Hardware overview

HW1 will keep coming out

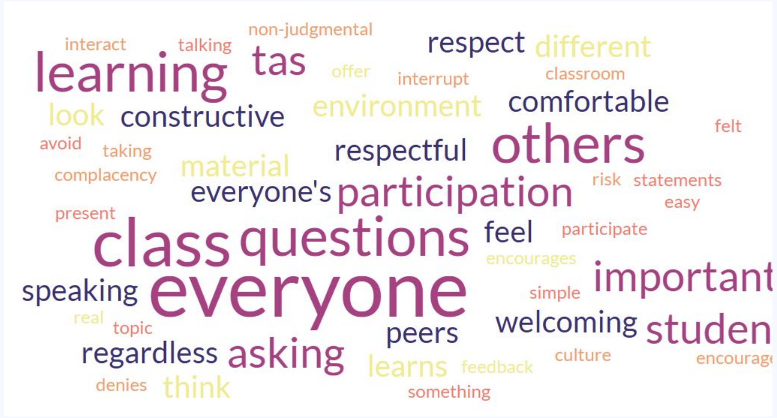
HW0 responses

Excited about: a lot!

Nervous about: hardware, assembly, course structure, novelty of topics

Helps your learning: in-class activities, diagrams, low-stakes activities

Community: committed to welcoming environment + sustained communication! We want to focus on everyone's success and growth



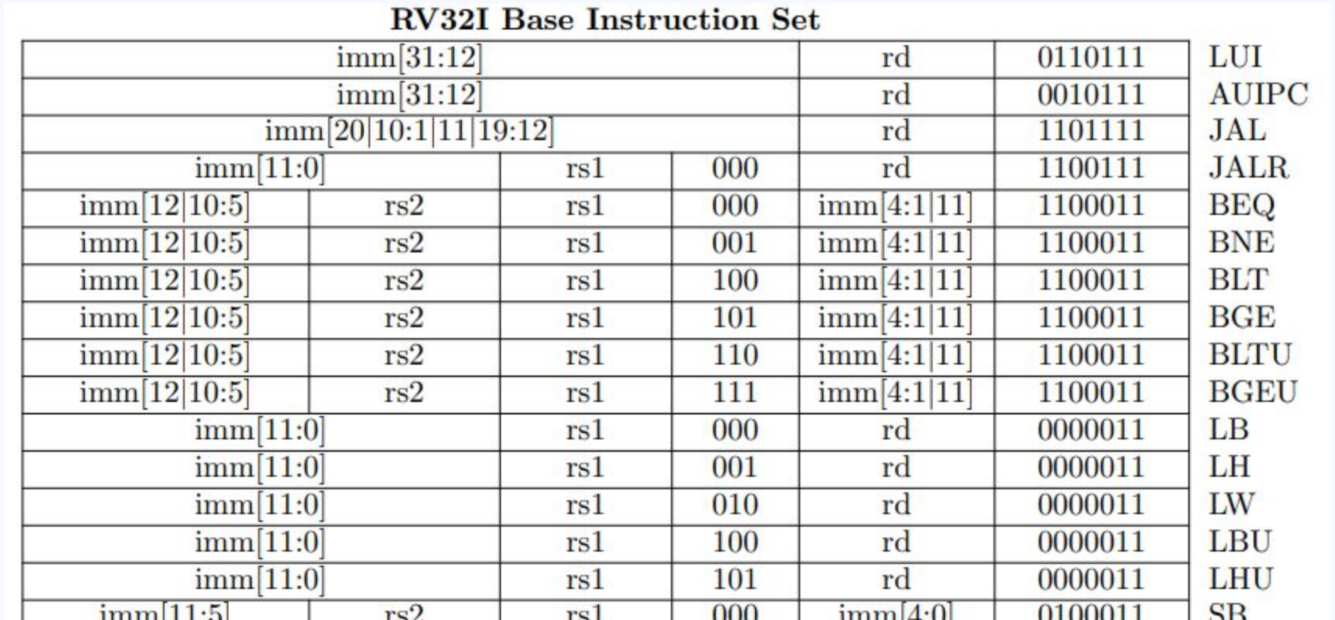
source // another visualization // Babbage’s analytical engine

? ? ?

