FIT1047 - Week 3

Central Processing Units

(part 2)

Recap

In the previous lecture we saw

- Basic CPU architecture
- MARIE assembly code
- Combinational circuits (adders, MUXes, decoders)
- ALUs

Overview

- Sequential Circuits
  - flip flops, registers, counters
  - memory
- Control
  - executing a program

Sequential Circuits

Sequential Circuits

(output depends on sequence of inputs)

Sequences

How can a circuit remember the past?

Sequences

How can a circuit remember the past?

Feed the output back into the input!

Sequences

How can a circuit remember the past?

Feed the output back into the input!

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!

Sequences

How can a circuit remember the past?

Feed the output back into the input!

Sequences

How can a circuit remember the past?

Feed the output back into the input!

Can we implement a toggle?

Sequences

Sequences

Sequences

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

SR Latch

Truth table:

S	R	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	

SR Latch

Truth table:

S	R	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	

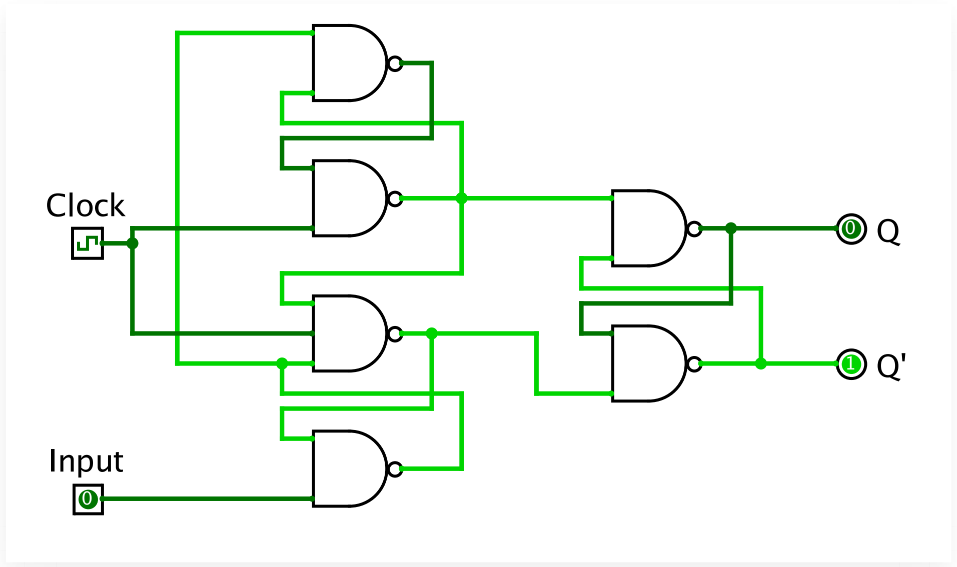
But digital circuits use a single input (bit).

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"

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

Registers

- Very fast memory inside the CPU

Registers

- Very fast memory inside the CPU
- Some special purpose registers
  - PC, IR, MBR, MAR (for MARIE)

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)

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

Register file

- Collection of registers

Register file

- Collection of registers
- Each implemented using n flip-flops (for n bits)

Register file

- Collection of registers
- Each implemented using n flip-flops (for n bits)
- n inputs and outputs

Register file

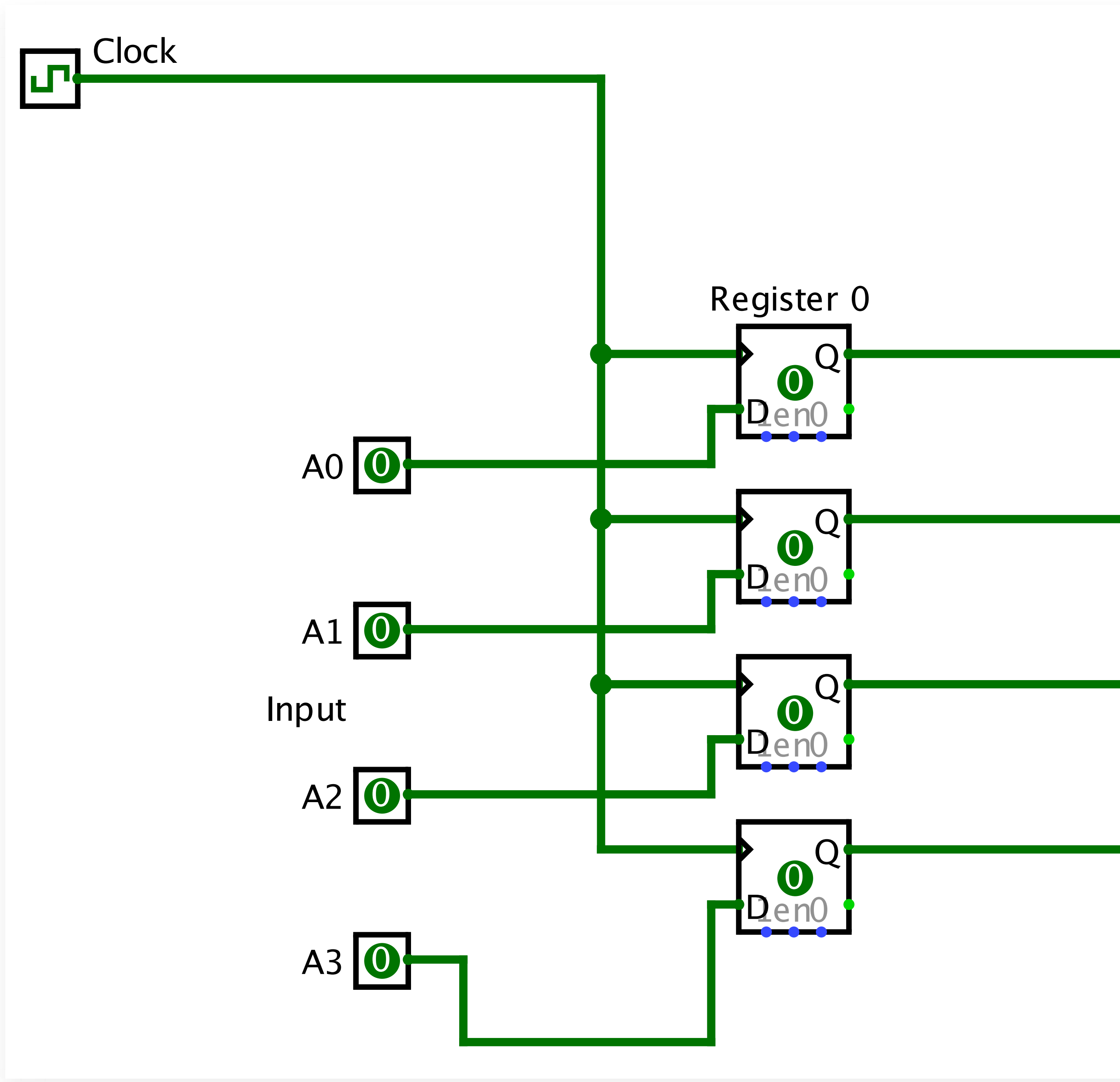
- Collection of registers
- Each implemented using n flip-flops (for n bits)
- n inputs and outputs
- Additional input: select register for writing

