

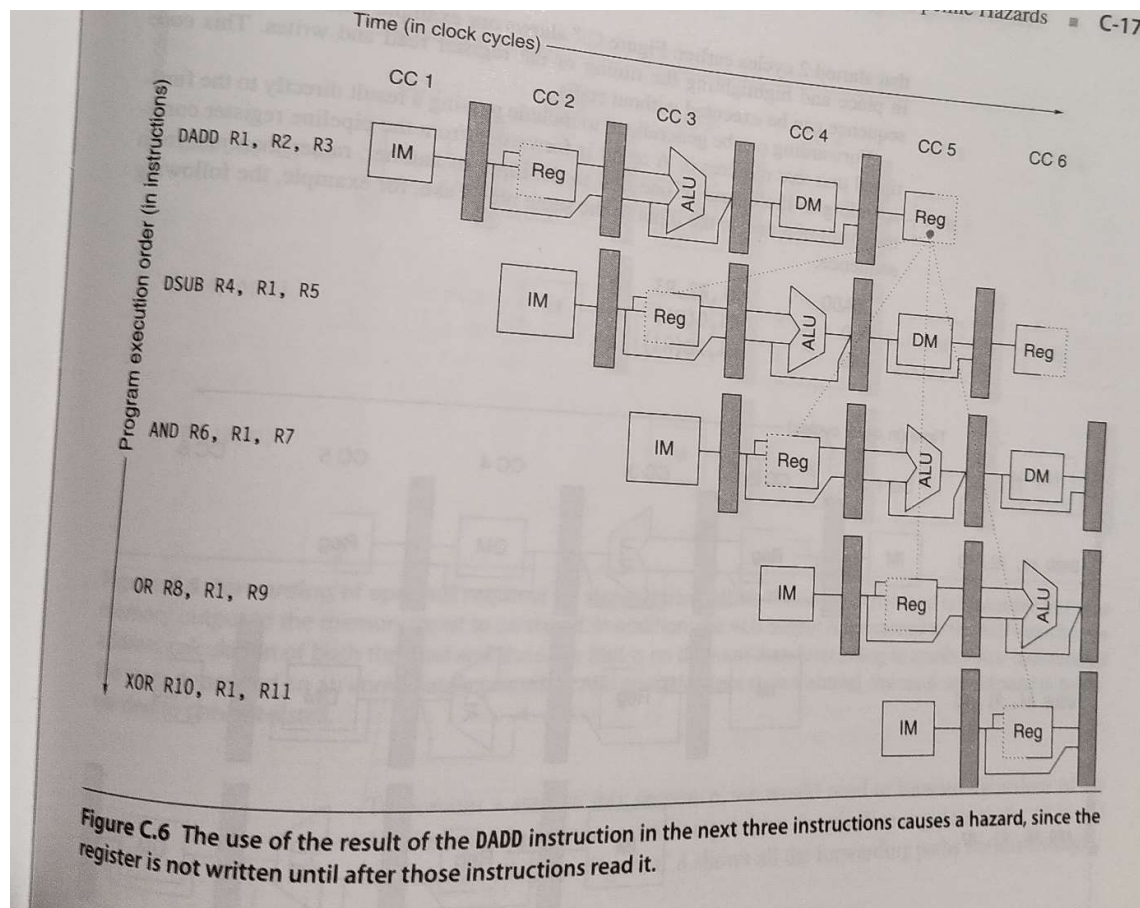
Notes-03

Data Hazards

Consider the following program fragment:

```
DADD R1, R2, R3
DSUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
XOR R10, R1, R11
```

The DADD instruction writes its result in register R1; all subsequent instructions use R1 as an input operand. The pipelined execution is shown in the following timing diagram.



In Figure C.6, the DADD instruction writes its result to Register R1 in CC5, i.e. clock cycle 5. However, the second instruction, i.e., **DSUB**, reads its two inputs, i.e. Registers R1 and R5 in CC3, i.e. clock cycle 3, **BEFORE DADD has written !** Clearly, **DSUB doesn't read the value written into R1 by DADD**. This problem is called a **DATA HAZARD**. This is caused by overlapped execution: DSUB begins execution before its preceding DADD has finished.