Goal of a CPU

Interpret an instruction (bits in memory/signals on wires) as an action it should take

What does that look like in hardware?



RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB

HW assumptions we’re working with

CPU can read bits from memory as electrical signals (one “wire” per bit)

Everything is a pure low/high signal, no noise/interference

For now, we’re not worried about constraints (space, complexity, power)

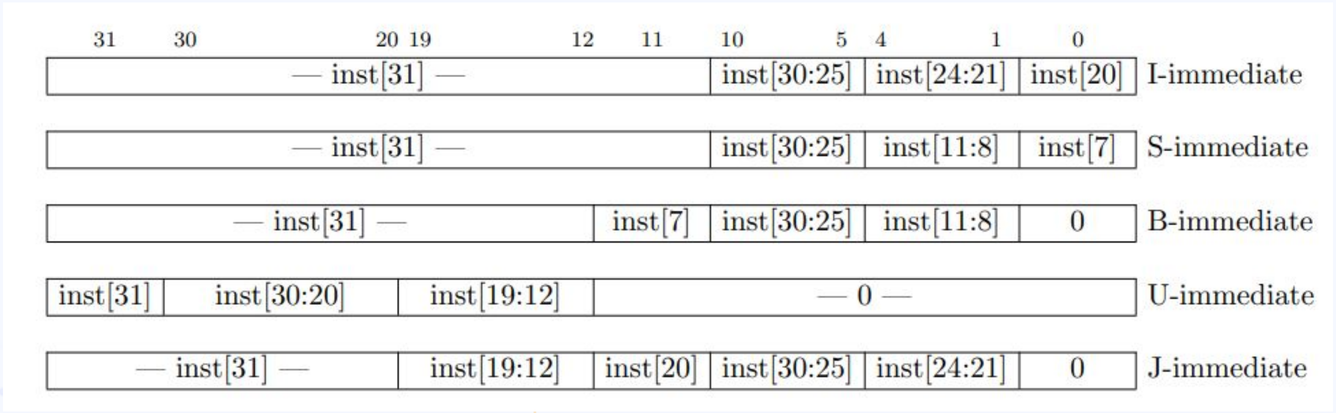
Each “step” leaves enough time for circuit to stabilize

To run a program, CPU HW needs:

A way to extract/rearrange bits - pull out the relevant fields of an instruction

A way to implement combinational logic – arithmetic/logical, comparison

A way to keep track of state - what is the value of the PC at the current step?

