Hardware overview

HW1 will keep coming out
TA hours are on website
Moving my hours to Fridays at 11 (next week)

HW0 responses

Excited about: a lot!

Nervous about: low-level details, new course/workload, heard scary things about the topic, using the simulators

Helps your learning: in-class activities, having access to resources

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

comments
assume
material
students
difficult
community
questions
new
milda
classroom
assignments
believe
act
help
peers
casual
find
try
class
best
environment
knowledge
constructive
backgrounds
people
course
think
systems
understanding
one
around
feel
previous
understand
everyone
expectation
talk
create
positive
interact
learn
respectful
especially
welcoming
judgement
topic
others
individuals
feedback
give

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

? ? ?

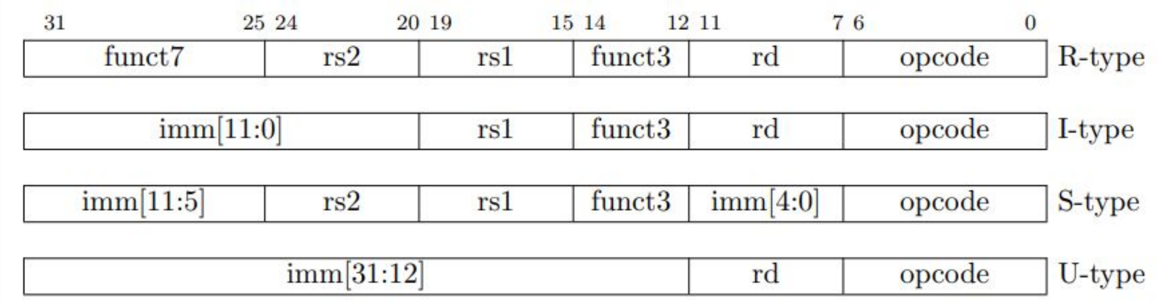
Stored-program computers

Modern computers hinge on two principles:

- Instructions are represented in memory the same way as numbers
- Memory can be altered by programs

First principle means that:

- Instructions live in memory
- The CPU needs to have a way of interpreting an instruction, just as it would any other data in memory



31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[31:12]				rd	opcode	U-type

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)

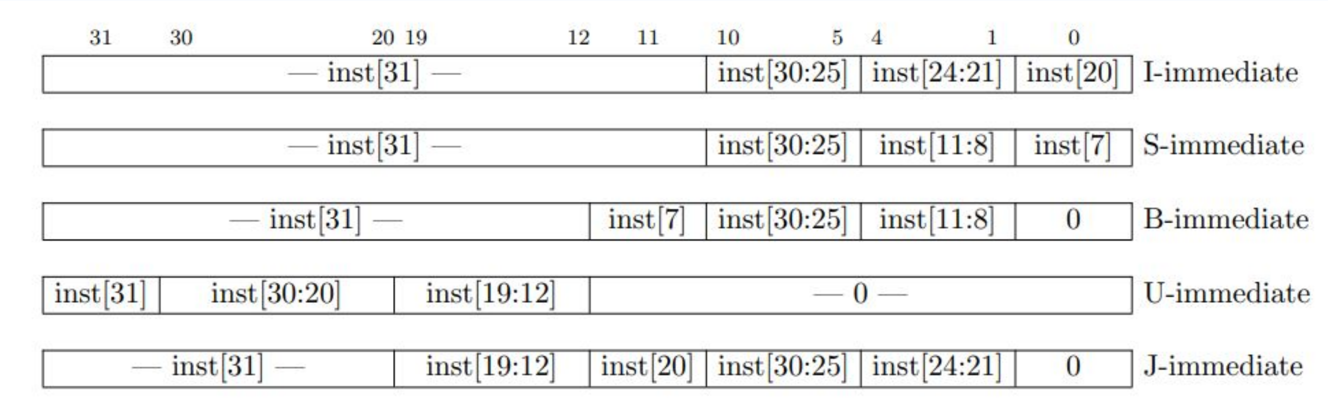
Each “step” leaves enough time for circuit to stabilize

