Week 4

Let's Build a Computer!

Aditya (@nebu)

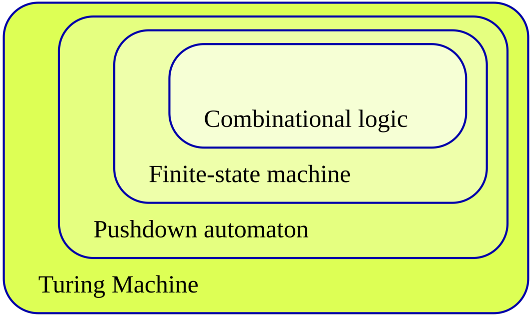
Outline

- Combinational Logic
- Feedback and FSMs
- Building a Computer

Why Are We Doing This?

- Understanding automata theory means understanding, in part, what the things in this figure mean:

Automata theory



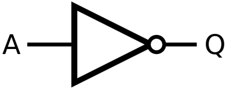
Why Are We Doing This?

- Understanding automata theory means understanding, in part, what the things in the figure on the previous slide mean.
- Teaches an application of what we're learning.
- Shows how general and useful the ideas we're covering are. (@Hassam compilers soon? 👀)
- Is very rewarding since computers are everywhere.

Section 1

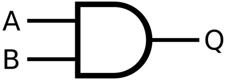
Combinational Logic

NOT Gate



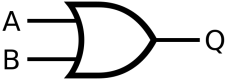
A	$Q = \overline{A}$
0	1
1	0

AND Gate



A	B	$Q = AB$
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate



A	B	$Q = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

That’s It!*

- AND, OR, and NOT are functionally complete.
- This means we can turn any “Boolean” function into one that’s made up of **only** ANDs, ORs, and NOTs.

Constructive Proof

A	B	C	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Constructive Proof

A	B	C	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$Q = \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}\,\overline{C} + ABC$$

This works because we “look for” all combinations of A, B, and C that’ll make Q high. If any of those combinations are high, Q is high.

Questions?

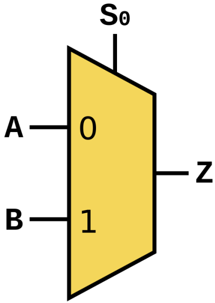
Questions!

Write F as an expression of A, B, and S.

S	A	B	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Solution

$$F = \overline{S}A + SB$$



Takeaways Before We Move On

- We can now map any input bits to output bits.
- You can build any Boolean function using only AND, OR, and NOT.
- Using this knowledge, you can build adders, multipliers, decoders, priority MUXes...whatever you want really.

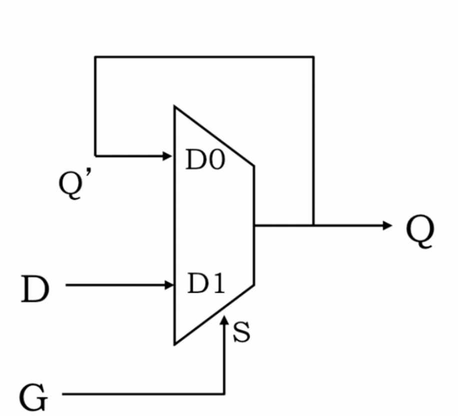
Section 2

Feedback and FSMs

A Fly in the Soup

- When I said you could build anything you wanted, what did I miss?
- **Storage.**
- Any ideas?

Consider This: No Longer Combinational



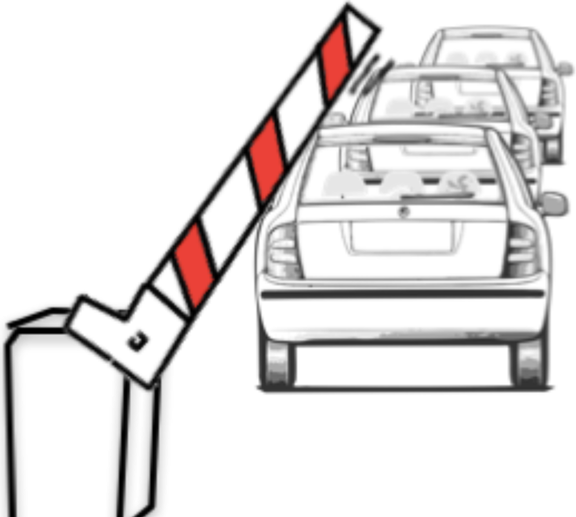
That’s a Latch

It’s rarely used as a memory element — can you guess why?

