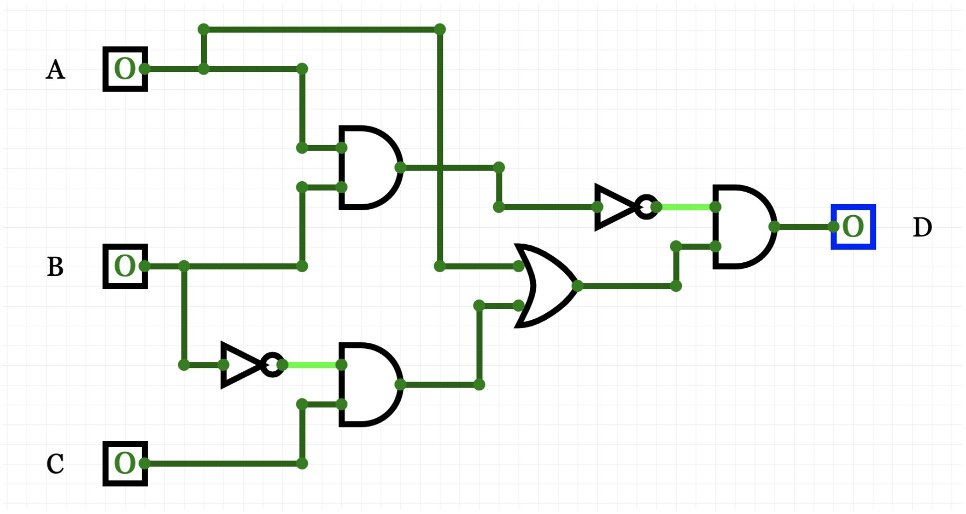
CSM 61C — Logic/FSMs/Datapath

Fall 2020 — Mentoring 5: October 12, 2020

1 Boolean Logic

1.1 Given the circuit diagram below, inputs A, B, and C, and output D, write down the truth table.



A	B	C	D
1	1	1	
1	1	0	
1	0	1	
1	0	0	
0	1	1	
0	1	0	
0	0	1	
0	0	0	

A	B	C	D
1	1	1	0
1	1	0	0
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	1
0	0	0	0

1.2 Convert the truth table above to a boolean logic expression.

Using the Sum of Products rule:

$$A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C$$

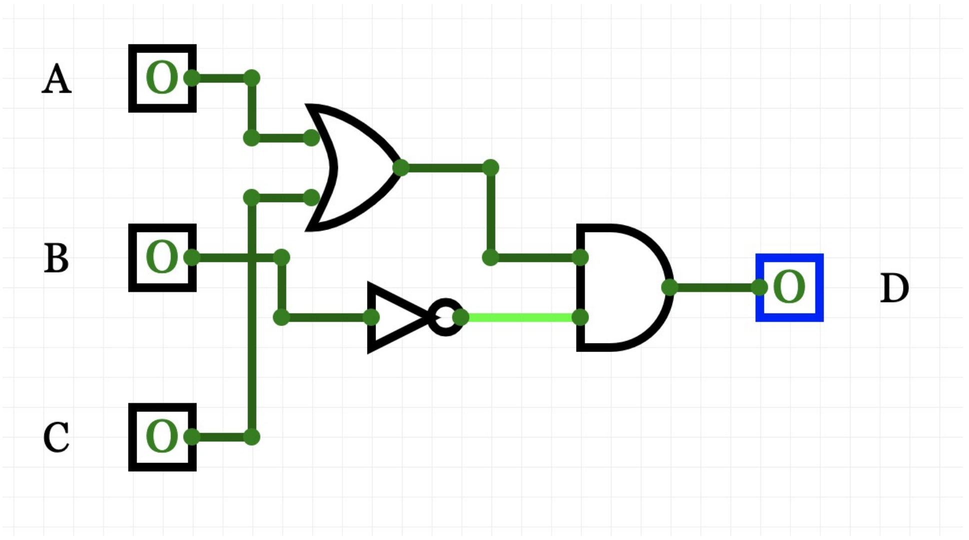
1.3 Simplify the above boolean expression.

