CSC258 Winter 2016
Lecture 6

Today's outline

- Processor components: ALU
- Review for midterm test

Lab 5 Tip

- In simulation, I'm getting indeterminate output values all the time. What can I do?
- This is because in the handout circuit the input is never explicitly set. You can:
  - Add a RESET signal which ANDs with the inputs of the flip-flops, so when RESET is 0, the inputs are forced set to 0.
  - Take a leap of faith and try the DE-2 board directly. (Risky)

A Peek of Lab 6

- Implement a James Bond gadget.

https://www.youtube.com/watch?v=Vi4LmILZUog

- This involves the most pre-lab design work so far, so make sure to spend some time in your reading week working on it.

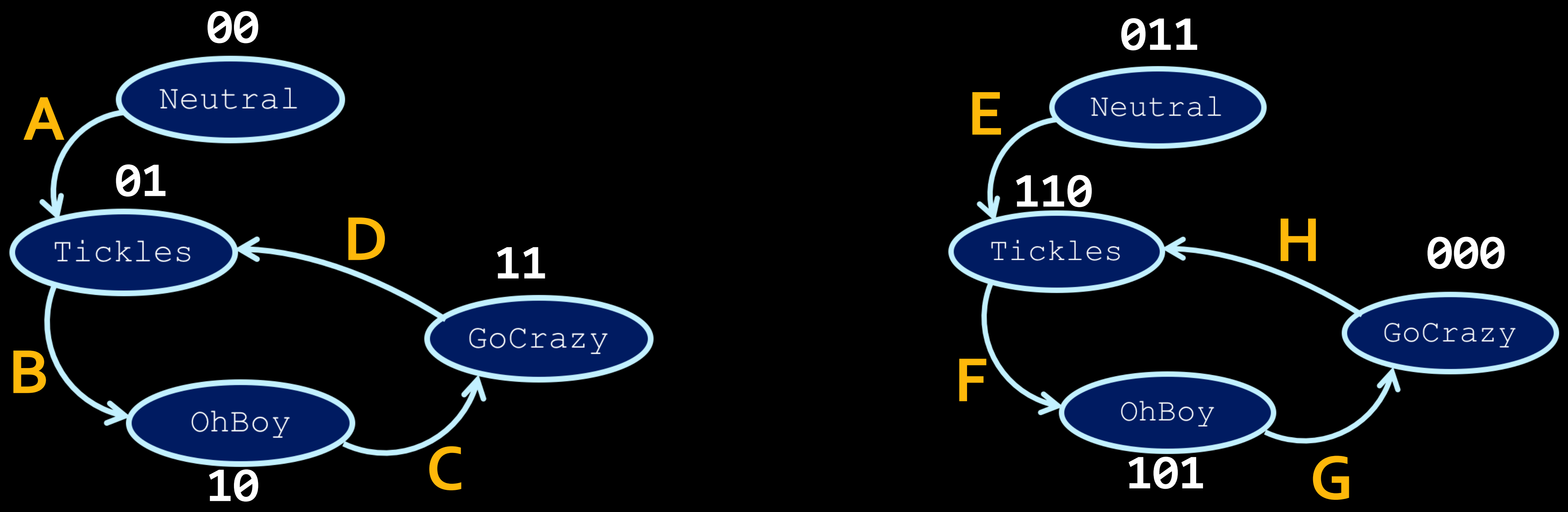
Quiz Time!

In this quiz, we will go through the steps of designing the Tickle-Me-Elmo circuit …

Question 1

Below is the state transition diagram of Elmo, with two different ways of assigning flip-flop values.

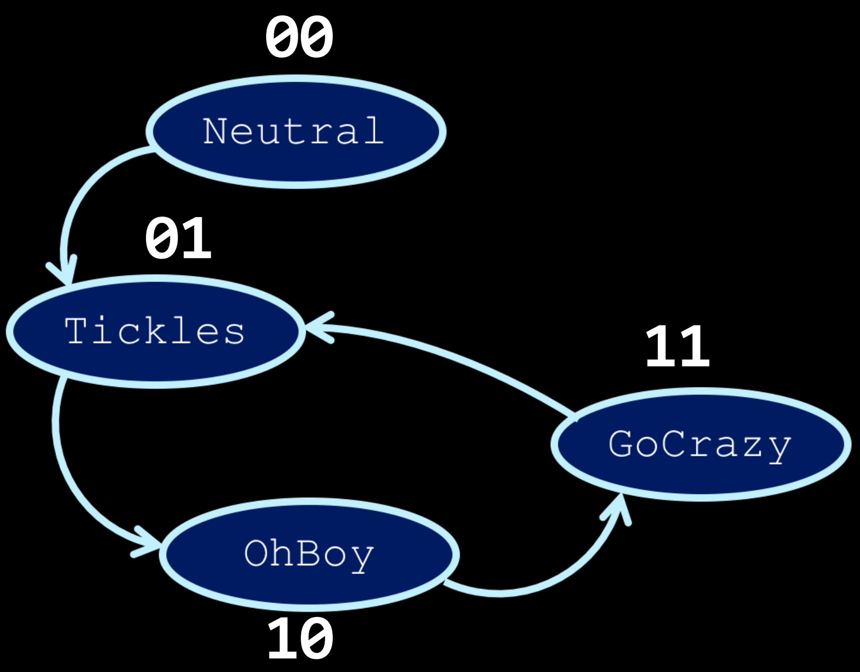
Which one of the transitions (A-H) will cause **unexpected behaviour** that cannot be avoided?



Question 2

Suppose we decided to go with the following flip flop assignment. And on the right side is the generated State Table.

Which row of the State Table is **wrong**?



	F_1F_0	F_1F_0
A	0 0	0 1
B	0 1	1 0
C	1 0	1 1
D	1 1	0 0

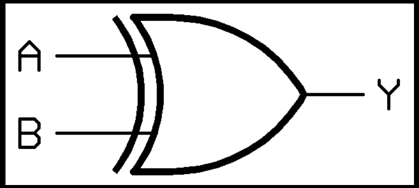
Question 3

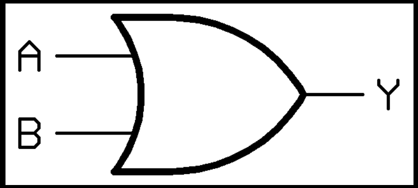
Suppose the circuit expression we get from the State Table is the following:

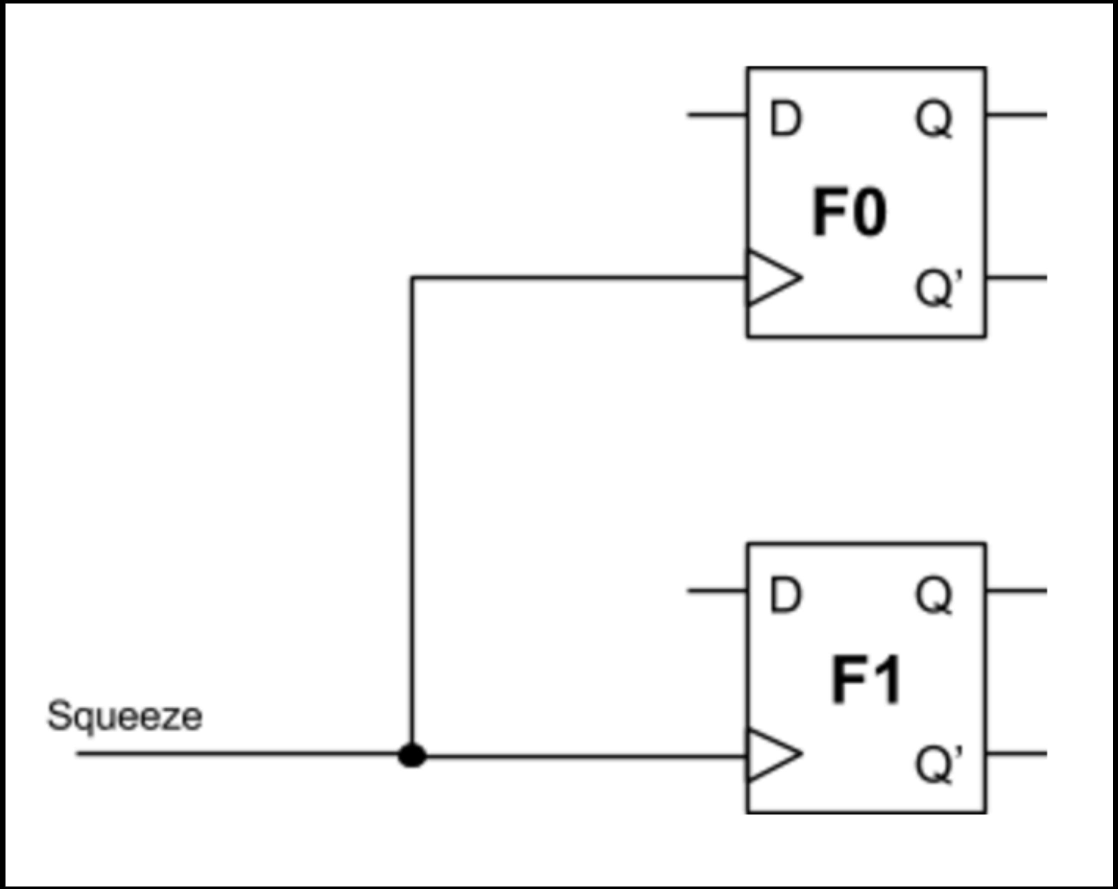
$$F1 = F0 \oplus F1$$

$$F0 = F1 + F0'$$

Complete the circuit.







