Notes-02

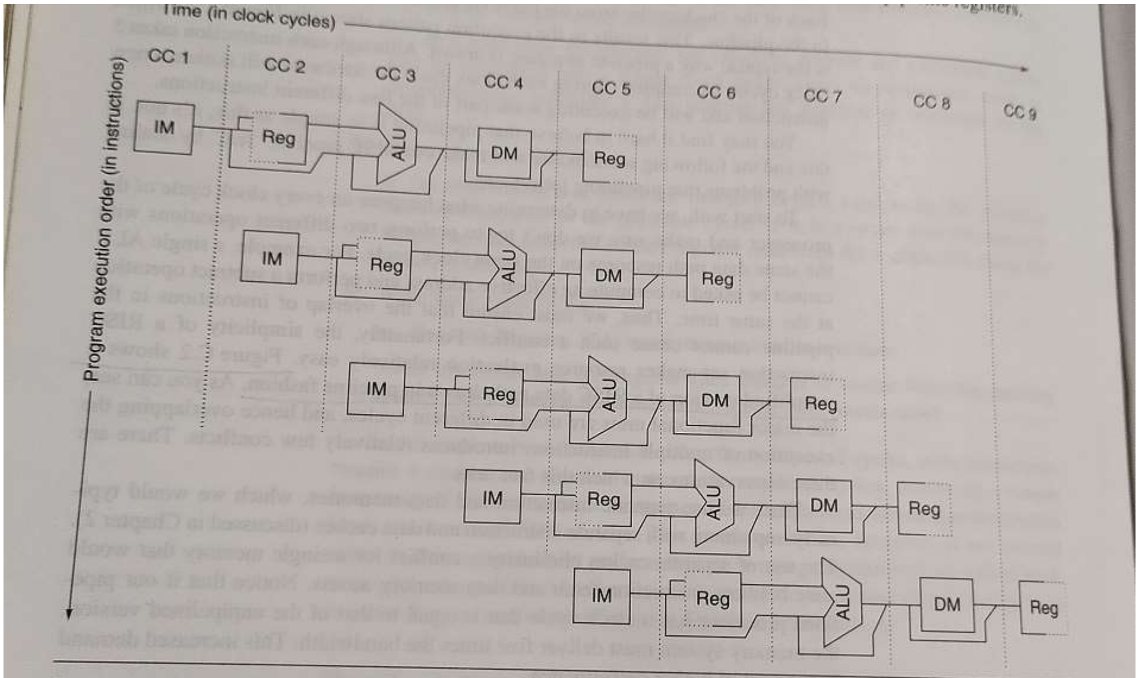


Figure C.2 The pipeline can be thought of as a series of data paths shifted in time. This shows the overlap among the parts of the data path, with clock cycle 5 (CC 5) showing the steady-state situation. Because the register file is used as a source in the ID stage and as a destination in the WB stage, it appears twice. We show that it is read in one part of the stage and written in another by using a solid line, on the right or left, respectively, and a dashed line on the other side. The abbreviation IM is used for instruction memory, DM for data memory, and CC for clock cycle.

In Figure C.2, the instruction pipeline is drawn as a series of data paths shifted in time.

We observe that in clock cycle-4, CC 4, the 1st instruction accesses DM, the data memory, while the 4th instruction accesses IM, the instruction memory. In order to avoid memory conflict, we use **separate instruction and data memories**.

Again, in clock cycle-5, CC 5, the 1st instruction performs the WB pipeline stage and **writes** the registers (unit Reg), while the 4th instruction performs the register-**read** pipeline stage (simultaneously with instruction decoding) by accessing unit Reg. Thus the registers are accessed by two instructions in the same cycle. In order to avoid a conflict, **register writes are done in the FIRST HALF of the cycle while register reads are done in the SECOND HALF of the cycle.**