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, branching

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

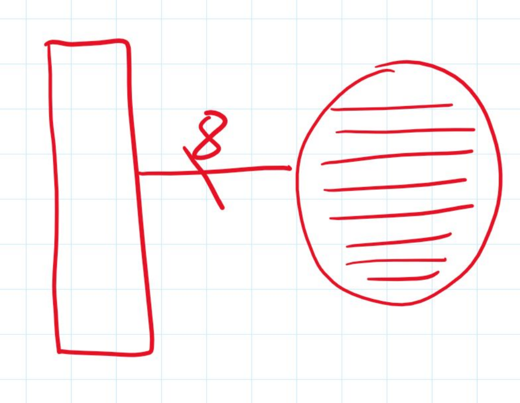


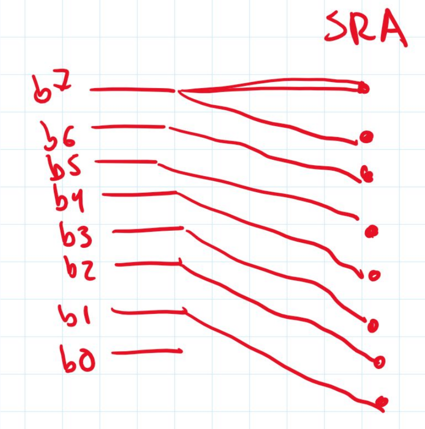
31	30 20	19 12	11	10 5	4 1	0	
— inst[31] —				inst[30:25]	inst[24:21]	inst[20]	I-immediate
— inst[31] —				inst[30:25]	inst[11:8]	inst[7]	S-immediate
— inst[31] —			inst[7]	inst[30:25]	inst[11:8]	0	B-immediate
inst[31]	inst[30:20]	inst[19:12]	— 0 —				U-immediate
— inst[31] —		inst[19:12]	inst[20]	inst[30:25]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

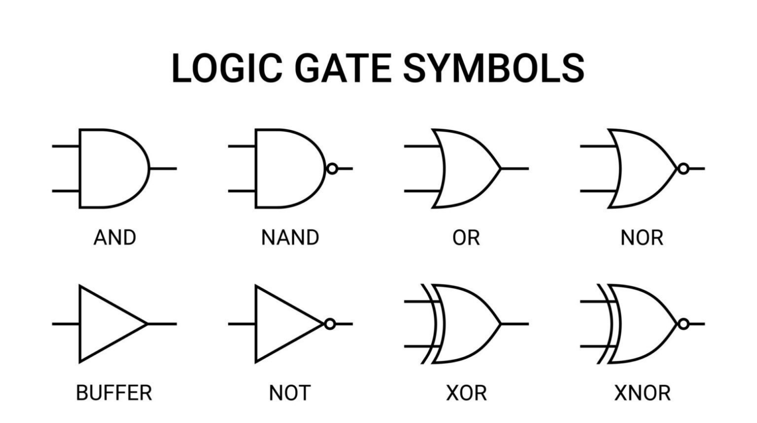
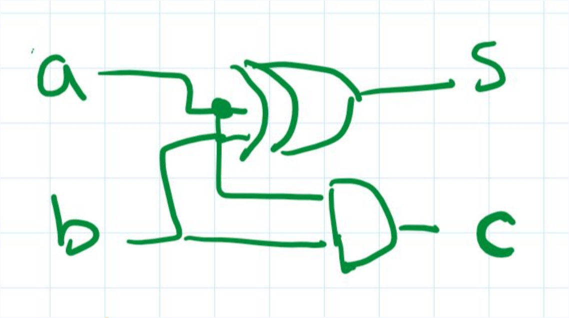