Latches are level triggered.

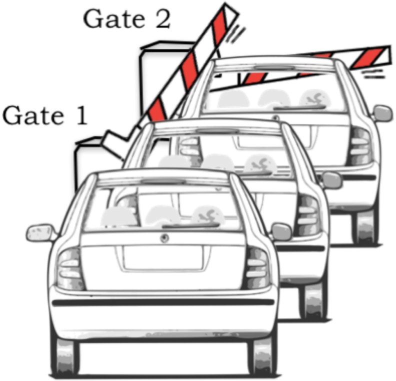
To Explain This, Let’s Talk About Cars

Timing is Key, Or Else...

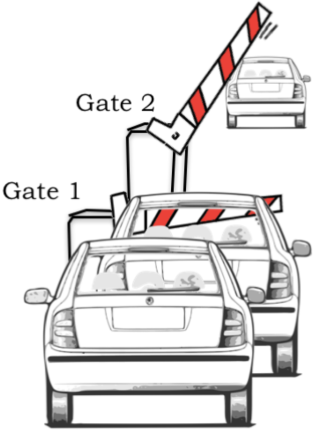


Any ideas?

Use Two Barriers!



Gate 1: open
Gate 2: closed

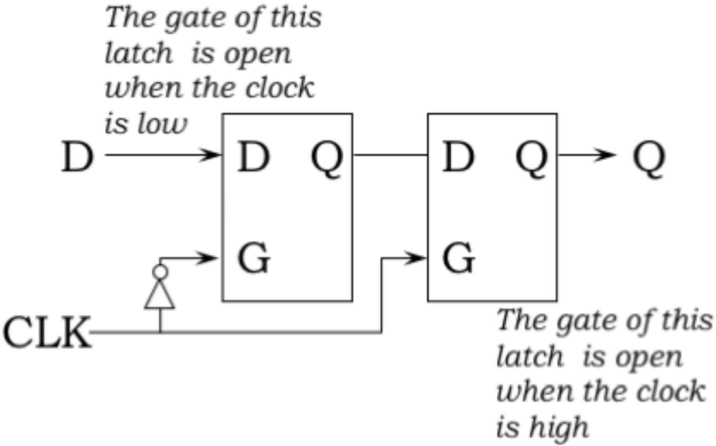


Gate 1: closed
Gate 2: open

How Does This Help?

- Timing is now easy — we just need to push one switch to swap the states of the barriers.
- Actually making two barriers in hardware isn't very tricky...

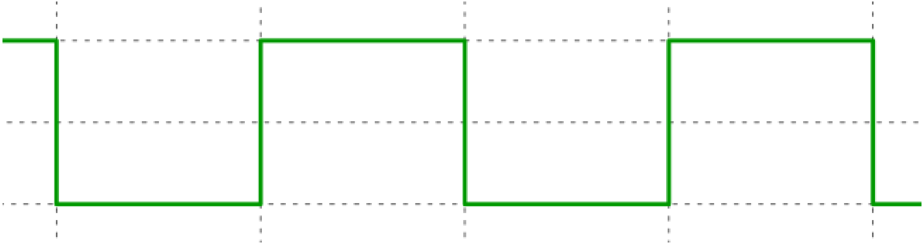
D Flip Flop/ D Register...Finally a Nice Storage Element!



Who Flips the Switch?

What we really need is a **low to high transition**, to load data correctly into our flip flop.

We use an alternating signal, called a clock, to give us these transitions at regular intervals.



Synchronous Digital Logic

A synchronous digital circuit is made up of flip flops/latches and combinational logic, all run by a single clock.