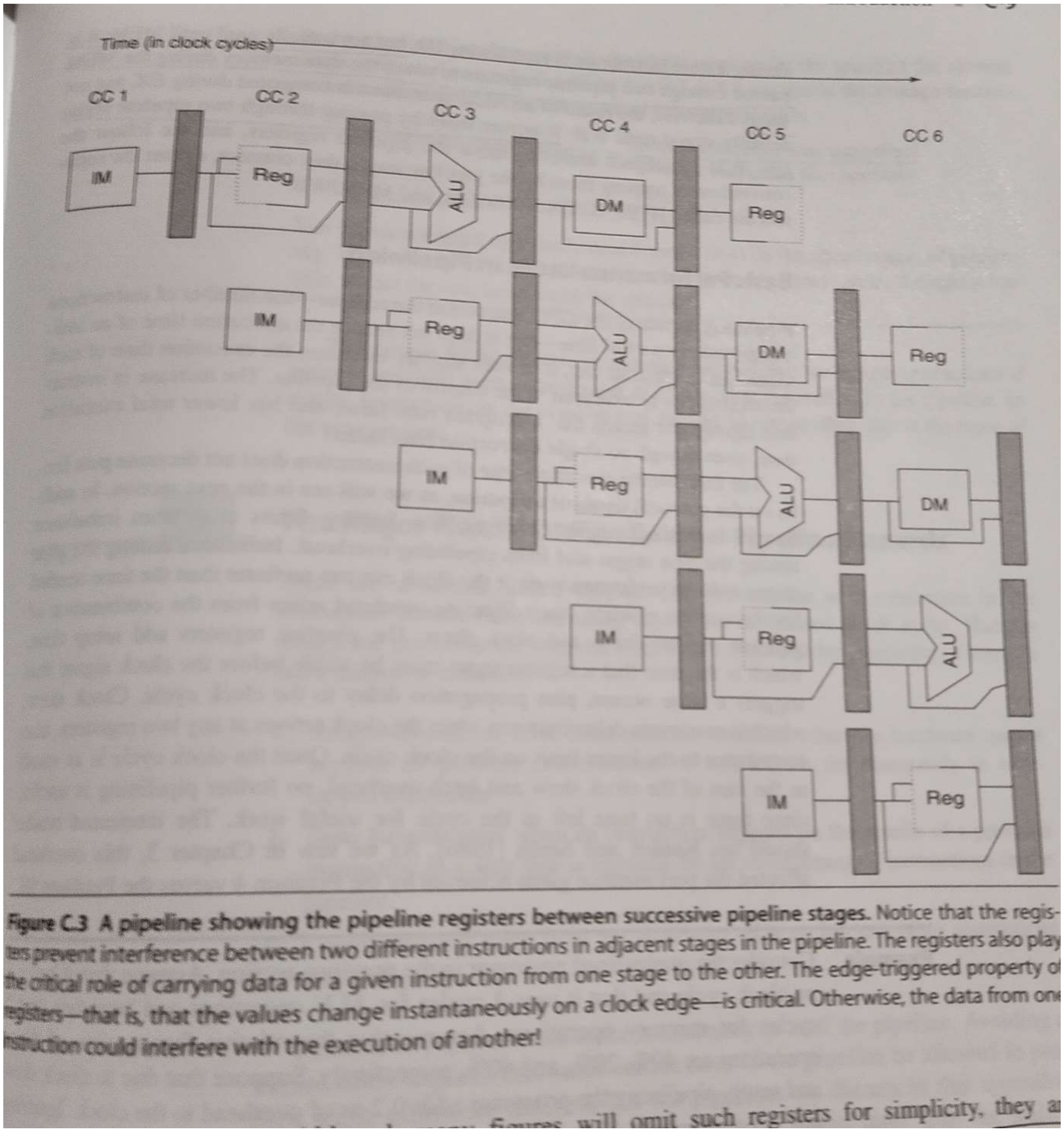


Figure C.3 A pipeline showing the pipeline registers between successive pipeline stages. Notice that the registers prevent interference between two different instructions in adjacent stages in the pipeline. The registers also play the critical role of carrying data for a given instruction from one stage to the other. The edge-triggered property o registers—that is, that the values change instantaneously on a clock edge—is critical. Otherwise, the data from one nstruction could interfere with the execution of another!

figures will omit such registers for simplicity, they a

In Figure C.3, **pipeline registers** are shown between successive stages of the pipeline. The results from one stage are stored in such registers and used by the next stage in the following cycle. These registers are also used to carry intermediate results from one stage to another when the two stages are not adjacent.

Pipeline Hazards

Hazards prevent the ideal speedup of a pipeline. They force an instruction to be delayed.

There are three classes of hazards:

1. *Structural hazards* arise fro resource conflicts.
2. *Data hazards* arise due to data dependences between instructions.

