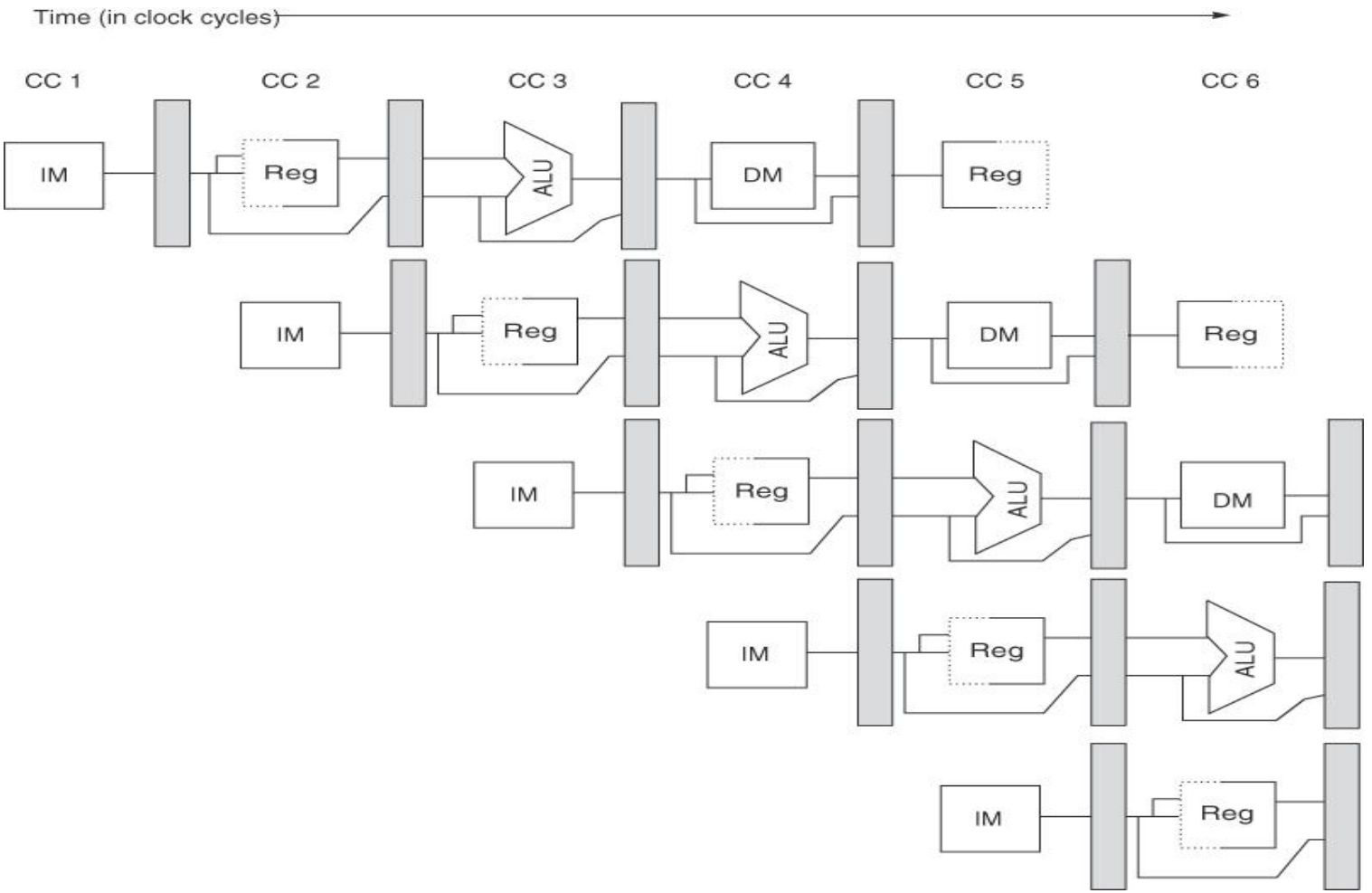
Lecture 4: Advanced Pipelines

- Data hazards, control hazards, multi-cycle in-order pipelines (Appendix C.4-C.8)

