

Notes-02

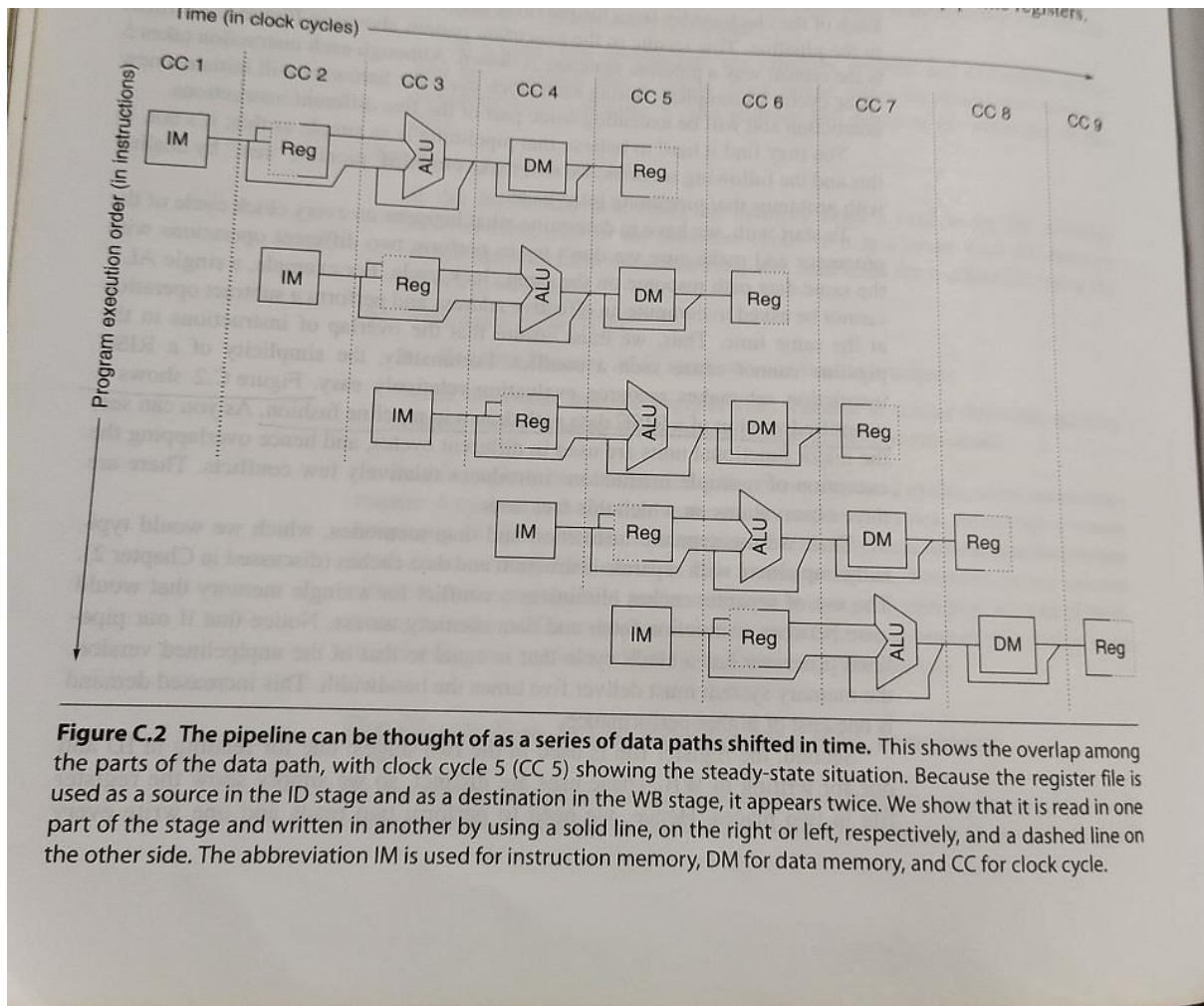
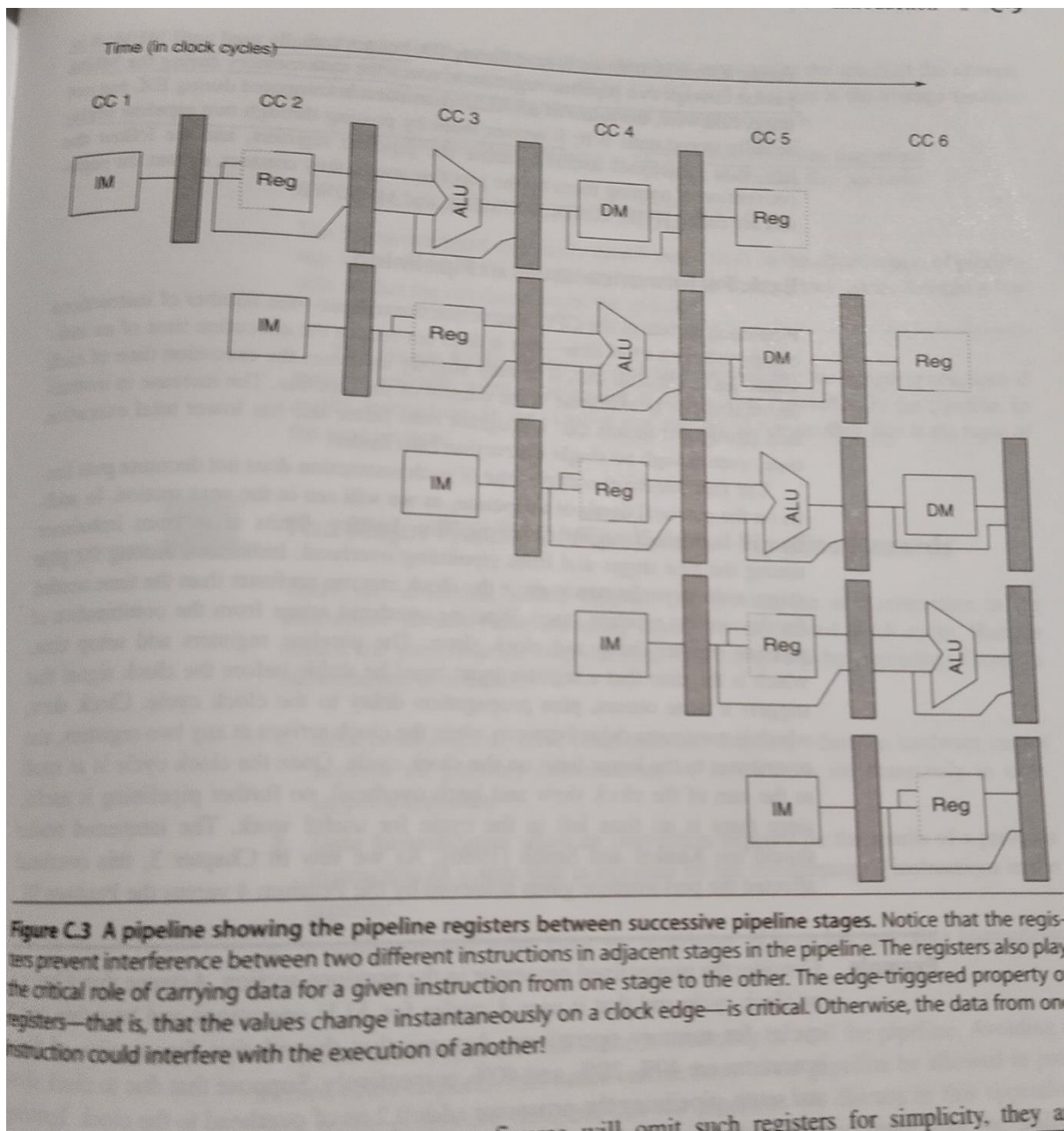