3. *Control hazards* arise from the pipelining of branches and other instructions that change the PC, i.e. the program counter.

Structural Hazards

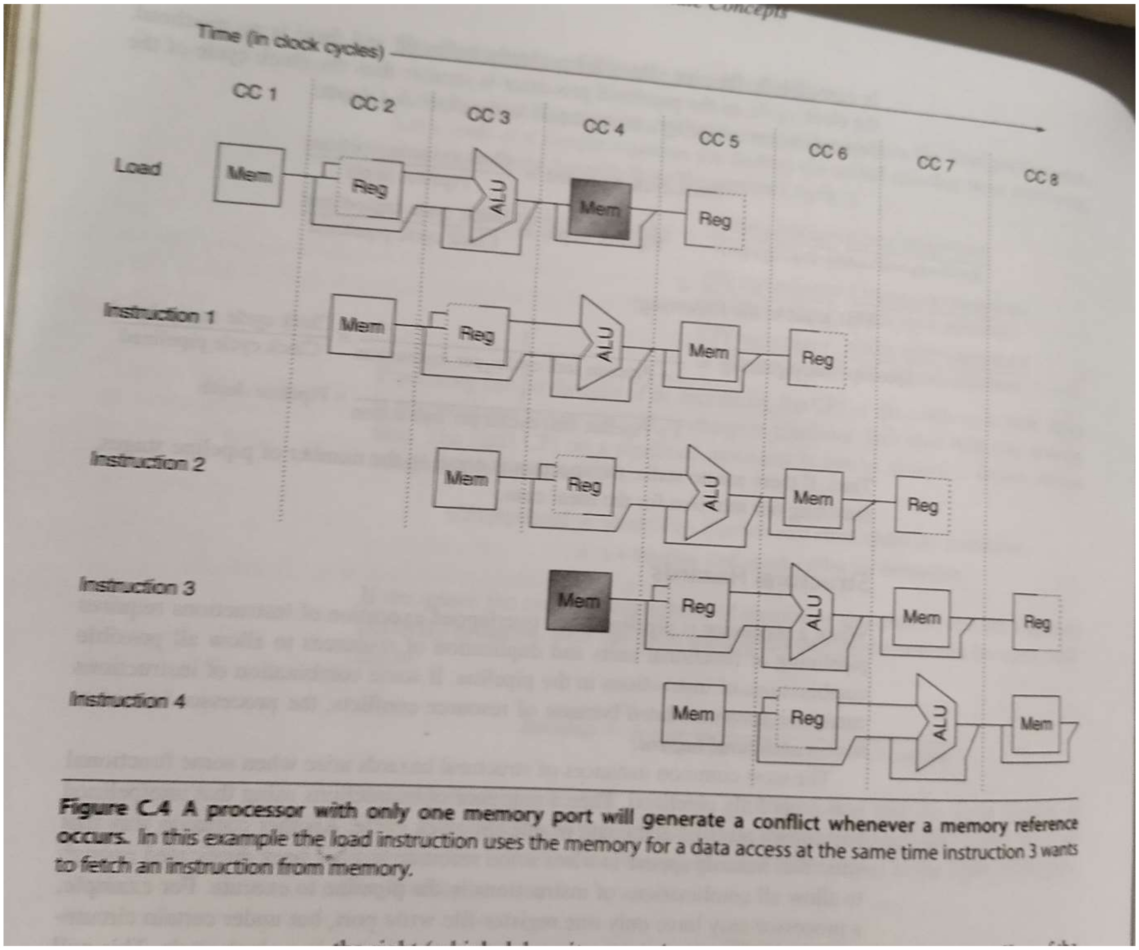
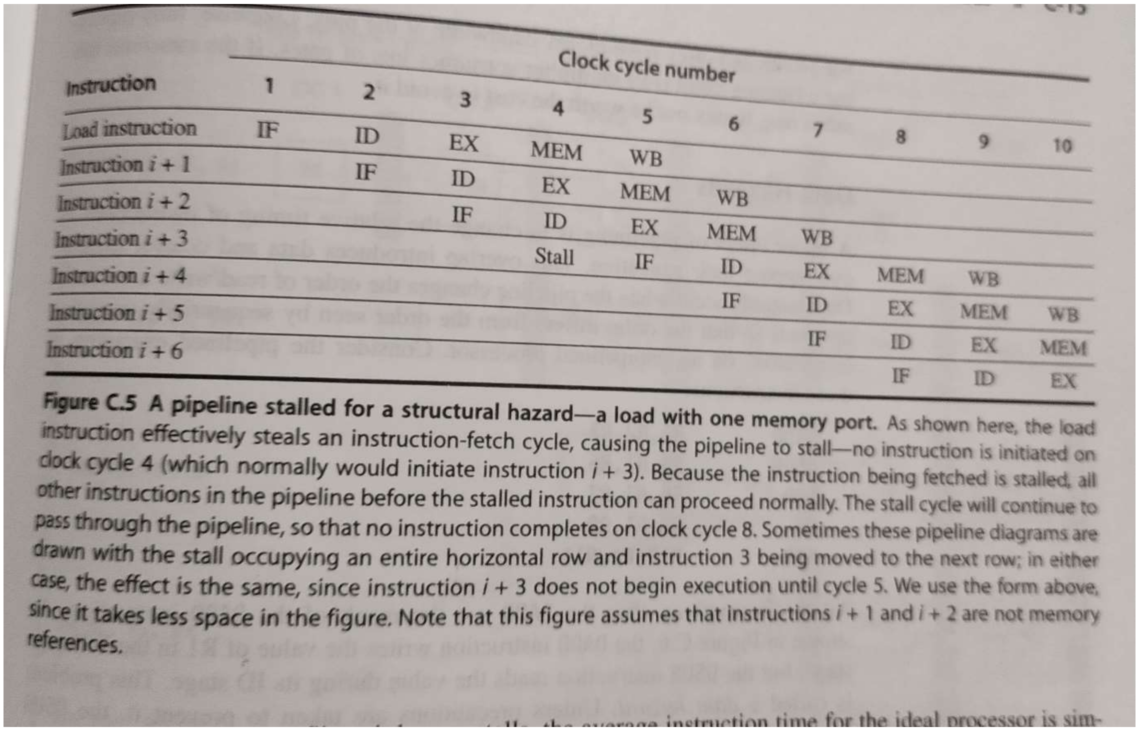