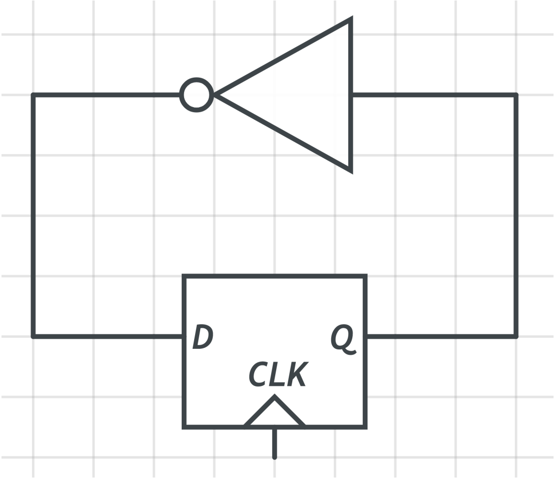
Questions?

Questions!

You're given a D flip flop, a NOT gate, and a clock signal. Make a circuit that, on every 0 to 1 transition of the clock, inverts the value stored in the flip flop. You may assume the flip flop is initialized with a valid value.

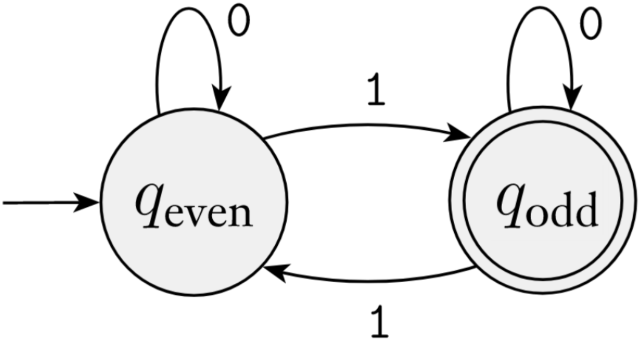
Hint: Think about when the flip flop loads data, and what data it should load.

Solution



We Can Now Build FSMs!

Recall that a DFA is simply something like:

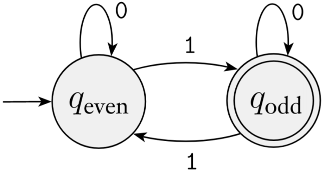


Let's build something similar[1] using our synchronous logic.

---

[1] Formally, a finite state transducer.

What is an FSM, *Really*



- An input alphabet: $\{0, 1\}$
- A bunch of states: $\{q_{even}, q_{odd}\}$
- An initial state: q_{even}
- A state transition function:

State	Input	Next State
q_{even}	0	q_{even}
q_{even}	1	q_{odd}
q_{odd}	0	q_{odd}
q_{odd}	1	q_{even}

Now, Let’s Make it in Hardware

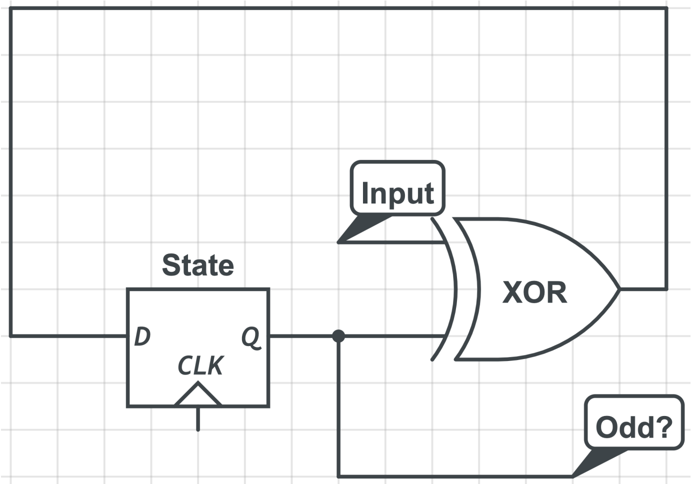
- An input alphabet: $\{0,\ 1\}$
- A bunch of states: $\{q_{even},\ q_{odd}\}$
- An initial state: q_{even}
- A state transition function:

State	Input	Next State
q_{even}	0	q_{even}
q_{even}	1	q_{odd}
q_{odd}	0	q_{odd}
q_{odd}	1	q_{even}

- An input alphabet: $\{0,\ 1\}$
- A bunch of states: use a flip flop, $(q_{\text{even}}, q_{\text{odd}}) = (0,\ 1)$.
- An initial state: initialize the flip flop to 0.
- A state transition function:

State	Input	Next State
0	0	0
0	1	1
1	0	1
1	1	0

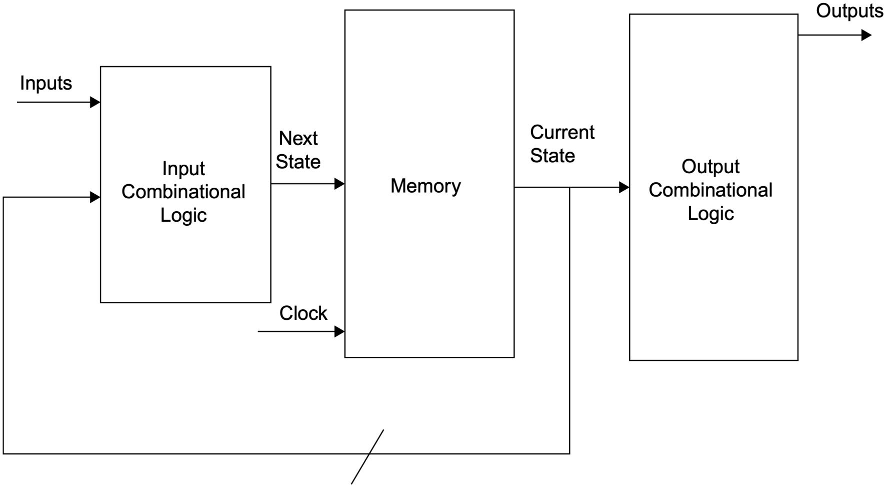
Our Final Hardware FSM Is



Hardware FSMs Also Have Outputs

- An output alphabet
- An output function that maps states to outputs

Generalized (Moore) FSM Architecture



Questions?

Questions!

Design in hardware an FSM that detects non-overlapping sequences of the string 101. (Your input alphabet is $\{0, 1\}$.)

Section 3

Building a Computer

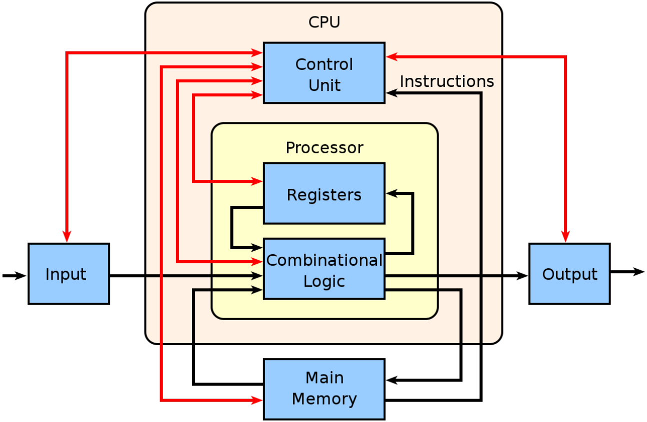
What is a Computer?

According to Wikipedia…

> *A computer is a digital electronic machine that can be programmed to carry out sequences of arithmetic or logical operations (computation) automatically.*

Surprisingly accurate description — of a CPU.

Von Neumann Architecture



A CPU Has

- **Memory**: We'll assume it's random access. The CPU can read/write values from here at will.
- **Registers**: a bunch of flip flops where it stores values that it's currently operating on.
- **Combinational logic** to "compute" things: adders, logical units, etc. whose inputs and outputs are the registers.
- **Instructions**: Stored in the memory, (logically) executed one after another.
- **Control Unit**: Orchestrates the whole thing.

