FIT1047 - Week 3

Central Processing Units

(part 2)

Recap

In the previous lecture we saw

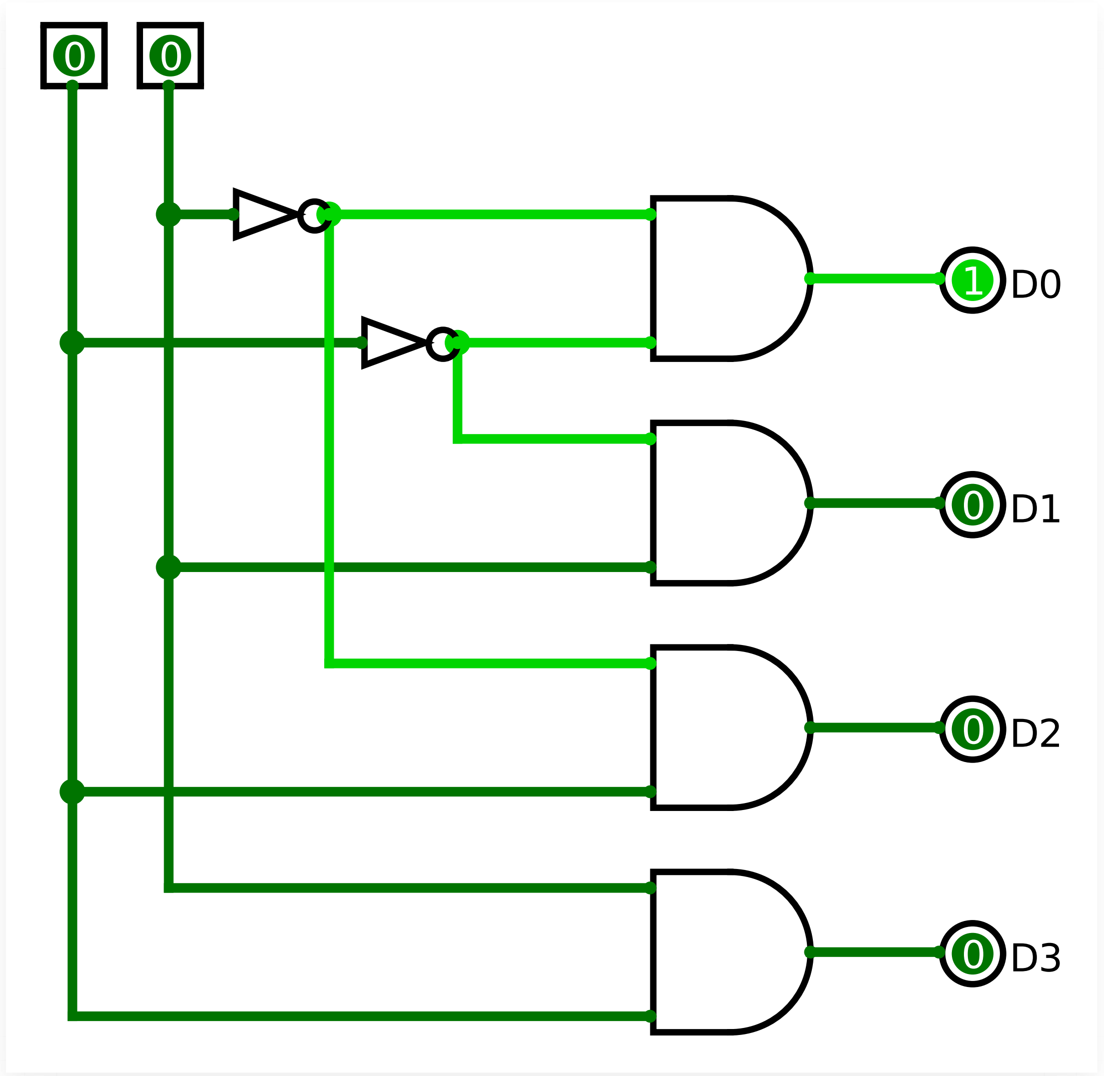
- Basic CPU architecture
- MARIE assembly code
- Combinational circuits (adders)

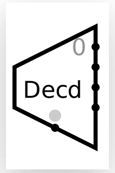
Overview

- Arithmetic/Logic Units (ALUs)
- Sequential Circuits
  - flip flops, registers, counters
  - memory
- Control
  - executing a program

Decoders

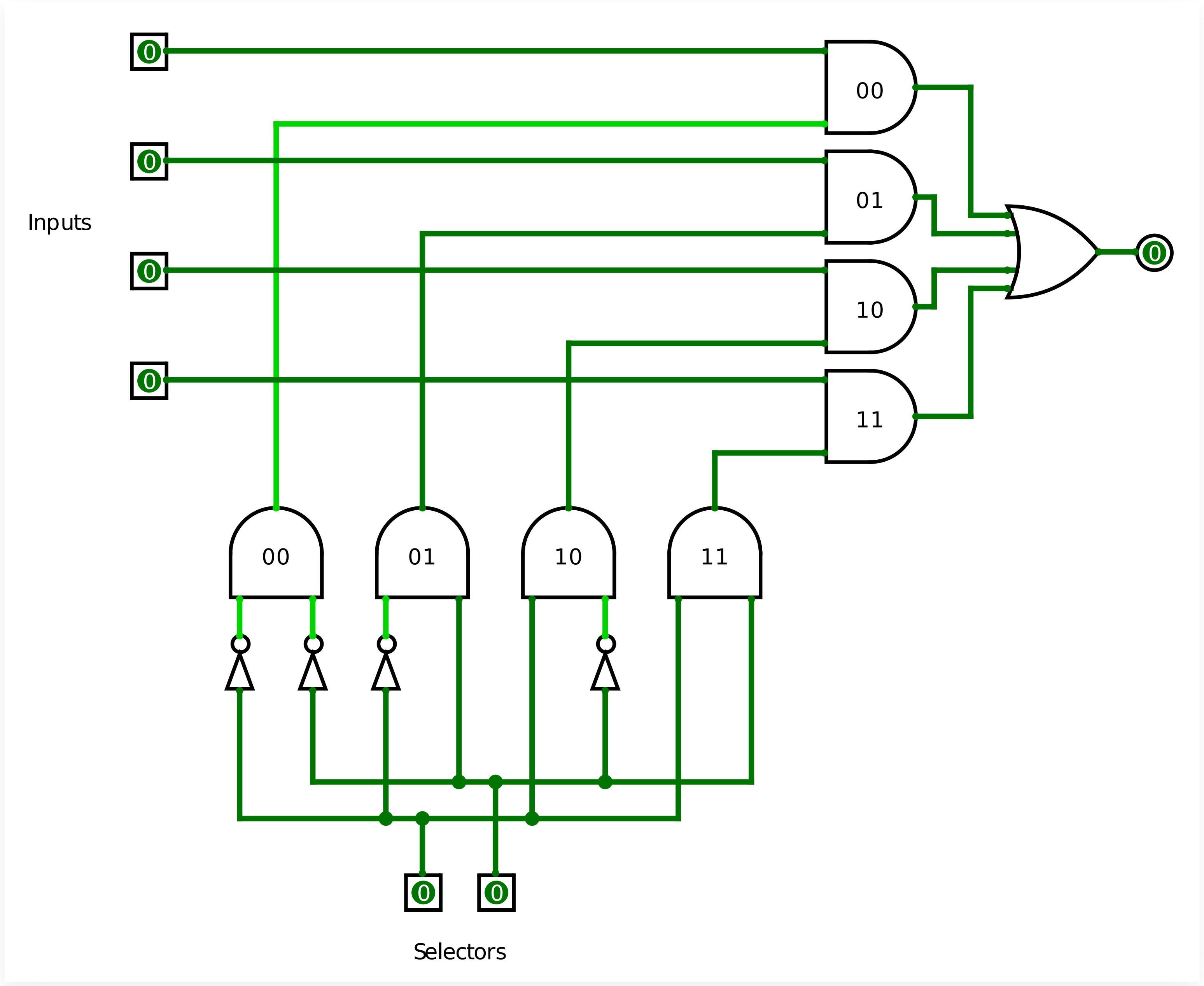
Activate one output based on a binary number.

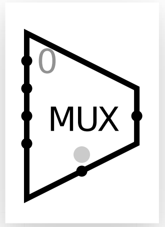




Multiplexers

Select one of several inputs.





Arithmetic Logic Unit

(ALU)

ALU

Implements basic computations:

- integer addition, subtraction (multiplication, negation)
- comparisons
- bitwise Boolean operations (and, or, not)
- shifting

Inputs:

- two n-bit operands
- op-code (which operation?)

Outputs:

- n-bit result and status flags (overflow? error?)

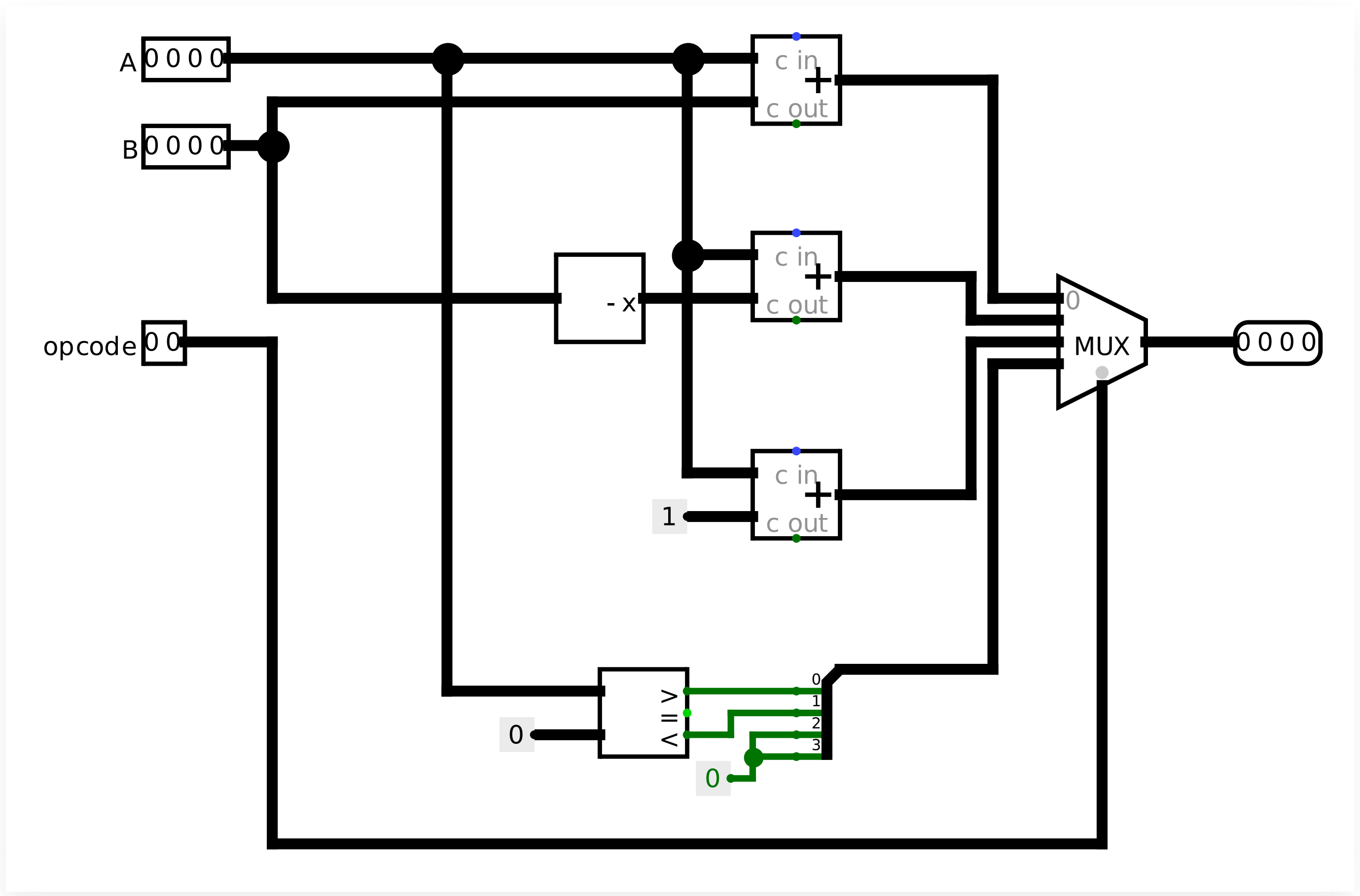
ALU

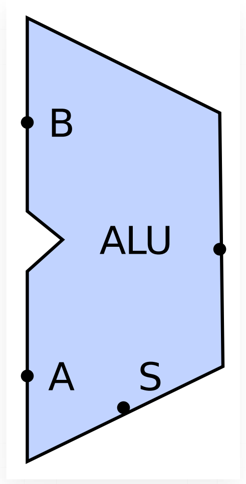
How does the ALU decide which operation to use?

- Simply do all in parallel
- And then only use the one prescribed by the op-code

Sounds like a job for a MUX!

ALU





Sequential Circuits

(output depends on sequence of inputs)

Sequences

How can a circuit remember the past?

Feed the output back into the input!

Let's add a switch to control the state.

Sequences

How can a circuit remember the past?

Feed the output back into the input!

Can we implement a toggle?

Sequences

This is called an SR latch (set-reset-latch).

SR Latch

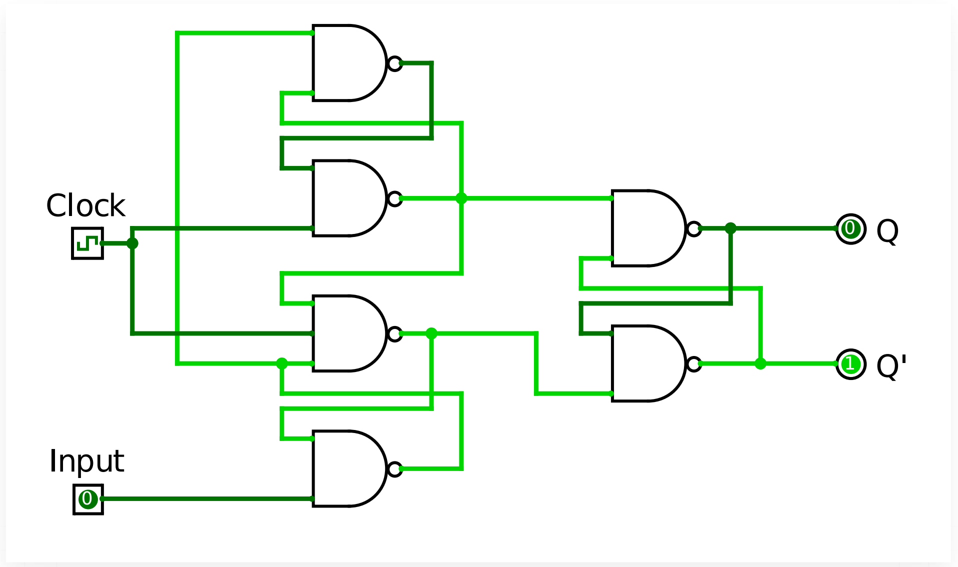
Truth table:

R	S	Q(t)	Q(t+1)
0	0	0	0
0	1	0	1
1	0	0	0
1	1	forbidden	
0	0	1	1
0	1	1	1
1	0	1	0
1	1	forbidden	

But digital circuits use a single input (bit).

D flip-flop

- One input: the data to be stored
- One output: the data currently stored
- Plus a clock: state only changes on "positive edge"



Registers

- Very fast memory inside the CPU
- Some special purpose registers
  - PC, IR, MBR, MAR (for MARIE)
- Some general purpose registers
  - AC (MARIE), AH/AL, BH/BL, CH/CL, DH/DL (x86)
- Fixed bit width
  - e.g. 16 bits in MARIE, 16/32 or 64 bits in modern processors

Register file

- Collection of registers
- Each implemented using n flip-flops (for n bits)
- n inputs and outputs
- Additional input: select register for writing
- Additional input: select register for reading

Register file

MARIE Data Paths

Register Transfer Language (RTL)

- Break down instructions into small steps
- CPU will perform one of these steps per cycle
- Transfer data between registers and/or memory
- Fetch cycle has four steps:

1: MAR ← PC

2: MBR ← M[MAR]

3: IR ← MBR

4: PC ← PC+1

RTL: Decode

Decoding has zero, one or two steps:

5: MAR ← X

6: MBR ← M[MAR]

RTL: Execute

Depends on the instruction (of course).

Example: Add X

7: AC ← AC + MBR

Example: Jump X

6: PC ← MAR

RTL: Add X

Full Add X instruction:

1: MAR ← PC

2: MBR ← M[MAR]

3: IR ← MBR

4: PC ← PC+1

5: MAR ← X

6: MBR ← M[MAR]

7: AC ← AC + MBR

Control

(the machine)

Control what?

- Fetch - decode - execute cycle
- Implement the RTL code for each instruction
  - RTL = Register transfer language
- Generate control signals for individual circuits

Control Signals

- "Wires" that switch components on or off, or select certain components
  - opcode for the ALU
  - read/write register number for the register file
  - memory read/write

Control Signals

Register read: P2,P1,P0

- None=000, MAR=001, PC=010, MBR=011, AC=100, IR=111

Register write: P5,P4,P3

- None=000, MAR=001, PC=010, MBR=011, AC=100, IR=111

Memory read, memory write: Mr, Mw

ALU operation: A2,A1,A0

- None=000, add=010, subt=001, clear=011, increment=100

Control Signals

Signals for Add X:

```
1. MAR ← PC                 P3    P1
2. MBR ← M[MAR]             P4 P3          Mr
3. IR ← MBR           P5 P4 P3    P1 P0
4. PC ← PC+1             P4       P1          A2
5. MAR ← X                  P3 P2 P1 P0
6. MBR ← M[MAR]             P4 P3          Mr
7. AC ← AC + MBR      P5          P2              A1
```

Next lectures

- Control
- More MARIE instructions
- Memory
- Interrupts
- Input/Output