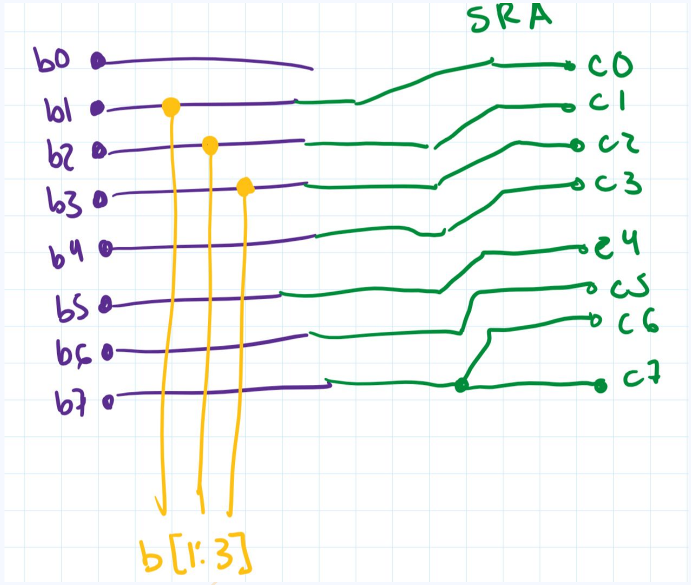


31	30 20	19 12	11	10 5	4 1	0	
— inst[31] —				inst[30:25]	inst[24:21]	inst[20]	I-immediate
— inst[31] —				inst[30:25]	inst[11:8]	inst[7]	S-immediate
— inst[31] —			inst[7]	inst[30:25]	inst[11:8]	0	B-immediate
inst[31]	inst[30:20]	inst[19:12]	— 0 —				U-immediate
— inst[31] —		inst[19:12]	inst[20]	inst[30:25]	inst[24:21]	0	J-immediate

Data as collections of wires

wire/data line: carries a single digital signal (on/off)

bus (P&H definition): a collection of data lines that is treated as one, multi-bit signal



Combinational logic circuits

Examples: adders, logical operators, control signal translation

Work like pure functions (no memory)

Combinatorial expressions can be automatically synthesized to circuits

Physically, logic gates are implemented using transistors (electrical switches)

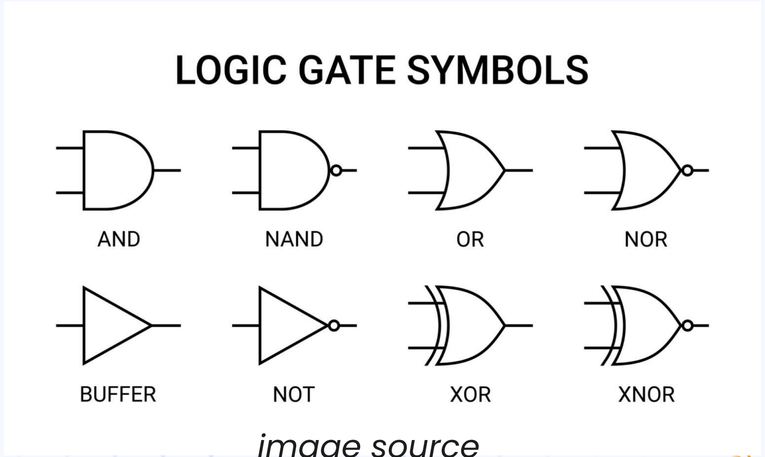
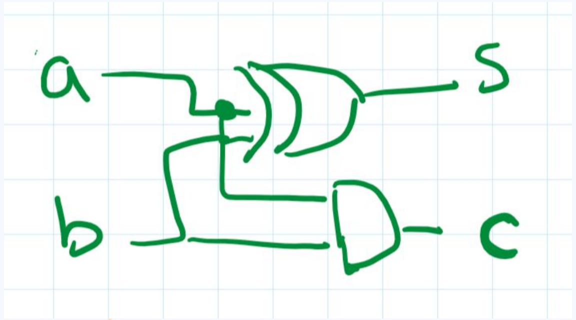


image source

"half adder" (**s**um and **c**arry output):