A 5-Stage Pipeline

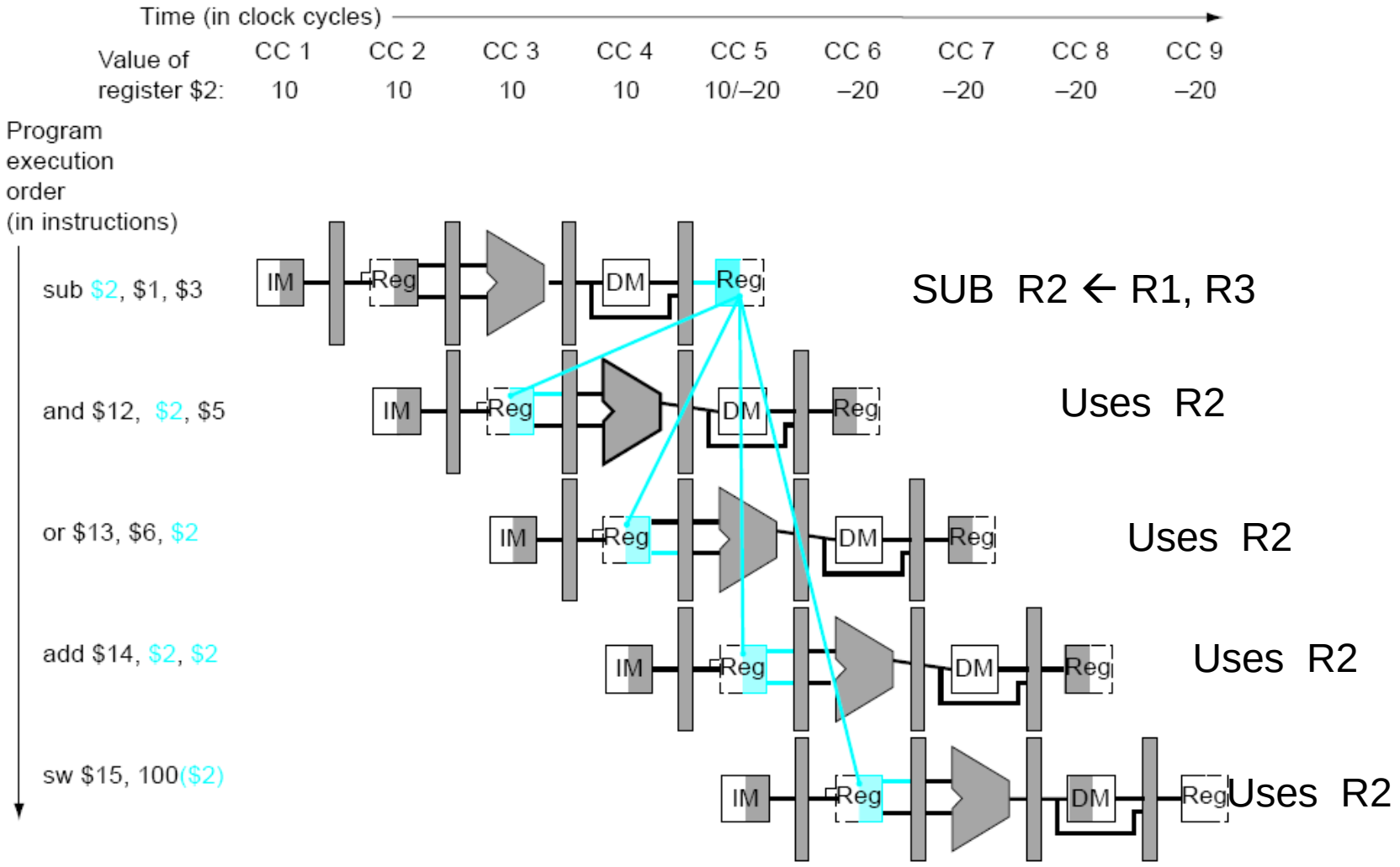


Source: H&P textbook

Hazards

- Structural hazards: different instructions in different stages (or the same stage) conflicting for the same resource
- Data hazards: an instruction cannot continue because it needs a value that has not yet been generated by an earlier instruction
- Control hazard: fetch cannot continue because it does not know the outcome of an earlier branch – special case of a data hazard – separate category because they are treated in different ways