Register file

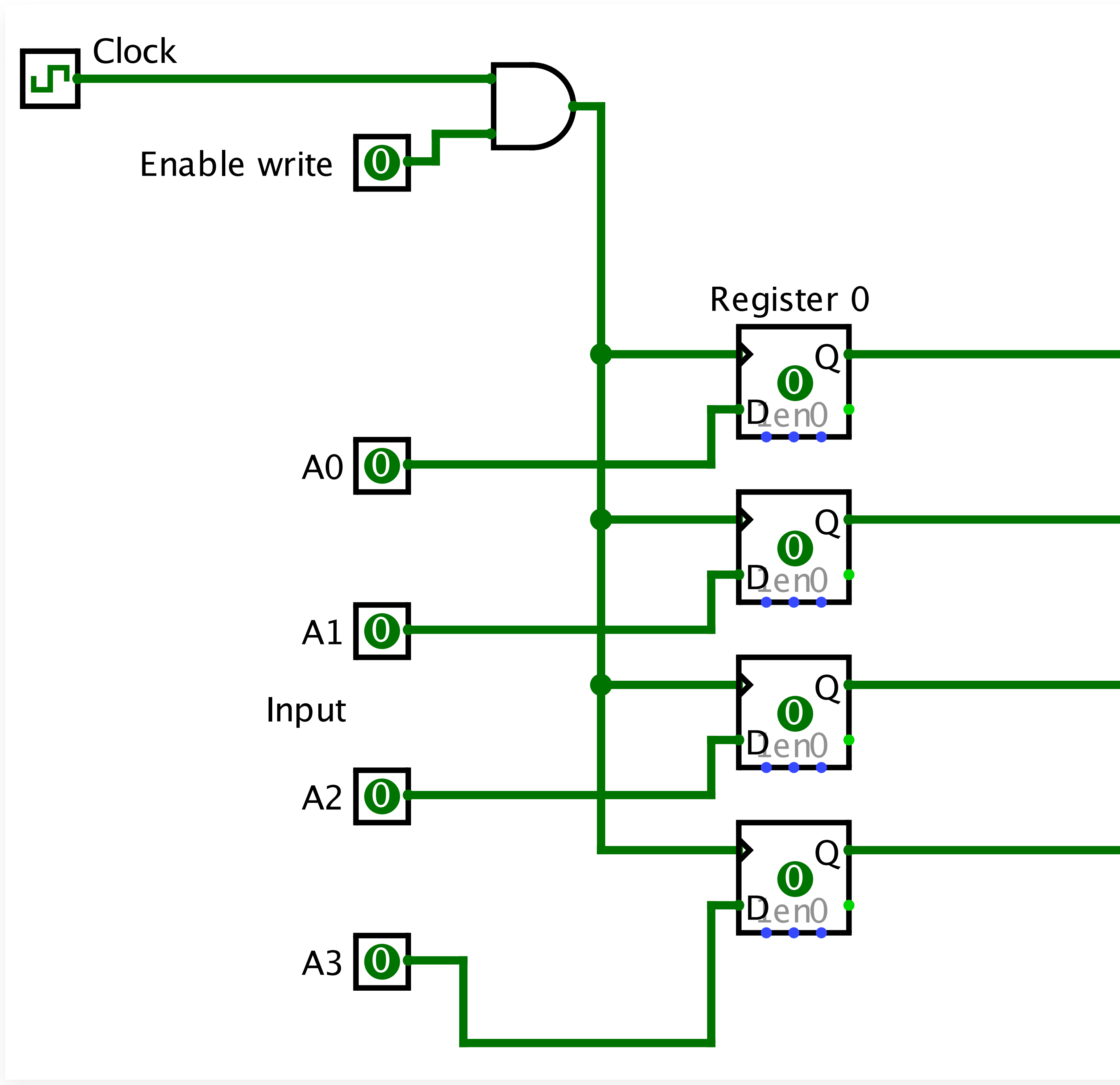
- Collection of registers
- Each implemented using n flip-flops (for n bits)
- n inputs and outputs
- Additional input: select register for writing
- Additional input: select register for reading