image source

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



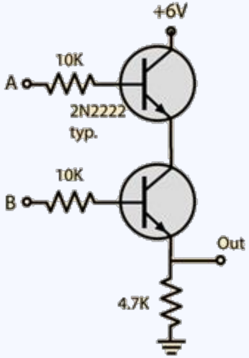


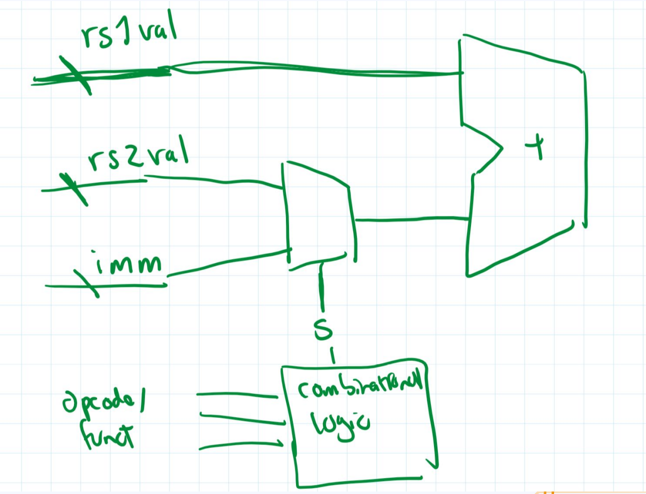
image source

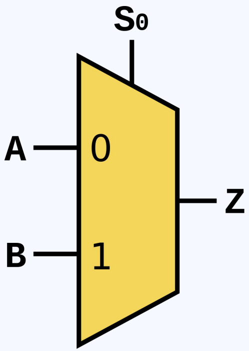
Multiplexers

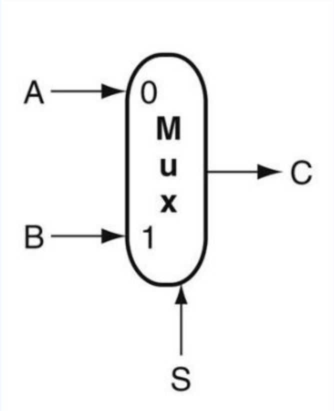
Used to select between multiple inputs