Figure C.2 The pipeline can be thought of as a series of data paths shifted in time. This shows the overlap among the parts of the data path, with clock cycle 5 (CC 5) showing the steady-state situation. Because the register file is used as a source in the ID stage and as a destination in the WB stage, 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**.



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

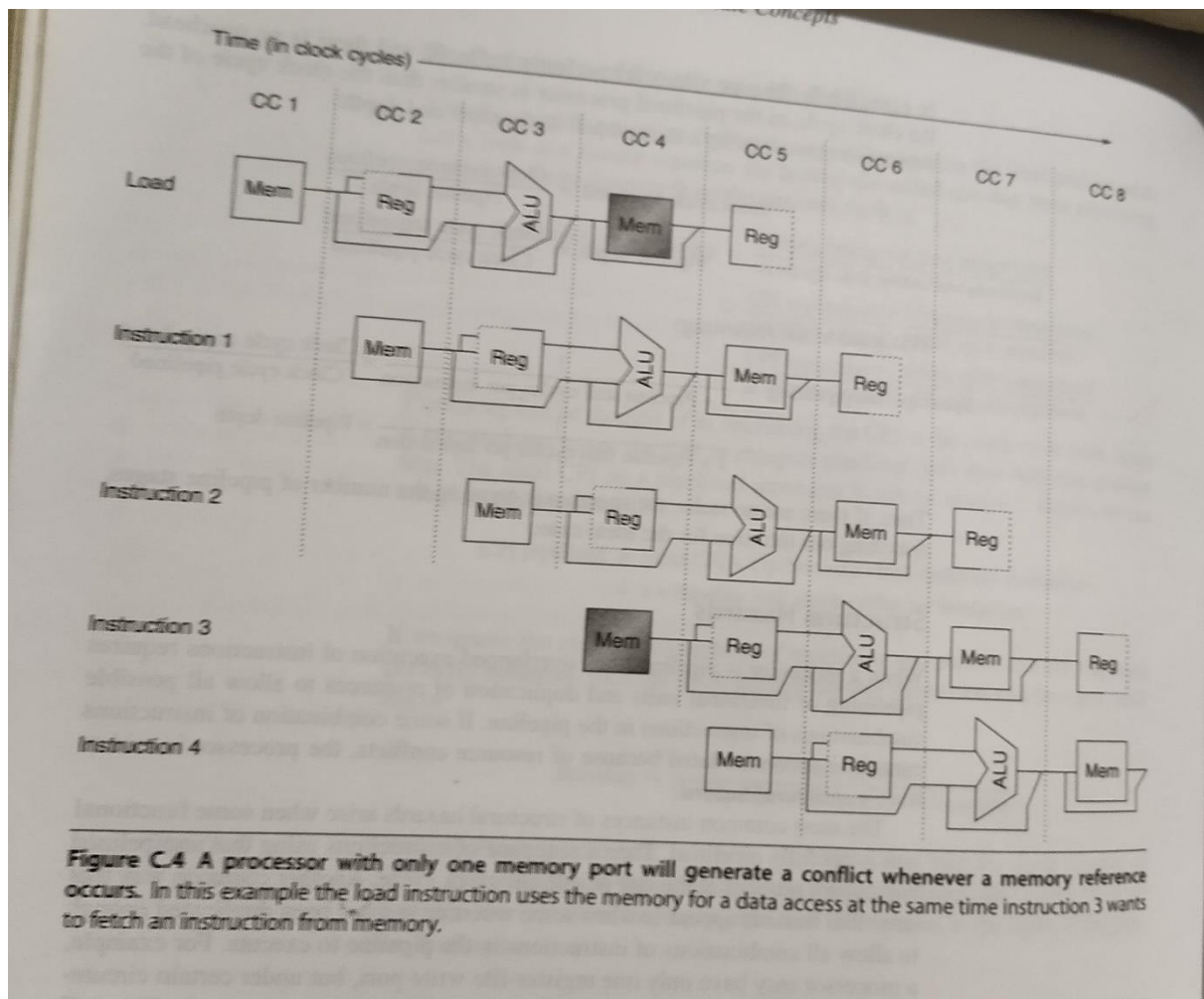
Hazards prevent the ideal speedup of a pipeline. They force an instruction to be delayed.

There are three classes of hazards:

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

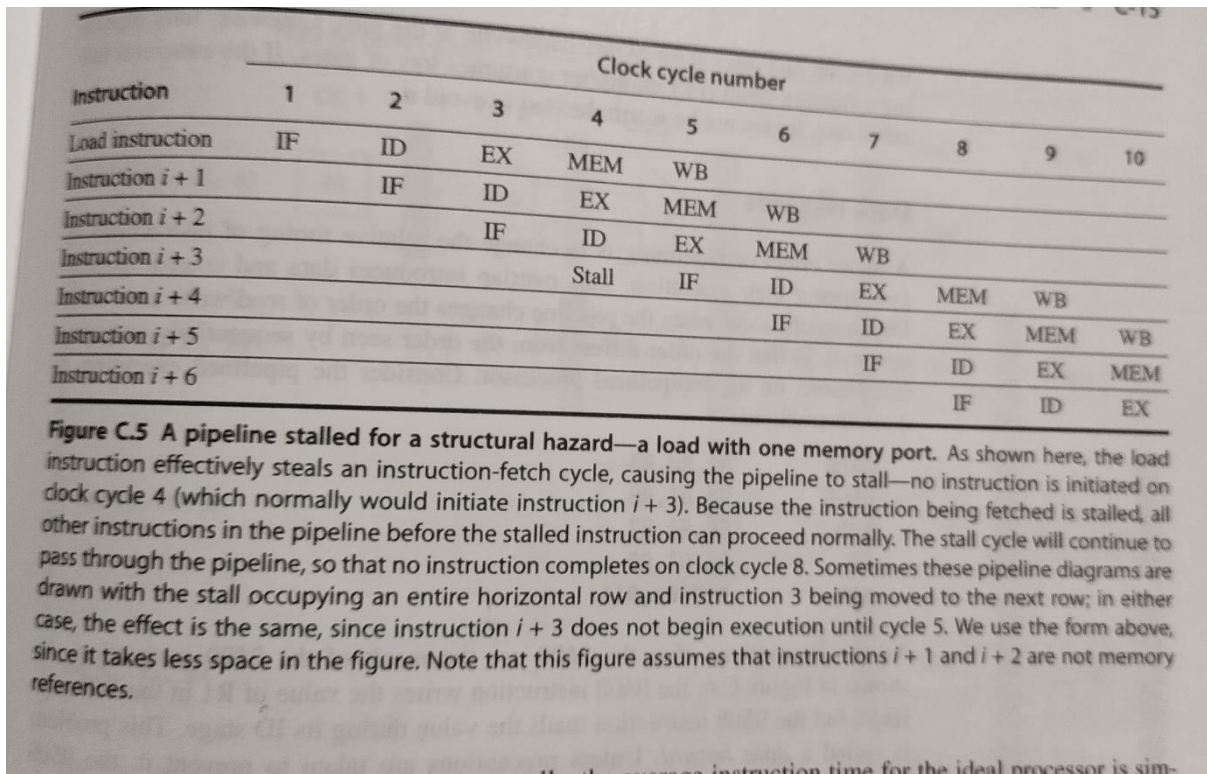
3. *Control hazards* arise from the pipelining of branches and other instructions that change the PC, i.e. the program counter.

Structural Hazards



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.



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.