Done!

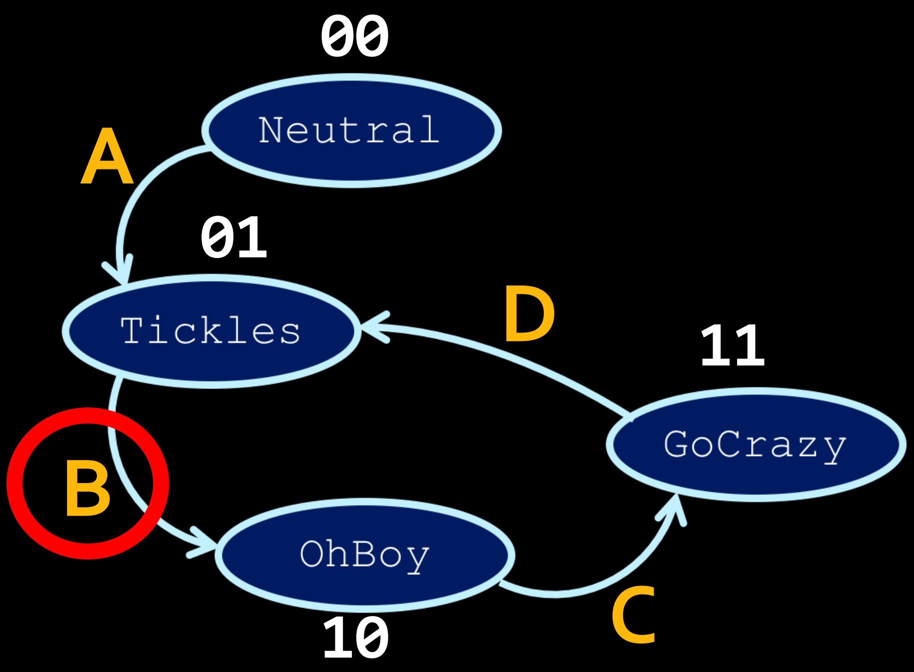
Taking it up…

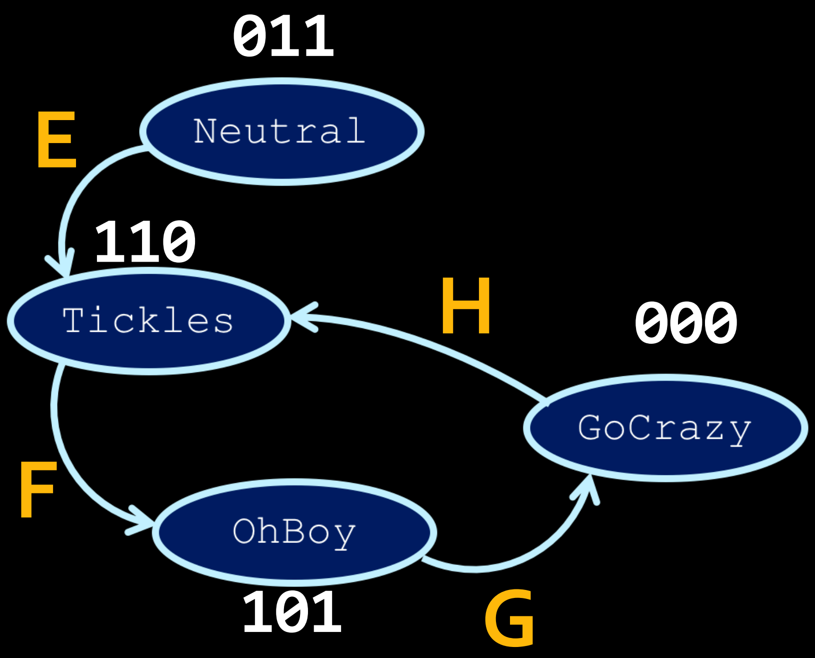
Question 1

Transition B can enter 11 temporarily and cause Elmo to go crazy

Transition E can enter **111** or **010**, both are **unused** state, so unexpected behaviours can be avoided.

Same argument for F, G, H.

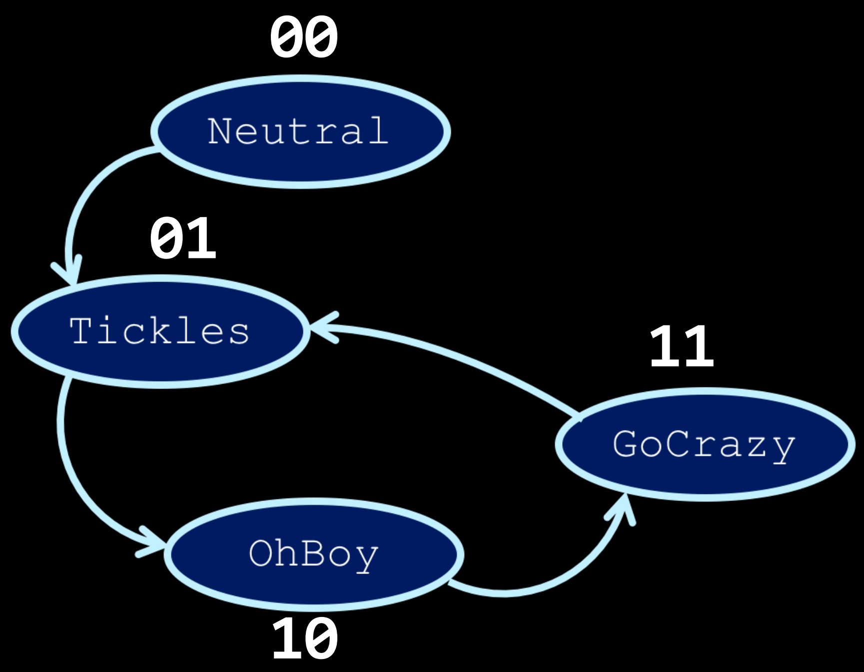




Question 2

Suppose we decided to go with the following flip flop assignment. And on the right side is the generated State Table.

Which row of the State Table is **wrong**?



	F_1F_0	F_1F_0
A	0 0	0 1
B	0 1	1 0
C	1 0	1 1
D	1 1	0 1

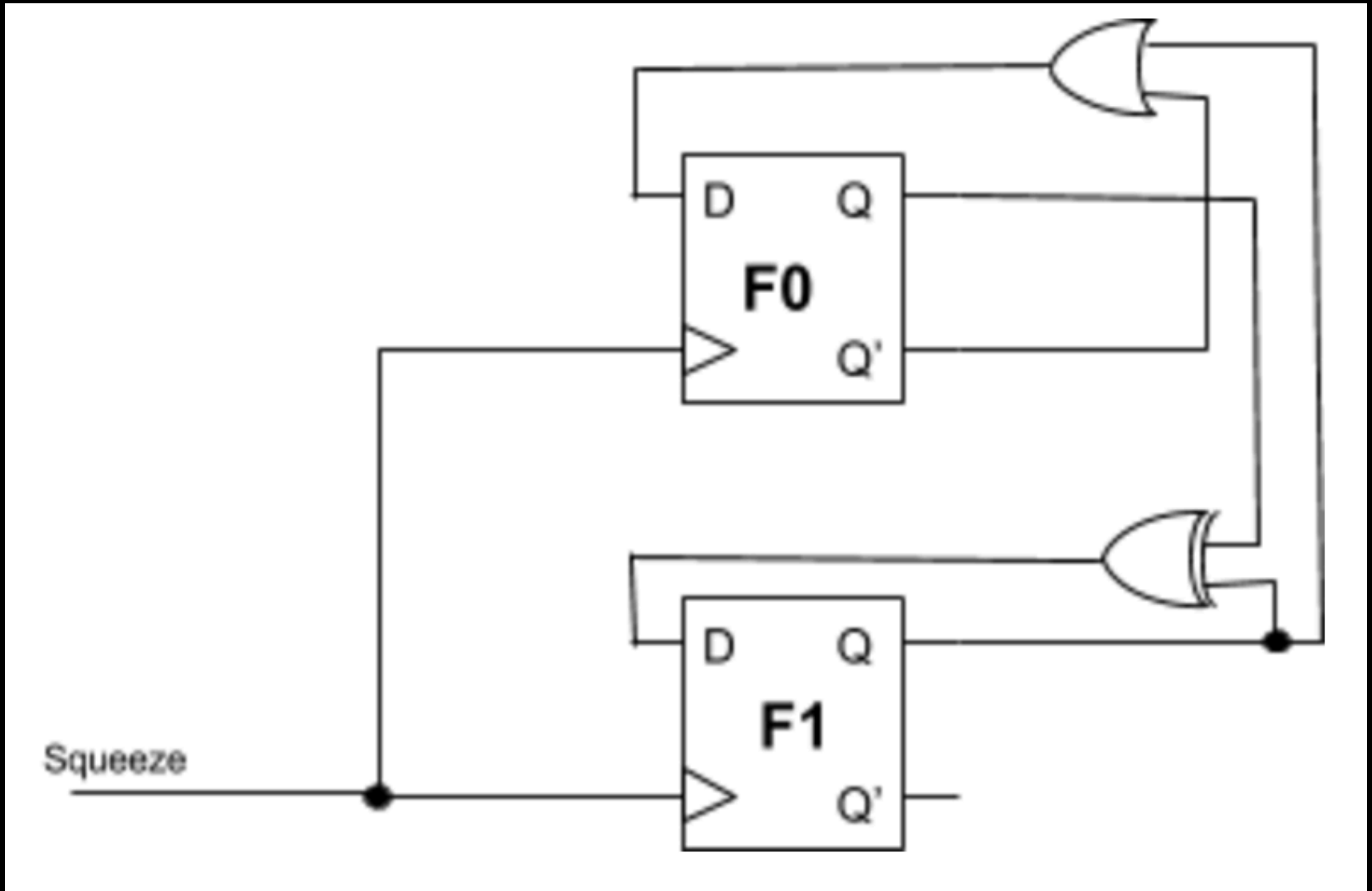
Question 3

Suppose the circuit expression we get from the State Table is the following:

F1 = F0 ⊕ F1

F0 = F1 + F0’

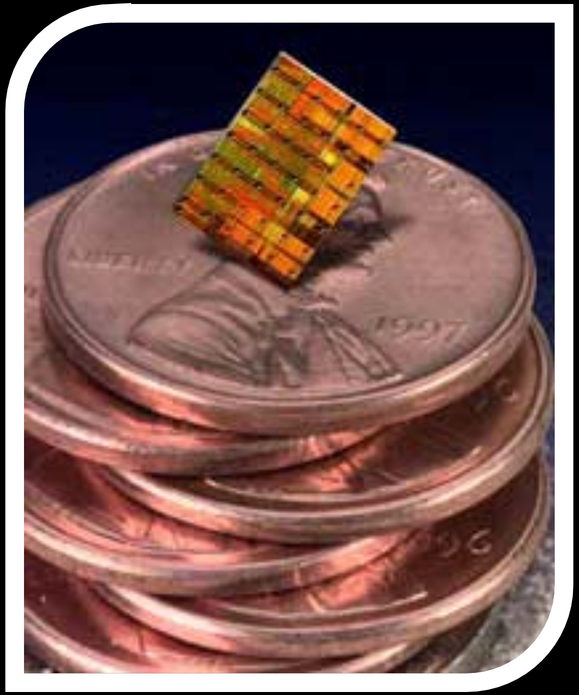
Complete the circuit.



New Topic: Processor Components

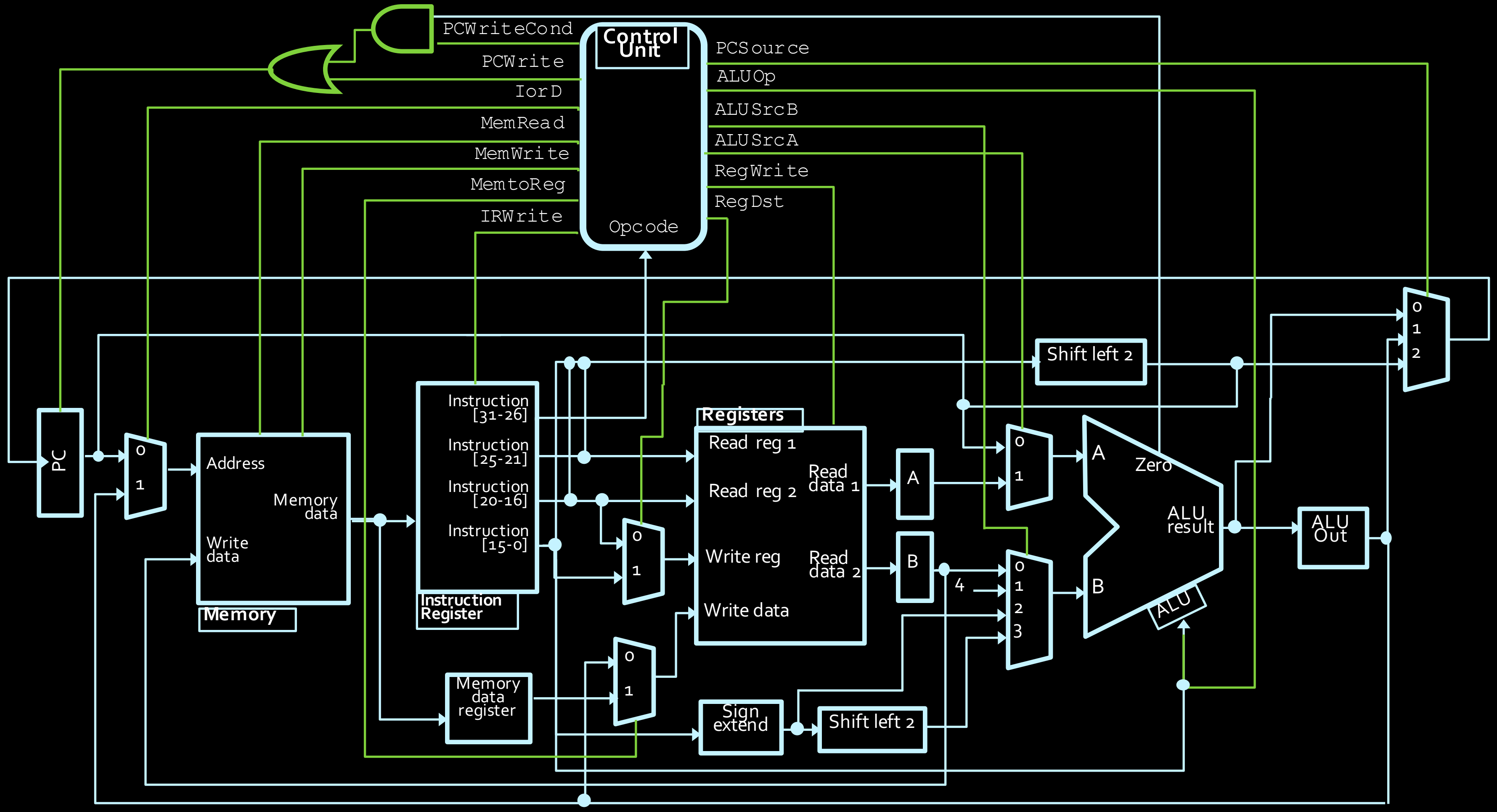
Using what we have learned so far (combinational logic, devices, sequential circuits, FSMs), how do we build a processor?

Microprocessors



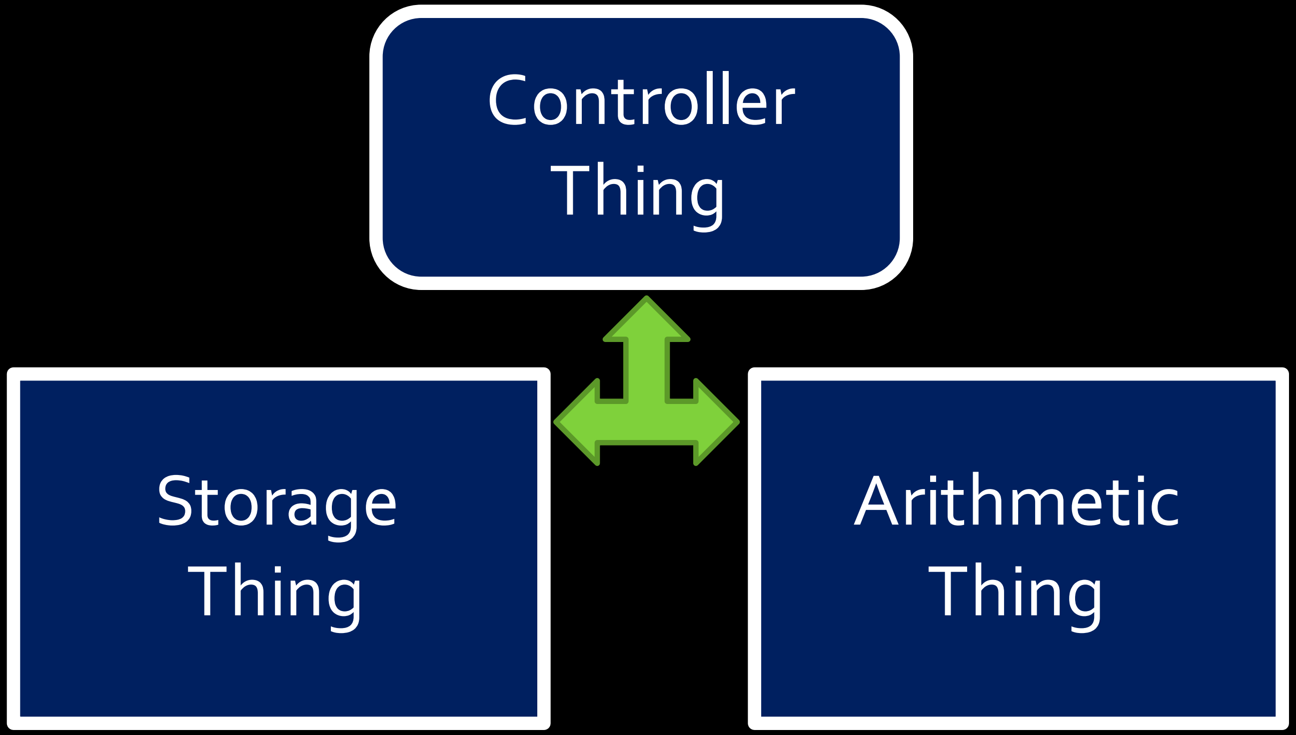
- So far, we've been talking about making devices, such as adders, counters and registers.
- The ultimate goal is to make a microprocessor, which is a digital device that processes input, can store values and produces output, according to a set of on-board instructions.