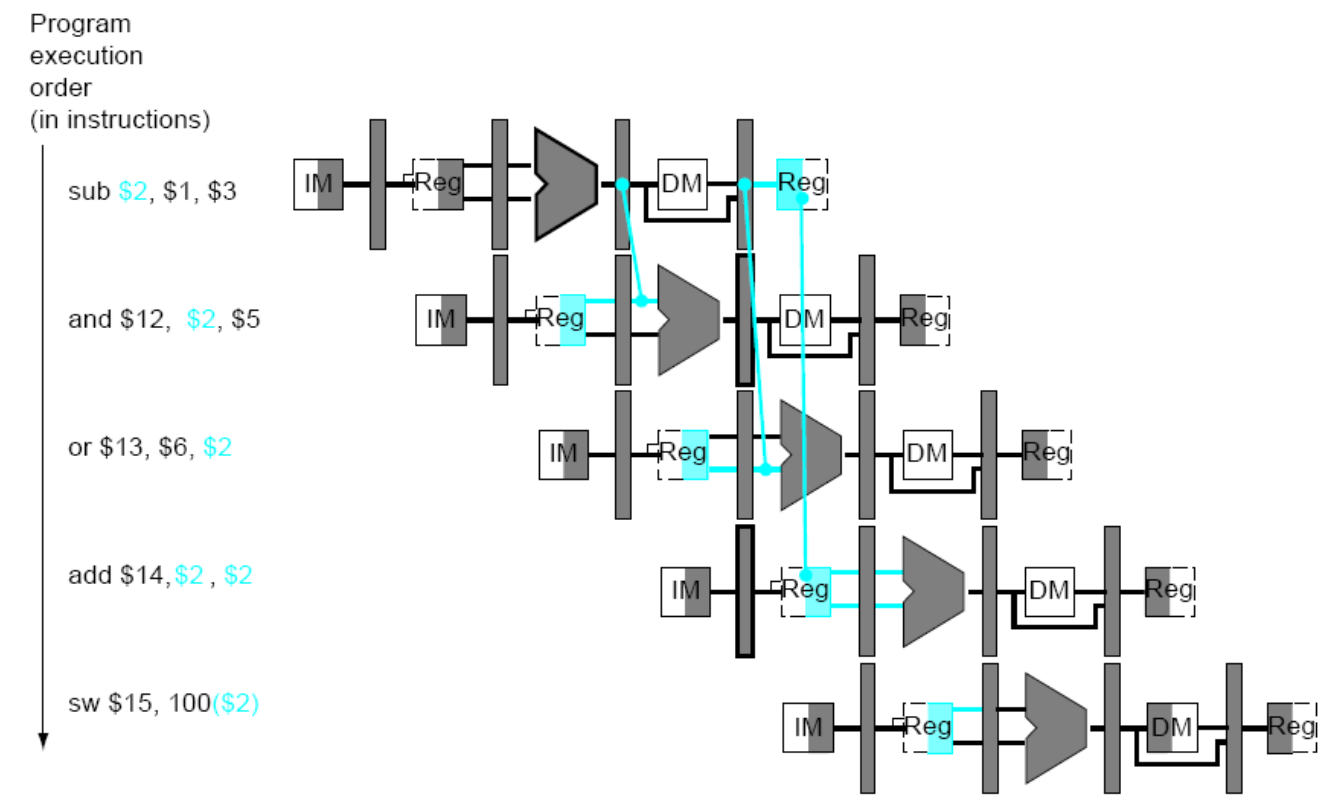
Data Hazards



Bypassing

Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X

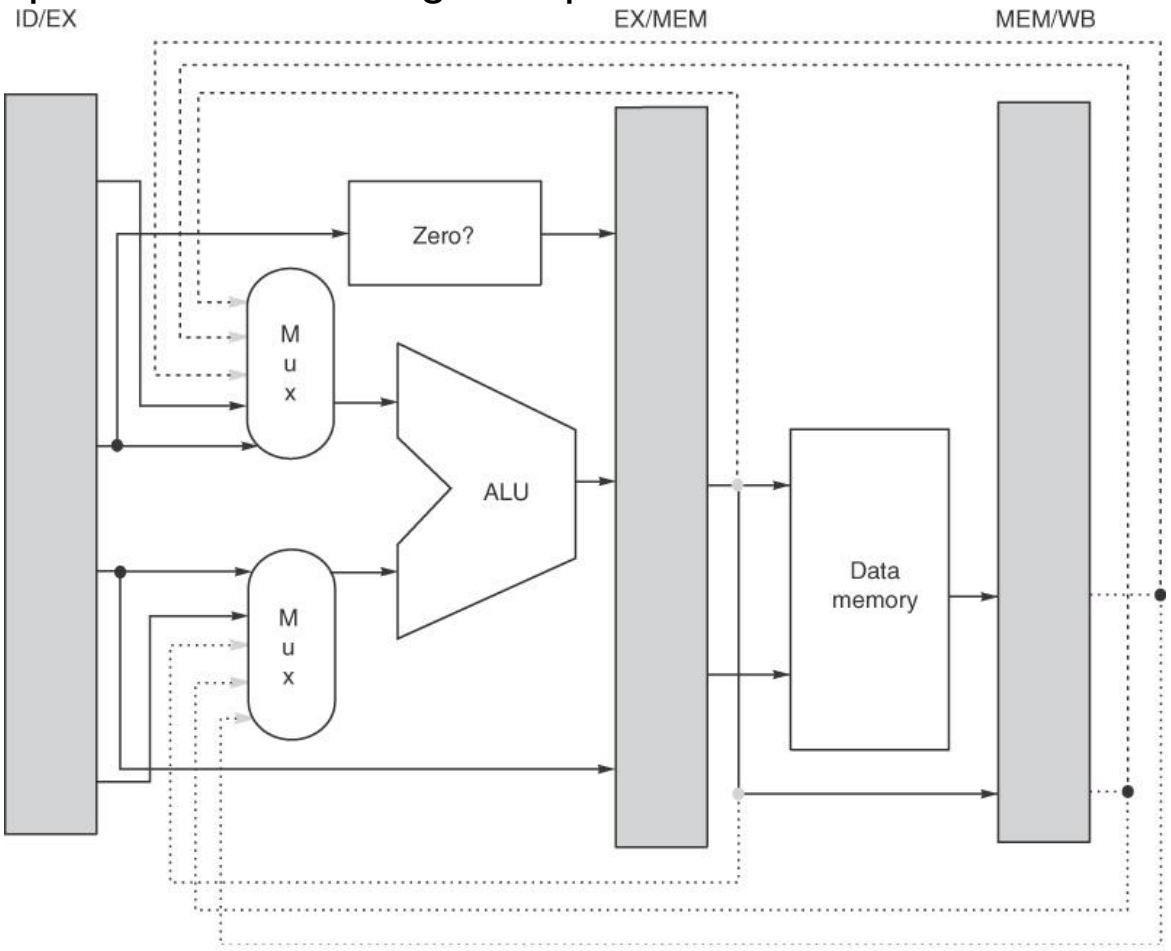


- Some data hazard stalls can be eliminated: bypassing

Bypassing

Pipeline Implementation

- Signals for the muxes have to be generated – some of this can happen during ID
- Need look-up tables to identify situations that merit bypassing/stalling – the number of inputs to the muxes goes up