$$\begin{aligned} A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C &= \bar{B}(AC + A\bar{C} + \bar{A}C) \\ &= \bar{B}(A\bar{C} + C(A + \bar{A})) \\ &= \bar{B}(A\bar{C} + C) \\ &= \bar{B}(A + C) \end{aligned} \tag{1}$$

See discussion 6, Q2.1 for one example of how to prove the final step. Alternatively:

$$\begin{aligned} A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C &= \bar{B}(AC + A\bar{C} + \bar{A}C) \\ &= \bar{B}(AC + A\bar{C} + AC + \bar{A}C) \\ &= \bar{B}(A(C + \bar{C}) + C(A + \bar{A})) \\ &= \bar{B}(A + C) \end{aligned} \tag{2}$$

1.4 Express the simplified boolean expression in circuit form.



2 Finite State Machines

Suppose we want to design a FSM that takes a single bit (1/0) as input, and outputs a single bit (1/0). We want the FSM to output true (1) only if it has seen three consecutive 1's as its input. Let's design it!

2.1 Given the following input streams, fill in what the FSM should output at each time step:

(a)

In	0	1	0	1
Out				

In	0	1	0	1
Out	0	0	0	0

The FSM has only seen individual 1's (non-consecutive), so it should output only zeros.

(b)

In	1	1	0	0
Out				

In	1	1	0	0
Out	0	0	0	0

The FSM has only seen two consecutive 1's, but not three. It should still output only zeros.

(c)

In	0	1	1	1
Out				

In	0	1	1	1
Out	0	0	0	1

The FSM has finally seen three consecutive 1's, but it should only output 1 (true) after it has seen the third 1.

2.2 Let's consider the design of the FSM with more formality.

(a) If a 0 is input into the FSM, what should the FSM output?

The FSM should always output 0. We only care about outputting 1 when the input is 1.

(b) If a 1 is input into the FSM, what does the FSM need to remember to make the correct decision?

The FSM needs to remember how many 1's were input before the current 1. More concretely, it needs to know whether no 1's, one 1, or two 1's were previously inputted.

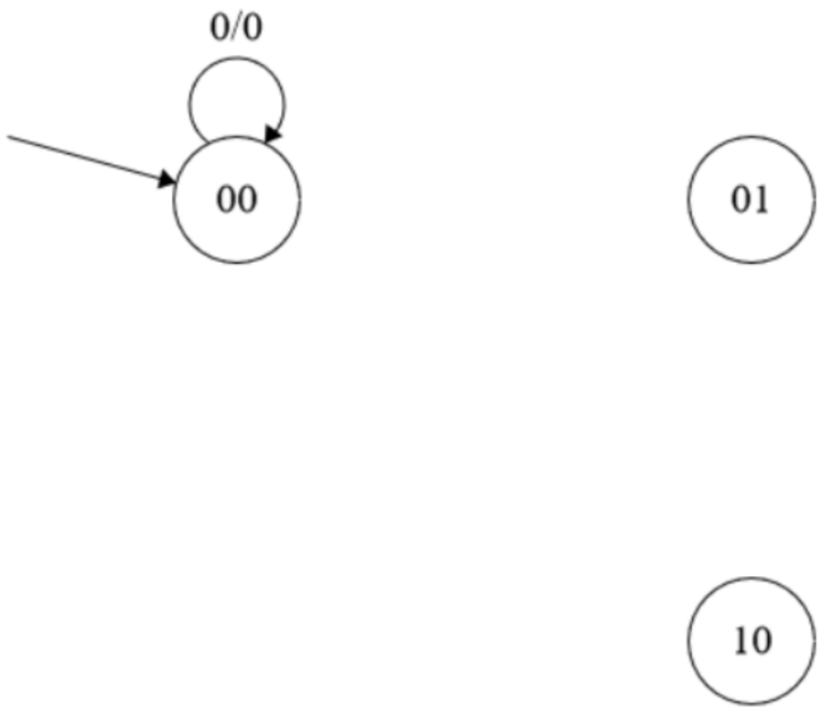
(c) How many unique states does the FSM need?