Register file

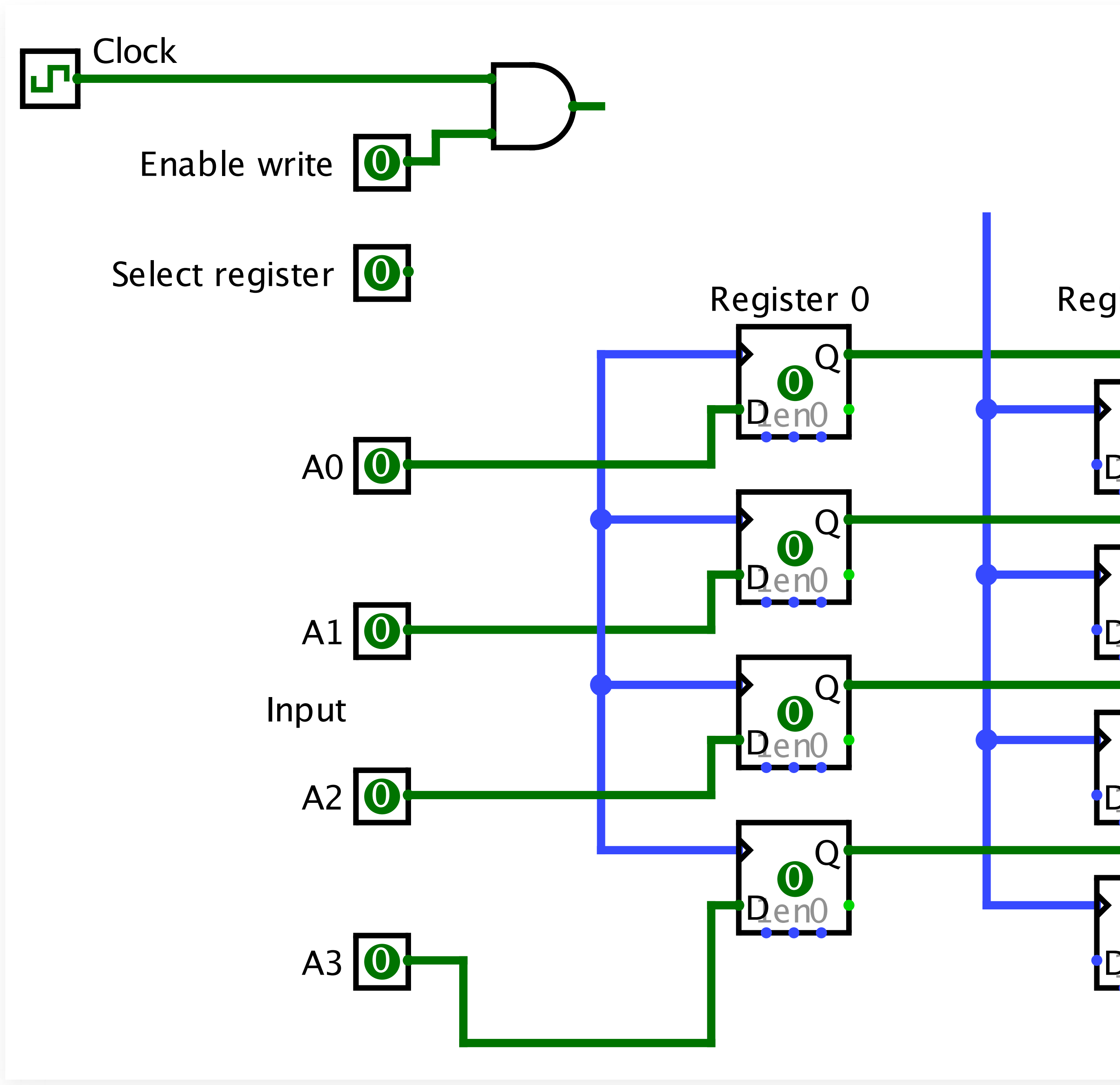
Register file



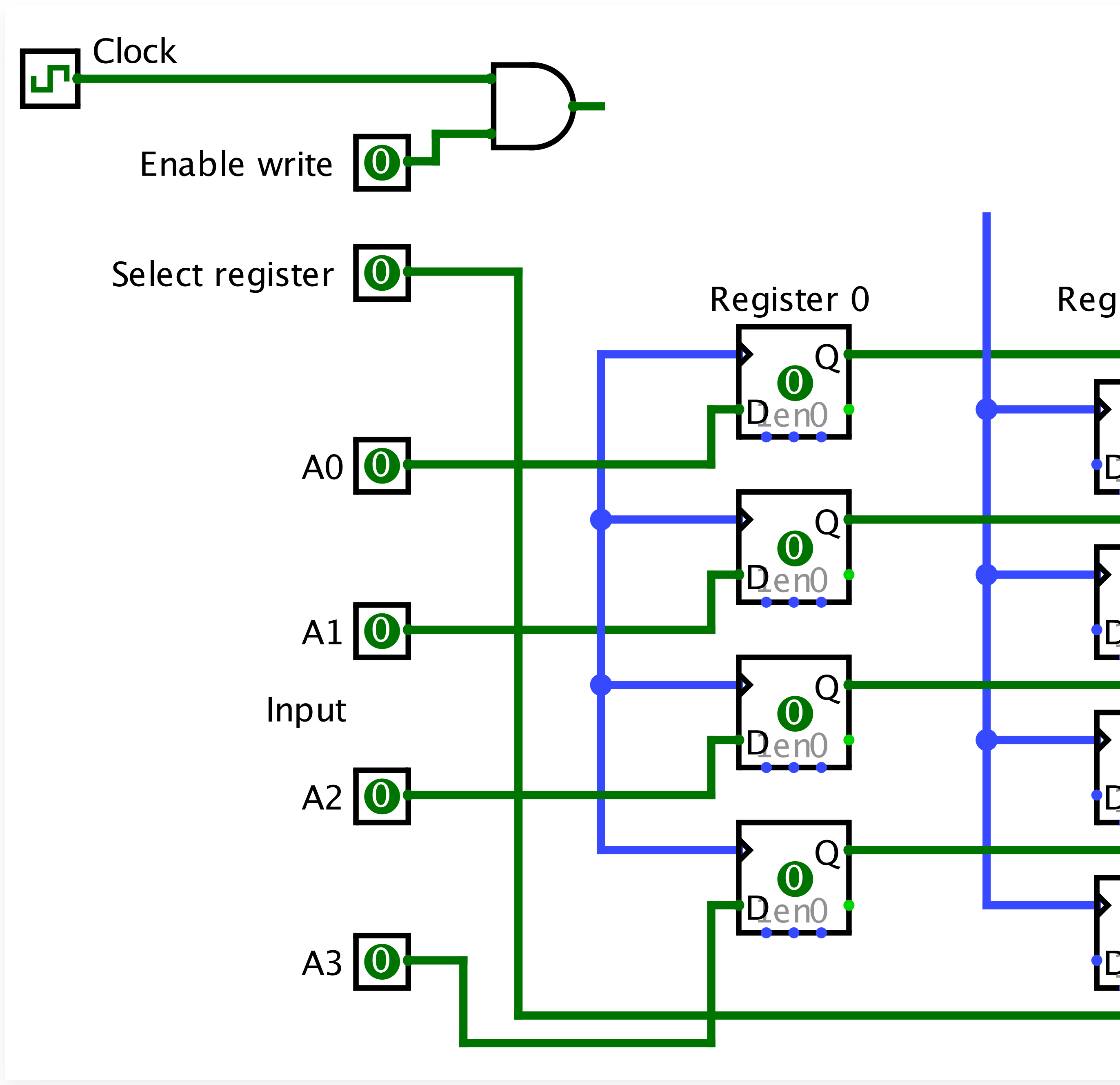
