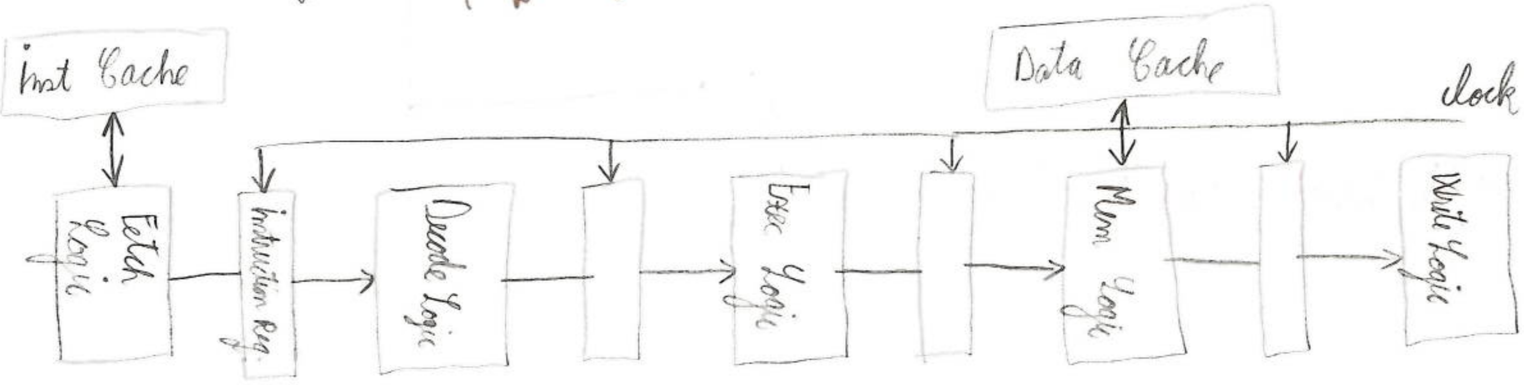
PIPELINES

Pipelining is a technique in which the instructions are split into different stages in a way that multiple instructions can overlap their execution. It provides a more efficient utilisation of the processor resources while at the same time allows increasing clock frequency.



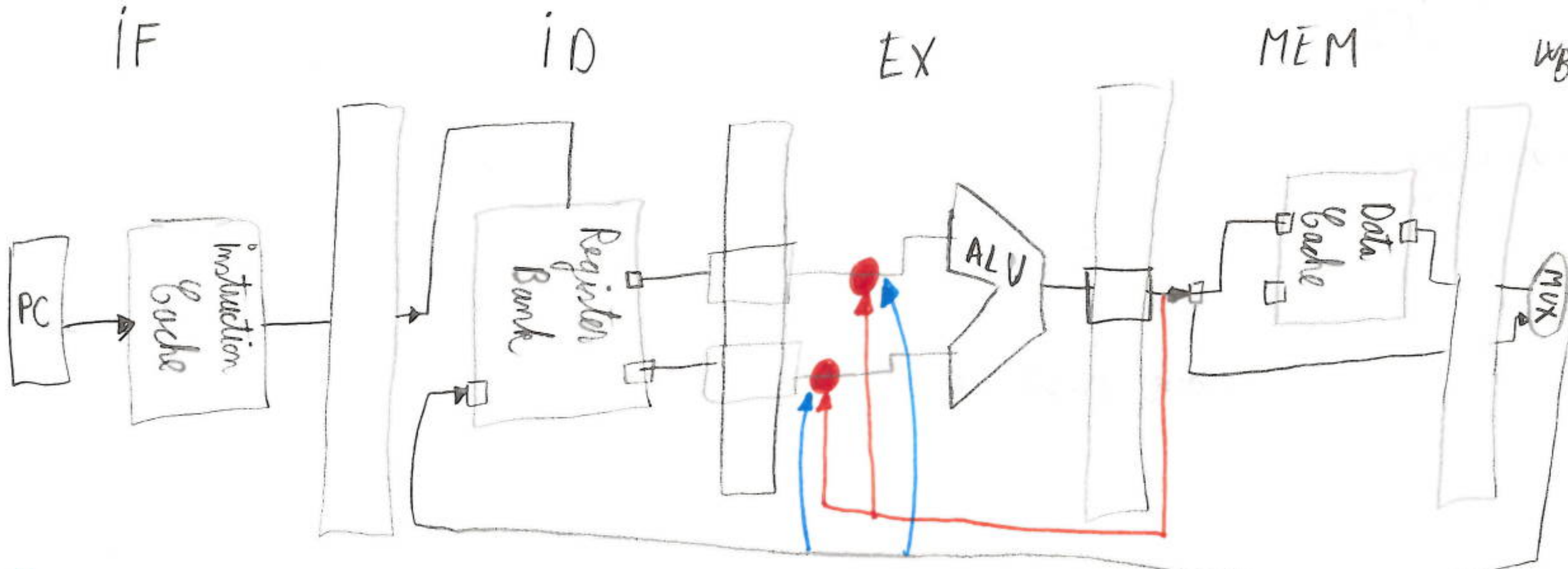
Problems that can arise with the use of pipelining