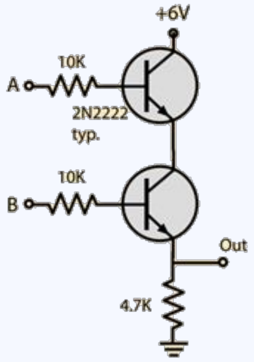


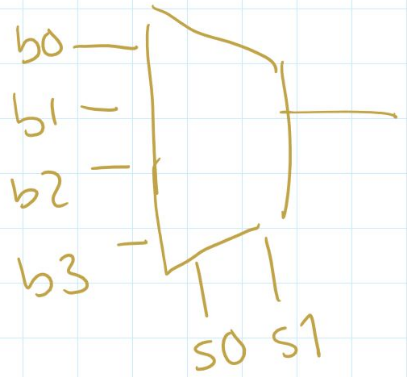
image source

Multiplexers

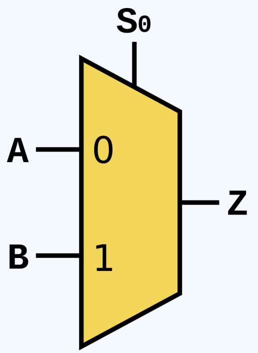
Used to select between multiple inputs

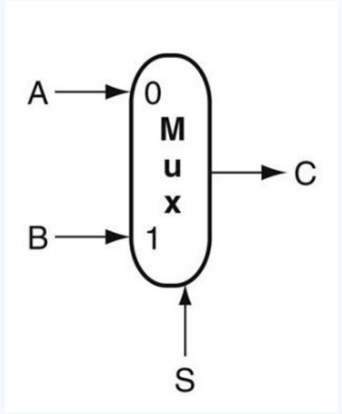
n-bit selector signal = select between 2^n inputs