Register file



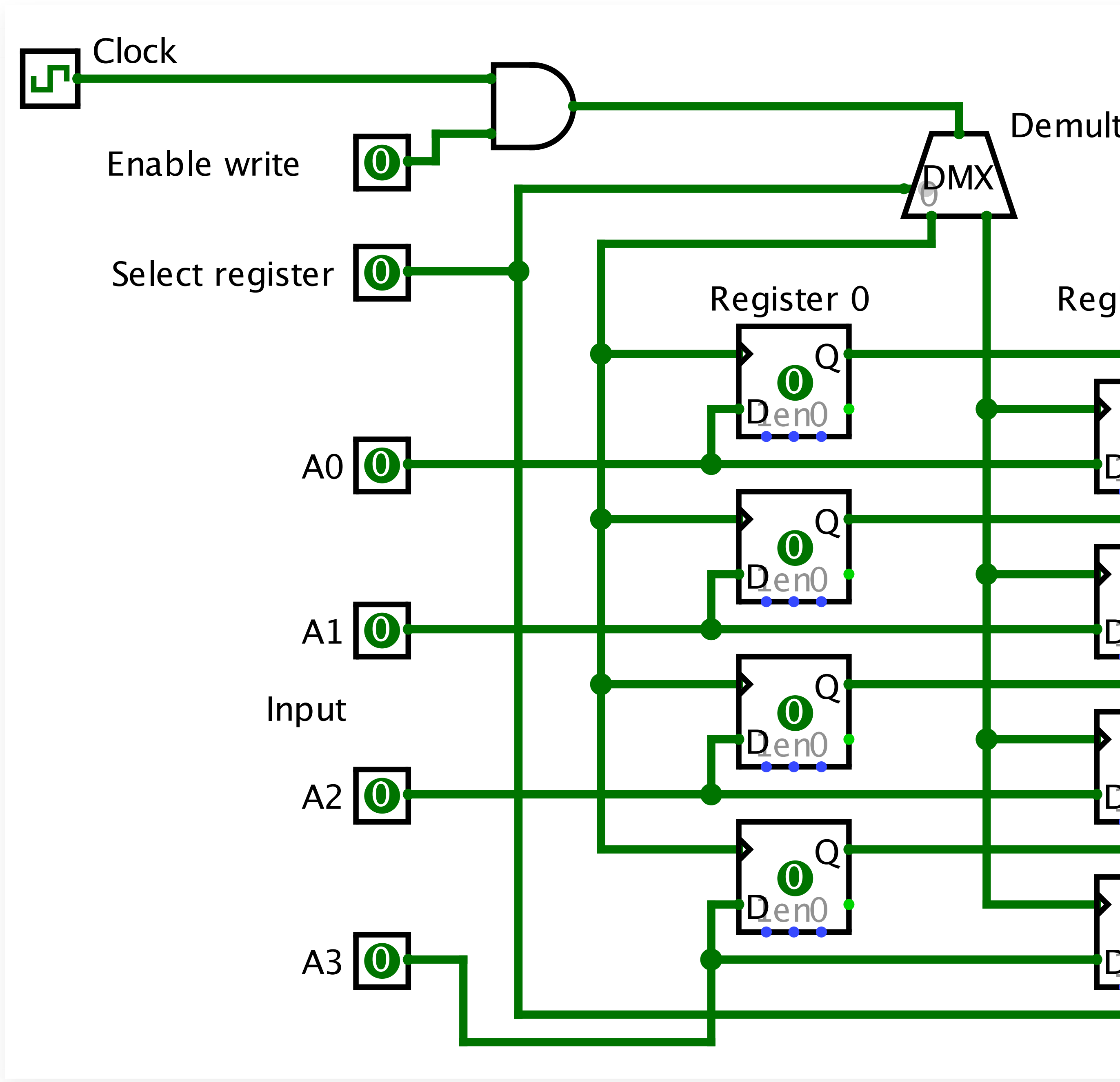
Register file



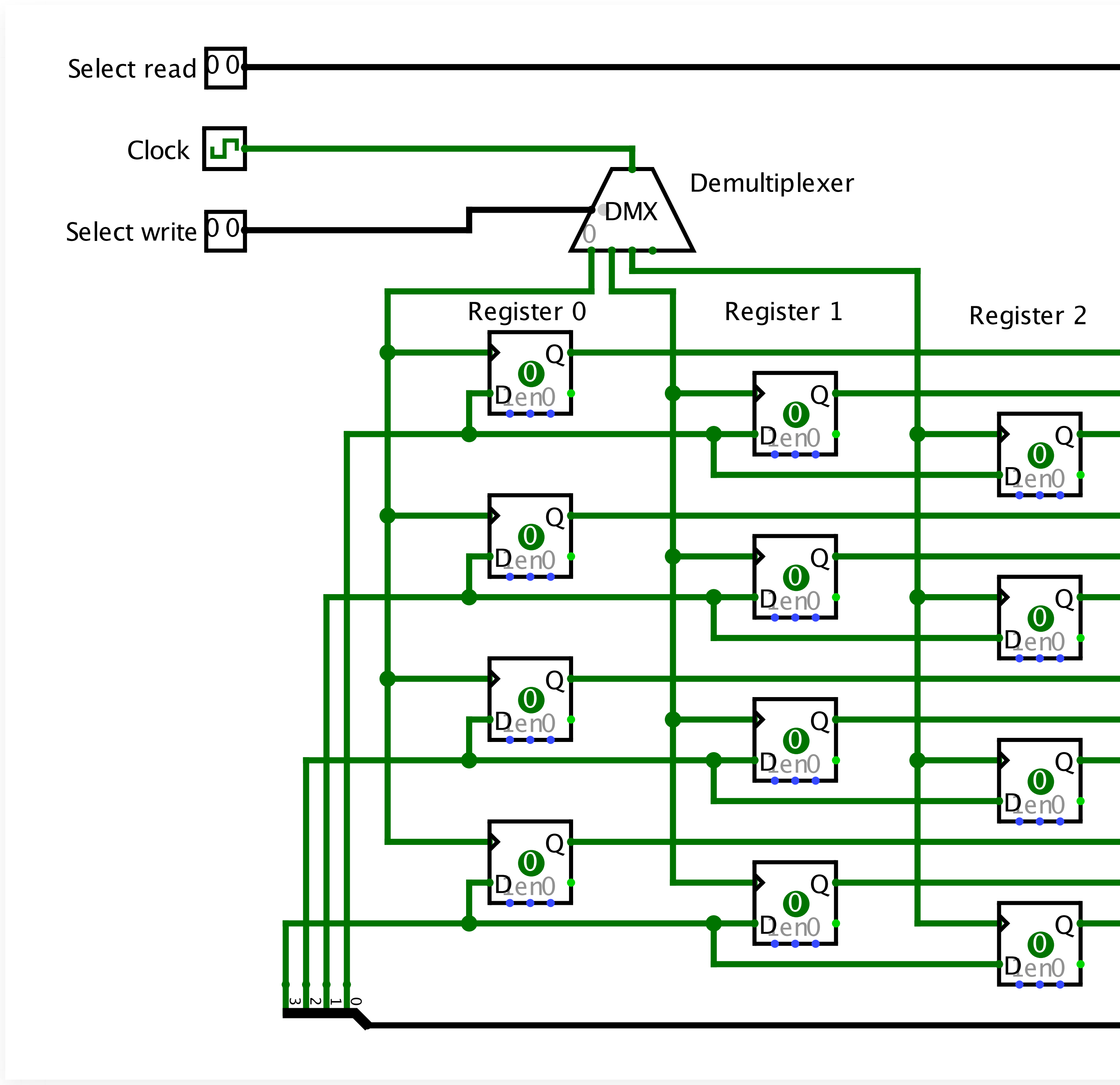
Register file



Register file



Register file



Control

Control

(the machine)

Control what?