The Control Unit Is

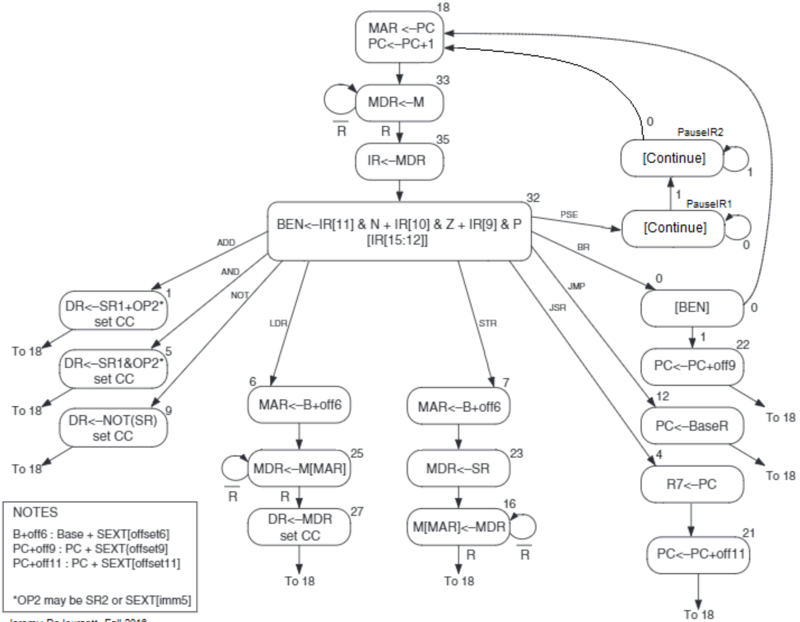
A giant FSM that does the following things:

- **Fetch**: Gets an “instruction” from memory.
- **Decode**: Figures out what to do according to the instruction.
- **Execute**: Actually do the instruction. Once it’s done, go decode the next instruction in memory — increment the program counter.

Instructions

What can the CPU do?

- `ADD`: Takes two registers, adds them, puts the sum in a register. Set condition codes.
- `AND`: Takes two registers, ANDs them, puts the result in a register. Set condition codes.
- `NOT`: Takes a register, NOTs it, puts the result in a register. Set condition codes.
- `BR`: Depending on "condition codes", move the program counter to the location specified.
- `JMP`: Move the program counter to the address specified.
- `LDR`: Load the contents of a memory address into a register.
- `STR`: Store the contents of a register into a memory address.

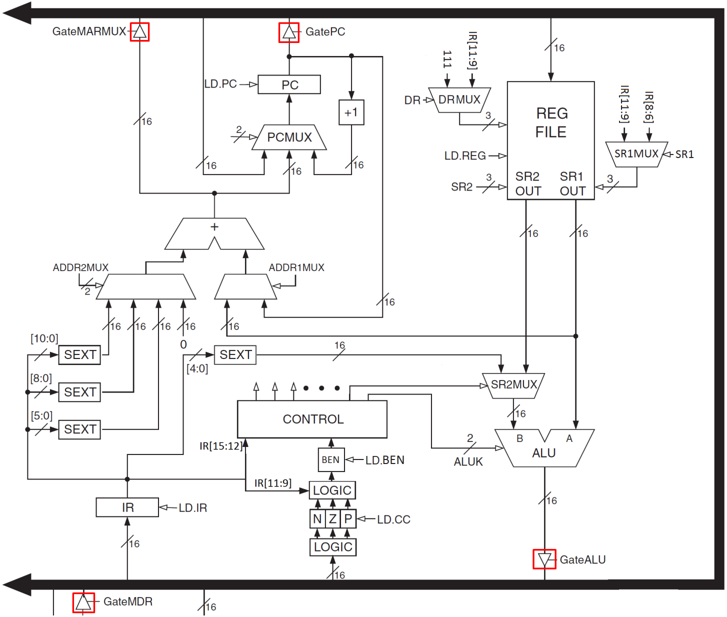


NOTES

B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT{offset9]
PC+off11 : PC + SEXT[offset11]

*OP2 may be SR2 or SEXT[imm5]

Jeremy DeJournett, Fall 2016


GateMARMUX
GatePC
LD.PC
PC
+1
PCMUX
2
16
111
IR[11:9]
DR
DRMUX
3
REG FILE
LD.REG
SR2
SR2 OUT
SR1 OUT
IR[11:9]
IR[8:6]
SR1MUX
SR1
+
ADDR2MUX
ADDR1MUX
0
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
SEXT
[4:0]
SR2MUX
CONTROL
B
A
ALU
ALUK
IR[15:12]
BEN
LD.BEN
IR[11:9]
LOGIC
N
Z
P
LD.CC
LOGIC
IR
LD.IR
GateALU
GateMDR

Questions?

42

> *The computer looked normal size for a black space-borne computer satellite — about a thousand miles across.*
>
> — DOUGLAS ADAMS (1979)