Example: 2nd operand for add vs addi



s0	s1	out
0	0	b0
1	0	b1
0	1	b2
1	1	b3



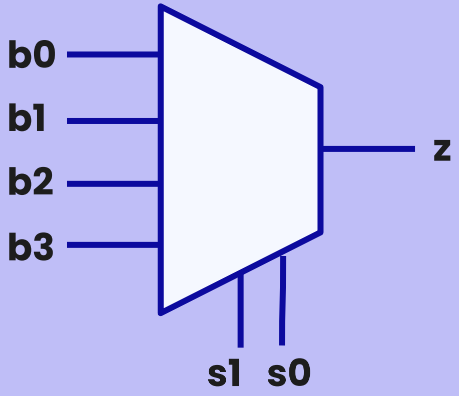


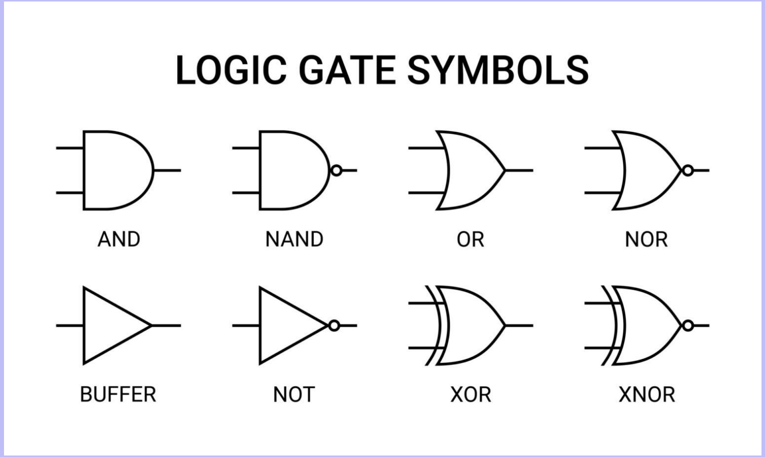
P&H Fig. A.3.2

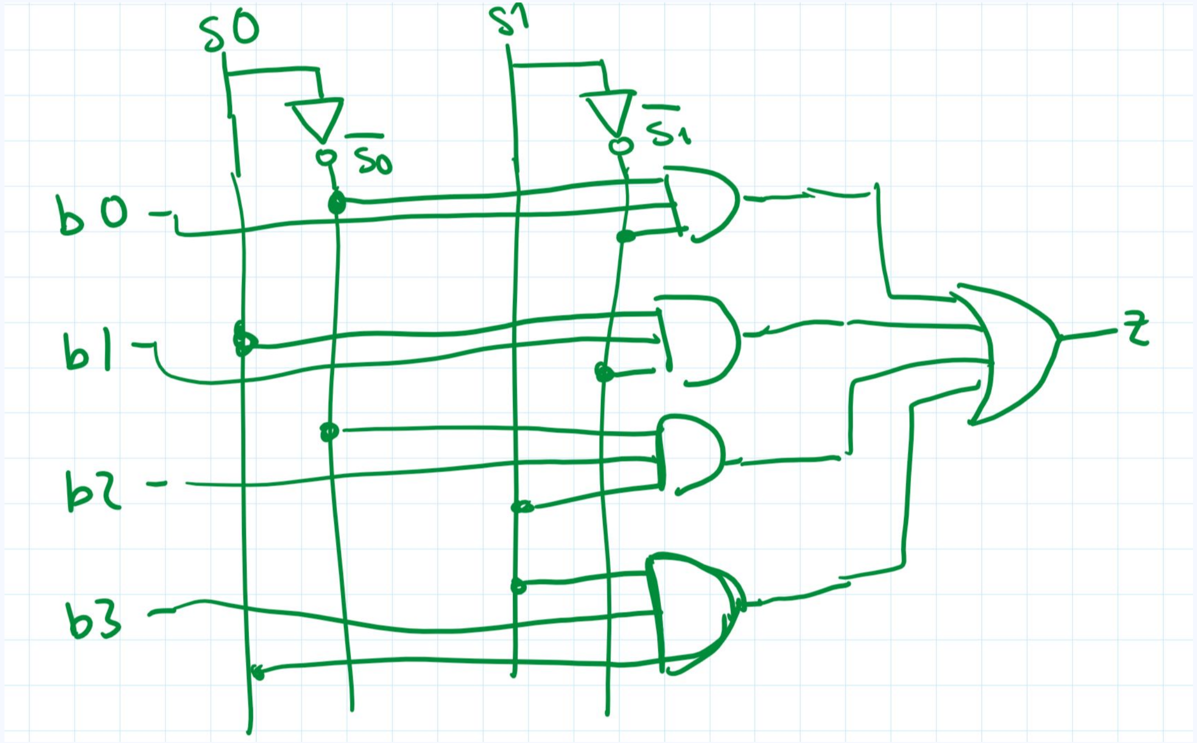
By en:User:Cburnett - This W3C-unspecified vector image was created with Inkscape ., CC BY-SA 3.0, link

? ? ?

