CS61C Fall 2025

Single-Cycle Datapath

Discussion 8

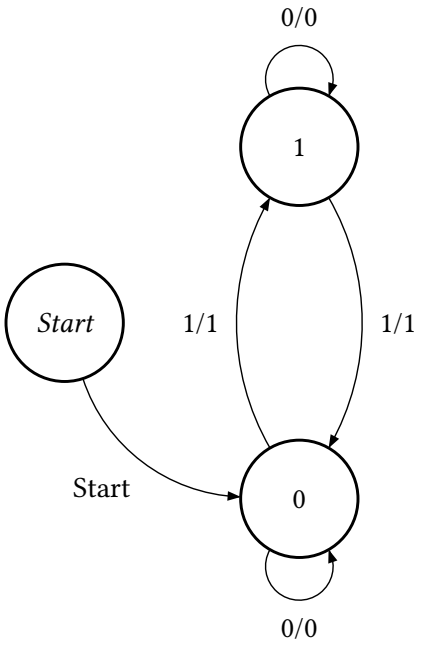
1 FSM

A finite state machine is a type of simple automaton where the next state and output depend only on the current state and input. Each state is represented by a circle, and every proper finite state machine has a starting state, signified either with the label "Start" or a single arrow leading into it. Each transition between states is labeled `[input]/[output]`.

For example, below is a finite state machine with two states (0 and 1). It outputs `1` when the state changes, and `0` when the state stays the same.

The machine starts in state `0`. When the input is `0`, it stays in its current state and outputs `0`. When the input is `1`, it switches to the other state and outputs `1`.

When in state `1`, the machine behaves the same way: it stays in state `1` and outputs `0` when the input is `0`, and switches back to state `0` with an output of `1` when the input is `1`.



With combinational logic and registers, any FSM can be implemented in hardware!

2 Pre-Check: T/F?

2.1 Register "clk-to-q" delay is the time between the rising edge of the clock signal and the register's hold time.

2.2 State elements only update their output on the rising edge of the clock, even if the inputs change between clock rising edges.

2.3 The single cycle datapath uses the outputs of all hardware units for each instruction.

2.4 It is possible to execute the stages of the single cycle datapath in parallel to speed up execution of a single instruction.

2.5 If the logic delay of reading from IMEM is reduced, then any (non-empty) program using the single cycle datapath will speed up.

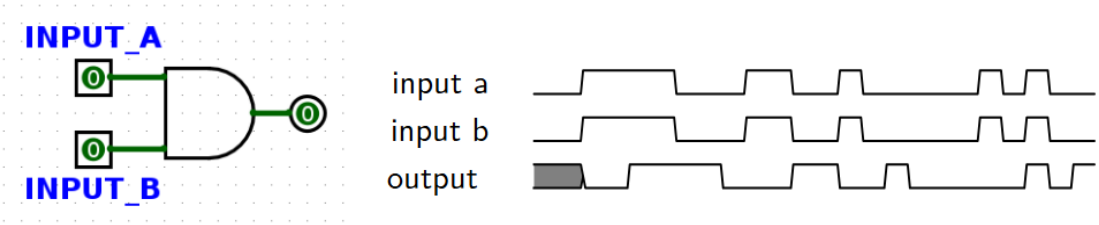
2.6 Stores and loads are the only instructions that require input/output from DMEM.

2.7 It is possible to feed both the immediate generator's output and the value in rs2 to the ALU in a single instruction.

3 SDS

There are two basic types of circuits: combinational logic circuits and state elements.

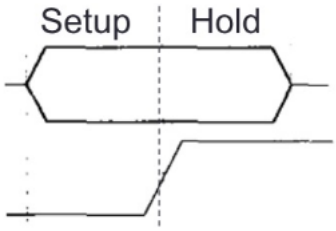
Combinational logic circuits simply change based on their inputs after whatever propagation delay is associated with them. For example, if an AND gate (pictured below) has an associated propagation delay of 2ps, its output will change based on its input as follows:



You should notice that the output of this AND gate *always* changes 2ps after its inputs change.

State elements, on the other hand, can *remember* their inputs even after the inputs change. State elements change value based on a clock signal. A rising edge-triggered register, for example, samples its input at the rising edge of the clock (when the clock signal goes from 0 to 1).

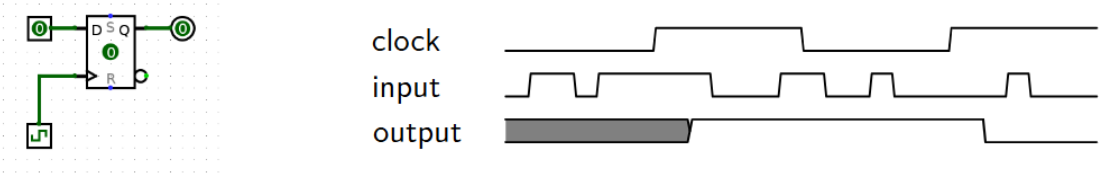
Like logic gates, registers also have a delay associated with them before their output will reflect the input that was sampled. This is called the **clk-to-q** delay. (“Q" often indicates output). This is the time between the rising edge of the clock signal and the time the register’s output reflects the input change.