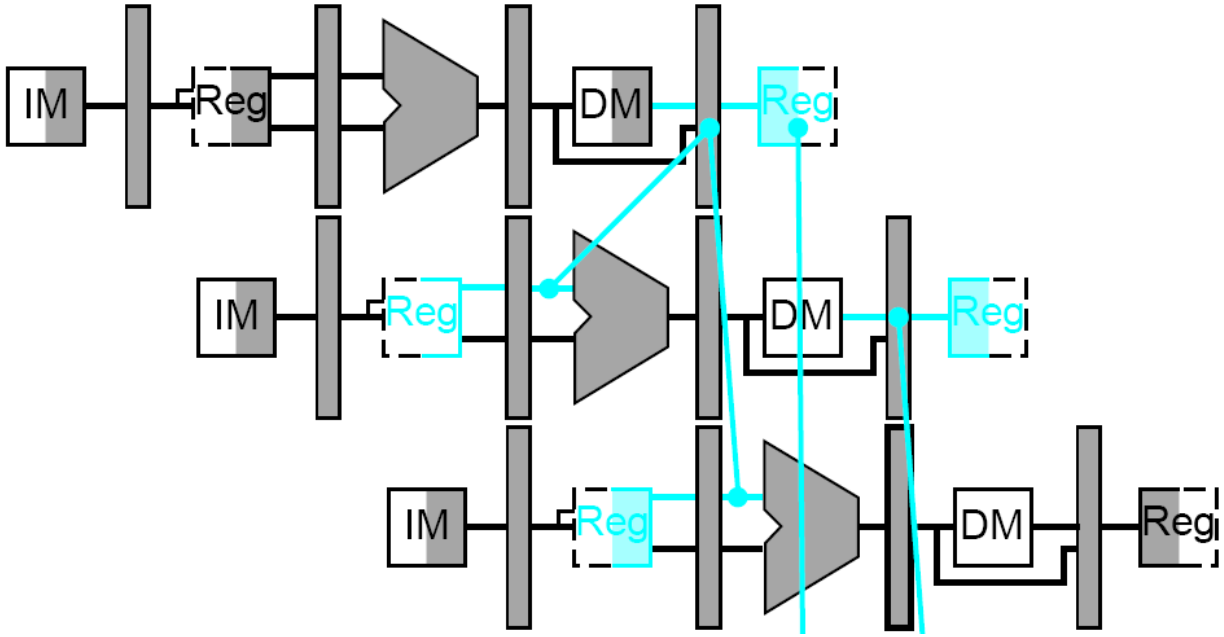
Detecting Control Signals

Situation	Example code	Action
No dependence	LD **R1**, 45(R2) DADD R5, R6, R7 DSUB R8, R6, R7 OR R9, R6, R7	No hazards
Dependence requiring stall	LD **R1**, 45(R2) DADD R5, **R1**, R7 DSUB R8, R6, R7 OR R9, R6, R7	Detect use of R1 during ID of DADD and stall
Dependence overcome by forwarding	LD **R1**, 45(R2) DADD R5, R6, R7 DSUB R8, **R1**, R7 OR R9, R6, R7	Detect use of R1 during ID of DSUB and set mux control signal that accepts result from bypass path
Dependence with accesses in order	LD **R1**, 45(R2) DADD R5, R6, R7 DSUB R8, R6, R7 OR R9, **R1**, R7	No action required

Example

add R1, R2, R3

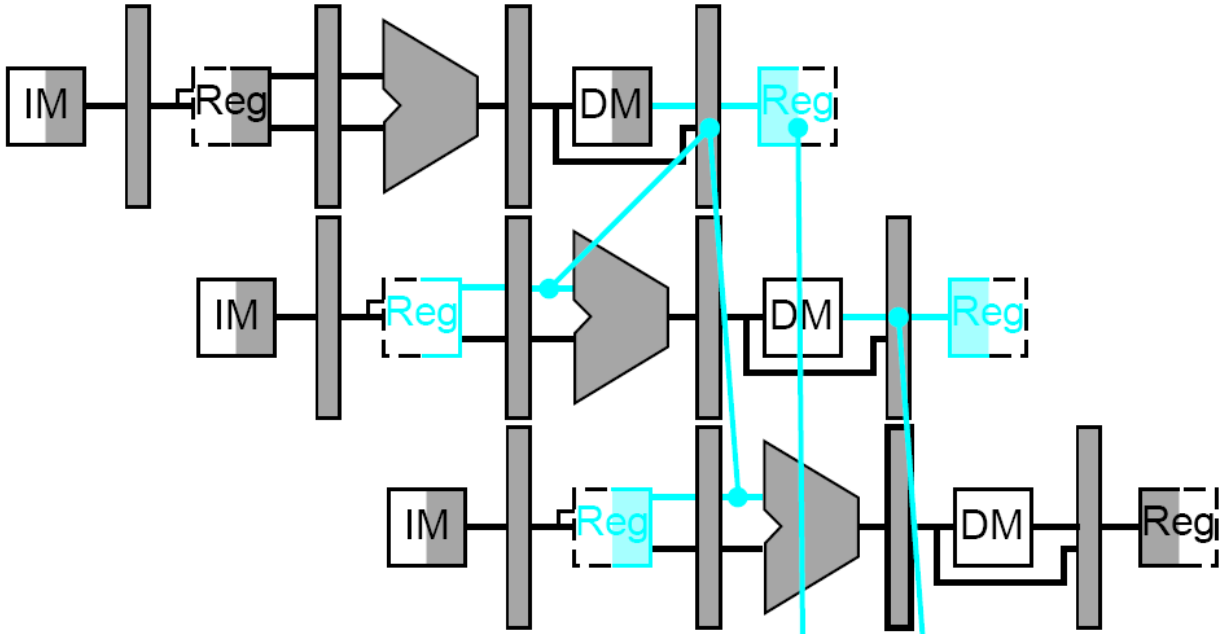
lw R4, 8(R1)



Example

lw R1, 8(R2)