The Final Destination

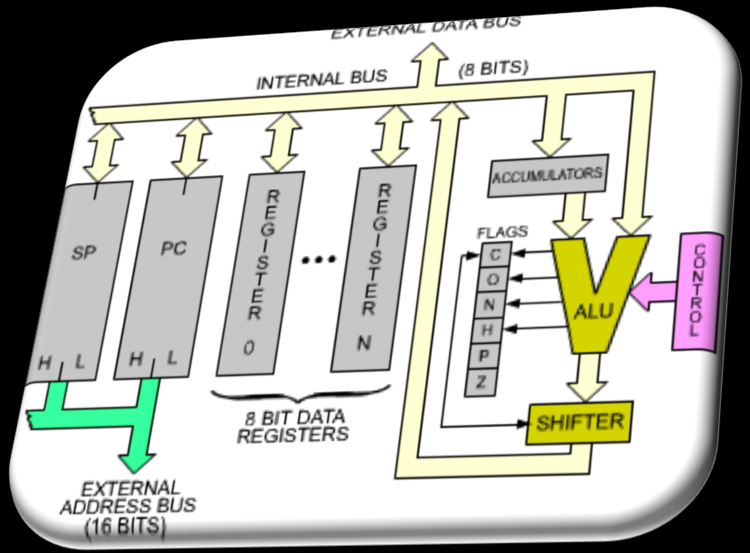


Deconstructing processors

- Processors aren't so bad when you consider them piece by piece:



Microprocessors

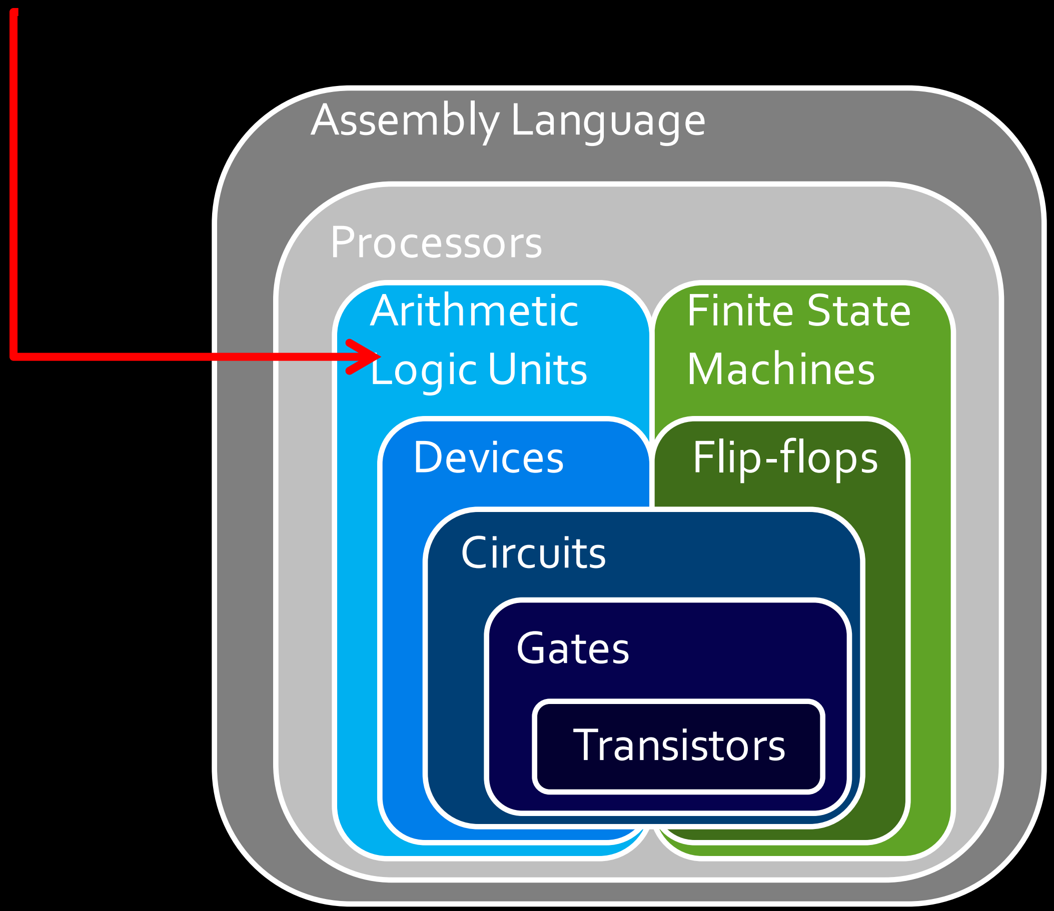


- These devices are a combination of the units that we've discussed so far:
  - Registers to store values.
  - Adders and shifters to process data.
  - Finite state machines to control the process.
- Microprocessors are the basis of all computing since the 1970's, and can be found in nearly every sort of electronics.

The “Arithmetic Thing”

aka: the Arithmetic Logic Unit (ALU)

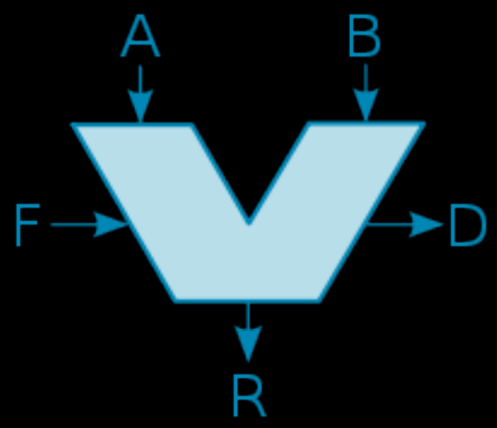
We are here



Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Gates
Transistors

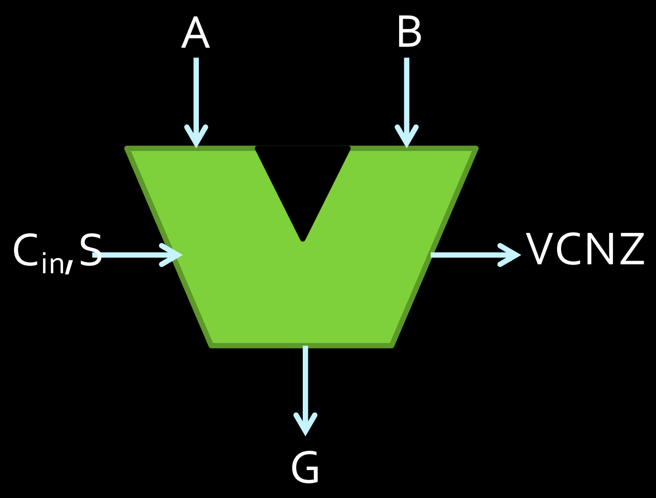
Arithmetic Logic Unit

- The first microprocessor applications were calculators.
  - Recall the unit on adders and subtractors.
  - These are part of a larger structure called the arithmetic logic unit (ALU).
- This larger structure is responsible for the processing of all data values in a basic CPU.



ALU inputs

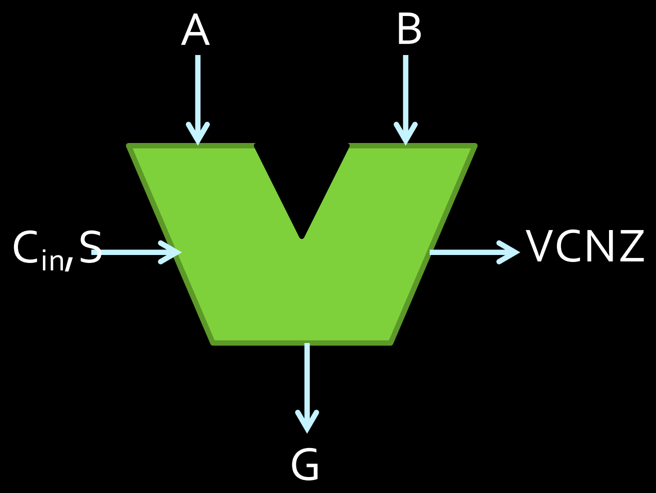
- The ALU performs all of the arithmetic operations covered in this course so far, and logical operations as well (AND, OR, NOT, etc.)



  - A and B are the oprands
  - The select bits (S) indicate which operation is being performed (S2 is a mode select bit, indicating whether the ALU is in arithmetic or logic mode).
  - The carry bit C_{in} is used in operations such as incrementing an input value or the overall result.