Control what?

- Fetch - decode - execute cycle

Control what?

- Fetch - decode - execute cycle
- Implement the RTL code for each instruction
  - RTL = Register transfer language

Control what?

- Fetch - decode - execute cycle
- Implement the RTL code for each instruction
  - RTL = Register transfer language
- Generate control signals for individual circuits
  - opcode for the ALU
  - read/write register number for the register file
  - memory read/write

Timing

Let's take a look at the Add X instruction.

Timing

Let's take a look at the Add X instruction.

- MAR   X
  - register read control: IR (X is in IR)
  - register write control: MAR

Timing

Let's take a look at the Add X instruction.

- MAR   X
  - register read control: IR (X is in IR)
  - register write control: MAR
- MBR   M[MAR]
  - memory: read (always reads at address MAR)
  - register write control: MBR

Timing

Let's take a look at the Add X instruction.

- MAR   X
  - register read control: IR (X is in IR)
  - register write control: MAR
- MBR   M[MAR]
  - memory: read (always reads at address MAR)
  - register write control: MBR
- AC   AC + MBR
  - ALU opcode: addition (always reads AC and MBR)
  - register write control: AC

Timing

How do we generate that sequence of signals?

Add a cycle counter!

- n outputs T_0 to T_n
- Cycle through these outputs at each clock cycle.
- Has an input C_n to reset to T_0.
- Now we can write RTL using Boolean statements!

Cycle counter

Cycle counter

Control signals

- Register read: $P_2\ P_1\ P_0$
  - Memory=0, MAR=1, PC=2, MBR=3, AC=4, IR=7
- Register write: $P_5\ P_4\ P_3$
- ALU: $A_1\ A_0$
  - Add=1, Subt=2, Clear=3

Add X with signals

MAR X

T_3 P_3 P_2 P_1 P_0

Add X with signals

MAR   X

$T_3 \; P_3 \; P_2 \; P_1 \; P_0$

- read IR ($P_2 = 1, P_1 = 1, P_0 = 1$)

Add X with signals

MAR   X

$T_3 \; P_3 \; P_2 \; P_1 \; P_0$

- read IR ($P_2 = 1, P_1 = 1, P_0 = 1$)
- write MAR ($P_5 = 0, P_4 = 0, P_3 = 1$)

Add X with signals

MAR X

$T_3 \; P_3 \; P_2 \; P_1 \; P_0$

- read IR ($P_2 = 1, P_1 = 1, P_0 = 1$)
- write MAR ($P_5 = 0, P_4 = 0, P_3 = 1$)

Note: T_0, T_1, T_2 are used for the fetch cycle.

Add X with signals

MBR M[MAR]

T_4 P_4 P_3 M_R

Add X with signals

MBR M[MAR]

T_4 P_4 P_3 M_R

- Read memory (always reads at address MAR)

Add X with signals

MBR   M[MAR]

$T_4\ P_4\ P_3\ M_R$

- Read memory (always reads at address MAR)
- write MBR ($P_5 = 0, P_4 = 1, P_3 = 1$)

Add X with signals

AC   AC + MBR

T_5 A_0 P_5 P_1 P_0

Add X with signals

AC   AC + MBR

$T_5 \ A_0 \ P_5 \ P_1 \ P_0$

- ALU: addition ($A_1 = 0, A_0 = 1$)

Add X with signals

AC   AC + MBR

$T_5 \; A_0 \; P_5 \; P_1 \; P_0$

- ALU: addition ($A_1 = 0$, $A_0 = 1$)
- read MBR ($P_2 = 0$, $P_1 = 1$, $P_0 = 1$)

Add X with signals

AC   AC + MBR

$T_5 \; A_0 \; P_5 \; P_1 \; P_0$

- ALU: addition ($A_1 = 0, A_0 = 1$)
- read MBR ($P_2 = 0, P_1 = 1, P_0 = 1$)
- write AC ($P_5 = 1, P_4 = 0, P_3 = 0$)

Add X with signals

AC   AC + MBR

T_6 C_r

Add X with signals

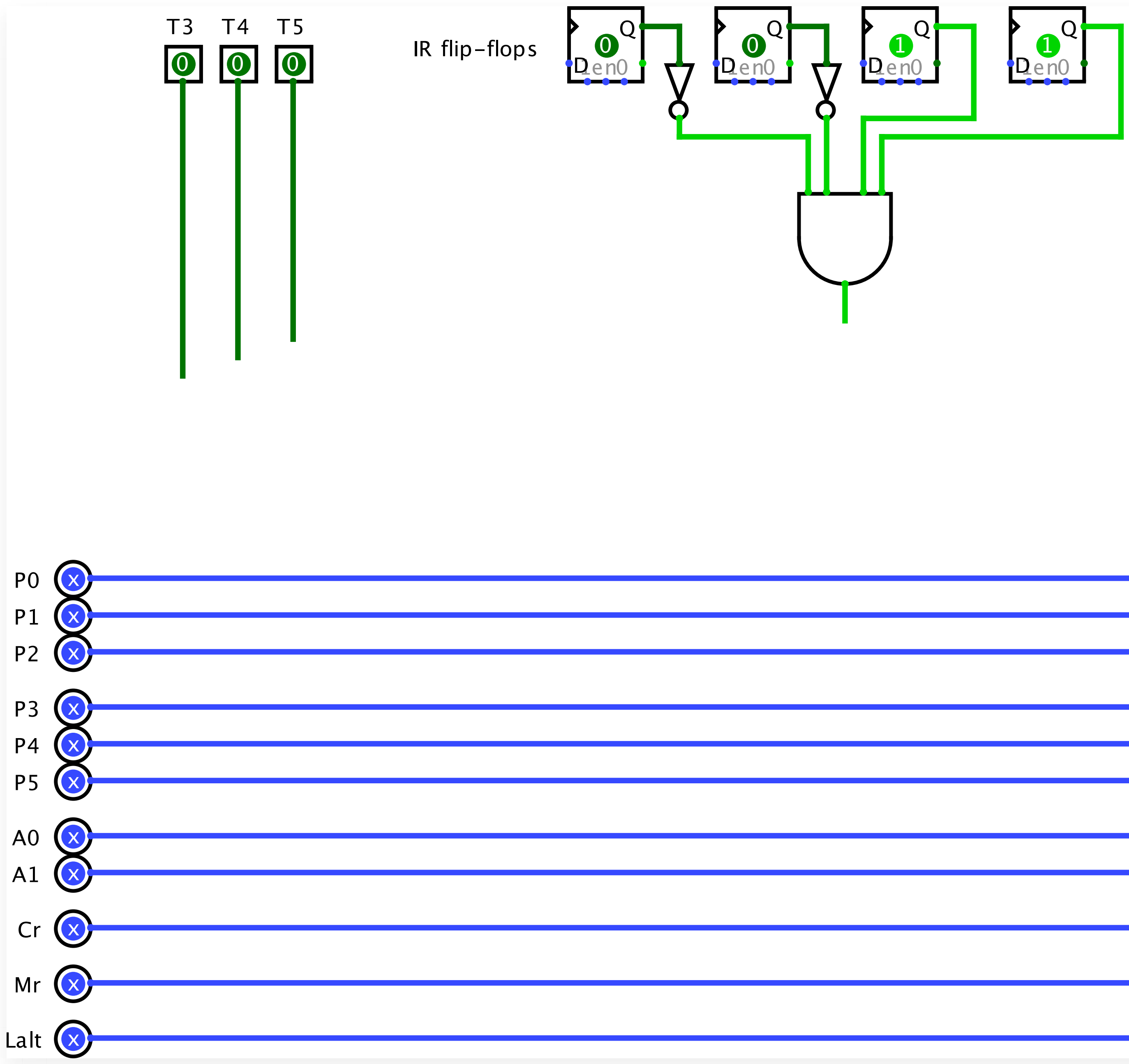
AC   AC + MBR

T_6 C_r