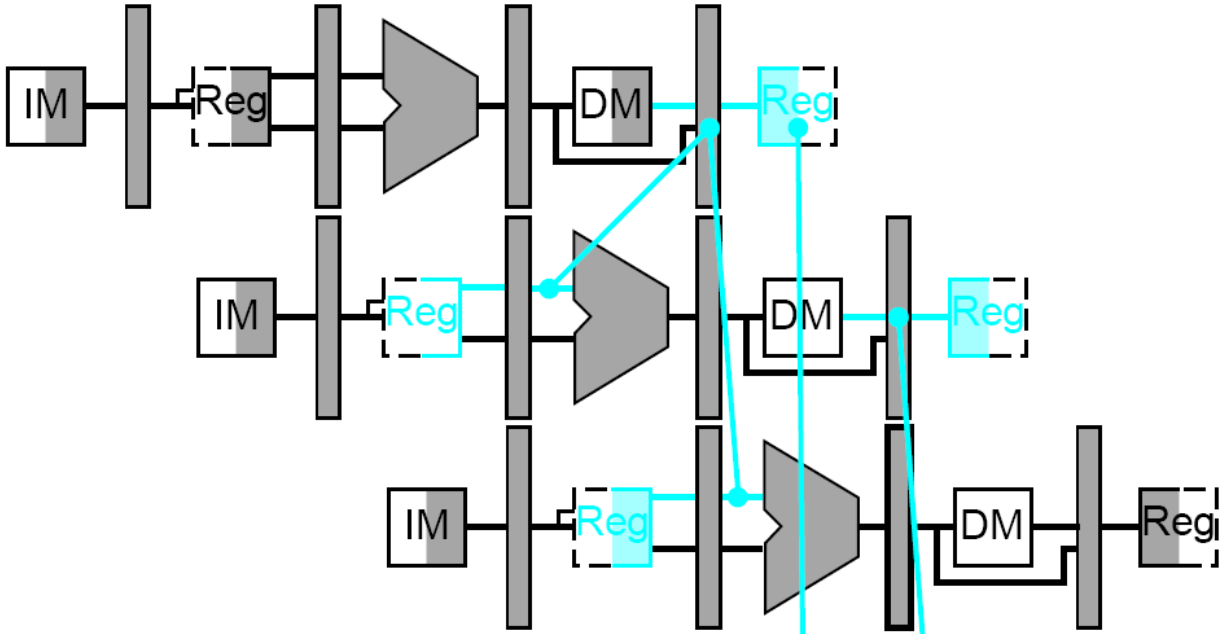
lw R4, 8(R1)



Example

lw R1, 8(R2)

sw R1, 8(R3)



Summary

- For the 5-stage pipeline, bypassing can eliminate delays between the following example pairs of instructions:

```
add/sub          R1, R2, R3
add/sub/lw/sw    R4, R1, R5

lw       R1, 8(R2)
sw       R1, 4(R3)
```

- The following pairs of instructions will have intermediate stalls:

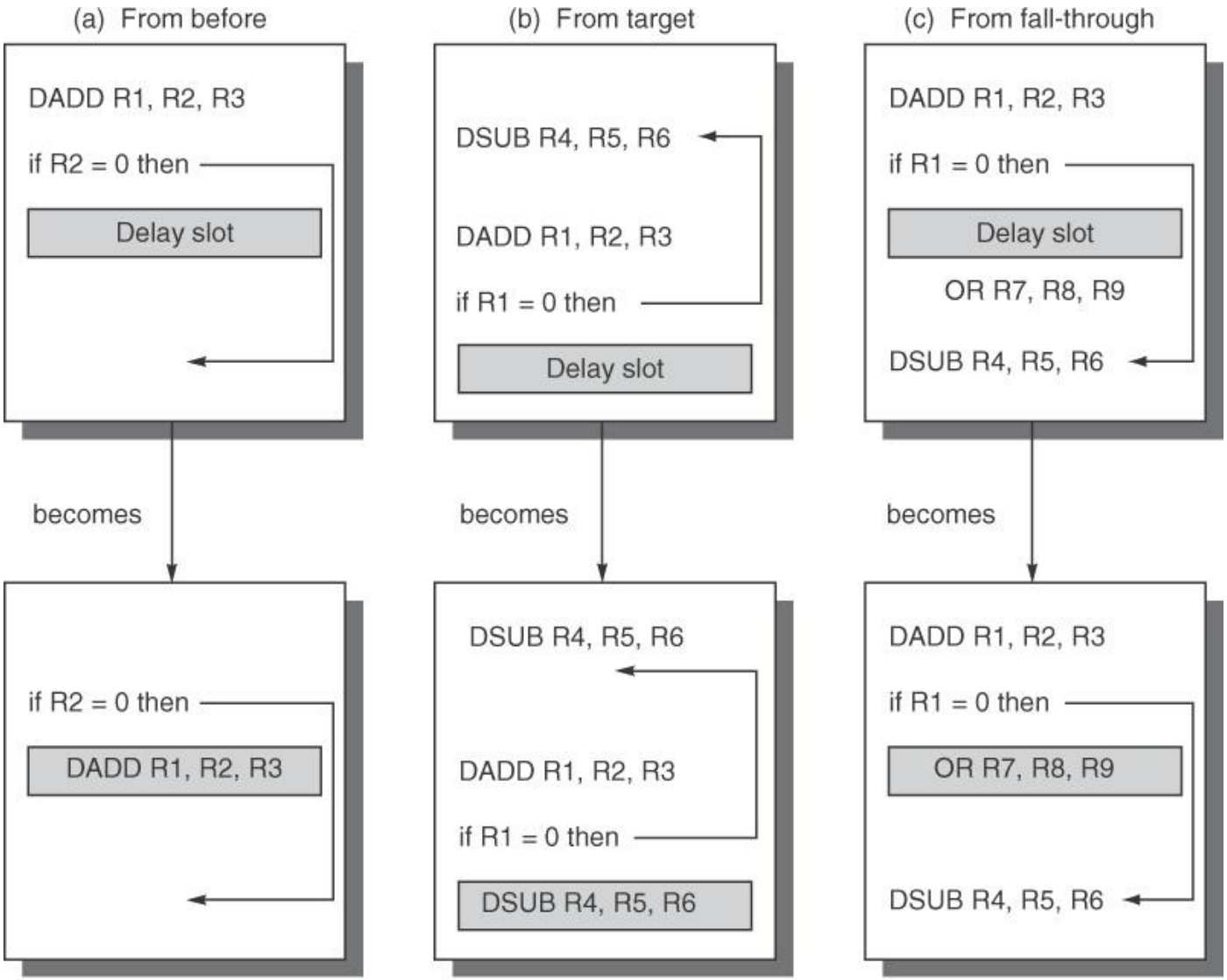
```
lw                R1, 8(R2)
add/sub/lw        R3, R1, R4      or   sw   R3, 8(R1)

fmul      F1, F2, F3
fadd      F5, F1, F4
```

Control Hazards

- Simple techniques to handle control hazard stalls:
  - for every branch, introduce a stall cycle (note: every 6th instruction is a branch!)
  - assume the branch is not taken and start fetching the next instruction – if the branch is taken, need hardware to cancel the effect of the wrong-path instruction
  - fetch the next instruction (branch delay slot) and execute it anyway – if the instruction turns out to be on the correct path, useful work was done – if the instruction turns out to be on the wrong path, hopefully program state is not lost

Branch Delay Slots



Slowdowns from Stalls

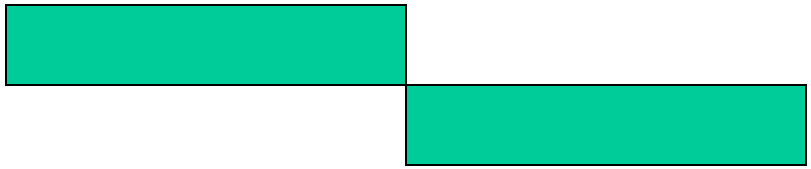
- Perfect pipelining with no hazards → an instruction completes every cycle (total cycles ~ num instructions)
  → speedup = increase in clock speed = num pipeline stages

- With hazards and stalls, some cycles (= stall time) go by during which no instruction completes, and then the stalled instruction completes

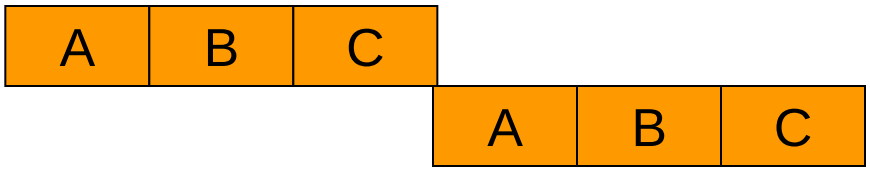
- Total cycles = number of instructions + stall cycles

- Slowdown because of stalls = 1/ (1 + stall cycles per instr)