Data Hazard = an instruction needs to read from a register that is written by a previous instruction before it is stored in the register bank which will require to stall the pipeline until the data is available

Solution: - forwarding / bypassing (adding connections from the output of the ALU[stage 3] and Memory Access[stage 4] to the input of the ALU)
- reordering of instructions during compilation

Control Hazard = when executing a branch, the following instruction is not decided until stage 2 if it is unconditional or stage 3 if it is conditional, so the instruction fetched after a branch may not be the one that needs to be executed after the branch instruction.

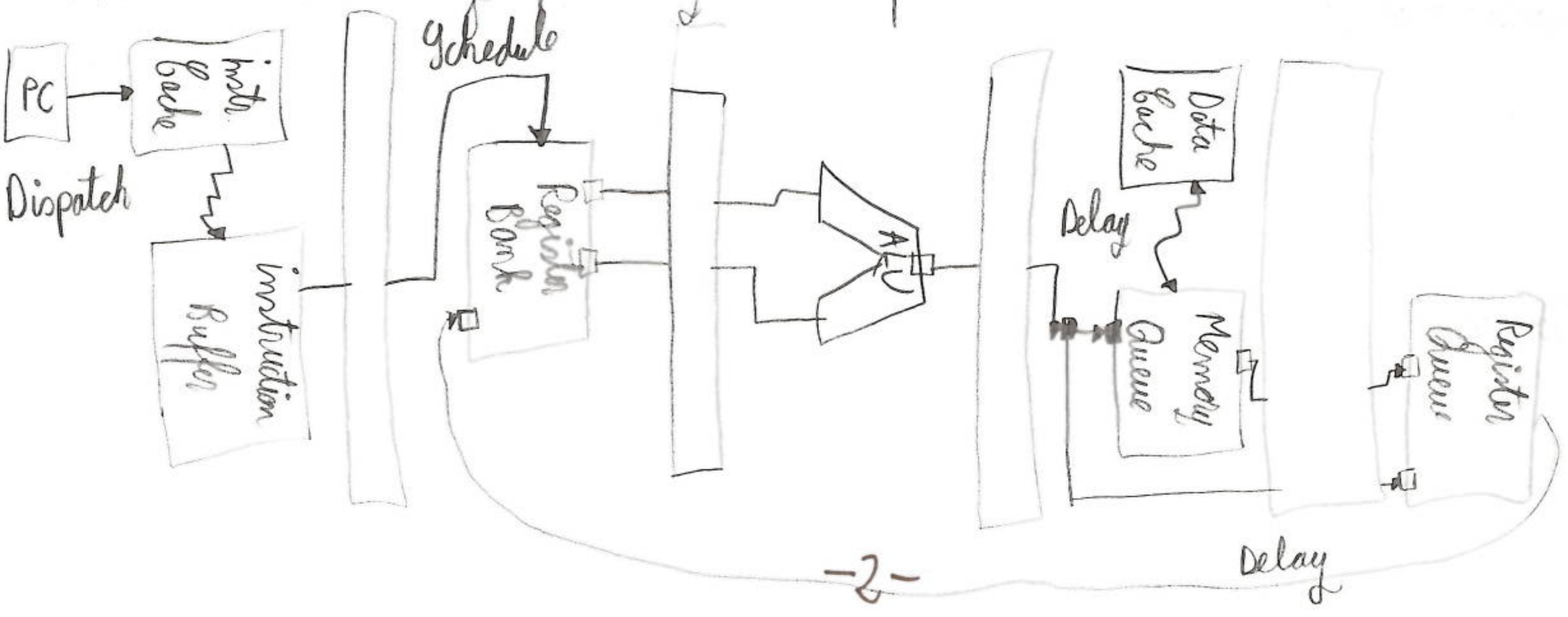
Solution: - stall the pipeline once a branch instruction is decoded
- add NOP instructions after each branch
- use branch prediction



Instruction Level Paralelism (ILP)
- need to fetch multiple instructions per cycle and to decode multiple instructions per cycle.
- need multiple ALUs

Superscalar
- two instr. can execute in parallel, but the access rate to registers and cache will be doubled (need a dual ported register bank and dual ported cache)
- use a 'dispatch unit' in the fetch stage which uses hardware to examine the instruction dependencies; only issue if they are independent

Out of order execution processor
- an instruction buffer needs to be added to store all issued instructions
- a scheduler is in charge of sending non-conflicted instruction to execute


PC
instr. Cache
Dispatch
instruction Buffer
Schedule
Register Bank
ALU
Delay
Data Cache
Memory Queue
Register Queue
Delay