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

Example: 2nd operand for add vs addi





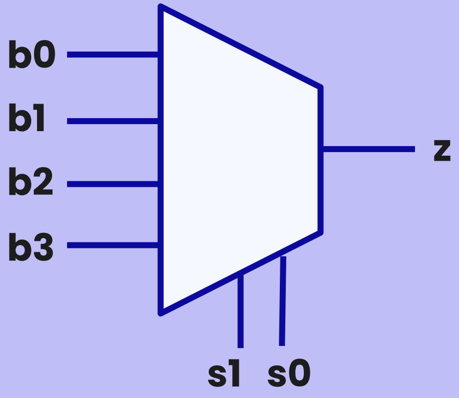


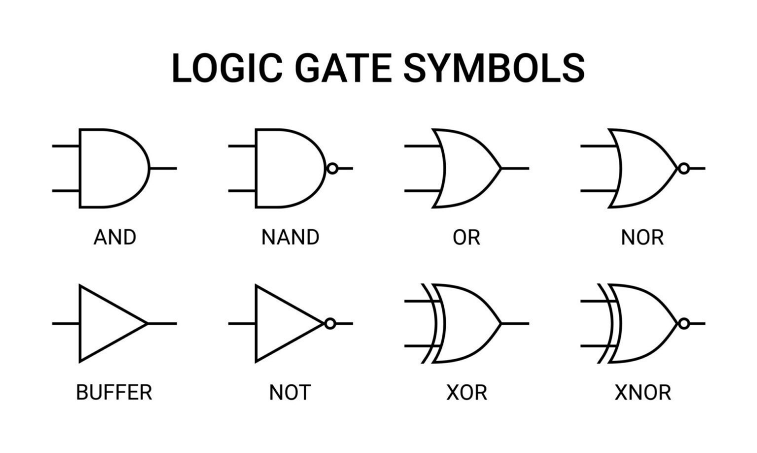
P&H Fig. A.3.2

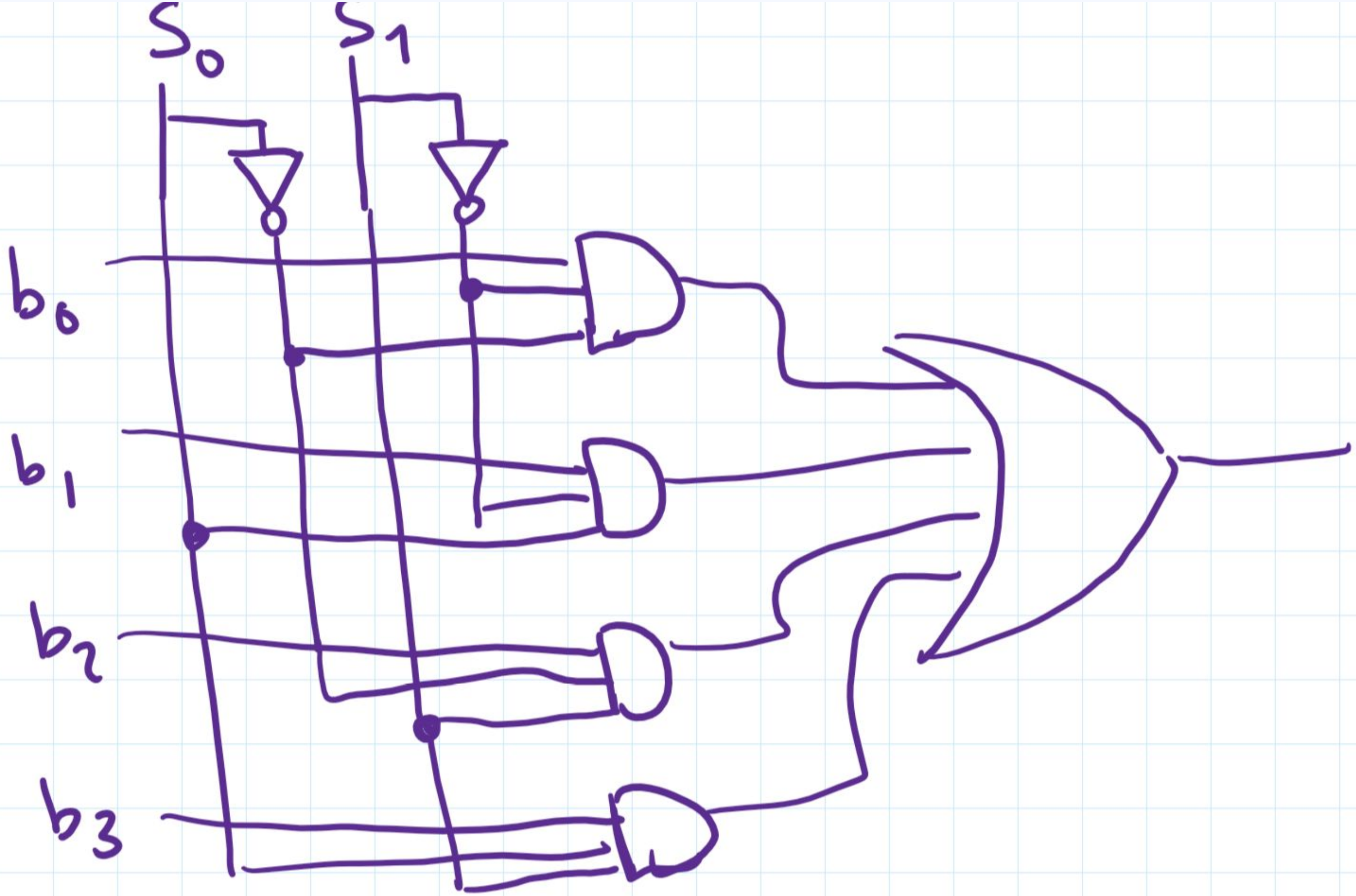
By en:User:Cburnett - This W3C-unspecified vector image was created with Inkscape ., CC BY-SA 3.0, link

? ? ?