Building off of part b), we need 3 states in total for the following purposes:

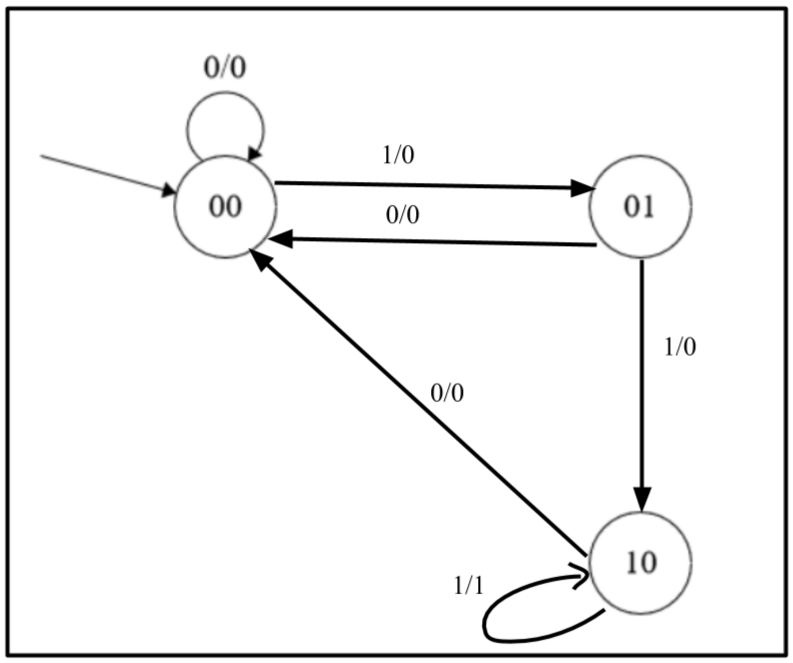
- Start/Reset: A starting point for the machine as well as a reset if we ever see 0
- One 1: We've seen one 1 so far.
- Two 1's: We've seen two 1's so far. This needs to be distinguishable from only seeing one 1.

Note that three consecutive 1's is indistinguishable from seeing four, five, six, etc. consecutive 1's.

2.3 Fill in the FSM.

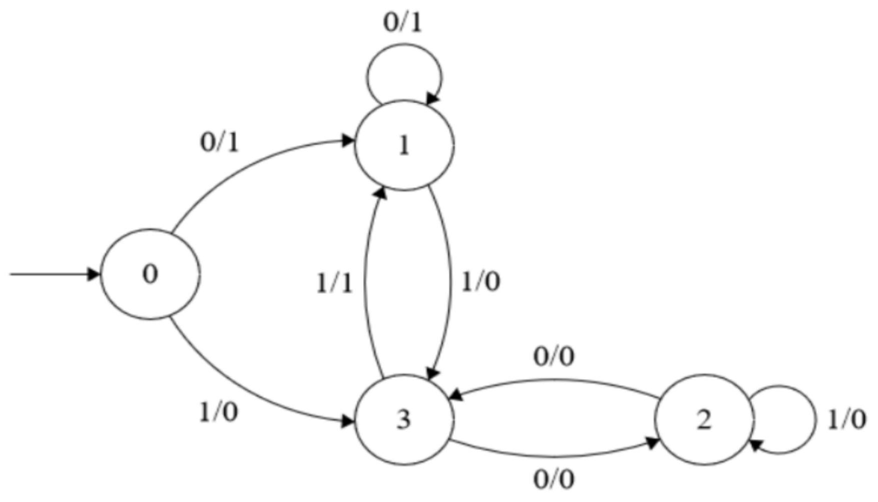


Solution:



Meta: The states can be named in decimal, not necessarily binary. Meta 2: Please excuse the poorly drawn self-loops. No (simple) drawing application likes them apparently.

2.4 Consider the following FSM. What does it do?



Meta: This will likely need more scaffolding. The FSM outputs a 1 whenever the input is a multiple of 3. Some ways to approach analyzing:

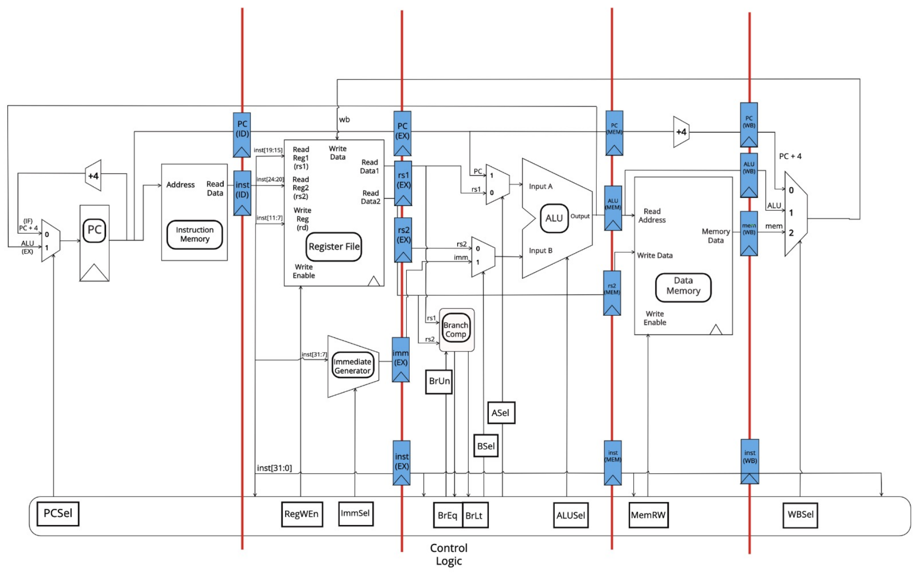
- 1 is a less common output than 0 → what causes a 1 to be output?
- Trace along the paths → what causes a 0 to be output?
  - Consider 0, 0, 1 → 0b001 = 1 is not a multiple of 3
  - Consider 1, 1 → 0b11 = 3 is a multiple of 3
  - Consider 1, 1, 1 → 0b111 = 7 is not a multiple of 3
- Brute-force: draw out the truth table

3 Single-Cycle Datapath and Control

3.1 **5 Stages of a Single Cycle CPU:**

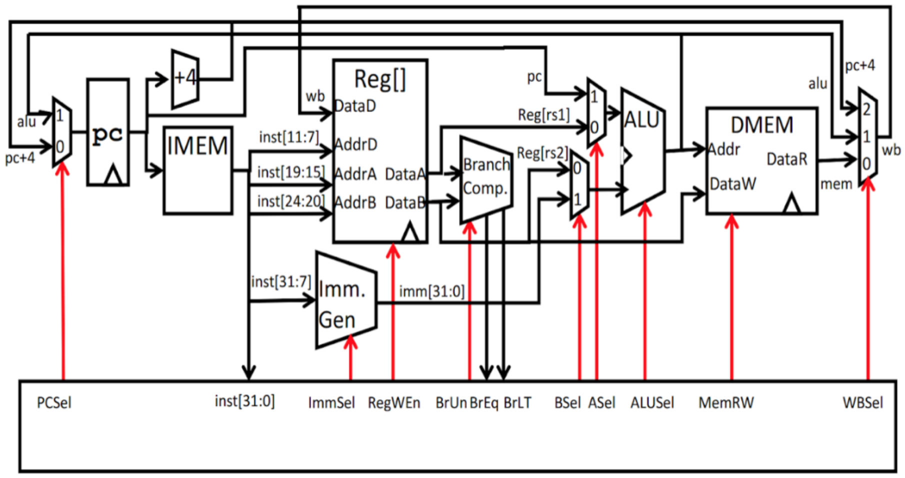
- Instruction Fetch (IF) - Fetch from memory (IMEM)
- Instruction Decode (ID) - Decode instruction
- Execute (EX) - Execute operation (arithmetic, shifting, etc) using ALU
- Memory Access (MEM) - Load and store instructions access memory
- Write Back to Register (WB) - Write instruction back to RegFile

Datapaths: A Visual Approach



3.2 Control Logic

A controller send signals to our circuit, telling which pieces to perform what operations. Not all control signals matter for every instruction: for example, R-type instructions ignores the output from the immediate generator. Control signals are used to pick between mux inputs in order to perform the correct operation. They are embedded within the actual machine code for an instruction.



Control Inputs

Signal:	**inst[31:0]**	**BrEq**	**BrLT**
Purpose:	Sends the current instruction to control	(DataA == DataB) ? 1 : 0	(DataA < DataB) ? 1 : 0

Control Outputs

Signal:	**Purpose:**
PCSel	Next instruction location
ALUSel	What operation to perform.
RegWEn	Do we allow the register value to be updated by enabling writes?
ImmSel	Format the immediate properly.
MemRW	Read or write to mem.
WBSel	What value to write back.
BrUn	Branch signed or unsigned.
ASel/BSel	Pick between the inputs for ALU.

4 Game of Signals

4.1 Fill out the control signals for the following instructions (put an X if the signal does not matter). For ImmSel, write the corresponding instruction type.

Instr	BrEq	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	X	0	X	X	0	0	Add	0	1	1
lw										
bge										
sw										
auipc										
jal										

Instr	BrEq	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	X	0	X	X	0	0	Add	0	1	1
lw	X	0	I	X	0	1	Add	0	1	0
bge	0/1	0/1	SB	0	1	1	Add	0	0	X
sw	X	0	S	X	0	1	Add	1	0	X
auipc	X	0	U	X	1	1	Add	0	1	1
jal	X	1	UJ	X	1	1	Add	0	1	2

4.2 We want to expand our instruction set from the base RISC-V ISA (RV32I) to support some new instructions. You can find the canonical single-cycle datapath above. For the proposed instruction below, choose ONE of the options below.

1. Can be implemented without changing datapath wiring, only changes in control signals are needed. (i.e. change existing control signals to recognize the new instruction)
2. Can be implemented, but needs changes in datapath wiring, only additional wiring, logical gates and muxes are needed.
3. Can be implemented, but needs change in datapath wiring, and additional arithmetic units are needed (e.g. comparators, adders, shifters etc.).
4. Cannot be implemented.

(Note that the options from 1 to 3 gradually add complexity; thus, selecting 2 implies that 1 is not sufficient. You should select the option that changes the datapath the least (e.g. do not select 3 if 2 is sufficient). You can assume that necessary changes in the control signals will be made if the datapath wiring is changed.)

(a) Allowing software to deal with 2's complement is very prone to error. Instead, we want to implement the negate instruction, `neg rd rs1`, which puts `-R[rs1]` in `R[rd]`.

A. This is a tricky question! Notice `neg` doesn't use all available bits, so we could make `neg rd, rs1` into a special R-type instruction `neg rd, x0, rs1` such that the instruction does `R[rd] = x0 - R[rs1]`. Notice that subtraction is supported by our default datapath. So, we only need to add the new control signal neg which will produce the same `ALUsel`, `Asel`, `Bsel`, ... signals a `sub` does.

(b) Sometimes, it is necessary to allow a program to self-destruct. Implement `segfault rs2, offset(rs1)`. This instruction compares the value in `R[rs2]` and the value in `MEM[R[rs1]+offset]`. If the two values are equal, write `0` into the `PC`; otherwise treat this instruction as a `NOP`.

C. Need to 1) Add a comparator after memory unit and wire the output to `PCSel`. 2) Add a zero wired to `mux` before `PC`. 3) Change corresponding `PCSel` signal width.