Build a 4-input (2-bit selector) mux out of logic gates



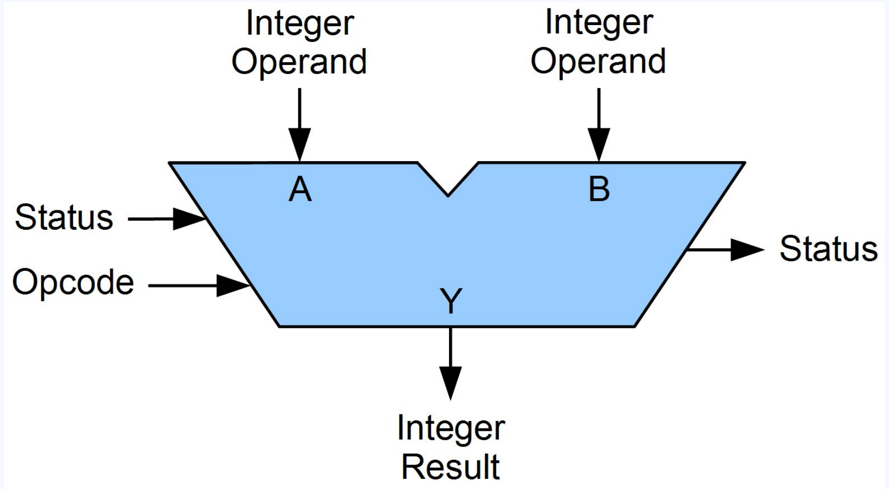



s0
$\overline{s0}$
s1
$\overline{s_1}$
b0
b1
b2
b3
z

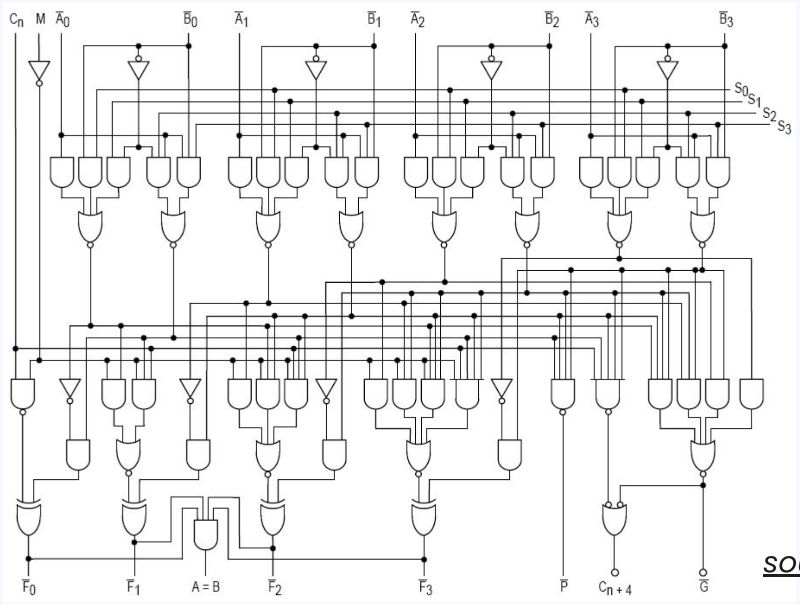
Arithmetic Logic Unit

"ALU"

Takes in two operands and a control signal for the operation, produces result of applying operation on operands (status input/output signals optional)



By Lambtron - Own work, CC BY-SA 4.0, link



source

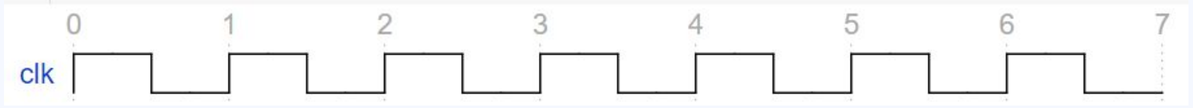
Clocks

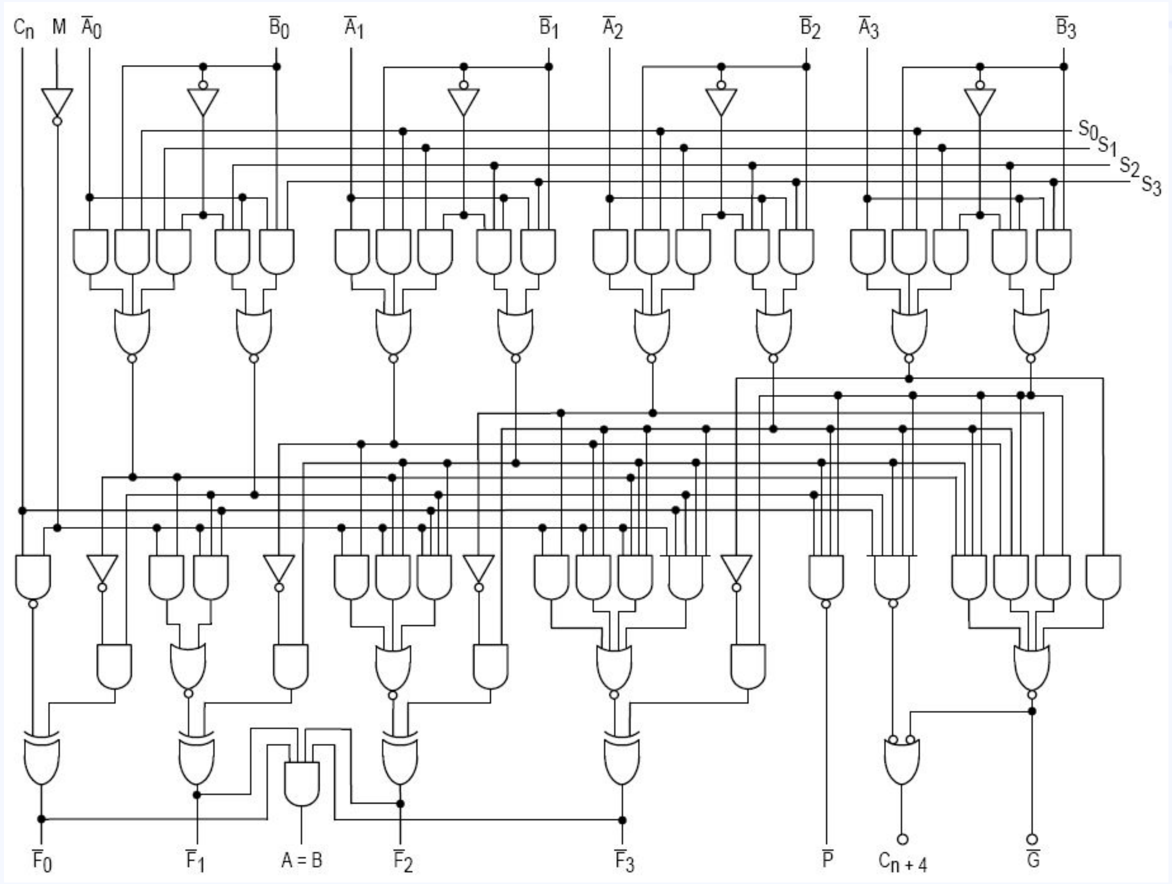
In a circuit, many things happen in parallel

Synchronization signal (wire) that allows necessary components to know when to move on to the next "step"

Clock cycle time is long enough to allow for signals to stabilize

> i.e. allow electrons to travel through the longest possible path of wires/transistors



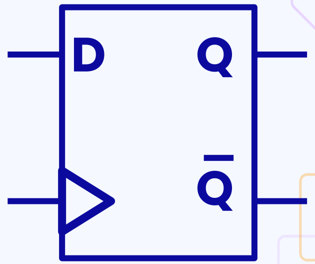

C_n
M
$\overline{A}_0$
$\overline{B}_0$
$\overline{A}_1$
$\overline{B}_1$
$\overline{A}_2$
$\overline{B}_2$
$\overline{A}_3$
$\overline{B}_3$
S_0
S_1
S_2
S_3
$\overline{F}_0$
$\overline{F}_1$
A = B
$\overline{F}_2$
$\overline{F}_3$
$\overline{P}$
C_{n+4}
$\overline{G}$

Components that have state

How do we express “at each step, increment the PC by 4?”

Memory elements, such as flip flops and latches, have internal state that updates on clock tick (D flip-flop pictured)

Our abstraction of registers: each bit is stored in a D flip-flop



? ? ?

How do we express “at each clock tick, increment the PC by 4” using a PC register and an adder?

Takeaways

Bits of an instruction = electrical signals CPU uses to execute a program

CPU is just a (very, very) big circuit made up of wires, combinational logic elements, and memory elements

Can implement modules we need (multiplexers, ALUs, bit selectors, registers) using these elements

Basically: we have an “existence proof” of the hardware we need, so we can start working one level of abstraction higher to implement a CPU

Hardware description languages

Used to describe circuits (often for synthesis into a circuit, such as on an FPGA)

Examples: Verilog, VHDL

Defines behavior of combinational components and memory components

Updates in a block are done in *parallel* – Verilog example:

```
reg a, r;
always @(posedge clk) begin
    r <= r + 1;
    a <= ~r;
end
```

We won’t be working in HDL – but a C++ approximation of it in simulation

Let’s build a CPU!

What do we need to get started?