Figure C.4 A processor with only one memory port will generate a conflict whenever a memory reference occurs. In this example the load instruction uses the memory for a data access at the same time instruction 3 wants to fetch an instruction from memory.

In Figure C.4 above, there is a memory conflict in CC 4 when the load instruction accesses the data memory while instruction 3 accesses instruction memory. (no separate IM and DM).

A solution to this problem is to **stall** the pipeline for 1 clock cycle. A stall is also called a **pipeline bubble** or **bubble**; it floats through the pipeline, occupying space but doing no useful work.



Instruction	Clock cycle number									
	1	2	3	4	5	6	7	8	9	10
Load instruction	IF	ID	EX	MEM	WB					
Instruction *i* + 1		IF	ID	EX	MEM	WB				
Instruction *i* + 2			IF	ID	EX	MEM	WB			
Instruction *i* + 3				Stall	IF	ID	EX	MEM	WB	
Instruction *i* + 4						IF	ID	EX	MEM	WB
Instruction *i* + 5							IF	ID	EX	MEM
Instruction *i* + 6								IF	ID	EX

Figure C.5 A pipeline stalled for a structural hazard—a load with one memory port. As shown here, the load instruction effectively steals an instruction-fetch cycle, causing the pipeline to stall—no instruction is initiated on clock cycle 4 (which normally would initiate instruction *i* + 3). Because the instruction being fetched is stalled, all other instructions in the pipeline before the stalled instruction can proceed normally. The stall cycle will continue to pass through the pipeline, so that no instruction completes on clock cycle 8. Sometimes these pipeline diagrams are drawn with the stall occupying an entire horizontal row and instruction 3 being moved to the next row; in either case, the effect is the same, since instruction *i* + 3 does not begin execution until cycle 5. We use the form above, since it takes less space in the figure. Note that this figure assumes that instructions *i* + 1 and *i* + 2 are not memory references.

instruction time for the ideal processor is sim-

In Figure C.5 above, instruction i+3 enters the pipeline in cycle-5 and not cycle-4. This strategy prevents memory conflict in cycle-4. It is assumed that instructions i+1 and i+2 **do not access memory for data; otherwise** there would have been conflicts in cycles 5 and 6.