The input to the register samples has to be stable for a certain amount of time around the rising edge of the clock for the input to be sampled accurately. The amount of time before the rising edge the input must be stable is called the **setup** time, and the time after the rising edge the input must be stable is called the **hold** time. Hold time is generally included in clk-to-q delay, so clk-to-q time will usually be greater than or equal to hold time.

Logically, the fact that clk-to-q $\geq$ hold time makes sense since it only takes clk-to-q seconds to copy the value over, so there’s no need to have the value fed into the register for any longer.

Examine the register circuit and assume **setup** time of 2.5ps, **hold** time of 1.5ps, and a **clk-to-q** time of 1.5ps. The clock signal has a period of 13ps.



Notice that the value of the output in the diagram doesn’t change immediately after the rising edge of the clock. Until enough time has passed for the output to reflect the input, the value held by the output is garbage; this is represented by the shaded gray part of the output graph. Clock

cycle time must be small enough that inputs to registers don't change within the hold time and large enough to account for clk-to-q times, setup times, and combinational logic delays.

A few important SDS relationships are below:

$$\tau_{\text{critical path delay}} = \tau_{\text{clk-to-q}} + \tau_{\text{combinational logic delay}} + \tau_{\text{setup time}}$$

where $\tau_{\text{combinational logic delay}}$ is the maximum combinational logic delay for any register $\longrightarrow$ register path in the circuit. The path with the maximum delay is called the "critical path".

Additionally, circuits must satisfy hold-time constraints because hold times may be violated if data propagates too quickly (see above):

$$\tau_{\text{clk-to-q}} + \tau_{\text{smallest combinational delay}} \geq \tau_{\text{hold time}}$$

4 Single-Cycle Datapath

Our single-cycle datapath is a synchronous digital system (SDS) that has the capabilities of executing RISC-V instructions. It is divided into multiple stages of execution, where each stage is responsible for a completing a certain task.

IF Instruction Fetch:

- Send address to the instruction memory (IMEM), and read IMEM at that address.
- **Hardware units**: PC register, +4 adder, PCSel mux, IMEM

ID Instruction Decode:

- Generate control signals from the instruction bits, generate the immediate, and read registers from the RegFile.
- **Hardware units**: RegFile, ImmGen

EX Execute:

- Perform ALU operations, and do branch comparison.
- **Hardware units**: ASel mux, BSel mux, branch comparator, ALU

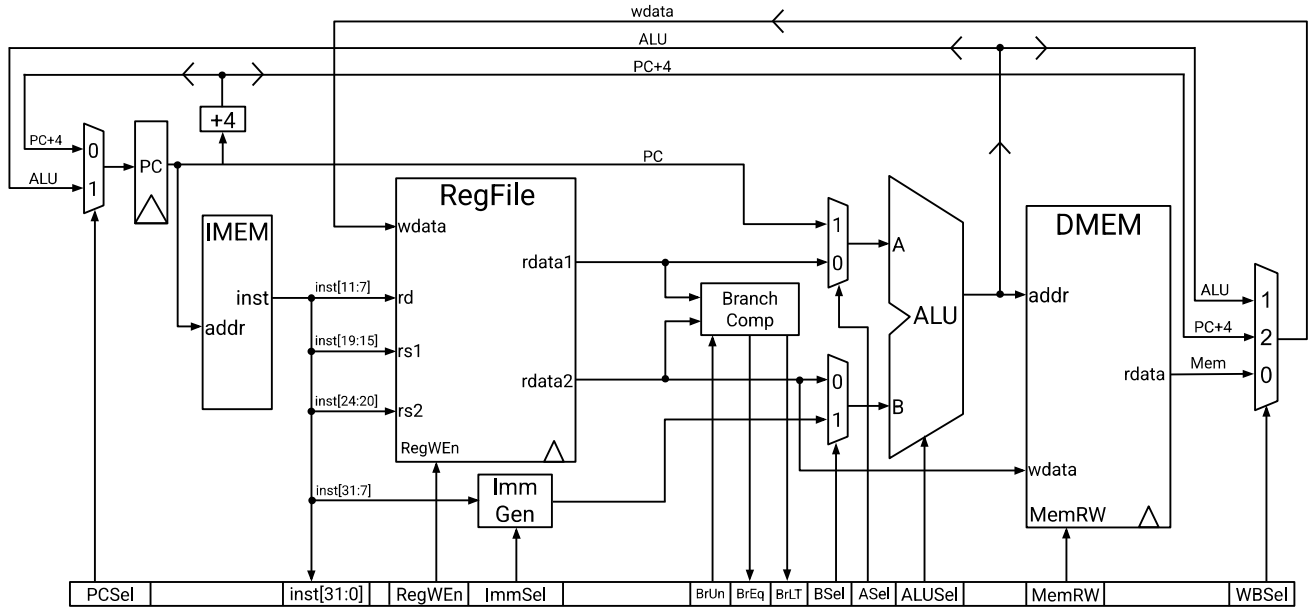
MEM Memory

- Read from or write to the data memory (DMEM).
- **Hardware units**: DMEM

WB Writeback

- Write back either PC + 4, the result of the ALU operation, or data from memory to the RegFile.
- **Hardware units**: WBSel mux, RegFile

Single-Cycle Datapath Diagram



5-Stage Datapath Diagram