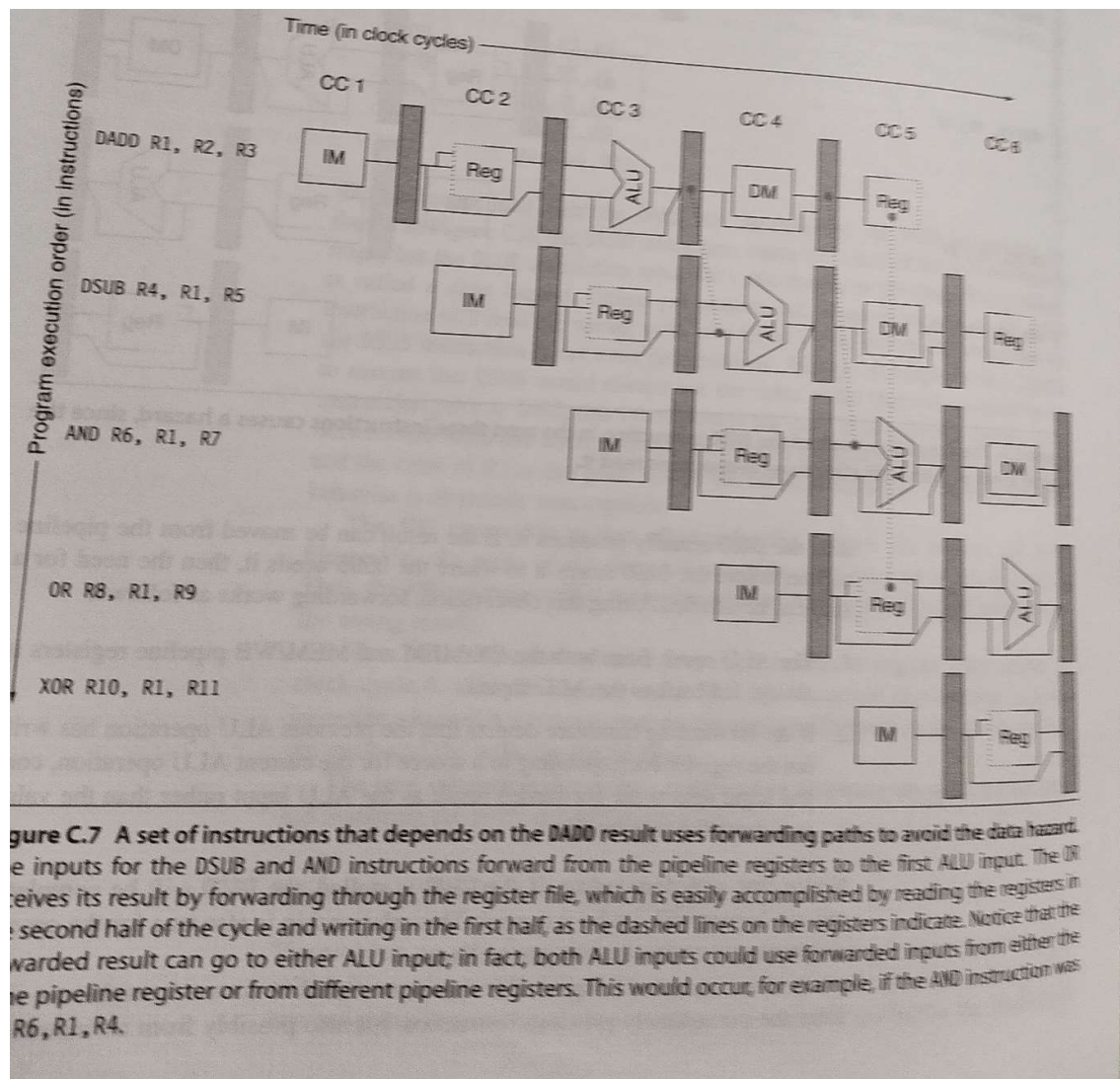
A similar problem occurs between DADD and AND, which reads R1 in CC4.

Fortunately, there is no problem involving DADD and OR. Although they access R1 in the same cycle CC5, DADD writes to R1 in the **FIRST HALF** cycle CC5 while OR reads from R1 in the **SECOND HALF** of CC5, i.e. after DADD has finished writing.

Similarly, there is no problem involving DADD and XOR because XOR reads R1 in cycle CC6, **AFTER** DADD has written into R1 in cycle CC5.

Resolving Data Hazard by forwarding

A solution to this problem can be obtained by observing that the output R1 of the DADD operation is obtained at the end of CC3 when DADD has finished using the ALU stage while DSUB needs R1 in CC4 when it uses the ALU stage. If we can somehow copy the ALU output of CC3 to the ALU input of CC4, the data hazard can be resolved. This is indeed the solution used by the hardware technique of **forwarding** (also called **bypassing** or **short-circuiting**). DSUB need not wait for DADD to write into Register R1 in cycle CC5; a copy of the sum produced by DADD is **already available** at the end of cycle CC3.

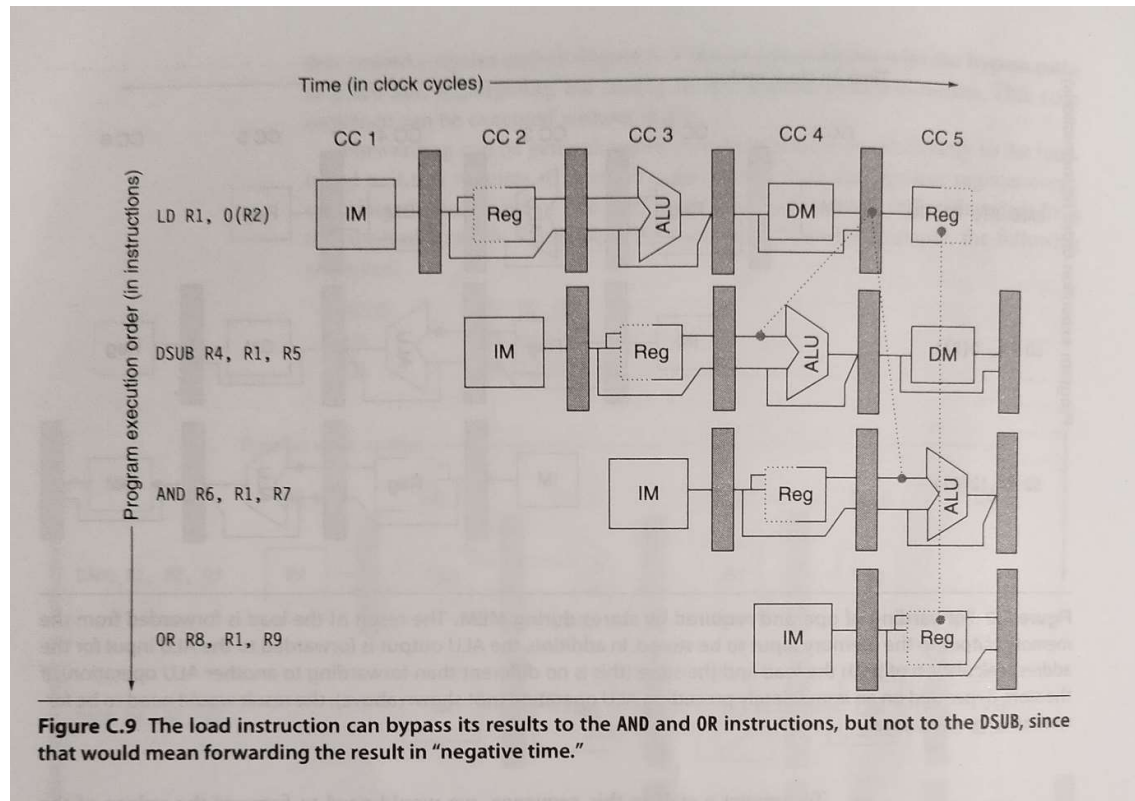


Resolving Data Hazards by stalling

Consider the following program fragment:

```
LD    R1, 0(R2)
DSUB  R4, R1, R5
AND   R6, R1, R7
OR    R8, R1, R9
```

The timing diagram for this program is given below.



The LD instruction receives the memory content at the end of cycle CC4 while the DSUB instruction needs the memory content at the beginning of cycle CC4. So **even with forwarding**, the hazard **CANNOT** be resolved; data cannot travel backward in time !

This hazard can be resolved by **stalling** the pipeline as shown below.

LD	R1,0(R2)	IF	ID	EX	MEM	WB						
DSUB	R4,R1,R5		IF	ID	EX	MEM	WB					
AND	R6,R1,R7			IF	ID	EX	MEM	WB				
OR	R8,R1,R9				IF	ID	EX	MEM	WB			

LD	R1,0(R2)	IF	ID	EX	MEM	WB						
DSUB	R4,R1,R5		IF	ID	stall	EX	MEM	WB				
AND	R6,R1,R7			IF	stall	ID	EX	MEM	WB			
OR	R8,R1,R9				stall	IF	ID	EX	MEM	WB		

Figure C.10 In the top half, we can see why a stall is needed: The MEM cycle of the load produces a value that is needed in the EX cycle of the DSUB, which occurs at the same time. This problem is solved by inserting a stall, as shown in the bottom half.

The DSUB instruction is stalled in CC4 and enters EX stage in CC5. By that time, LD has already fetched the memory data which is available in pipeline registers; this can be forwarded to DSUB. Similarly, the AND and OR instructions are stalled in CC4. The AND instruction reads R1 in CC5 (ID cum register read cycle) and the data can come directly from the register file in the middle of CC5; this is also forwarding. The OR instruction reads R1 in cycle CC6 AFTER LD has finished writing in CC5; no forwarding is needed.