Chapter 6: Datapath

- The *processor* is a part of the computer that manipulates data and makes its own decisions. It's split into two parts: datapath and control
  - The **datapath** is the part of the processor with the necessary hardware to perform the operations required the ISA
  - The control tells the datapath what needs to be done.

On every tick of the clock, processors execute a single instruction. Instruction execution takes place in 5 stages:

- **Instruction Fetch**
- **Instruction Decode**
- **Execute**
- **Memory Access**
- **Write Back**
- Disc 6: Single Cycle Datapath

Each datapath has several main state elements:

- *Register File* – an array of registers which the processor uses to keep information out of memory
- *Program Counter* (PC) – a special register which keeps track of where the processor is in the program
- *IMem* – Read-Only section of memory containing the instruction that needs to be executed
- DMem – section of memory which contains data the processor needs to be read/write

Data Path Elements:

- *Immediate Generator*: Creates immediate values.
- *Branch Comparator*: Decides branch instructions.
- *ALU*: Performs math and logic operations.
- *Synchronous Operation*: All components work simultaneously.
- *Control Mechanism*: Determines data path operation. Can be implemented using combinational logic or ROM.

6.1 Pipelined Datapath

- **Single-Cycle Datapath**: every instruction passes through the datapath one at a time
  - This is inefficient because faster stages (ex, register reading) are left unused while waiting for slower stages (ex, memory reading)
- **Pipelining the Datapath:** this can be fixed by allowing multiple instructions to use different parts of the datapath at once, speeding up the processor bc no stage is left unutilized.
  - All we need to do is add registers after each datapage stage

- **Hazards Introduced by Pipelining**: Registers between stages can cause data, structural, and control hazards
    - Need to insert stall cycles or forwarding logic to deal with hazards

6.1.1 Structural Hazards

- When instructions compete for same resource
- Solutions:
    - Take turns accessing resource
    - Add more hardware (e.g. extra read/write ports)

6.1.2 Data Hazards

- When instructions have data dependency
- Solutions:
    - Stall cycles
    - Forwarding - loop result back to input
    - Reorder instructions

6.1.3 Control Hazards

- When program flow changes and pipeline instructions become invalid
- Only a problem when branch is taken
- Solutions:
    - Flush pipeline by converting to no-ops
    - Branch prediction - predict if branch taken, load speculated instructions