FIT1047 – Week 3

Central Processing Units

Recap

In weeks 1 and 2 we saw

- Number systems, binary
- Boolean logic
- Basic gates

Now let's put them together to build a computer.

Overview

- CPUs
- Machine code and assembly language
- Combinational Circuits
  - Adders

CPUs

A central processing unit is the "brain" of a computer.

- built out of logic gates
- executes instructions
- connected to memory and I/O devices

Programs

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

Programs

```
import sys
name = sys.argv[1]
print 'Hello, ' + name + '!'
```

Programs

```
int: n;
array[0..n-1] of var 0..n: s;
constraint forall(i in 0..n-1) (
  s[i] = sum(j in 0..n-1)(s[j]=i)
);
solve satisfy;
```

Programs

None of these can be executed directly by the CPU!

They are compiled or interpreted.

The CPU can only execute machine code.

Machine Code

- A very simple computer language.
- Different for each CPU architecture.
- Programs = sequences of instructions.
  - stored in memory
  - each instruction is encoded into one or more words

Machine Code

Example memory contents:

```
0001000000000100
0011000000000101
0010000000000110
0111000000000000
0000000010001110
0000110110000000
0000000000000000
```

- each line is one binary 16-bit word
- could be a program, could be data!

Machine Code

The instructions that a particular type of CPU understands are called the

Instruction Set Architecture (ISA).

What does a CPU need to be able to do?

- do some maths (add, subtract, multiply, compare)
- move data between memory, CPU and I/O
- execute conditionals and loops

MARIE: A simple CPU

Very basic machine architecture:

- 16 bit words
- only 16 instructions
- one general purpose register
- instructions are composed of an opcode (4 bits) and an address (12 bits)

opcode	address
0001	000110001110

See Moodle for MARIE simulator link.

MARIE: Registers

A register is a storage cell for temporary data.

- AC (Accumulator): a "general purpose" register.
- MAR (Memory Address Register): stores memory addresses.
- MBR (Memory Buffer Register): holds data read from or written to memory.
- IR (Instruction Register): current instruction.
- PC (Program Counter): address of next instruction.

Assembly language

Machine code is hard to write and read.

What does 0010000000000110 mean?

So we use assembly language:

- one mnemonic per machine instruction
- can be translated 1:1 into machine code

MARIE assembly

(There's a few missing, we'll cover them later.)

Opcode	Instruction
0001	Load X
0010	Store X
0011	Add X
0100	Subt X
0111	Halt
1000	Skipcond
1001	Jump X

MARIE programming

Let's write a small program adding two numbers.

Pseudocode:

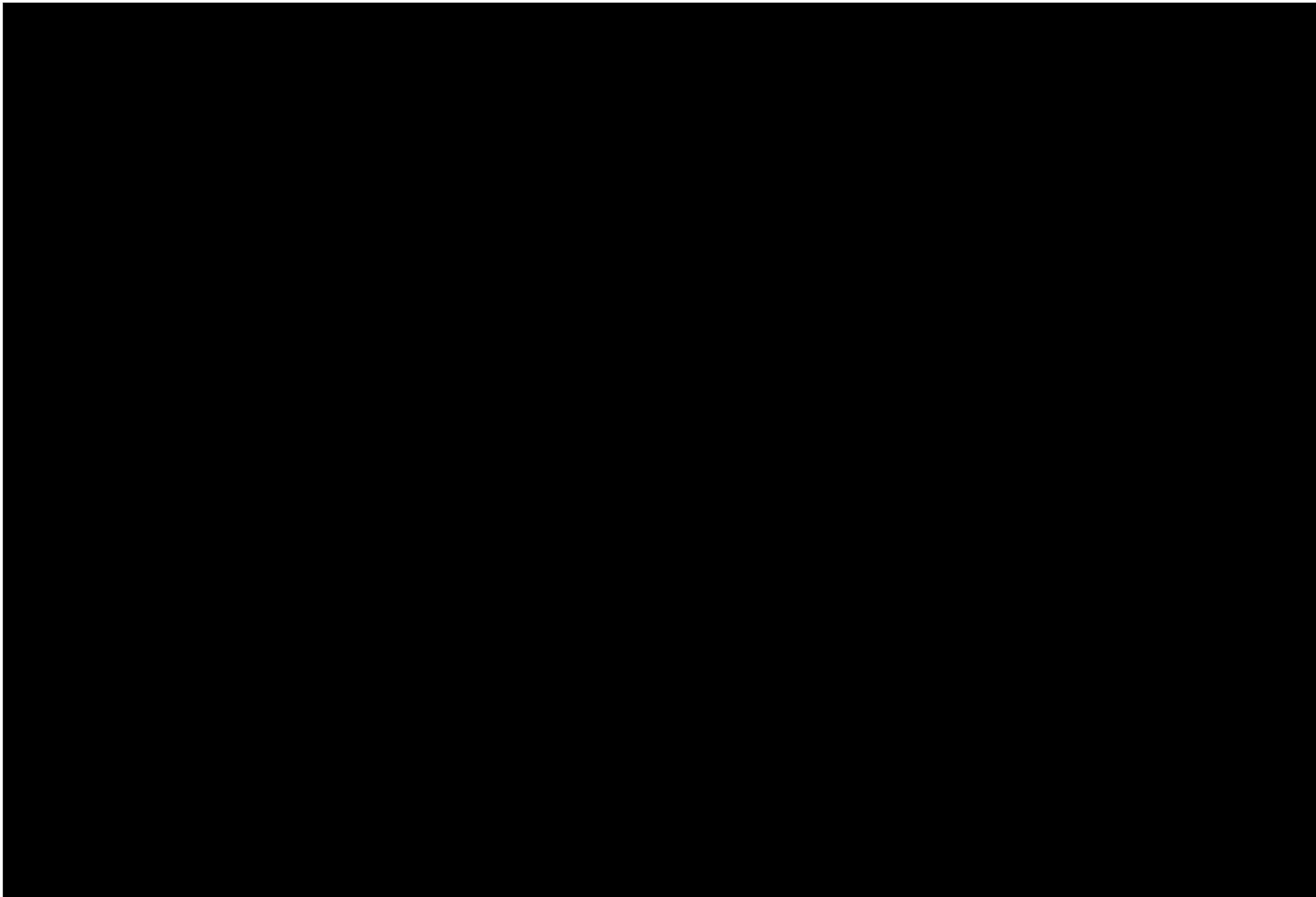
1. Load first number from memory into AC
2. Add second number from memory to AC
3. Store result from AC into memory
4. Stop execution.

MARIE programming

Let's write a small program adding two numbers.

Address	Instruction	Memory contents
0x000	Load 0x004	0001000000000100
0x001	Add 0x005	0011000000000101
0x002	Store 0x006	0010000000000110
0x003	Halt	0111000000000000
0x004	142	0000000010001110
0x005	3456	0000110110000000
0x006	0	0000000000000000

Note: program and data both reside in memory!



MARIE instructions

Let's go through the most important instructions.

We will use Register Transfer Language (RTL) to specify what each instruction means.

MARIE instructions

Opcode	Instruction
0001	Load X
0010	Store X
0011	Add X
0100	Subt X
0111	Halt
1000	Skipcond
1001	Jump X

Load data from address X into AC.

MAR ⟵ X
MBR ⟵ M[MAR]
AC ⟵ MBR

MARIE instructions

Opcode	Instruction
0001	Load X
0010	Store X
0011	Add X
0100	Subt X
0111	Halt
1000	Skipcond
1001	Jump X

Store data in AC into address X.

```
MAR    ⟵ X
MBR    ⟵ AC
M[MAR] ⟵ MBR
```

MARIE instructions

Opcode	Instruction
0001	Load X
0010	Store X
0011	Add X
0100	Subt X
0111	Halt
1000	Skipcond
1001	Jump X

Add data in address X to AC. Store result in AC.

MAR ⟵ X
MBR ⟵ M[MAR]
AC ⟵ AC + MBR

MARIE instructions

Opcode	Instruction
0001	Load X
0010	Store X
0011	Add X
0100	Subt X
0111	Halt
1000	Skipcond
1001	Jump X

Subtract data in address X from AC, store result in AC.

```
MAR ⟵ X
MBR ⟵ M[MAR]
AC  ⟵ AC - MBR
```

MARIE instructions

Opcode	Instruction
0001	Load X
0010	Store X
0011	Add X
0100	Subt X
0111	Halt
1000	Skipcond
1001	Jump X

Stop execution.

MARIE instructions

Opcode	Instruction
0001	Load X
0010	Store X
0011	Add X
0100	Subt X
0111	Halt
1000	Skipcond
1001	Jump X

If

- bits 10 and 11 in IR are 00 and AC < 0
- or bits 10 and 11 in IR are 01 and AC = 0
- or bits 10 and 11 in IR are 10 and AC > 0

then

PC ⟵ PC + 1

MARIE instructions

Opcode	Instruction
0001	Load X
0010	Store X
0011	Add X
0100	Subt X
0111	Halt
1000	Skipcond
1001	Jump X

Continue execution at address X.

PC ⟵ X

Constructing a MARIE CPU

Circuits required to build a MARIE CPU:

- Perform simple maths (addition, subtraction etc)
- Store and load data in registers and memory
- Fetch, decode and execute instructions

Let's start with the basics.

Combinational Circuits

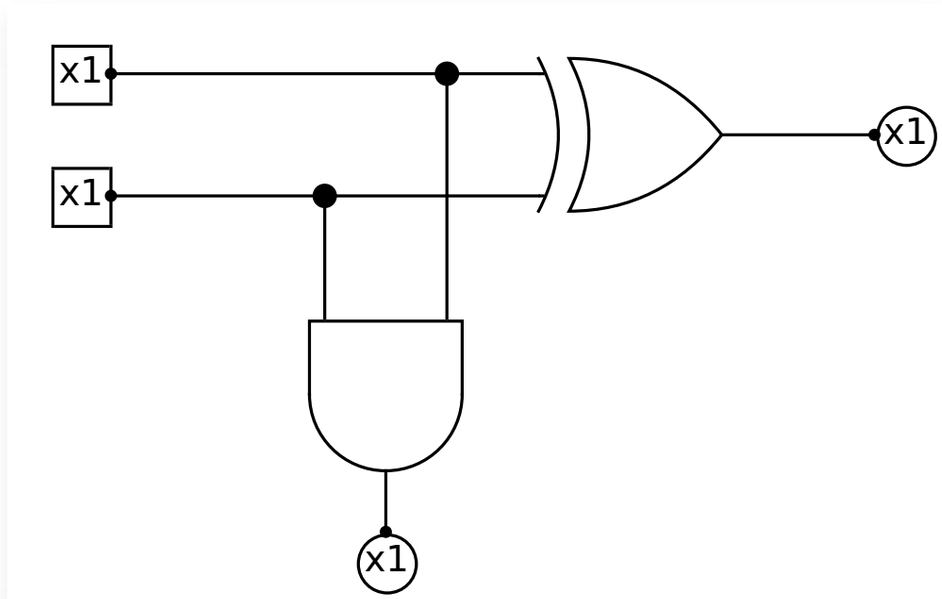
(output is a function of the inputs)

Adders

Here is the truth table for adding two bits:

Bit 1	Bit 2	Result	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Let's try this in Logisim.



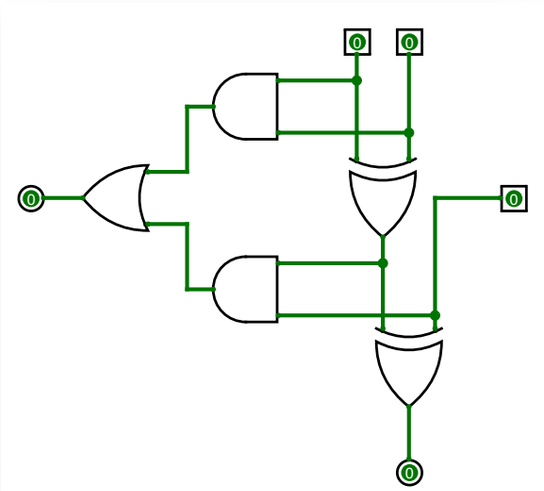
This is called a half-adder.

But what if we have an input carry?

Full Adders

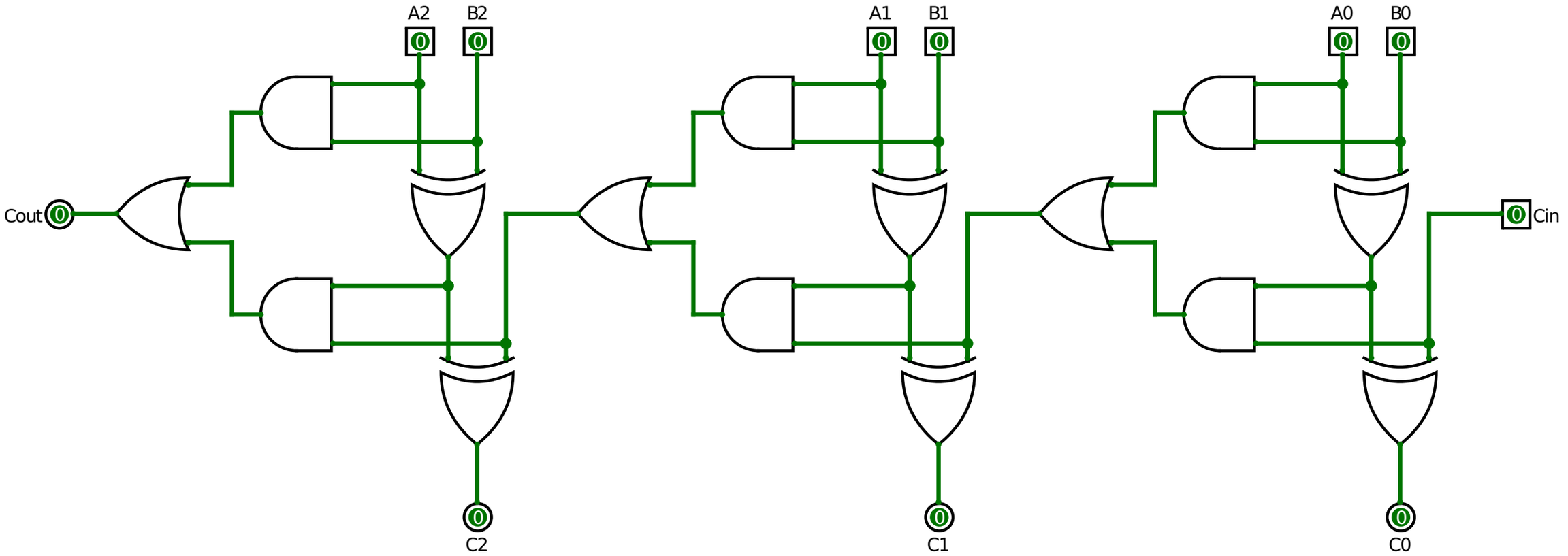
Adding three bits.

Bit 1	Bit 2	Carry In	Result	Carry Out
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1



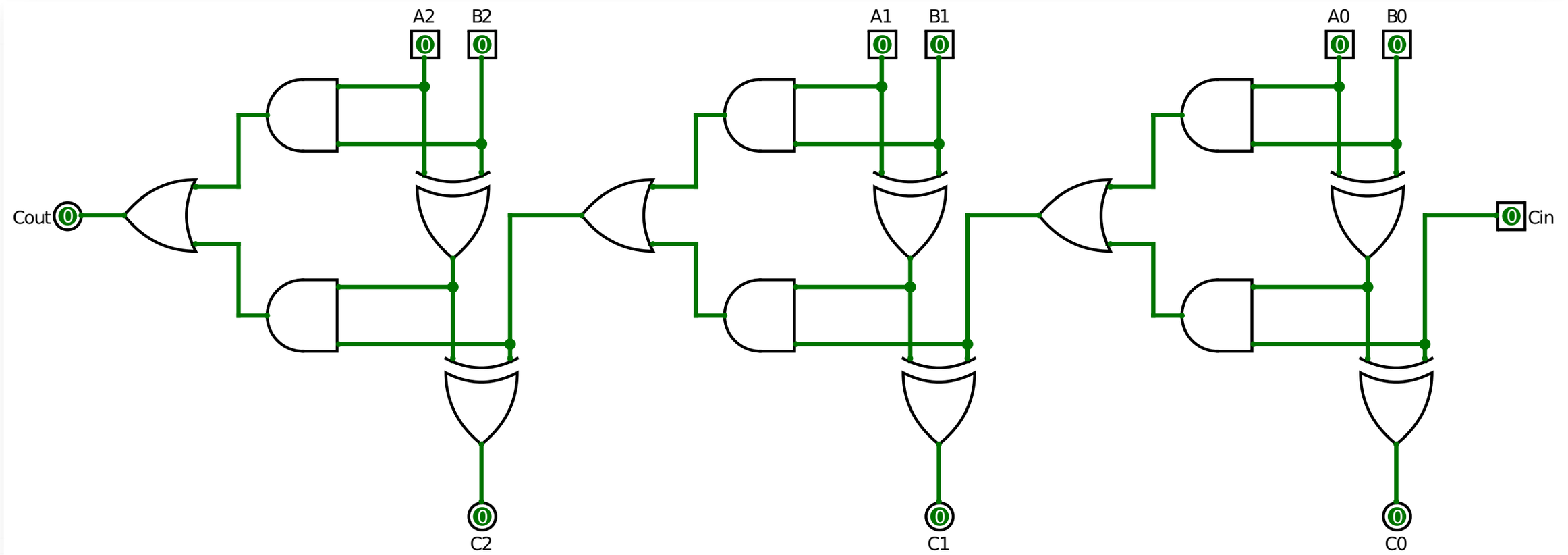
Ripple-Carry Adder

Join carry-out of several 1-bit-adders!



Delay

Gates require time to propagate the signals.



For n bit ripple carry: $2n + 1$

(There are more efficient adders.)

Tutorials this week

- Programming MARIE
- Circuits for adding and subtracting