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



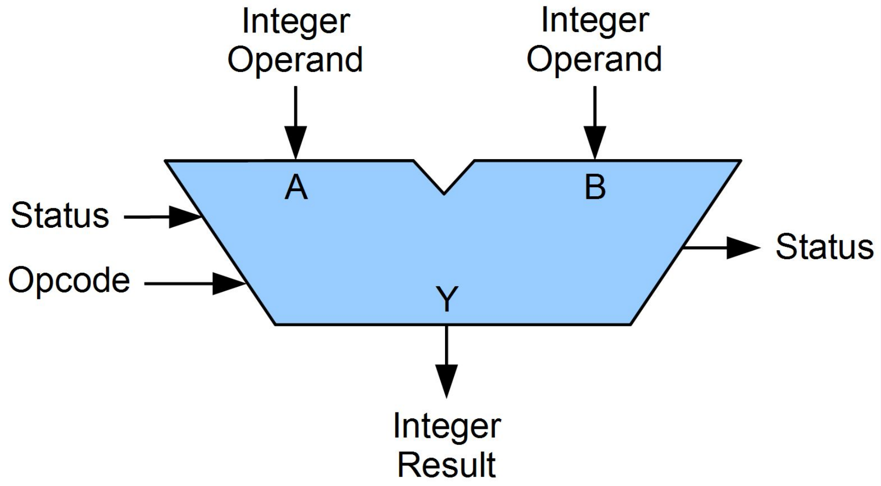



S_0
S_1
b_0
b_1
b_2
b_3

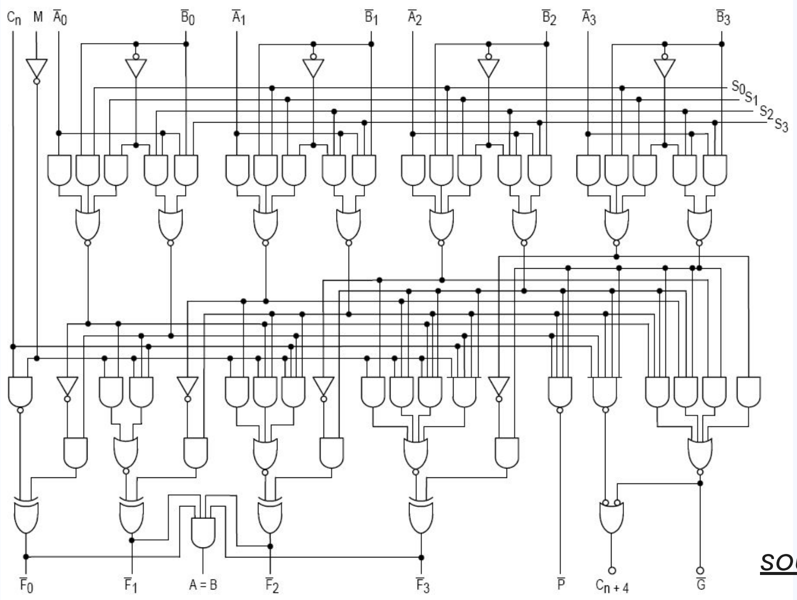
Arithmetic Logic Unit

“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

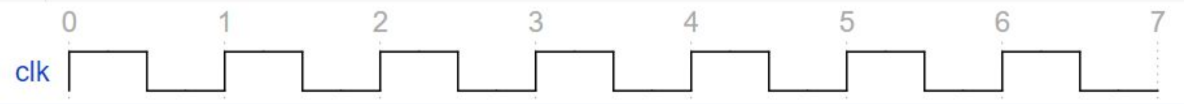


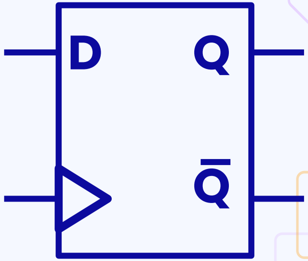
source

Components that have state

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

Need *clock signal* to control when state changes





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
    a <= ~r;
    r <= r + 1;
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?