

Pipelined Datapath & Control

Topics

- Single-cycle Pipelined datapath diagram
- Hazards in a pipelined datapath
- Pipeline registers
- How data flows through pipelined datapath
- 2 types of pipelined datapath diagrams & what they reveal
- How control signals flow through pipelined datapath

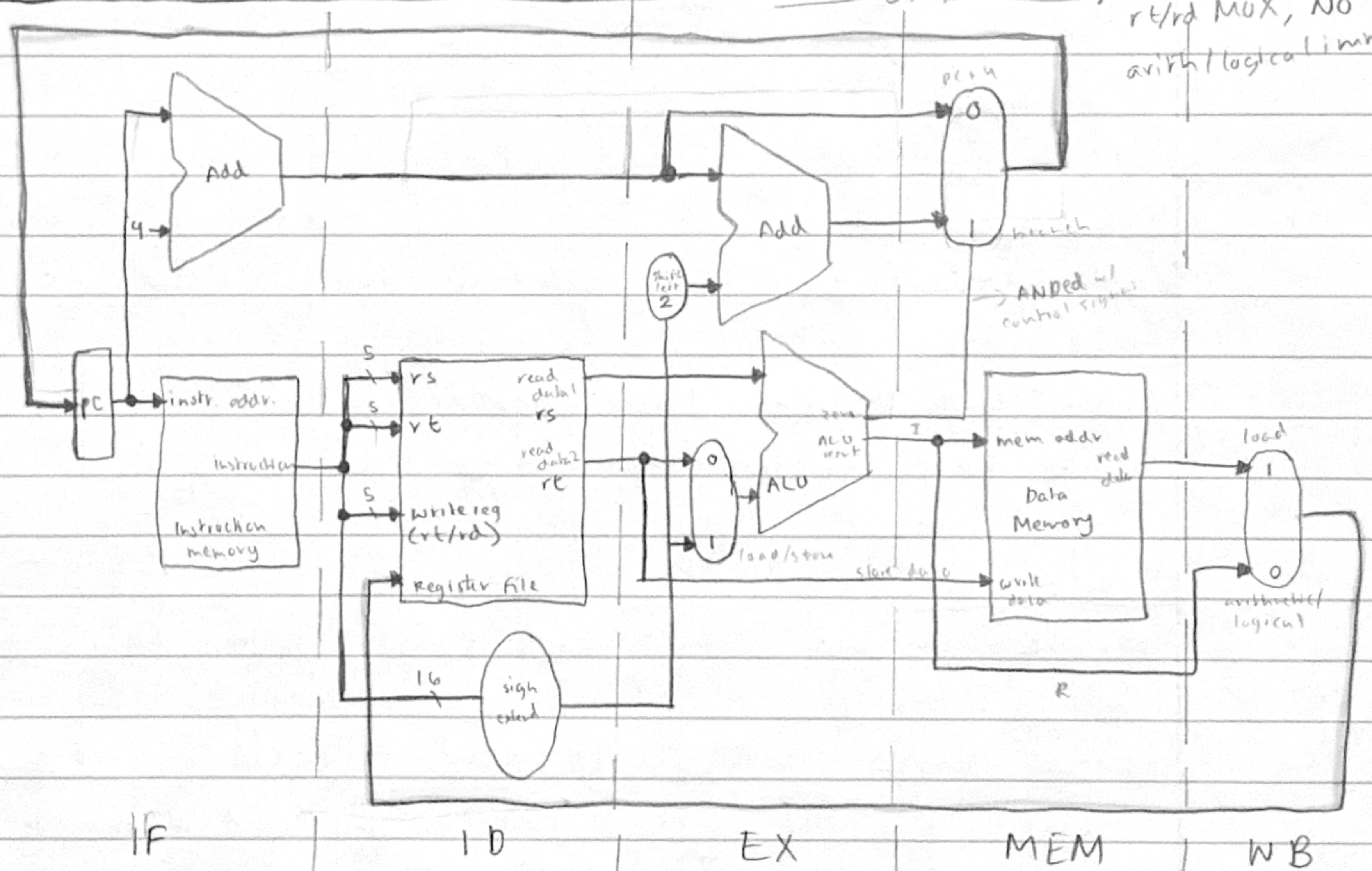
Pipelining Recap

- pipelining INCREASES throughput (jobs / period) & DECREASES reciprocal throughput / execution time (avg time / job) (GOOD)
- it may INCREASE latency (time / individual instruction) due to dividing instructions into stages & pipelined register overhead

due to temporal parallelism

(Single-cycle) Pipelined datapath diagram

simplified: shows arithmetic/logical, branching, load/store; NO jump, NO rt/rd MUX, NO arith/logical immediate



- fetch next instruction from mem
- $PC + 4$

- generate control signals
- decode/read register operands
- sign-extend immediates
- some implementation: calc branch/jump targets/tests

- compute arith/log results
- calculate EAs for load/store instrs.
- evaluate branch condition
- calc branch targets

- read or write data
- mux decides if branch addr is taken

- write arith/log or load result back to write data in register file

Hazards in a Pipelined datapath

• data flows left to right through datapath, w/ 2 exceptions:

① branch decisions: whether to use PC+4 or branch is determined in MEM stage, and modifies PC in IF stage

② write back: writing back arithmetic or load results in WB stage, modifies register file in ID stage.

• These backwards flows can cause 2 types of hazards due to dependencies in the pipeline:

① WB → ID: data hazards. IF subsequent instructions rely on the write register to be correct in the ID stage, they must wait for the instruction's WB phase to use the data. This is related to data value dependencies.

② MEM → IF: control hazards. The pipeline relies on the PC value, from the IF stage, to load the next instruction. In the case of branch statements (a jump, though they may be resolved @ ID), the pipeline must wait until the MEM stage when the branch addr is determined to fetch the next instruction.

• 1 more hazard: structure hazard. When multiple stages need to access the same resource @ the same time. This is why instruction & data memory are separated.

Managing hazards

• data hazards: forwarding data from one stage to another w/o waiting to write it to memory,

• control: static & dynamic branch prediction

• structure: separate data & instruction memory

Pipeline registers

• hold information stored in previous clock cycles (control signals, destination addresses, etc...)

• control signals derived in ID stage and held in registers between stages. All info is passed through, w/ used information being consumed.

ex: ID/EX holds EX, MEM, WB signals. MEM holds MEM, WB. WB holds WB.

EX: