Clocked Circuits

Dr. Charles R. Severance

www.ca4e.com

online.dr-chuck.com

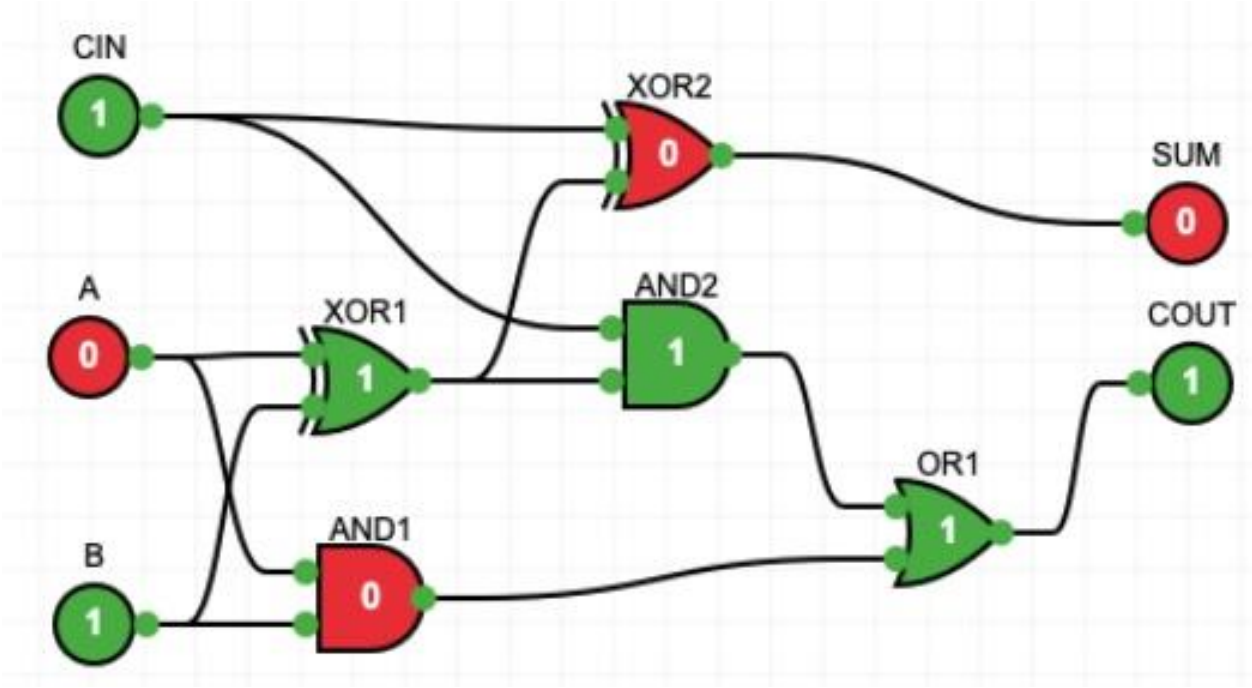
Outline

- Speed / Delay / Clocks
- Programming with gates – machine code (0's and 1's)
- Inside of a CPU

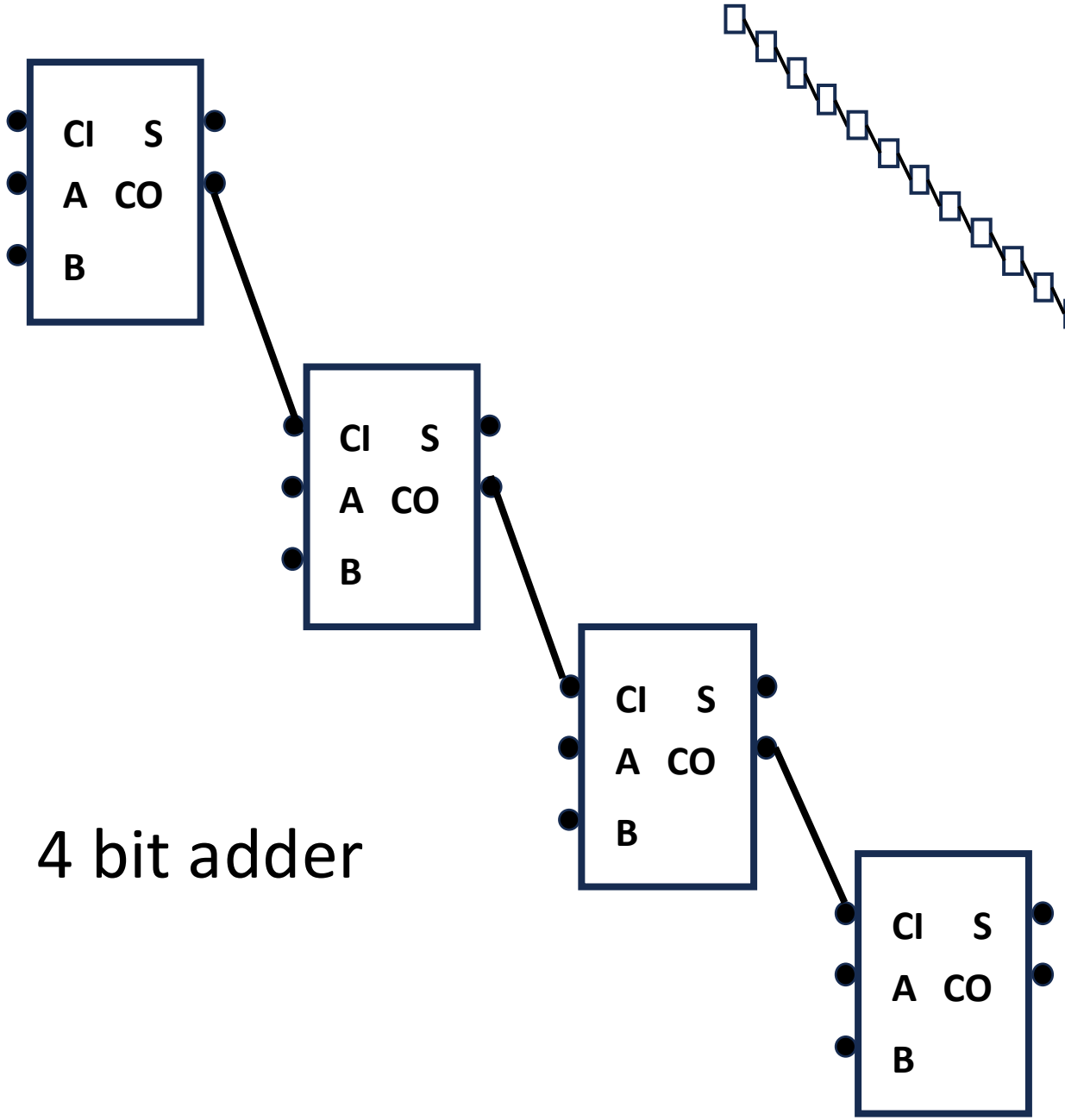
Delay

- The "longer" a circuit is, the longer it takes for the outputs to "settle" to their final values

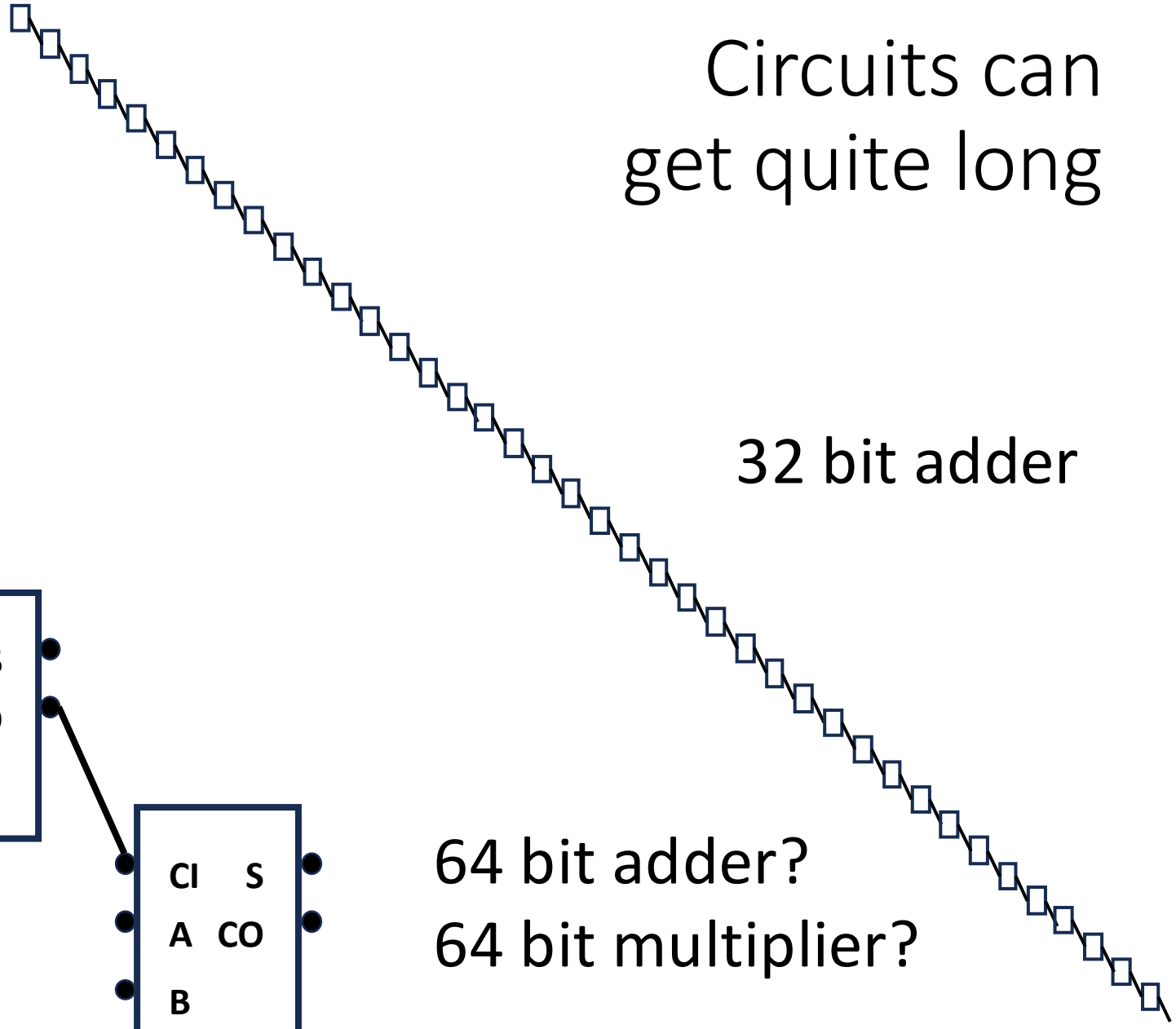
Full Adder



Circuits can get quite long



4 bit adder

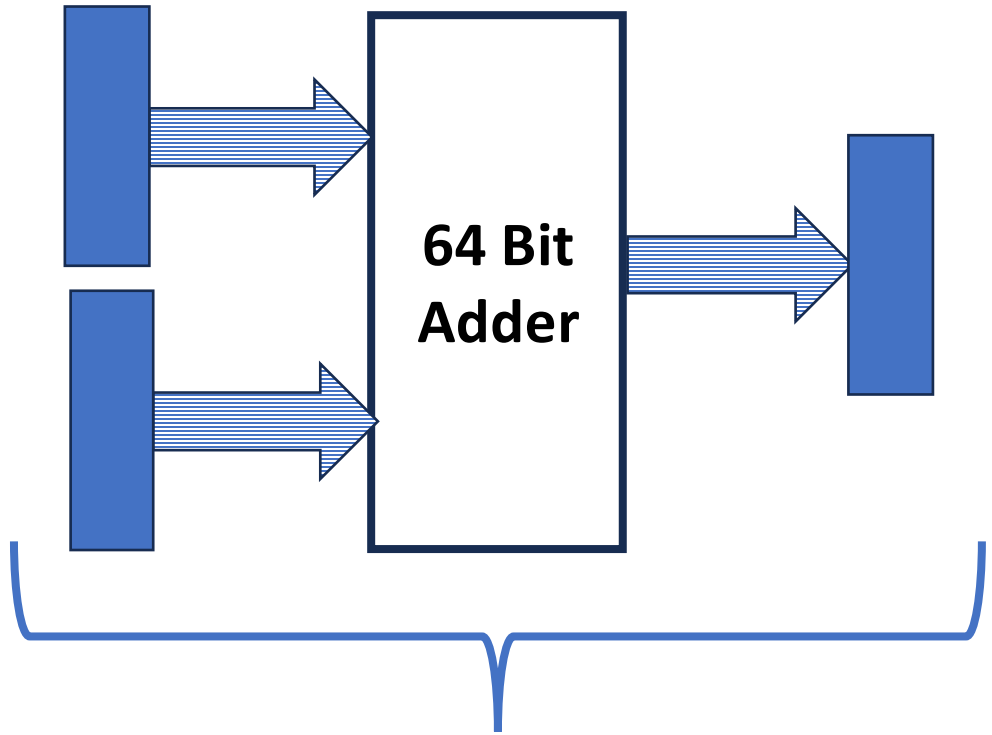


32 bit adder

64 bit adder?

64 bit multiplier?

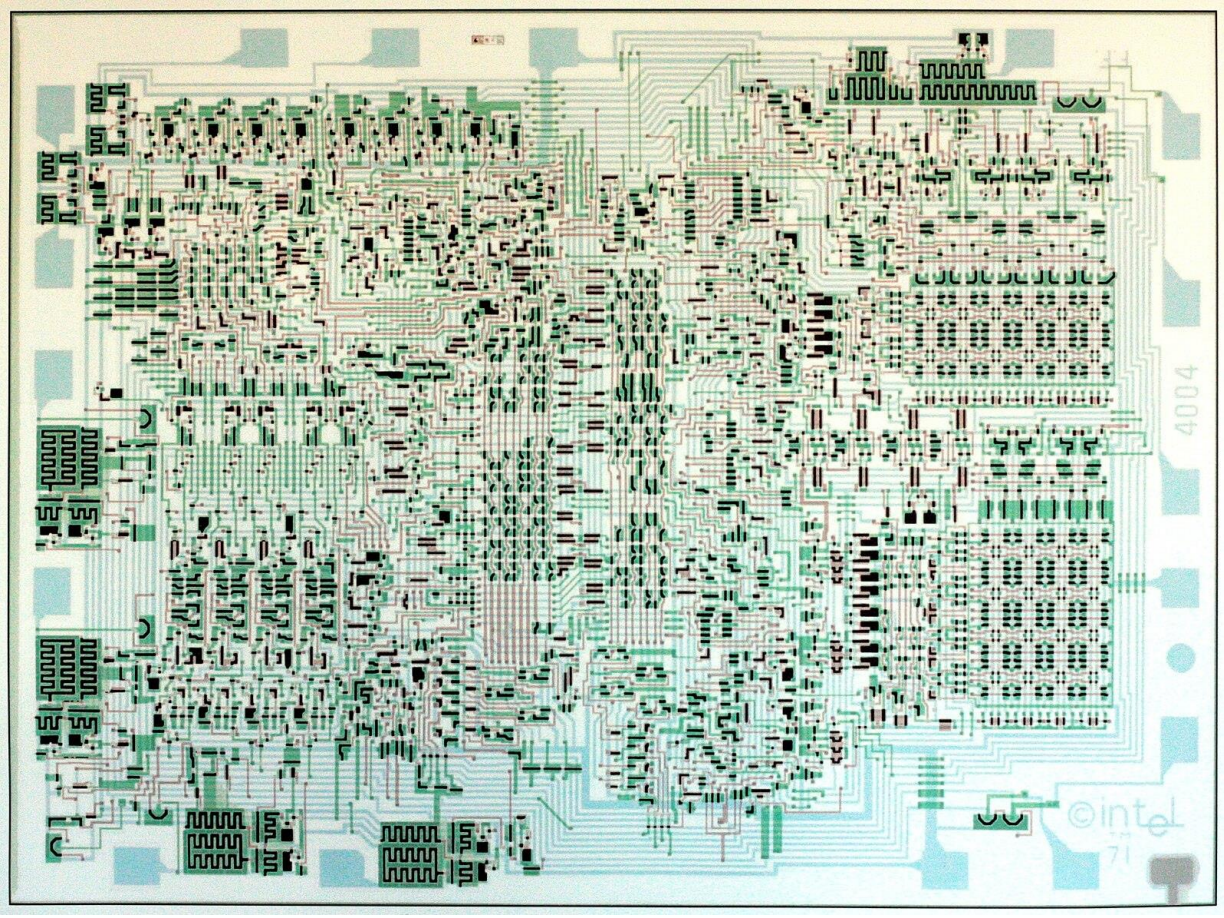
The "Length" of a circuit



Once a circuit is designed and laid out, circuit simulators can compute the time it takes for signals to propagate through the circuit. We can not trust at the results to be accurate until sufficient time has passed.

Clock Rate

- Once we compute the delay of the slowest element of the Arithmetic and Logic parts of the CPU (aka the ALU), we can determine the number of times per second we can reliably do computations.



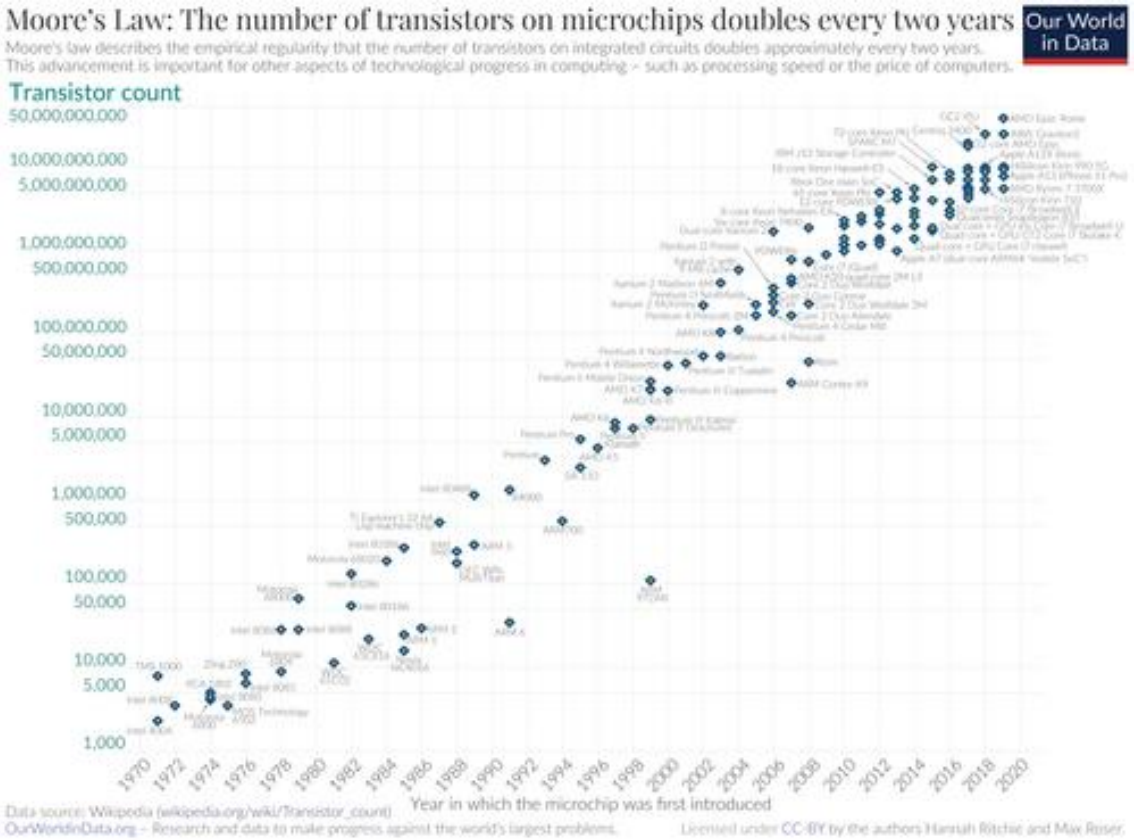
What Determines Clock Rate?

- The size of the chip (speed of light)
- The number of transistors in the longest path within the ALU
- The physical length of the longest path within the ALU (speed of light)
- The speed at which each transistor switches between 0 and 1 (depends on the technology used to build the chip)

- Chips are designed, redesigned, laid out, and simulated endlessly during development to produce the fastest clock rate

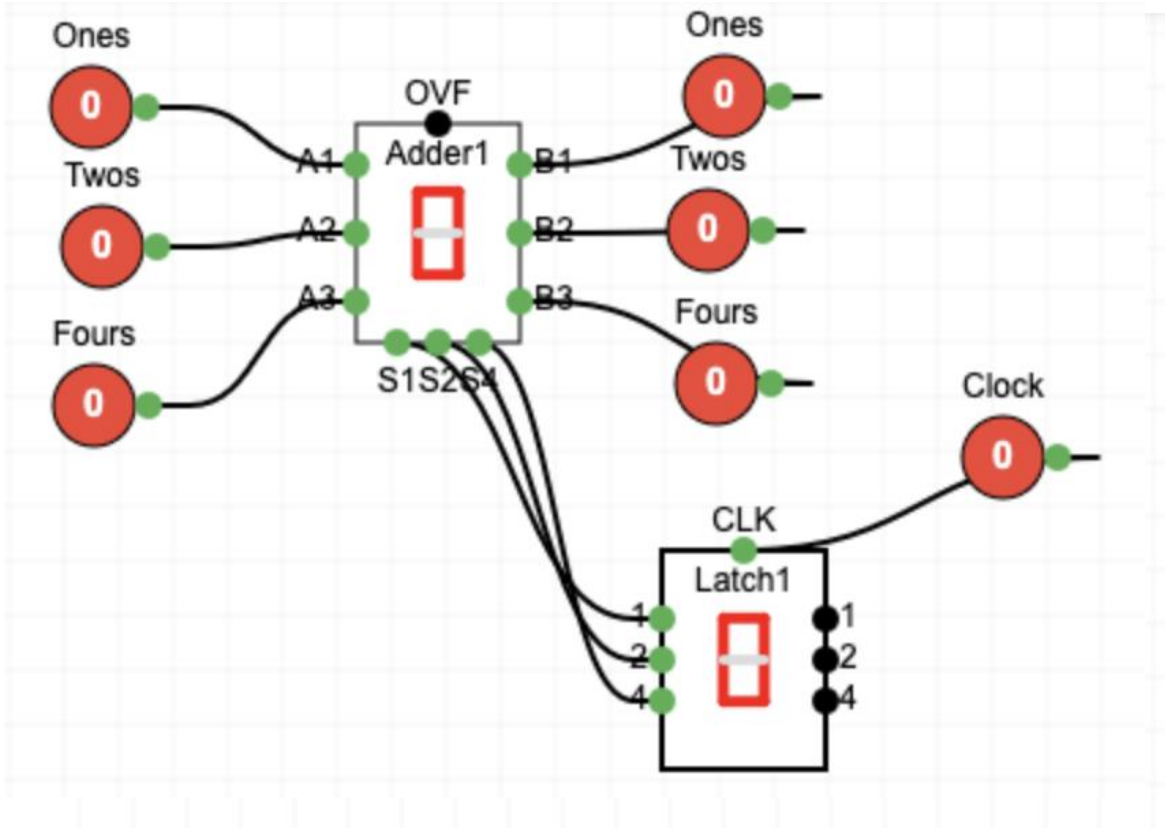
Moore's Law

- **Moore's law** is the observation that the number of transistors in an integrated circuit (IC) doubles about every two years.
- X-Axis 1970 - 2020
- Y-Axis 1000 - 50 billion



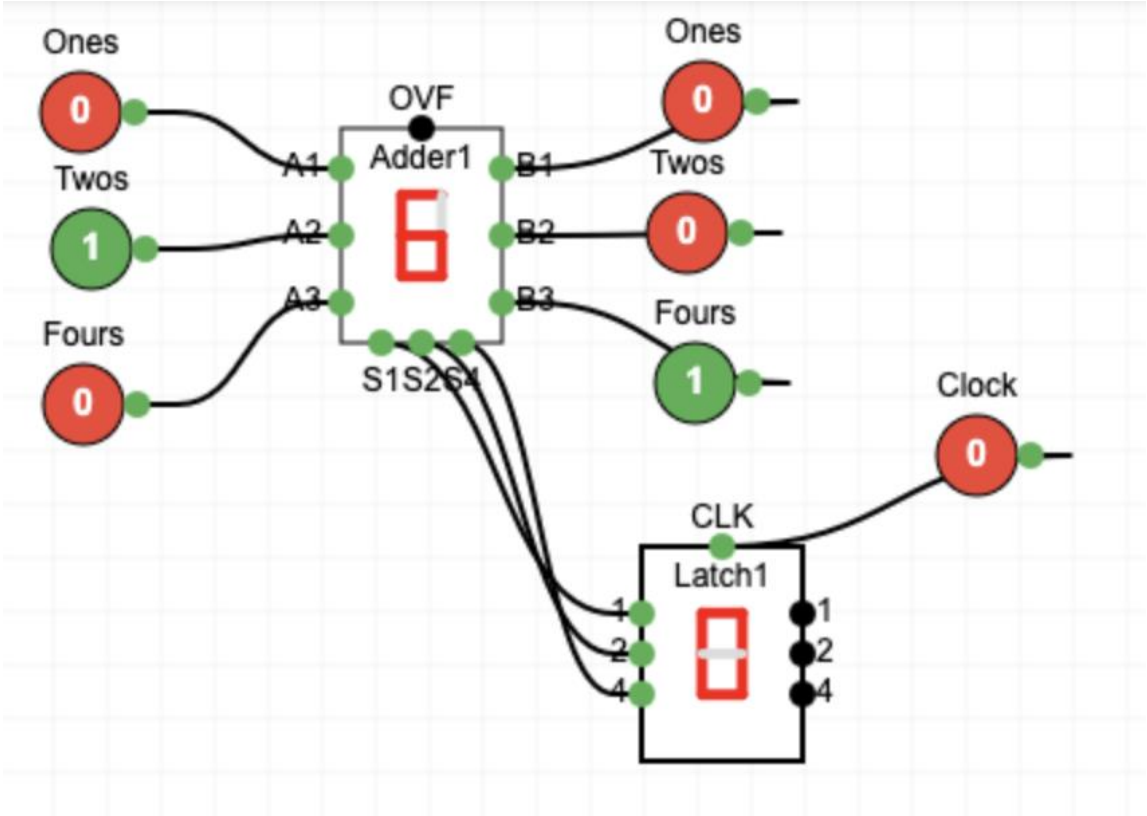
An Adder, Clock, and Register, Oh My!

- The adder and its inputs is a "combinational circuit"
- The latch is a "clocked circuit", ignoring its inputs while the clock is low, then "latching" the inputs in once the clock goes high
- When the clock is low input changes are ignored



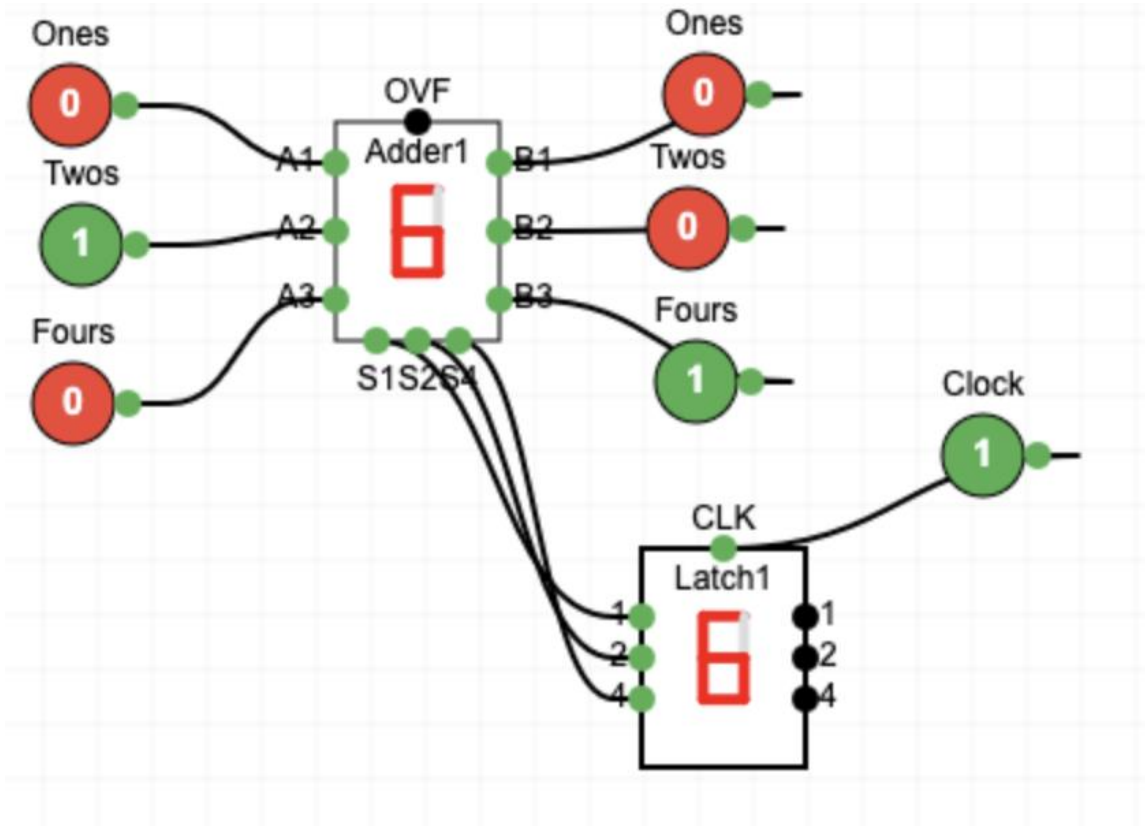
An Adder, Clock, and Register (Step 1)

- At this point, we have changed the adder input values.
- The adder is adding the values and producing the sum reacting to whatever we do with a small delay.
- The latch is ignoring all changes of the sum values



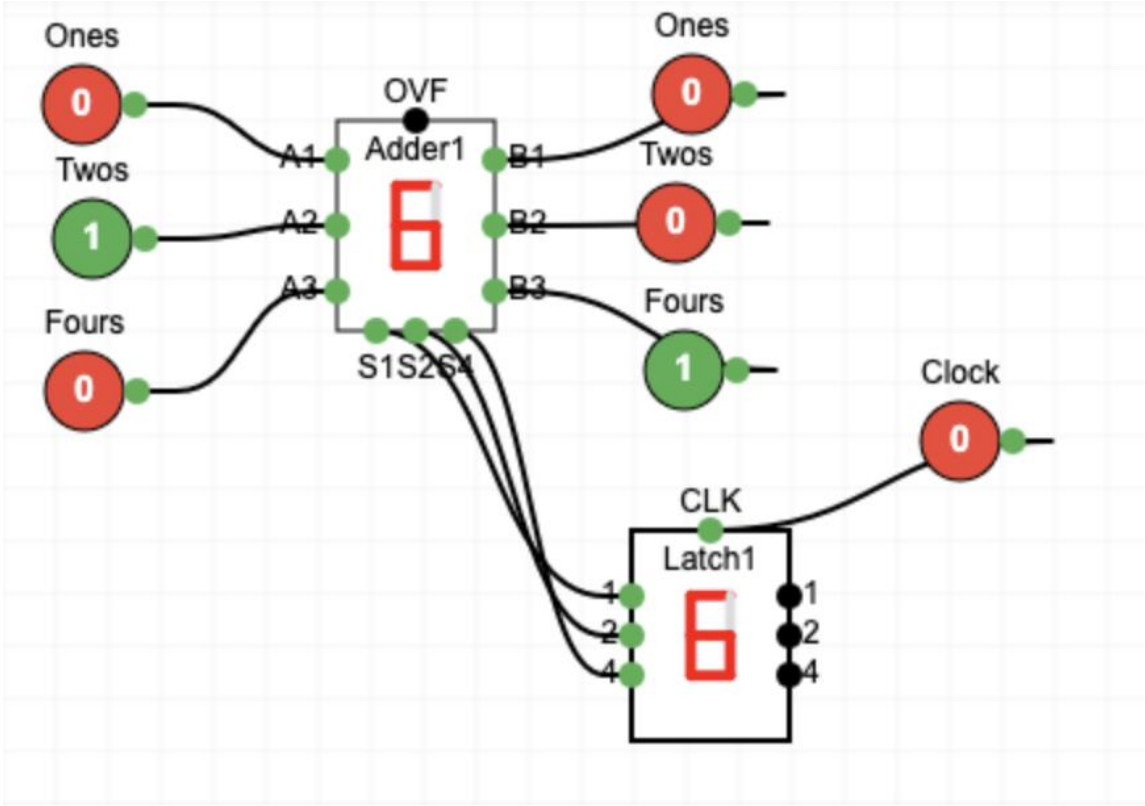
An Adder, Clock, and Register (Step 2)

- Once the clock is set to 1, the sum values are "latched" into the latch's internal memory.
- At this point if we change inputs the latch may or may not get new sum values depending on the delay of the adder



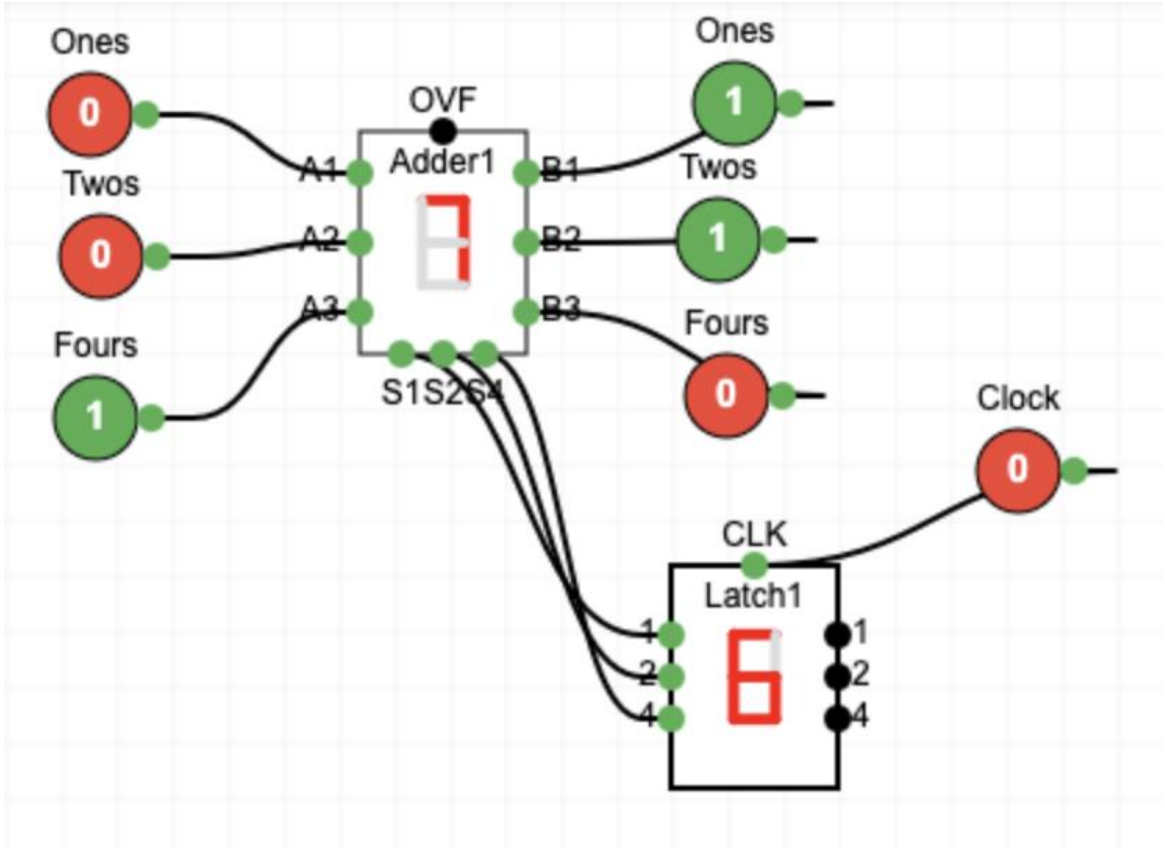
An Adder, Clock, and Register (Step 3)

- So once we copy the sum inputs into the latch, the clock goes back down
- We can once again change the adder inputs without affecting the latch since the clock is low.



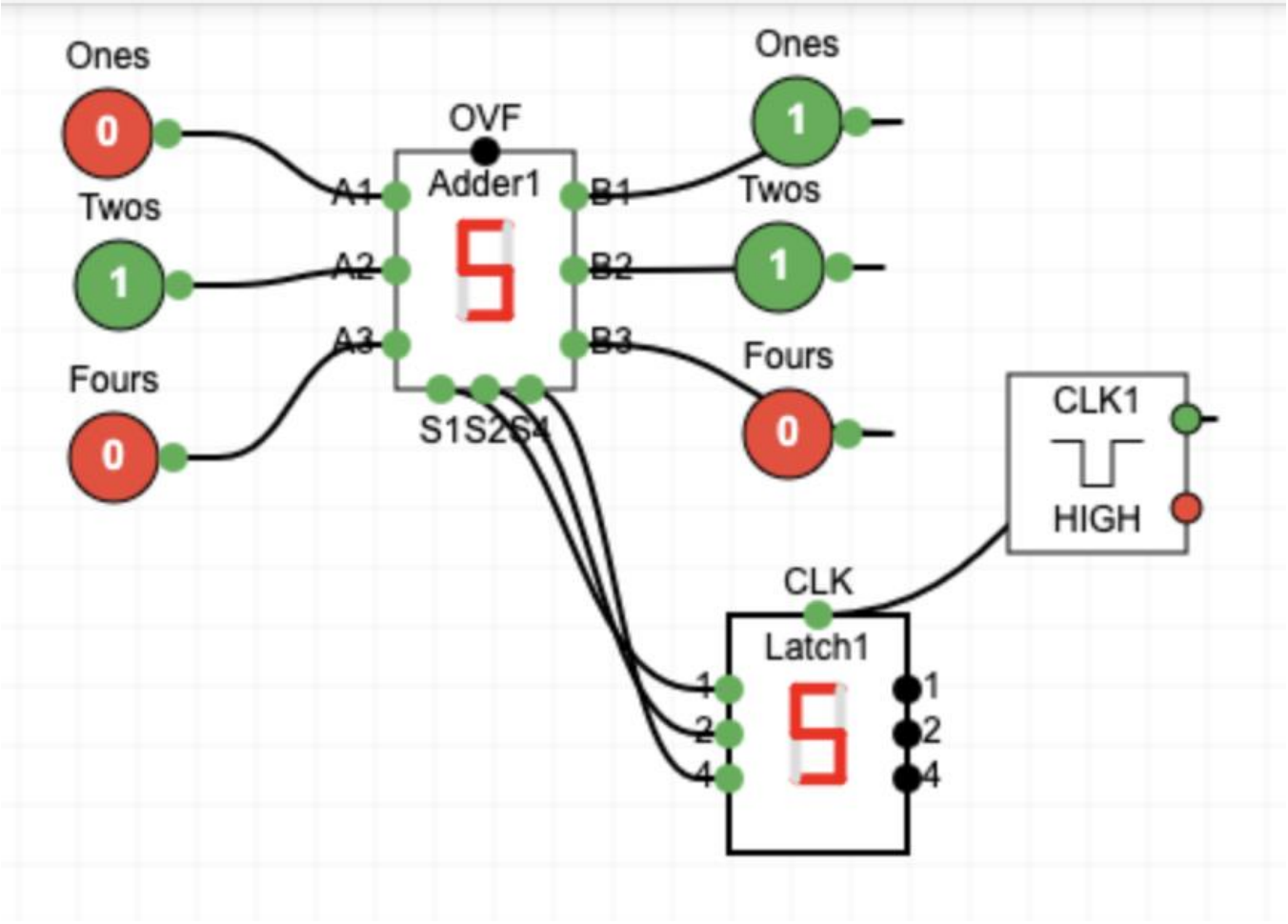
An Adder, Clock, and Register (Step 4)

- As we change the inputs to the adder, its sum outputs change to 7 (with a short delay) but the latch retains its stored value of 6 since the clock is low.
- If the clock goes high the 7 would be copied in as the new latch stored value



For a good time…

- Replace the clock button with an automatic 1 hertz clock (toggles once per second)
- Play with the adder inputs and watch as the clock causes them to be copied into the latch.
- See how many inputs you can change in one second
- You can start and stop the clock by clicking on it ☺



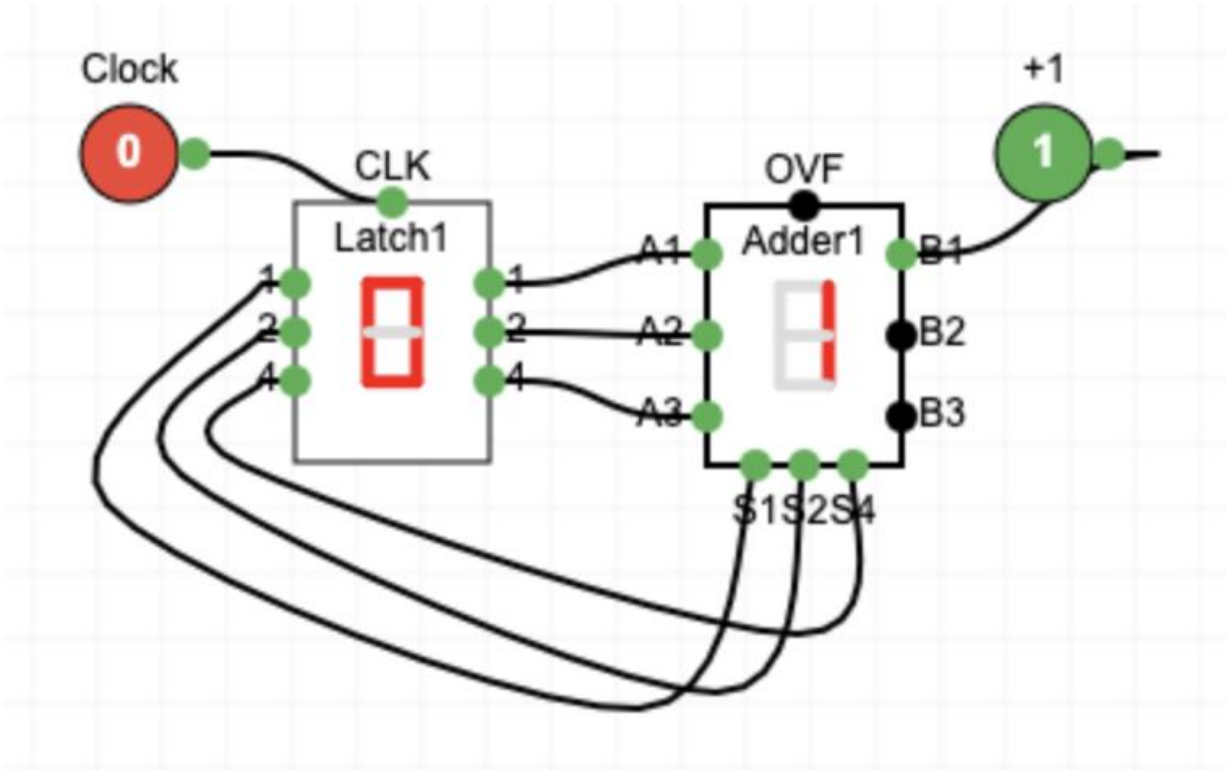
Counting using Gates

Let's make a counter

- If we take an adder, clock, and register, we can make a counting unit by adding 1 to the register each clock cycle.

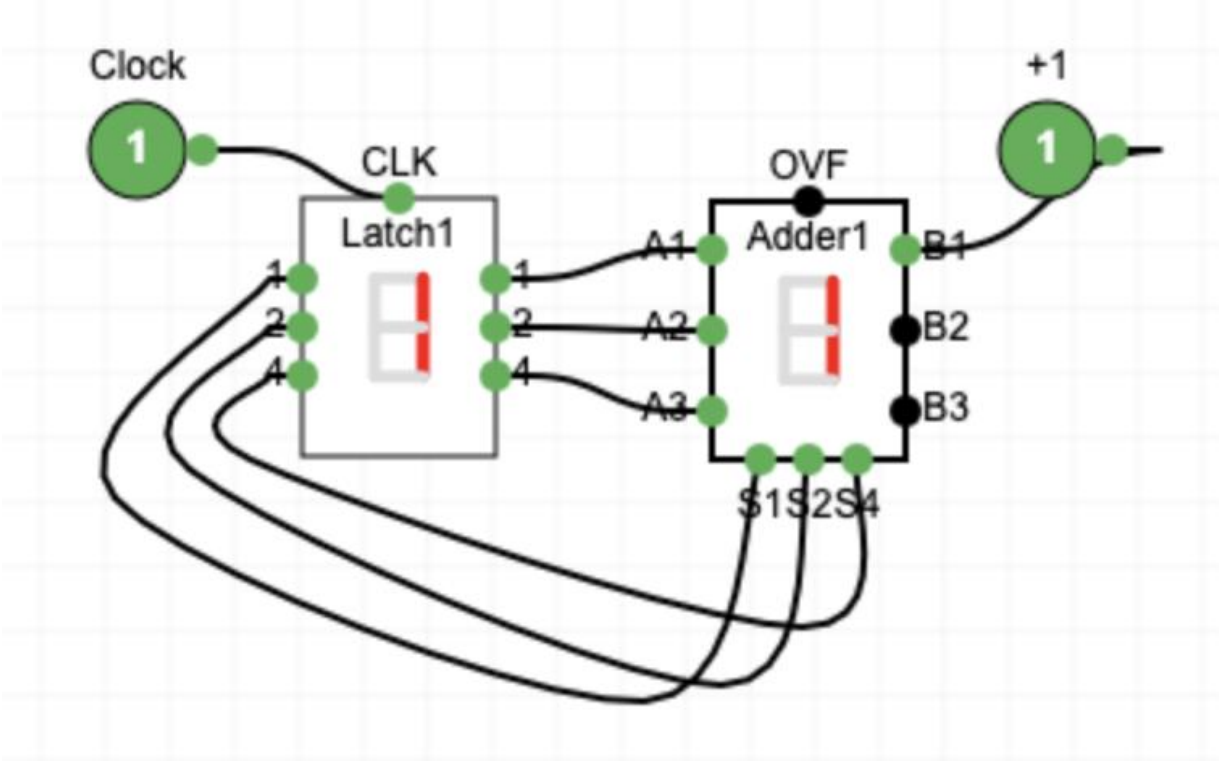
Counter Circuit (Step 1)

- If we take an adder, clock, and register, we can make a counting unit by adding 1 to the register each clock cycle.
- With the clock low and latch stored value of zero, the adder is continuously computing 0+1 but the latch is ignoring its inputs



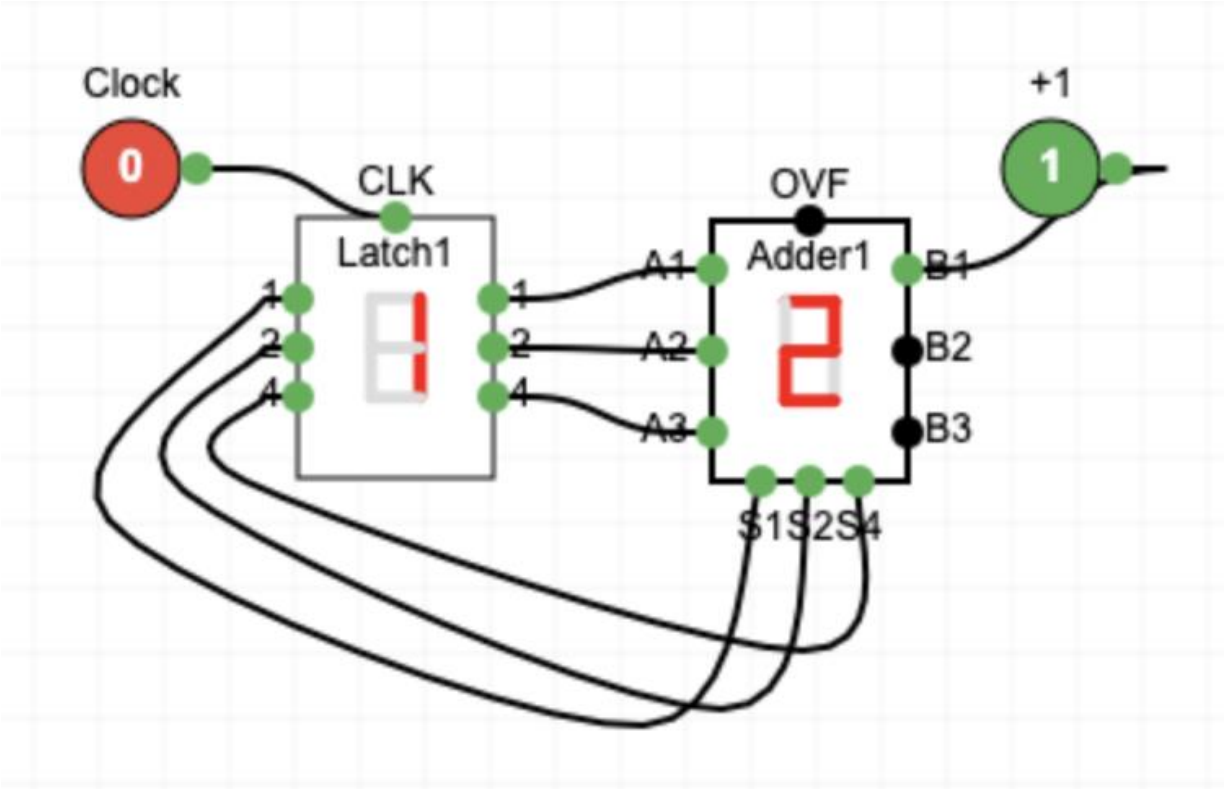
Counter Circuit (Step 2)

- When the clock goes high, the latch stores its inputs, but does not copy the stored data to the latch output until the clock goes low
- So, the adder is still computing 0+1 (i.e. not 1+1)



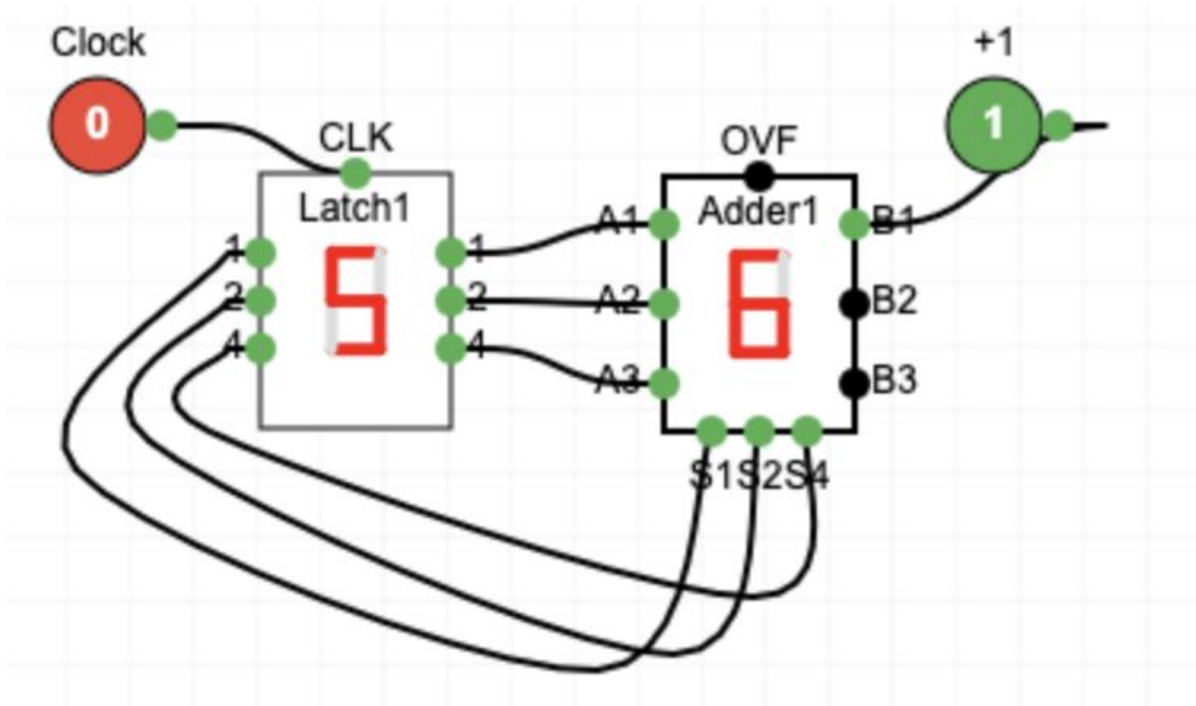
Counter Circuit (Step 3)

- When the clock goes low, the stored latch data is presented to the adder which computes 1+1=2 (after a short delay)



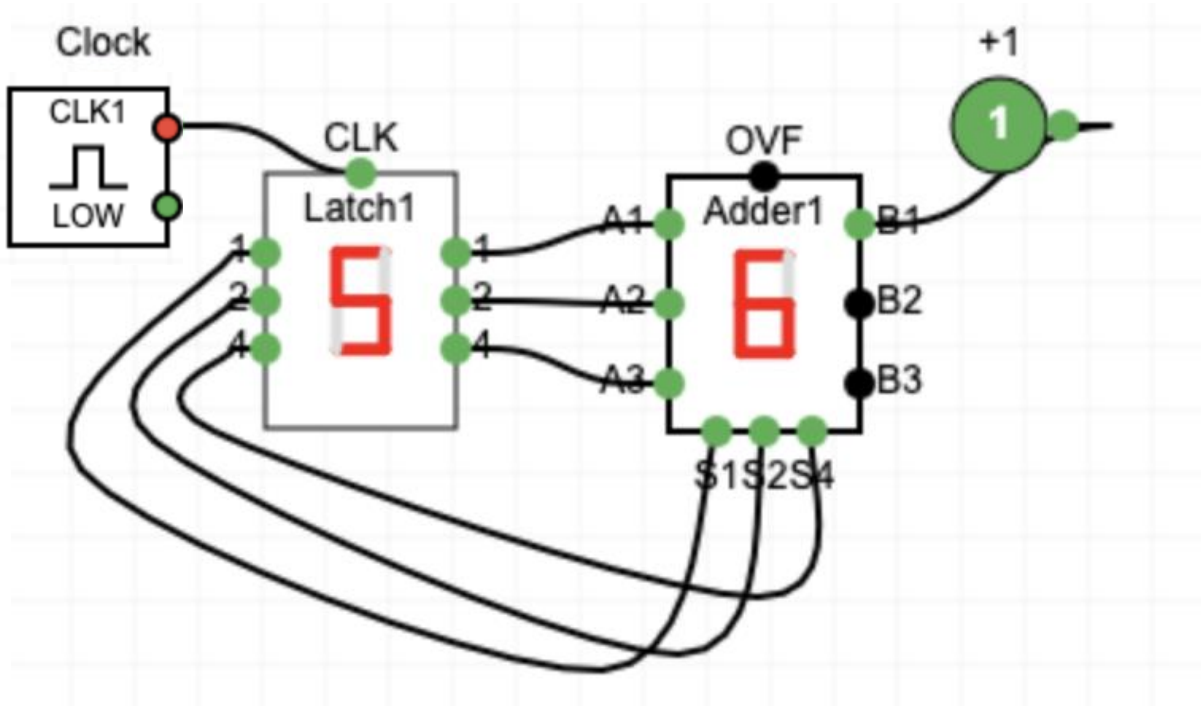
Counter Circuit (Step 4)

- If you keep toggling the clock the counter will count up to 7, overflow and restart the count at 0 again



We made a digital clock ☺

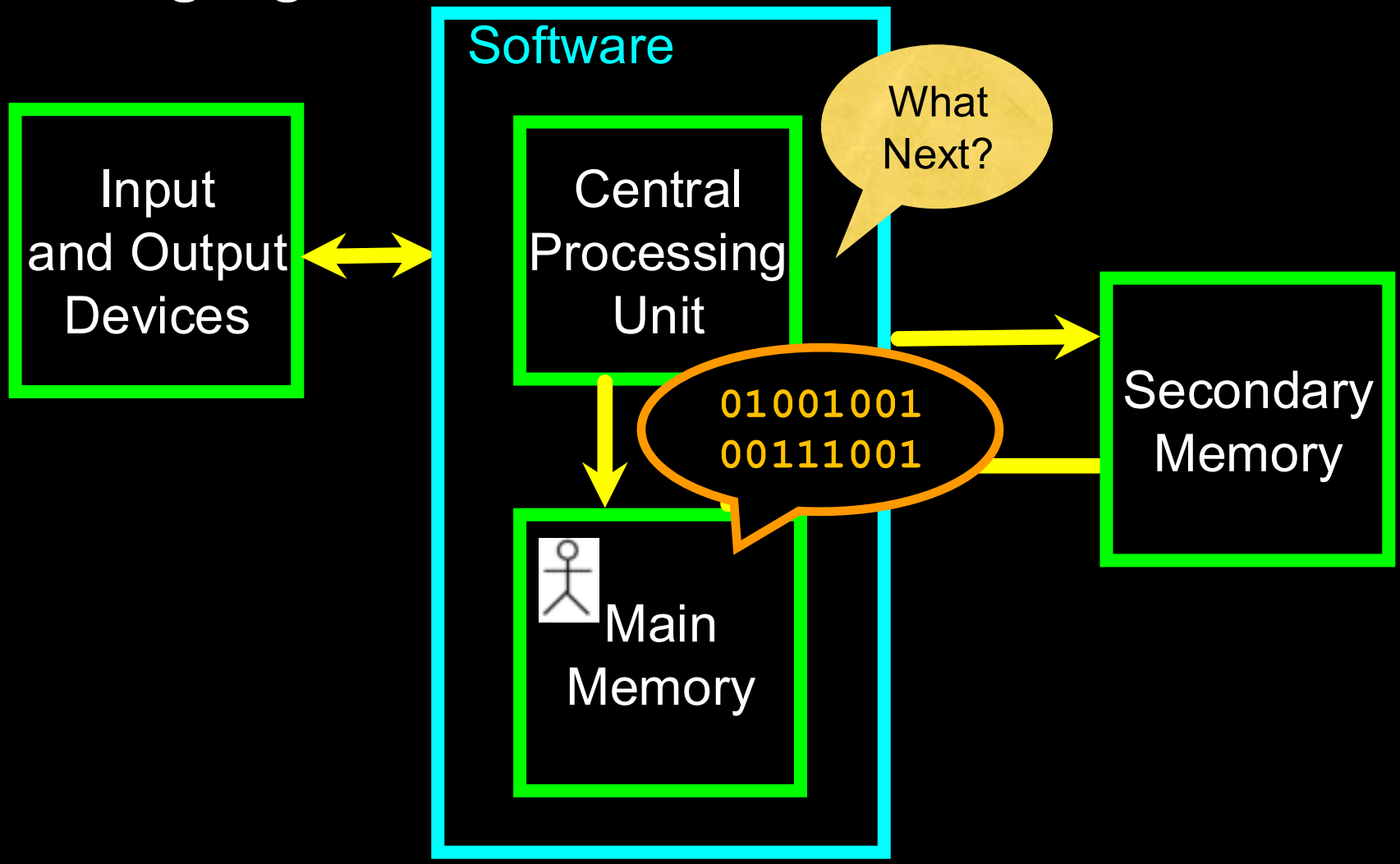
- Replace the clock button with an automatic 1 hertz clock (toggles once per second)
- You just made a simple seven-second clock!



Programming with gates

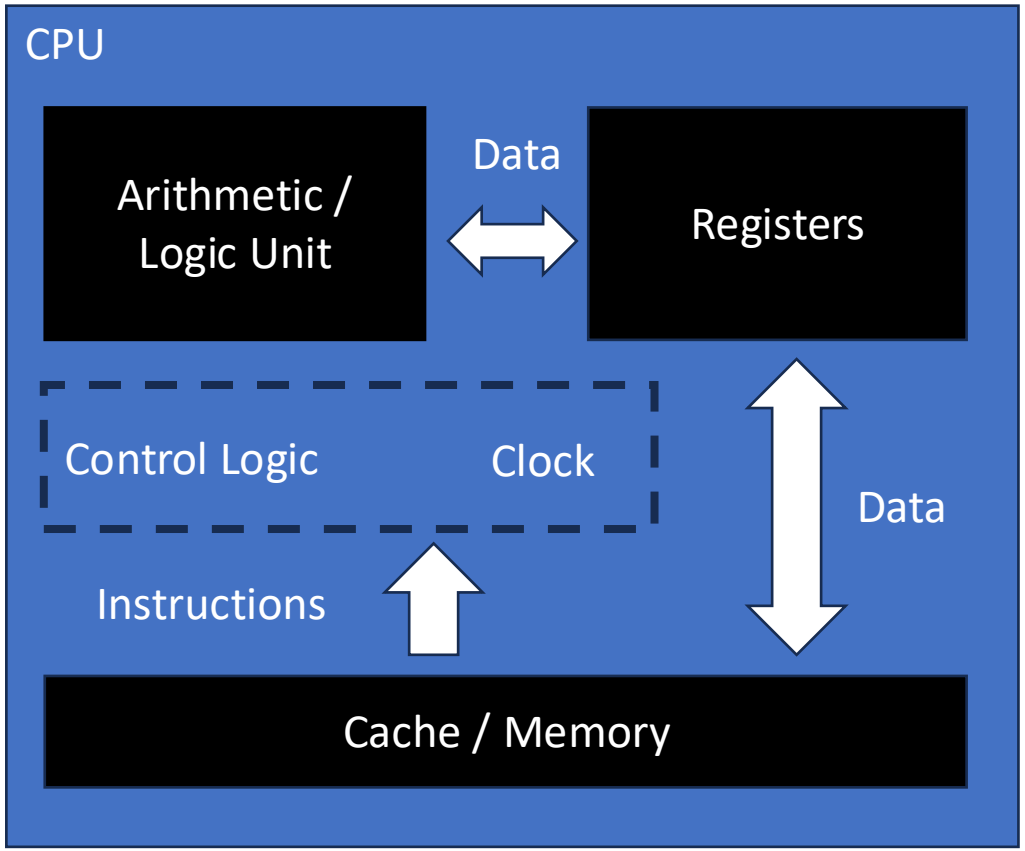
Machine code programming in (0's and 1's)

Machine Language



Inside a CPU

- The ALU is circuits like adders
- Registers are latches
- Control logic and clock connect everything together
- Instructions are taken from memory, decoded by the control logic which instructs the registers and ALU on what to do using wires, circuits and gates



Foreshadowing..

- Soon, we will design a simple CPU
  - CDC 8512 – Inspired by the Control Data Corporation 6500
- It will have a complete architecture
- It will have a completely defined machine language
- We will write real programs in CDC 8512 machine code and run them in an emulator

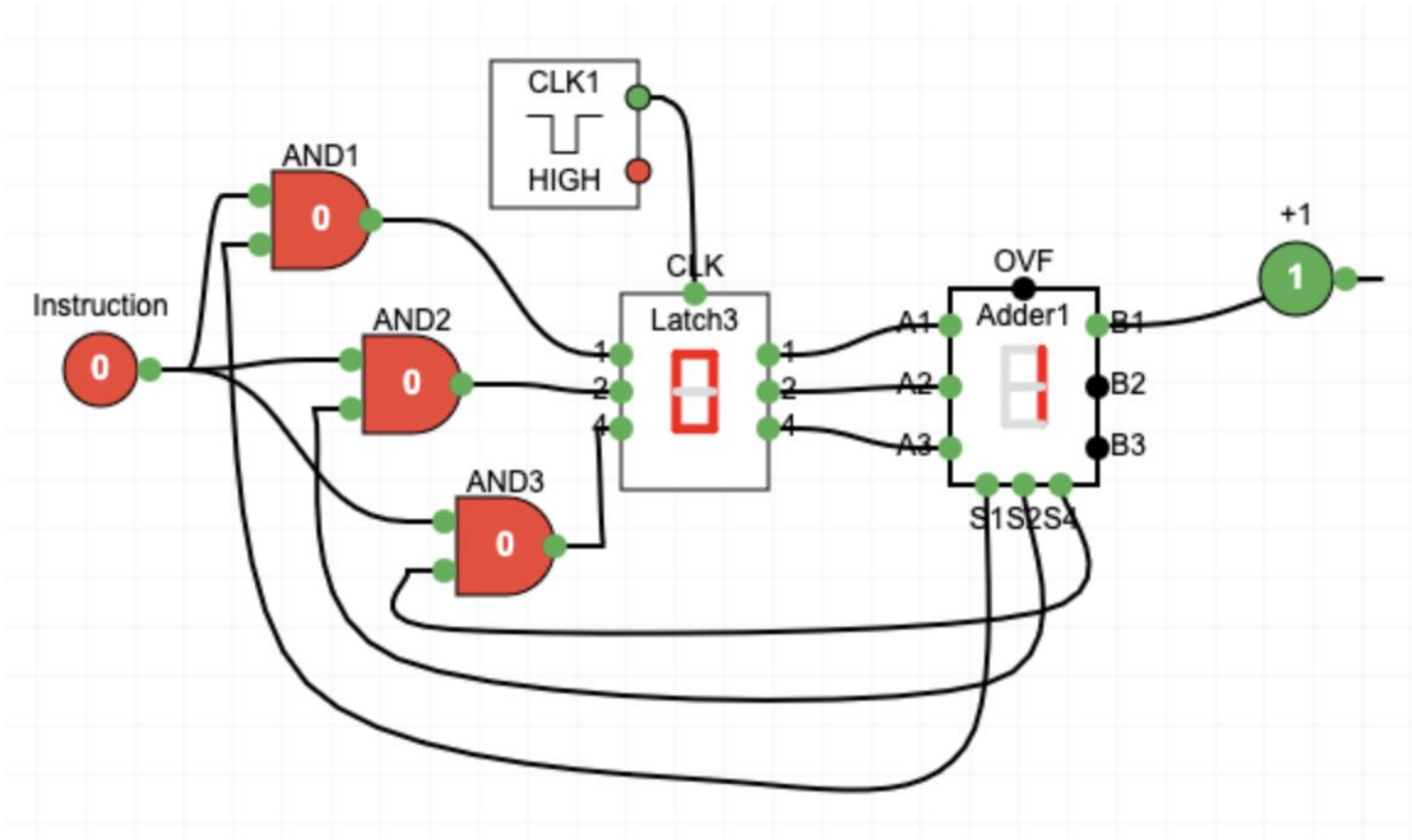
https://en.wikipedia.org/wiki/CDC_6000_series#Central_processor

For now, we will build a simple CPU

- One 3-bit latch
- Two machine language instructions
  - 0 – Clear the latch
  - 1 – Add 1 to the latch
- It will have a 1 Hz clock and be capable of executing one instruction per second
- We will build this CPU and run it in our emulator

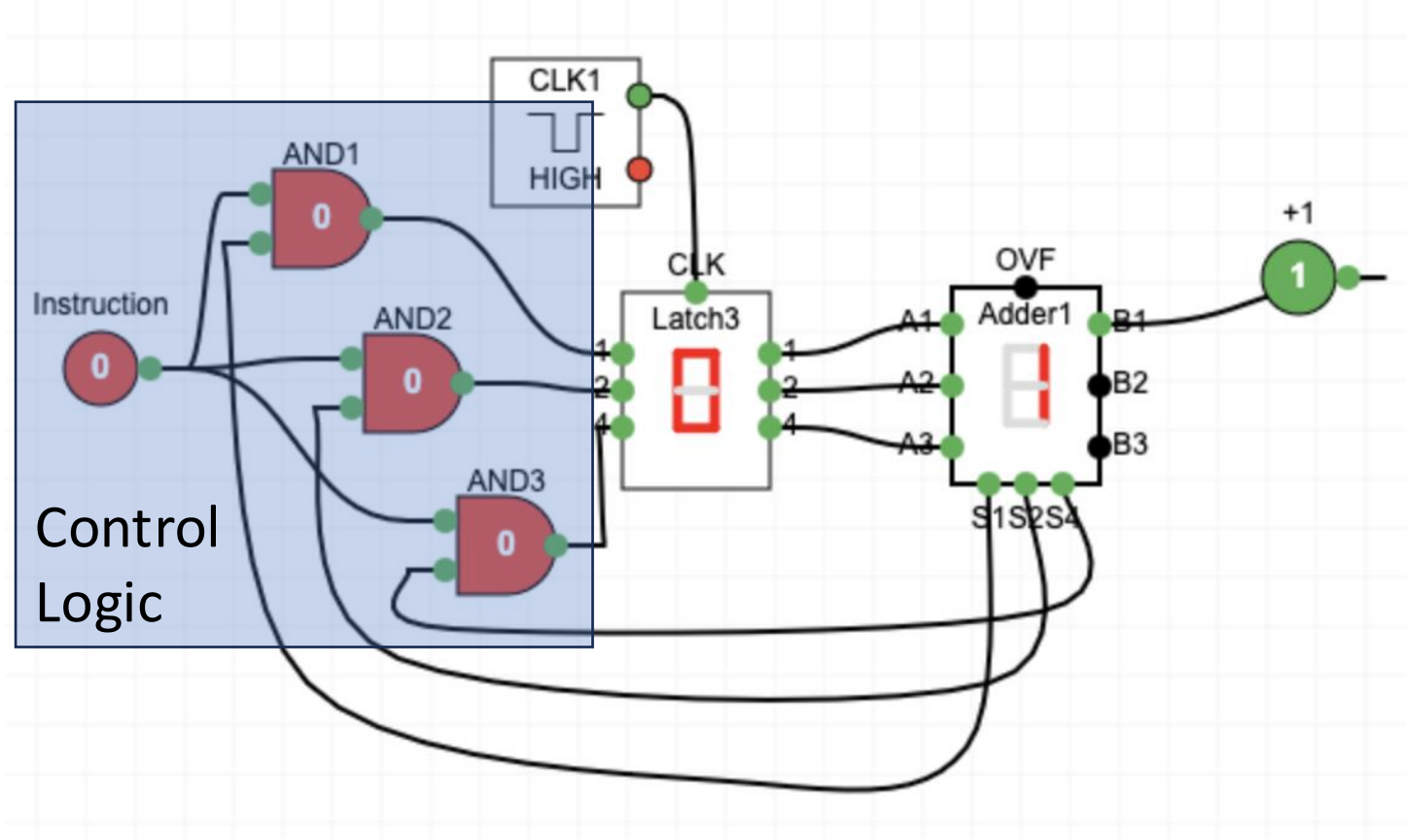
Start with our counter

- A latch feeds its stored value into one operand of an adder (clock low)
- We hard-code the other adder input to be 1
- We have a clock to tell the latch when to copy its new input to its internal state (clock high)



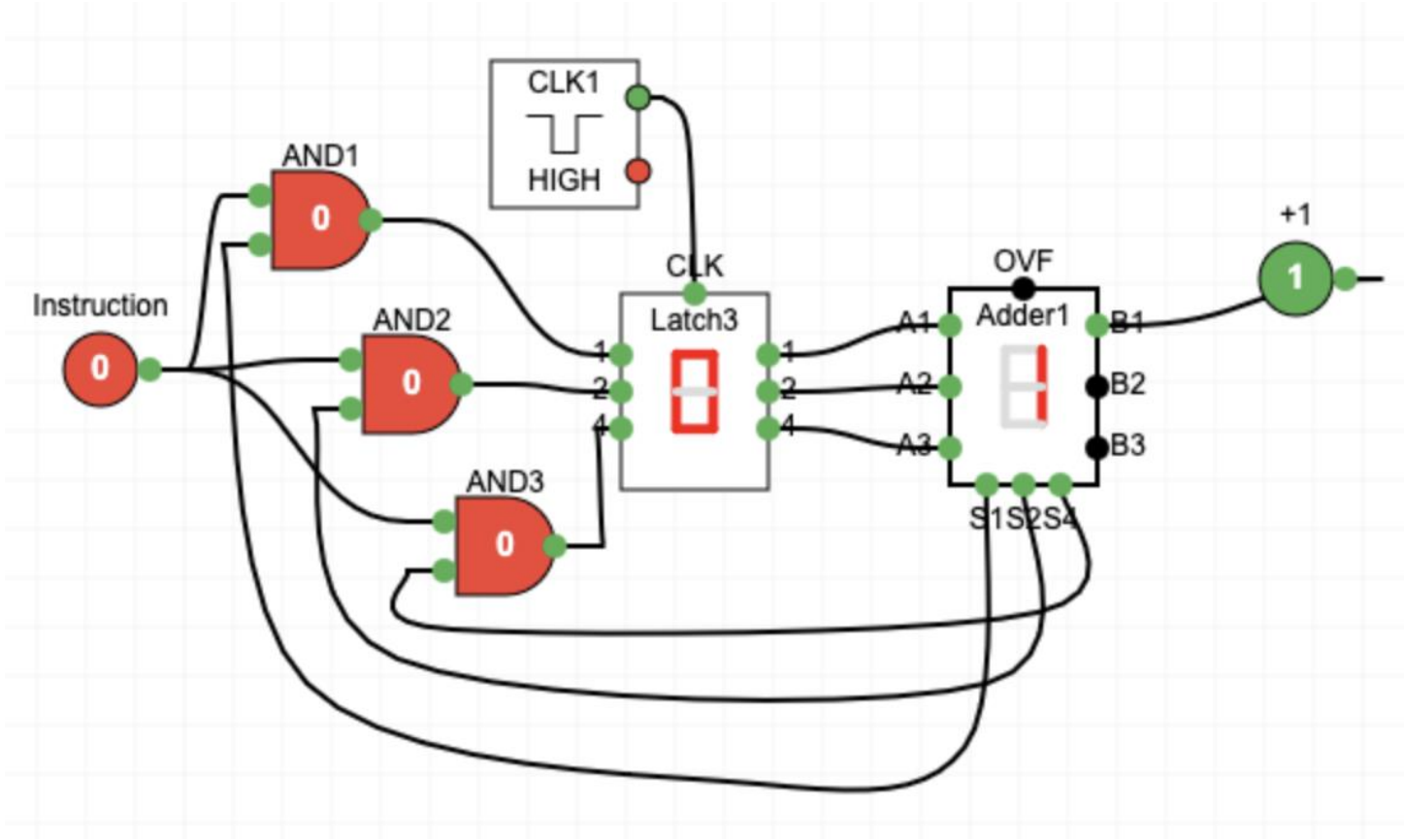
Add control logic

- Have an instruction input
  - 0 – Reset latch to zero
  - 1 – Copy output of adder
- The instruction is fed into three AND gates
- The sum values also go into the AND gates
- The output of the AND gates goes into the Latch



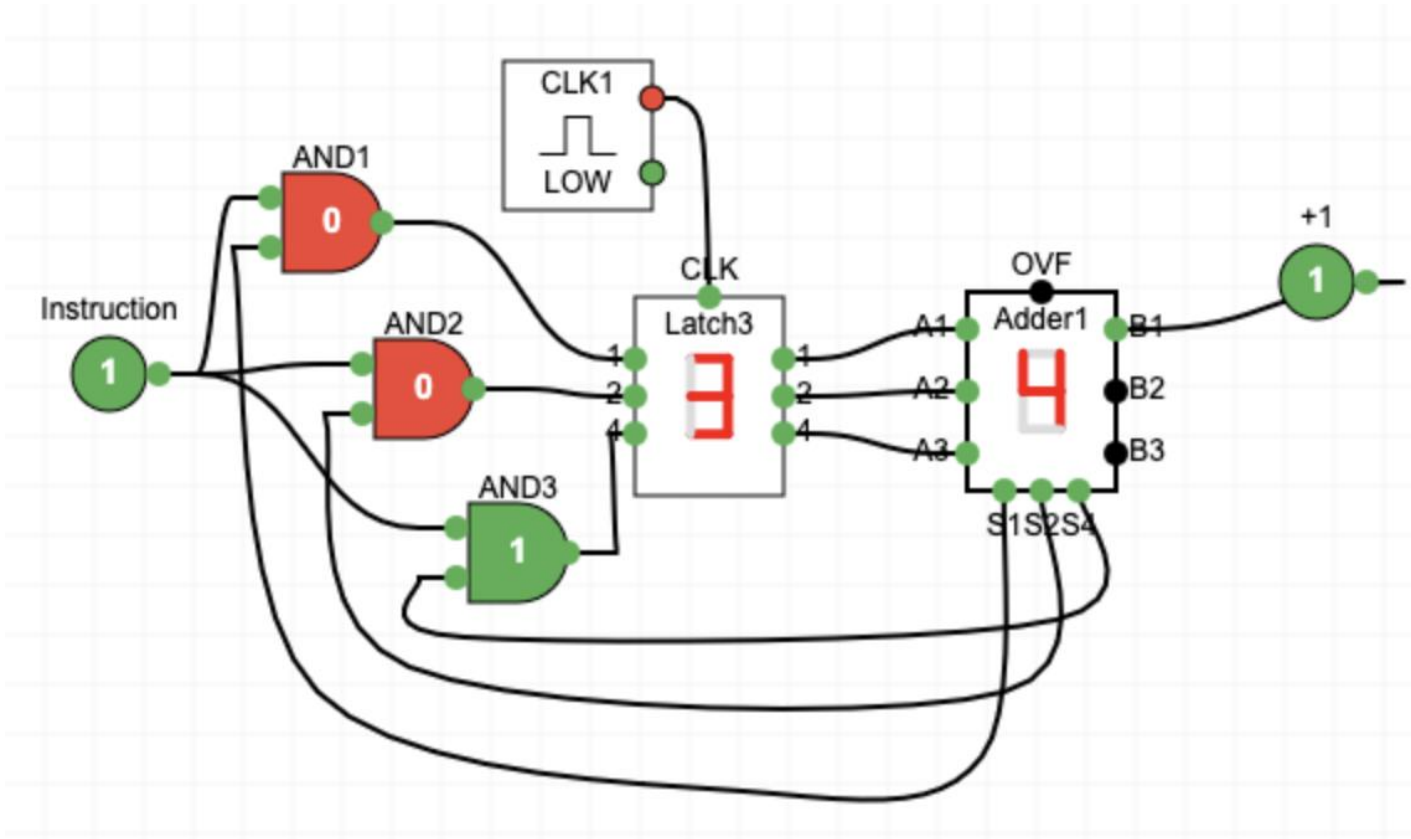
Instruction 0

- When instruction is zero, the output of the three AND gates is zero and the latch stays zero (reset)

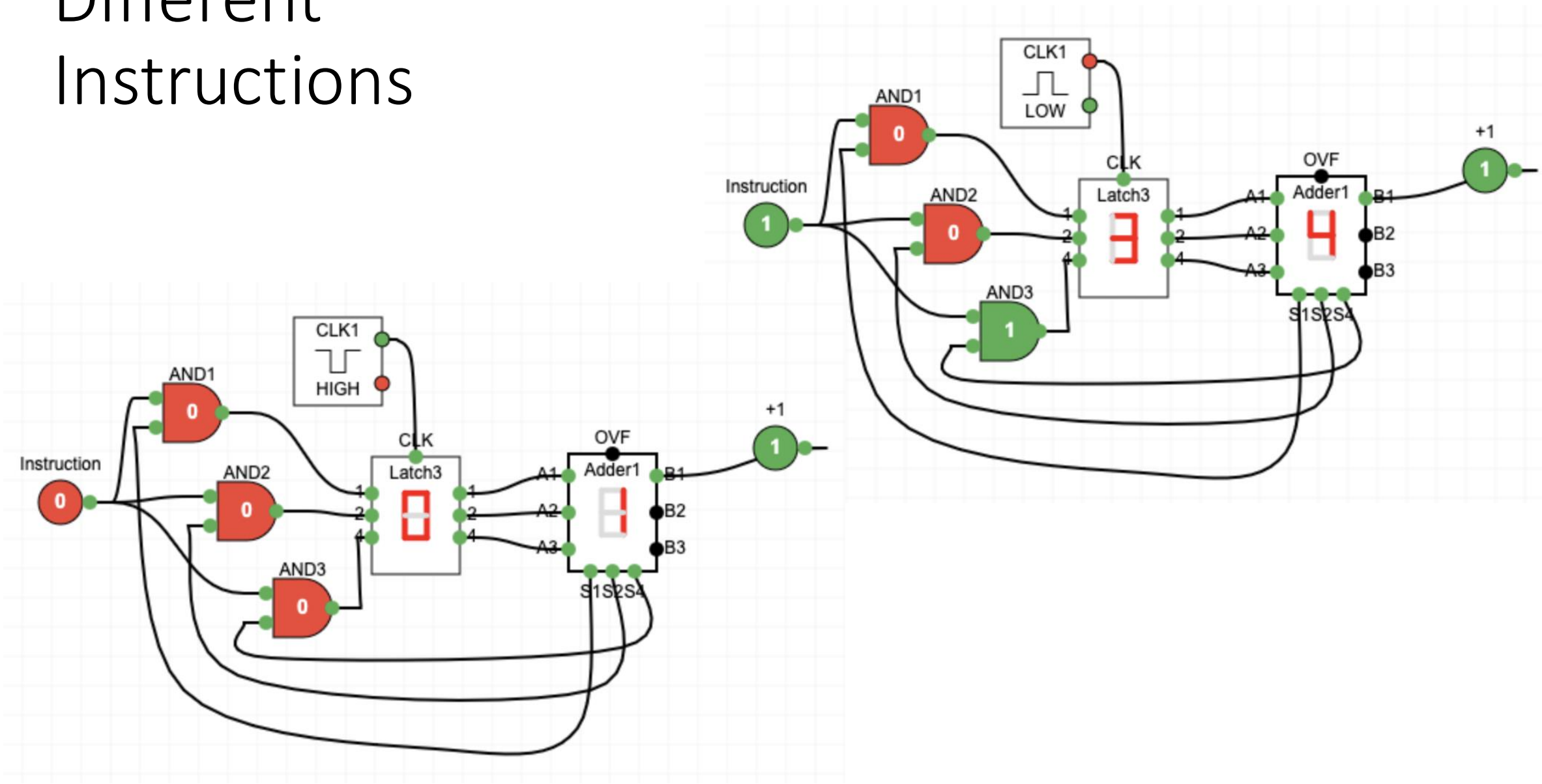


Instruction 1

- When instruction is one, the sum values are passed to the latch inputs
- And each clock rise/fall adds 1 to the latch value

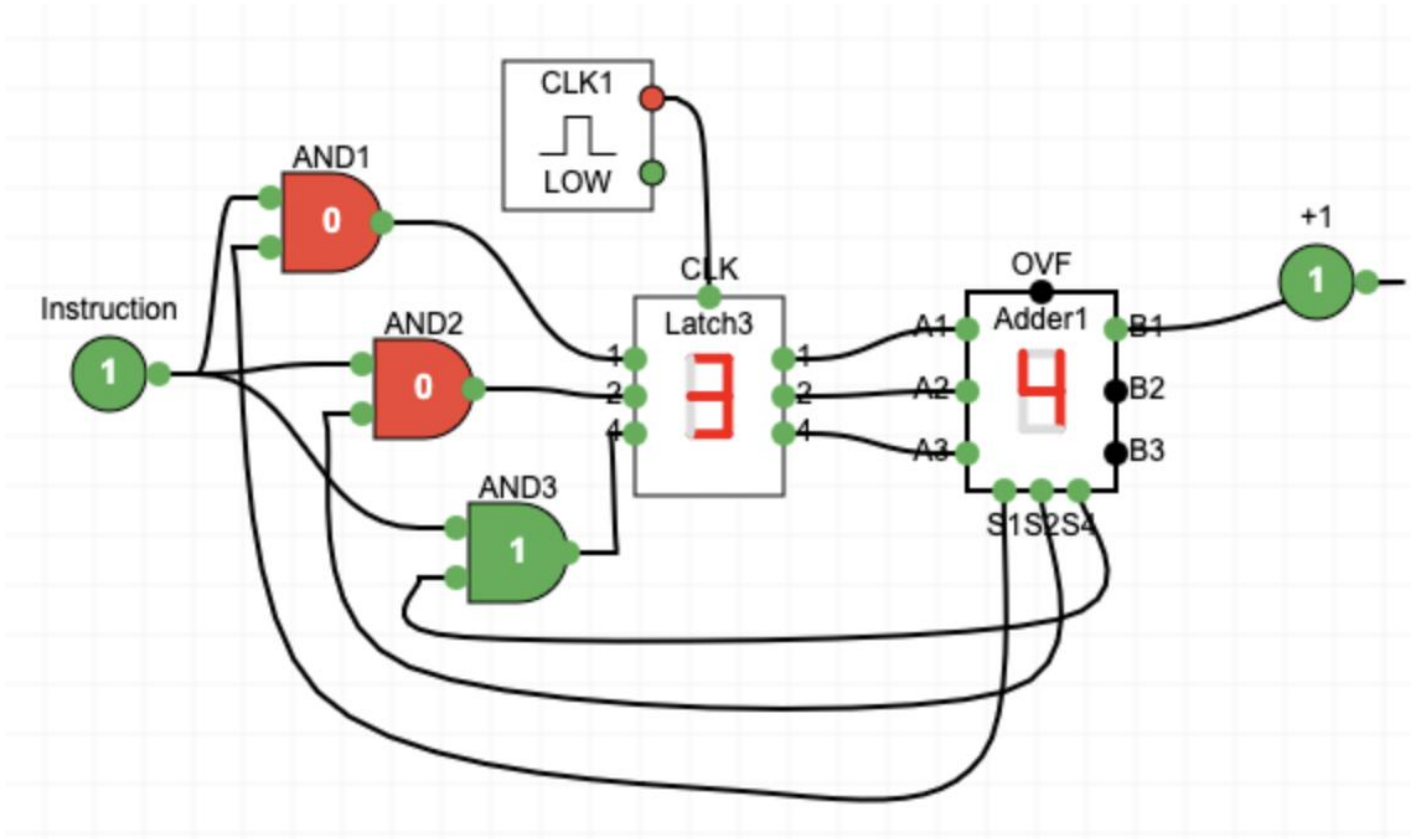


Different Instructions



What is missing?

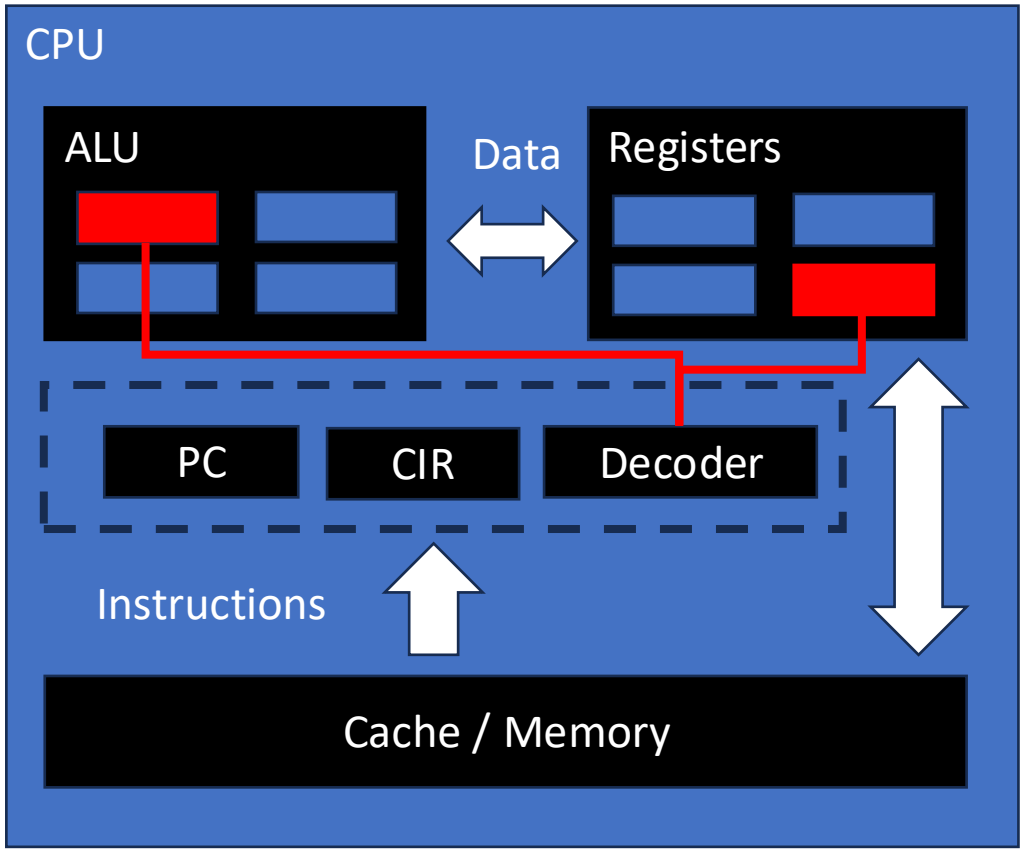
- The instructions are not being read from memory.
- It is like there is a switch on the front of the computer and it chooses between the two instructions
- We want a flexible sequence of many instructions executed one after another (a.k.a. Software)



Stored Sequences of Instructions

Fetch-Decode-Execute-Cycle

- Program Counter (PC) Register – Address of the next machine code instruction
- Current Instruction Register (CIR)
- The decoder looks at the bits of the CIR and using gates activates the correct ALU Element and register
- The Clock tells the logic when to compute and when to copy results back to the register



Machine Language Instructions

- As part of the documentation for a CPU, we are given the bit patterns of the machine code for each of its instructions
- There is usually a symbolic language that translates to machine language (assembler)

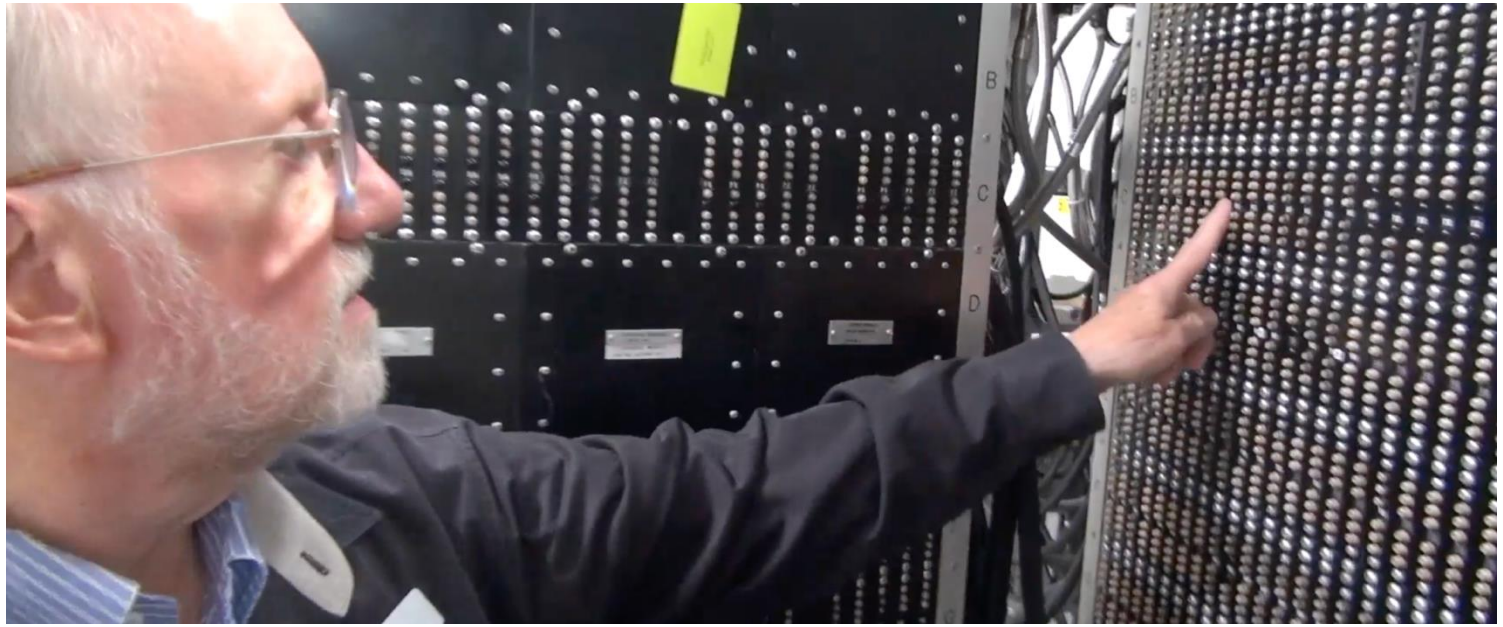
Assembly Language	Binary	Description
ZERO reg	01000rrr	Set register to zero
CMPZ reg	01001rrr	Compare register to zero
INC reg	01010rrr	Increment register by 1
DEC reg	01011rrr	Decrement register by 1

Up Next…

- Soon, we will design a simple CPU
  - CDC 8512 – Inspired by the Control Data Corporation 6500
- It will have a complete architecture
- It will have a completely defined machine language
- We will write real programs in CDC 8512 machine code and run them in an emulator

https://en.wikipedia.org/wiki/CDC_6000_series#Central_processor

Enough Foreshadowing



Summary

- We introduced the notion of a clock to wait until computation is complete
- We talked about how clock rate has improved over time
- We learned about the internals of a CPU – registers, arithmetic logic unit, program counter, and decoder logic
- Up next: Designing and implementing a CPU architecture and its machine language

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Insert new Contributors and Translators here including names and dates

Continue new Contributors and Translators here