Pipelined Execution → In each cycle, a new instruction can enter the pipeline.

		t1	t2	t3	t4	t5	t6	t7	---------- >t
Instruction	1	F1	D1	E1	W1				
	2		F2	D2	E2	W2			
	3			F3	D3	E3	W3		
	4				F4	D4	E4	W4	

Dependencies between instructions

i) Data dependency;
ii) Control dependency;
iii) Resource dependency.

i) DATA Dependencies

RAW dependency

i1: load r1, a
i2: add r2, r1, r1

i2 is RAW(Read after Write) dependent on i1
{i2 uses r1 as source; i1 loads r1}

WAR dependency

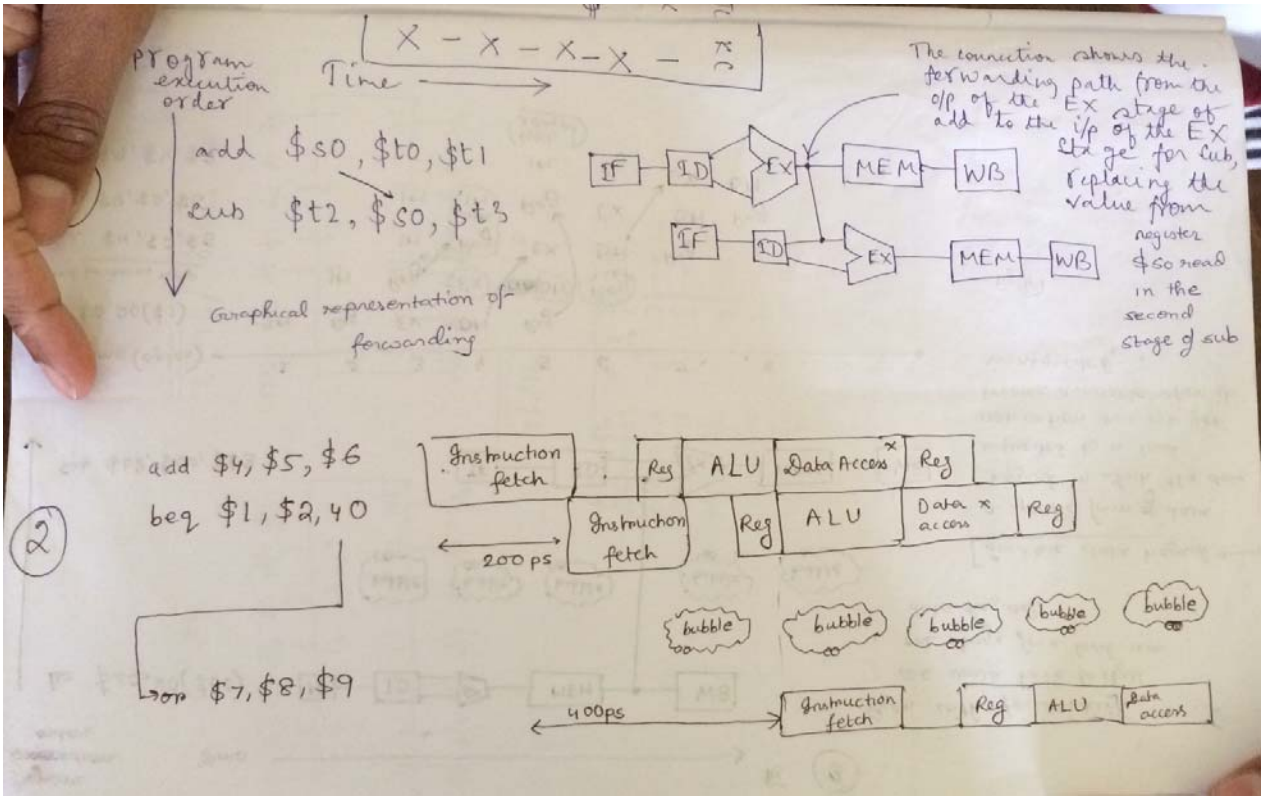
i1: mul r1, r2, r3
i2: add r2, r4, r5

If i2 executes BEFORE i1, the original content of r2 would be rewritten earlier than it is read by i1.

Add $so, $t0, $t1
Sub $t2, $s0, $t3

Solution: We don't need to wait for the instruction to complete before trying to resolve the data hazard. As soon as the ALU creates the sum for the add, we can supply it as input for the subtract. Adding extra hardware to retrieve the missing item early from the internal resources is called **forwarding or bypassing.**

PASTE HERE (Program execution order graph) FIG 1



Control Hazard (also called branch hazard):

An occurrence in which the proper instruction cannot execute in the proper clock cycle because the instruction that was fetched is not the one that is needed; that is, the flow of instruction addresses is not what the pipeline expected.