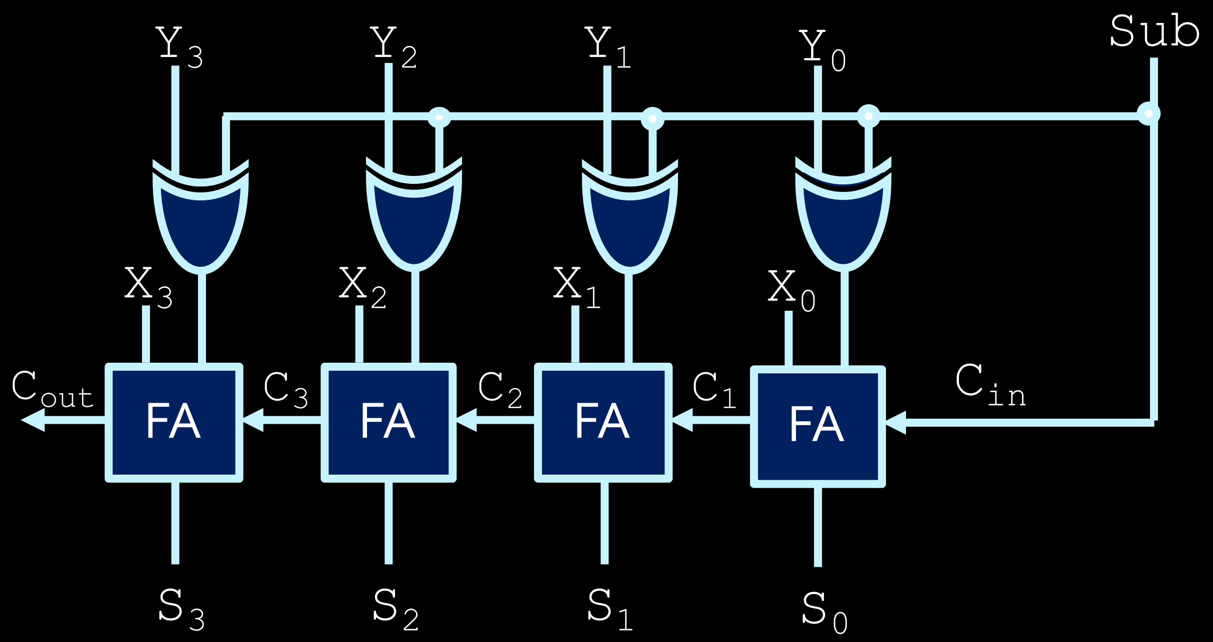
ALU outputs

- In addition to the input signals, there are output signals V, C, N & Z which indicate special conditions in the arithmetic result:
  - V: overflow condition
    - The result of the operation could not be stored in the n bits of G, meaning that the result is incorrect.
  - C: carry-out bit
  - N: Negative indicator
  - Z: Zero-condition indicator



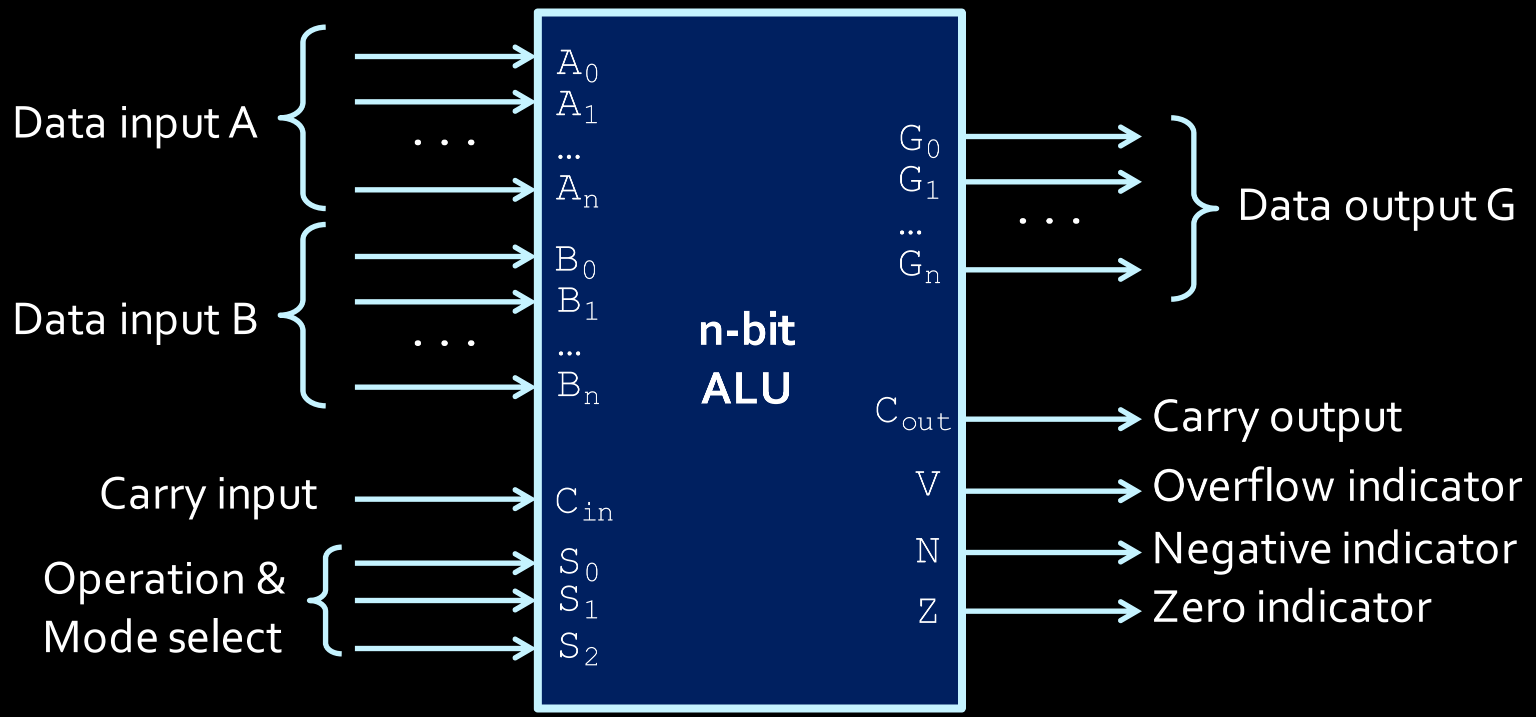
The "A" of ALU

- To understand how the ALU does all of these operations, let's start with the arithmetic side.
- Fundamentally, this side is made of an adder / subtractor unit, which we've seen already:

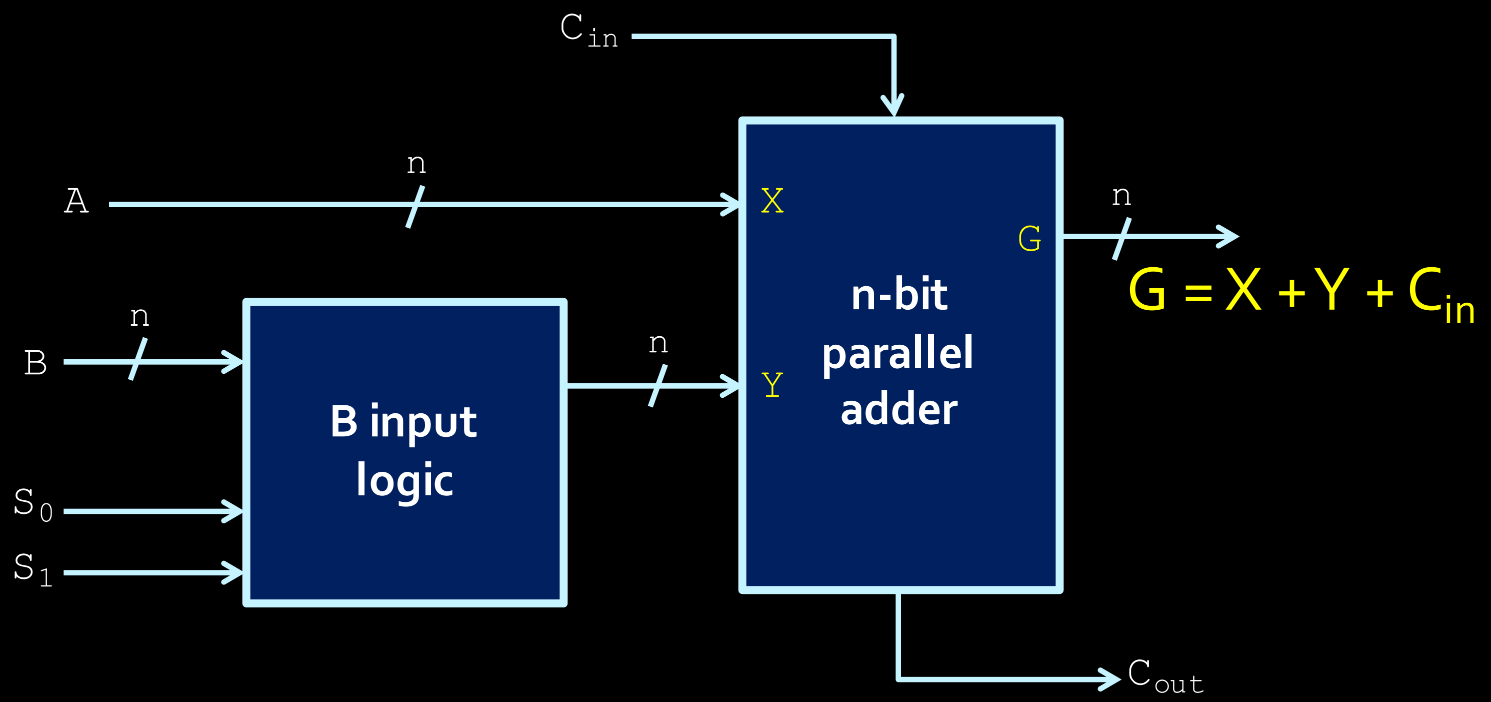


ALU block diagram

- In addition to data inputs and outputs, this circuit also has:
  - outputs indicating the different conditions,
  - inputs specifying the operation to perform (similar to `Sub`).

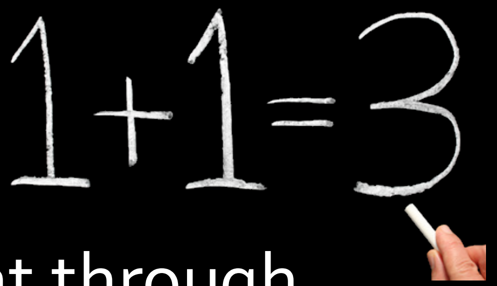


Arithmetic components



- In addition to addition and subtraction, many more operations can be performed by manipulating what is added to input A, as shown in the diagram above.

Arithmetic operations



- If the input logic circuit on the left sends B straight through to the adder, result is $G = A+B$
- What if B was replaced by all ones instead?
  - Result of addition operation: $G = A-1$
- What if B was replaced by $\overline{B}$?
  - Result of addition operation: $G = A-B-1$
- And what if B was replaced by all zeroes?
  - Result is: $G = A$. (Not interesting, but useful!)

→ Instead of a Sub signal, the operation you want is signaled using the select bits S_0 & S_1.

Operation selection

Select bits S_1	S_0	Y input	Result	Operation
0	0	All 0s	G = A	Transfer
0	1	B	G = A+B	Addition
1	0	$\overline{B}$	G = A+$\overline{B}$	Subtraction - 1
1	1	All 1s	G = A-1	Decrement

- This is a good start! But something is missing…
- Wait, what about the carry bit?

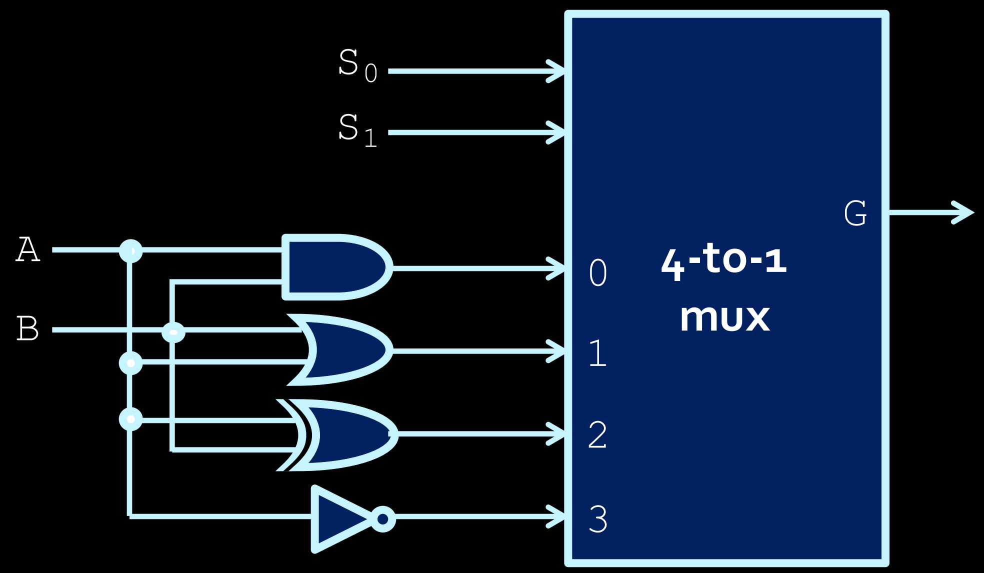
Full operation selection

Select		Input	Operation	
S_1	S_0	Y	$C_{in}=0$	$C_{in}=1$
0	0	All 0s	G = A (transfer)	G = A+1 (increment)
0	1	B	G = A+B (add)	G = A+B+1
1	0	$\overline{B}$	G = A+$\overline{B}$	G = A+$\overline{B}$+1 (subtract)
1	1	All 1s	G = A-1 (decrement)	G = A (transfer)

- Based on the values on the select bits and the carry bit, we can perform any number of basic arithmetic operations by manipulating what value is added to A.

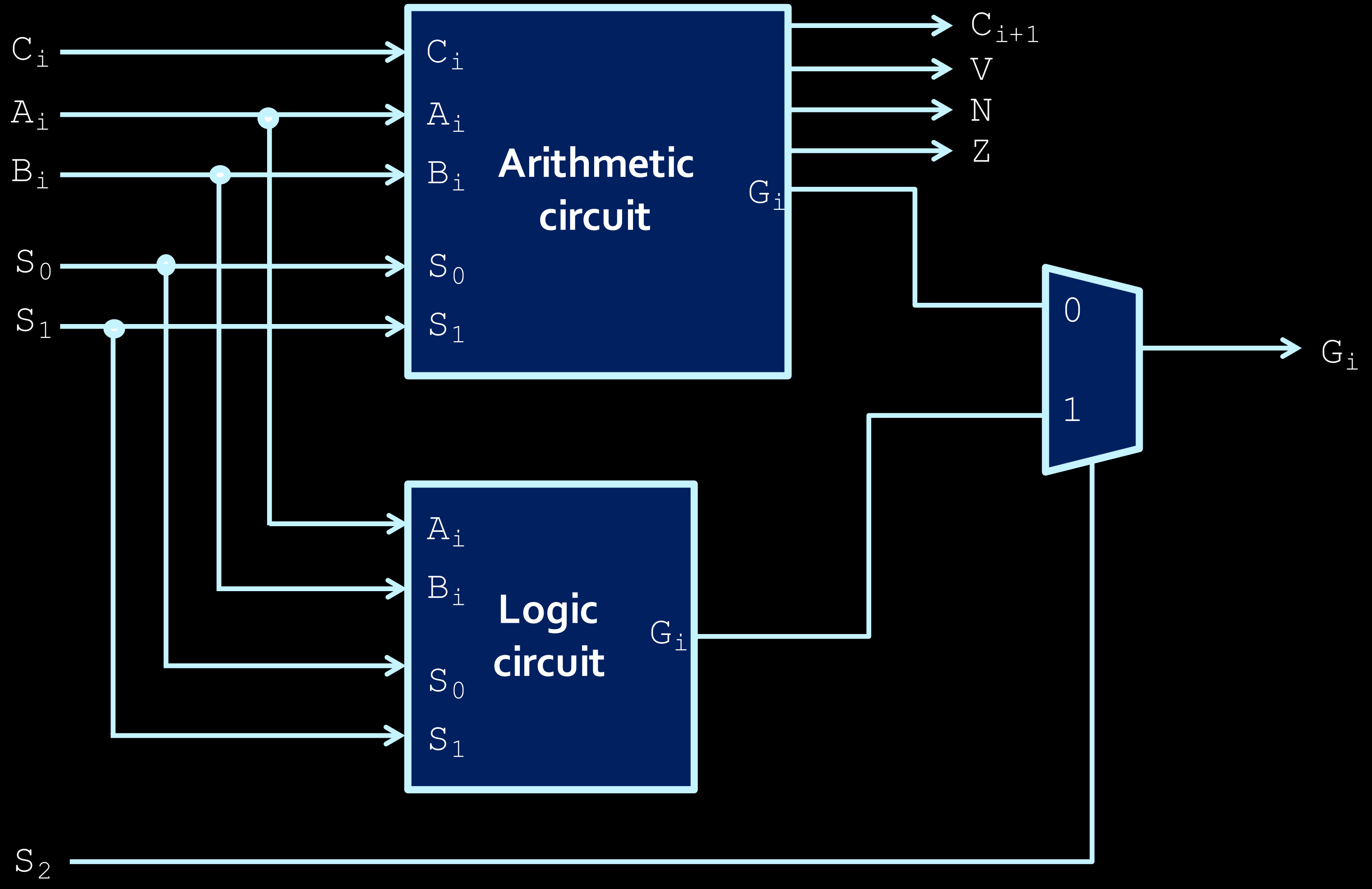
The "L" of ALU

- We also want a circuit that can perform **logical operations**, in addition to arithmetic ones.
- How do we tell which operation to perform?
  - Another select bit!



- If S_2 = 1, then logic circuit block is activated.
- Multiplexer is used to determine which block (logical or arithmetic) goes to the output.

Single ALU Stage

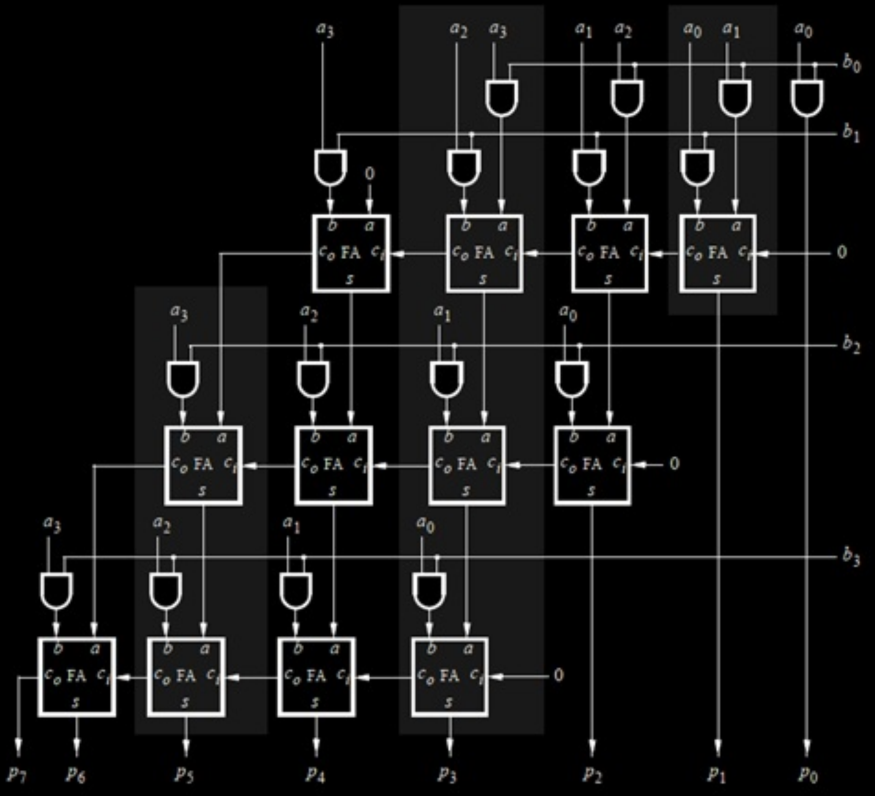


Multiplication

What about multiplication?

- Multiplication (and division) operations are always more complicated than other arithmetic (plus, minus) or logical (AND, OR) operations.
- Three major ways that multiplication can be implemented in circuitry:
  - Layered rows of adder units.
  - An adder/shifter circuit
  - Booth's Algorithm

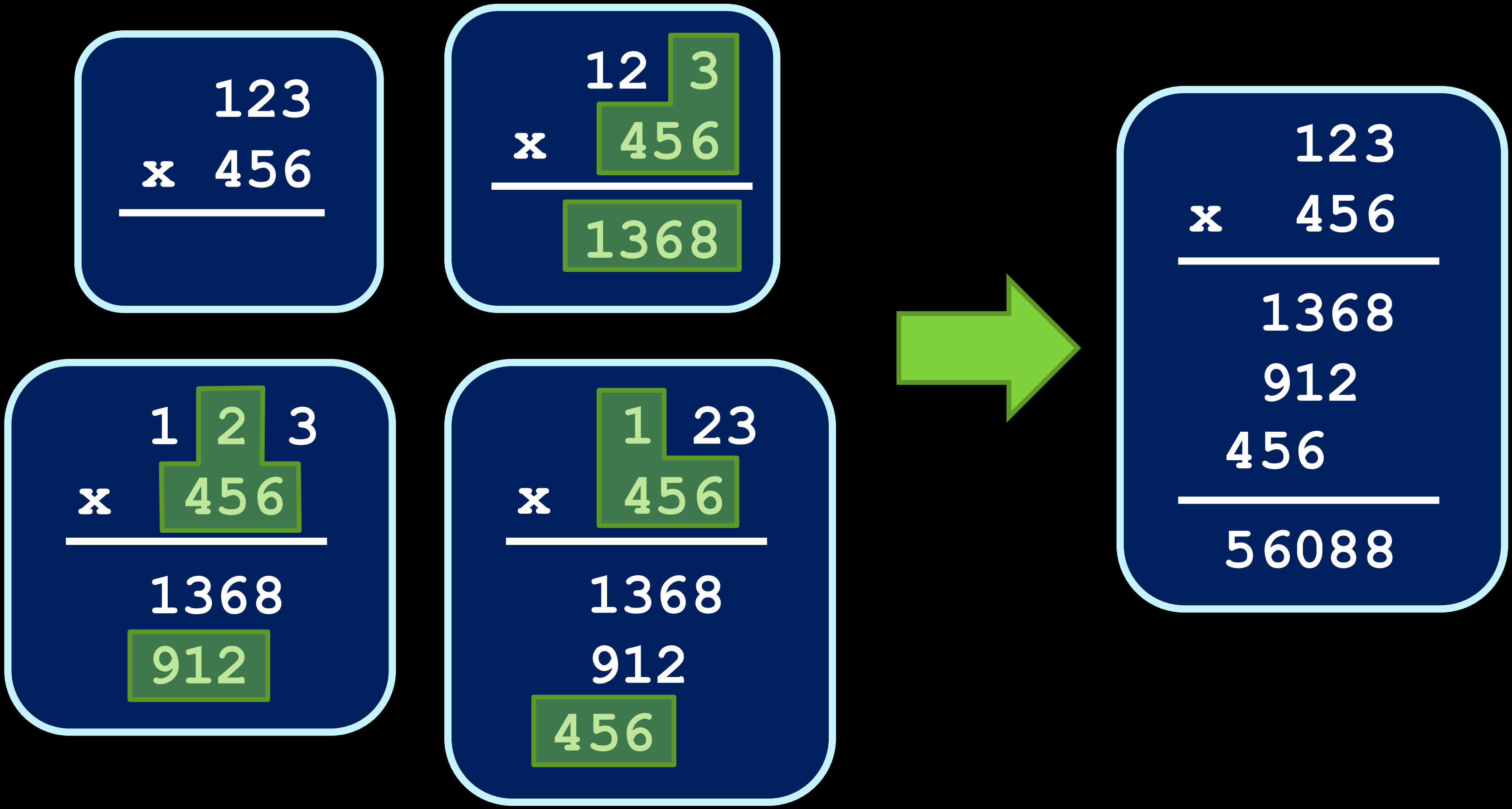
Multiplication



- Multiplier circuits can be constructed as an array of adder circuits.
- This can get a little expensive as the size of the operands grows.
- Is there an alternative to this circuit?

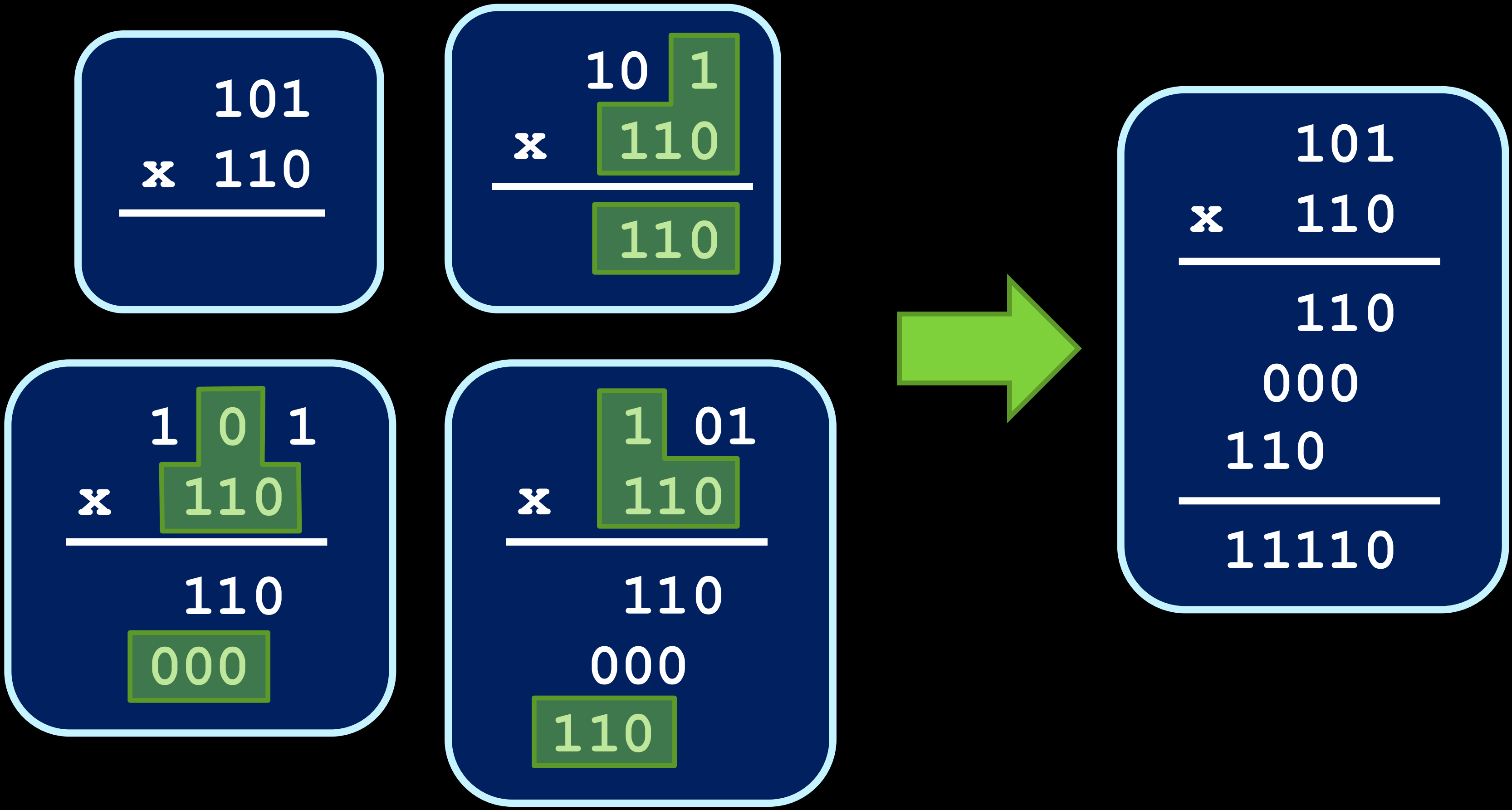
Multiplication

- Revisiting grade 3 math…



Multiplication

- And now, in binary…

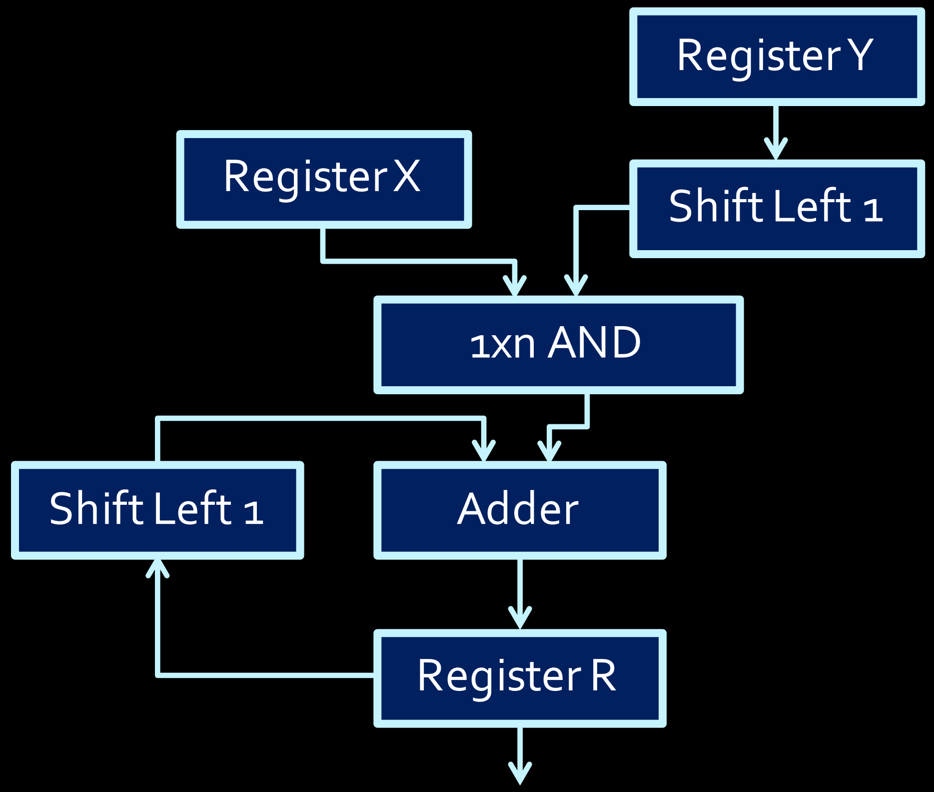


Observations

- Calculation flow
  - Multiply by 1 bit of multiplier
  - Add to sum and shift sum
  - Shift multiplier by 1 bit
  - Repeat the above
- What is "multiply by 1 bit of binary"?
  - `10101 x 1 ?`
  - `10101 x 0 ?`
  - `It's an` **`AND`**`!`

Accumulator circuits

- What if you could perform each stage of the multiplication operation, one after the other?
  - This circuit would only need a single row of adders and a couple of shift registers.



Make it more efficient

Think about 258 x 9999

- Multiply by 9, add to sum, shift, multiply by 9, add to sum, shift, multiple by 9, add to sum, shift, multiply by 9, add to sum.

- 258 x 9999 = 258 x (10000 - 1) = 258 x 10000 – 258
- Just shift 258, becomes 2580000, then do 2580000 – 258
- More efficient!

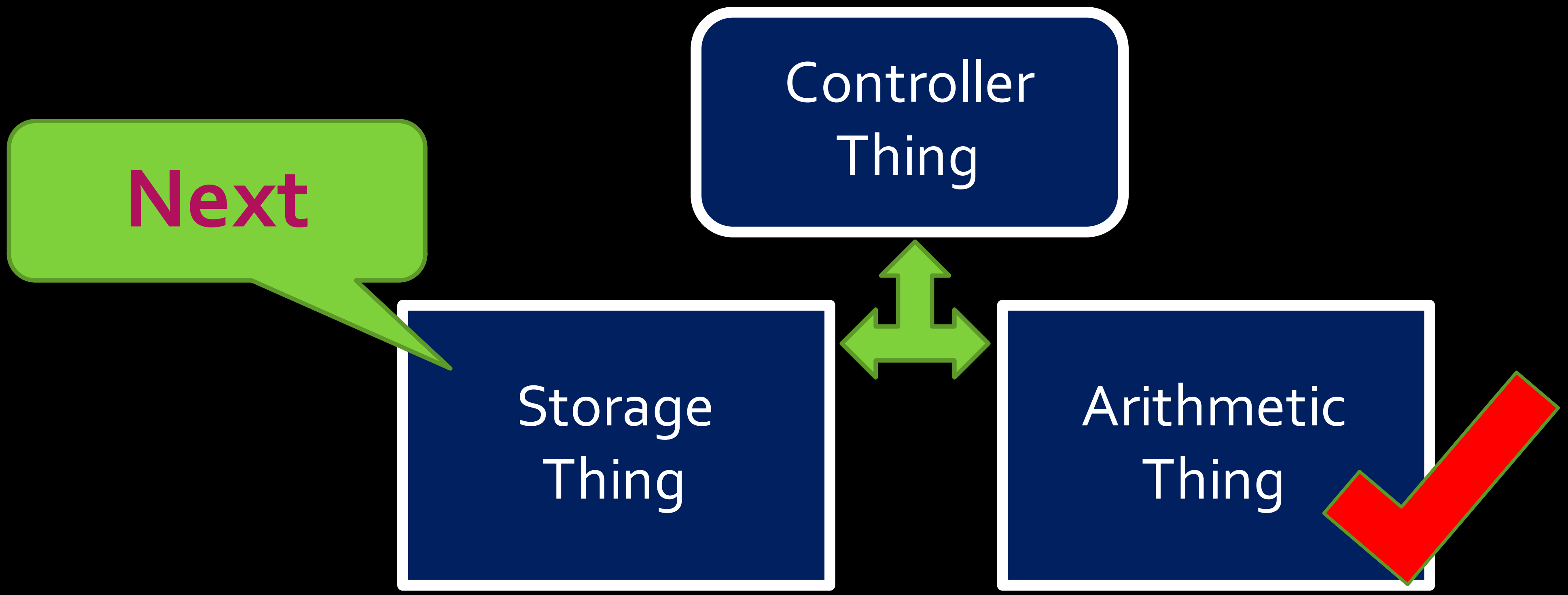
More efficient: Booth's Algorithm

- Take advantage of circuits where shifting is cheaper than adding, or where space is at a premium.
  - when multiplying by certain values (e.g. `99`), it can be easier to think of this operation as a difference between two products.
- Consider the shortcut method when multiplying a given decimal value `X` by `9999`:
  - `X*9999 = X*10000 - X*1`
- Now consider the equivalent problem in binary:
  - `X*001111 = X*010000 - X*1`
- More details: https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm

Reflections on multiplication

- Multiplication isn't as common an operation as addition or subtraction, but occurs enough that its implementation is handled in the hardware.

- Most common multiplication and division operations are powers of 2. For this, the shift register is used instead of the multiplier circuit.


Next
Controller Thing
Storage Thing
Arithmetic Thing

Midterm Review

Time & Location

Monday, Feb 22, 3:10pm to 4:00pm

No aid.

Bring your TCard

Types of questions

- Short answer: basic understanding
- Circuit analysis: given a circuit understand it
- Circuit design: given a requirement, design a circuit

How to study for midterm

1. Review slides
2. Review what you did for labs
3. Review quizzes
4. Practice with past test
    - Posted on course web page with solutions
    - Ignore Verilog questions
5. Whenever confused, ask on Piazza or go to office hours

Office hours in Reading Week:

No office hour on Monday (Family Day)

Tuesday as usual: 5pm ~ 6:30pm

Friday from 3pm ~ 6:30pm