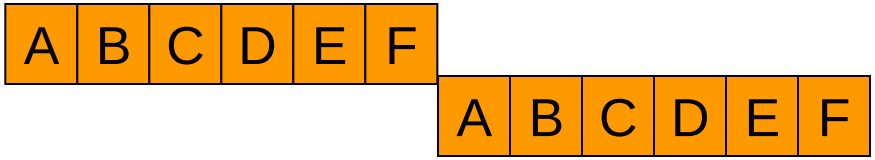
Pipelining Limits



Gap between indep instrs: $T + T_{ovh}$
Gap between dep instrs: $T + T_{ovh}$



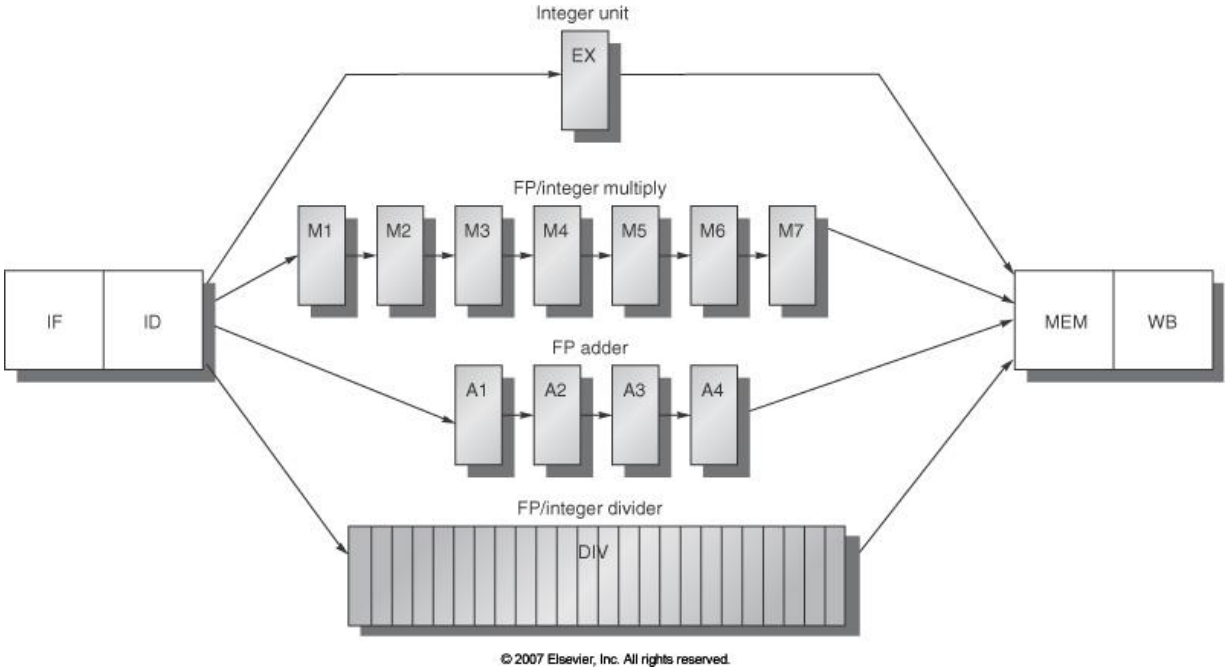
Gap between indep instrs:
$T/3 + T_{ovh}$
Gap between dep instrs:
$T + 3T_{ovh}$



Gap between indep instrs:
$T/6 + T_{ovh}$
Gap between dep instrs:
$T + 6T_{ovh}$

Assume that there is a dependence where the final result of the first instruction is required before starting the second instruction

Multicycle Instructions



Functional unit	Latency	Initiation interval
Integer ALU	1	1
Data memory	2	1
FP add	4	1
FP multiply	7	1
FP divide	25	25

Effects of Multicycle Instructions

- Structural hazards if the unit is not fully pipelined (divider)
- Frequent RAW hazard stalls
- Potentially multiple writes to the register file in a cycle
- WAW hazards because of out-of-order instr completion
- Imprecise exceptions because of o-o-o instr completion

Note: Can also increase the “width” of the processor: handle multiple instructions at the same time: for example, fetch two instructions, read registers for both, execute both, etc.

Precise Exceptions

- On an exception:
  - must save PC of instruction where program must resume
  - all instructions after that PC that might be in the pipeline must be converted to NOPs (other instructions continue to execute and may raise exceptions of their own)
  - temporary program state not in memory (in other words, registers) has to be stored in memory
  - potential problems if a later instruction has already modified memory or registers

- A processor that fulfils all the above conditions is said to provide precise exceptions (useful for debugging and of course, correctness)

Dealing with these Effects

- Multiple writes to the register file: increase the number of ports, stall one of the writers during ID, stall one of the writers during WB (the stall will propagate)
- WAW hazards: detect the hazard during ID and stall the later instruction
- Imprecise exceptions: buffer the results if they complete early or save more pipeline state so that you can return to exactly the same state that you left at

Title

- Bullet