FIG 2 (Above)

Pipeline showing stalling on every conditional branch as solution to control hazards. There is a one-stage pipeline stall, or bubble, after the branch. [PC is updated during the second stage of the pipeline.]

Data hazards and stalls

In addition to a forwarding unit, we need a hazard detection unit. It operates during the ID stage so that it can insert the stall between the load and its use.

Condition checked

If (instruction is a load)

/*the only instruction that reads data memory is a load*/

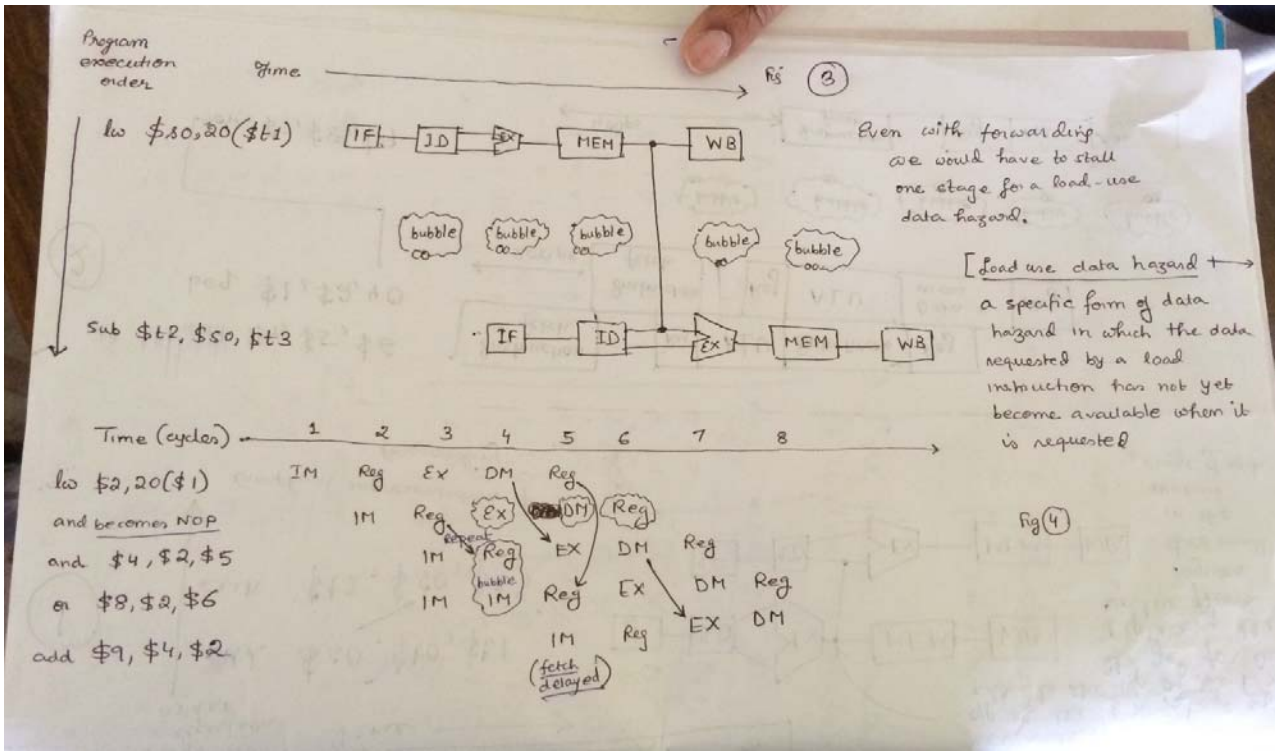
If (destination-register of load in EX matches either source register of instruction in ID stage)

Stall the pipeline;

After this one cycle stall, the forwarding logic can handle the dependency and execution proceeds.

FIG 3

FIG 4



A bubble is inserted beginning in clock cycle 4, by changing the AND instruction into a **NOP**. Note that the AND instruction is really fetched and decoded in clock cycle AND, but its EX stage is delayed until clock cycle 5 (would have been done in cycle 4 if stall was absent).

The hazard faces the AND and OR instructions to repeat in clock cycle 4 what they did in clock cycle 3.

Branch Statistics:

The frequency of branches affects the amount of parallelism that can be extracted from a program. The lower the frequency of branches, the longer, on average, the basic blocks are and, thus, the more parallelism that can be extracted by computers

Branches account for about 20% of general-purpose code and almost 5-10% of scientific/technical code. This means that in general-purpose code, on average, each fifth instruction or so is a branch.

The majority of branches are conditional.