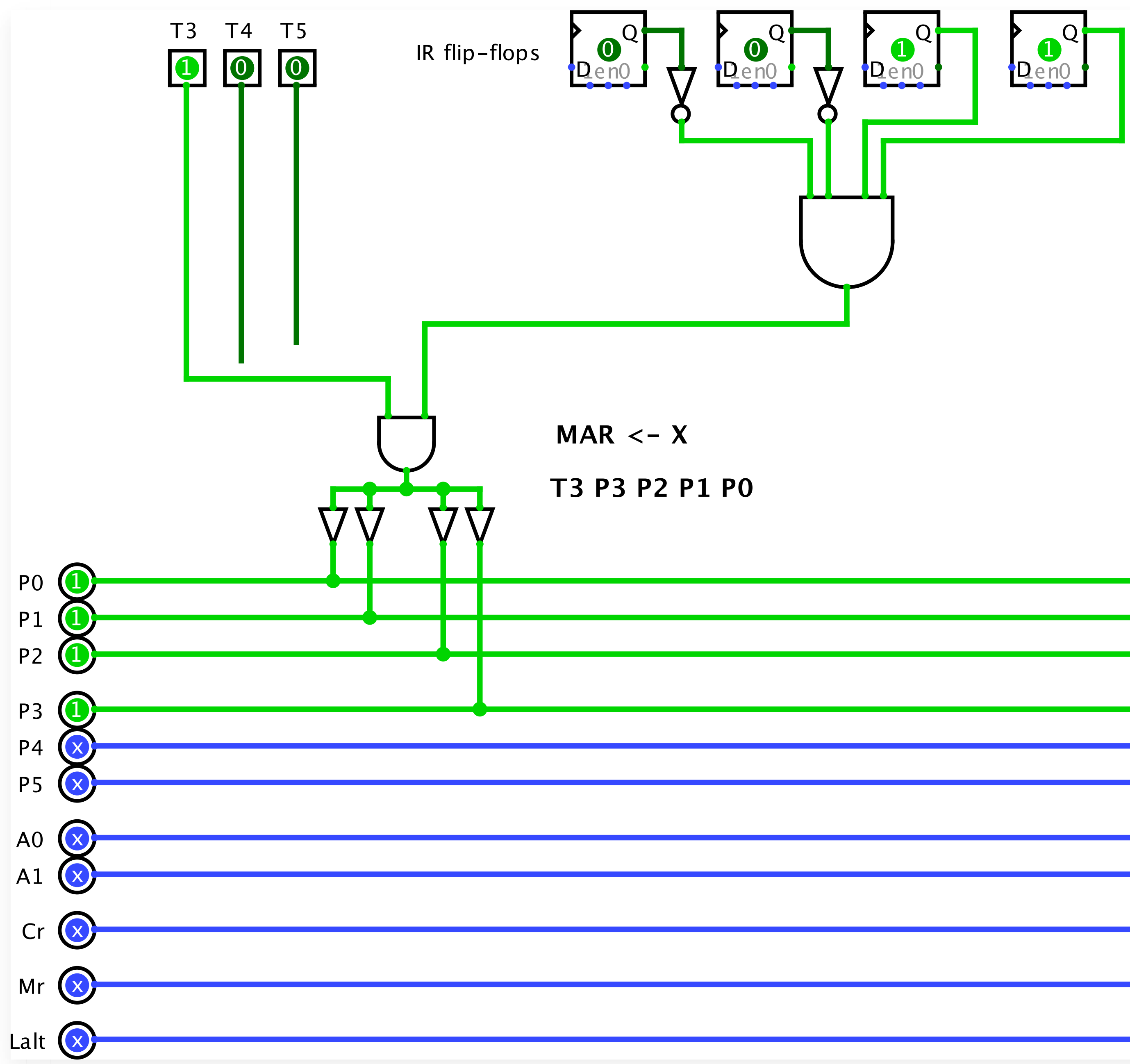
- reset cycle counter

Control circuit (Add X)

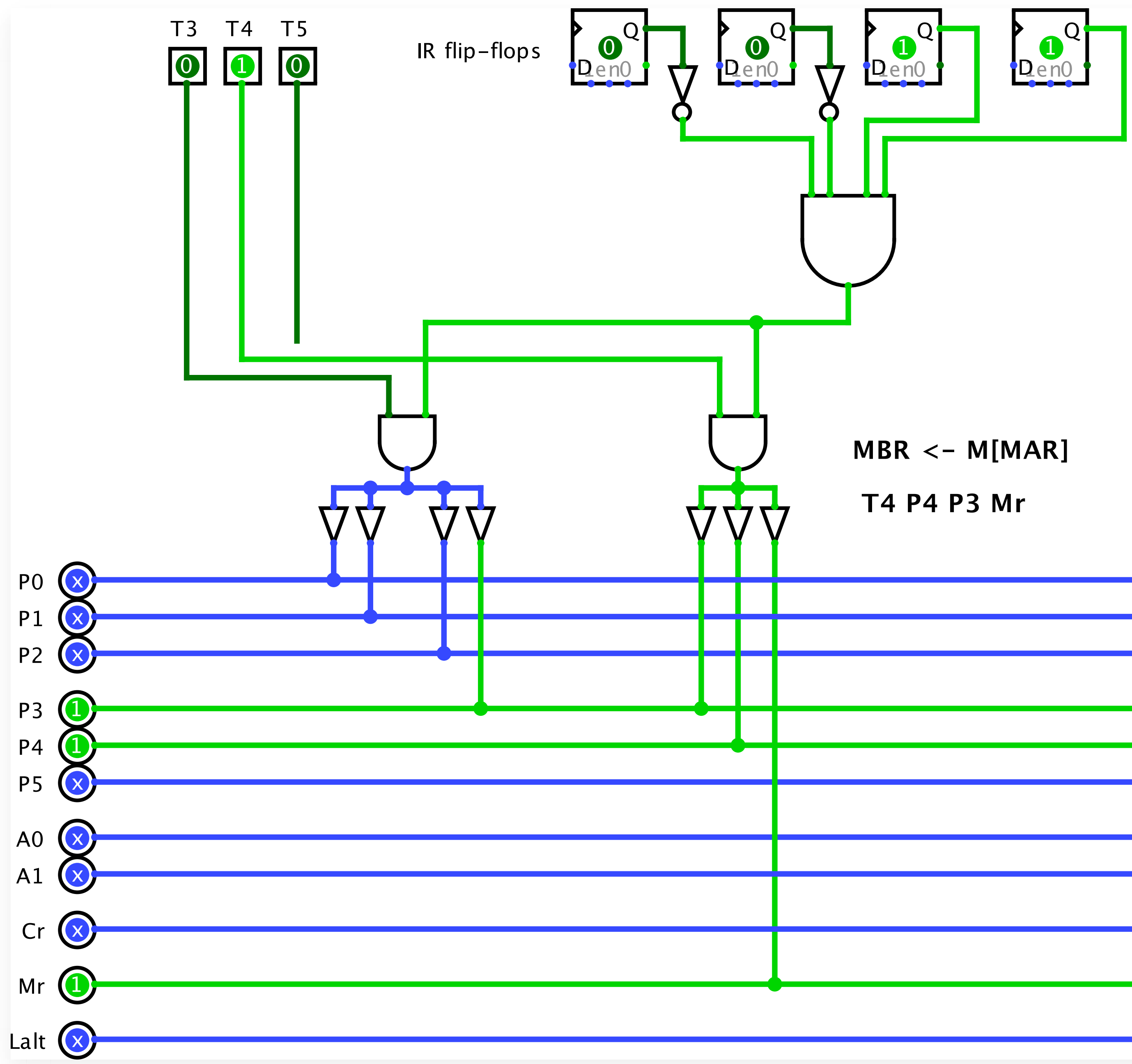
Control circuit (Add X)



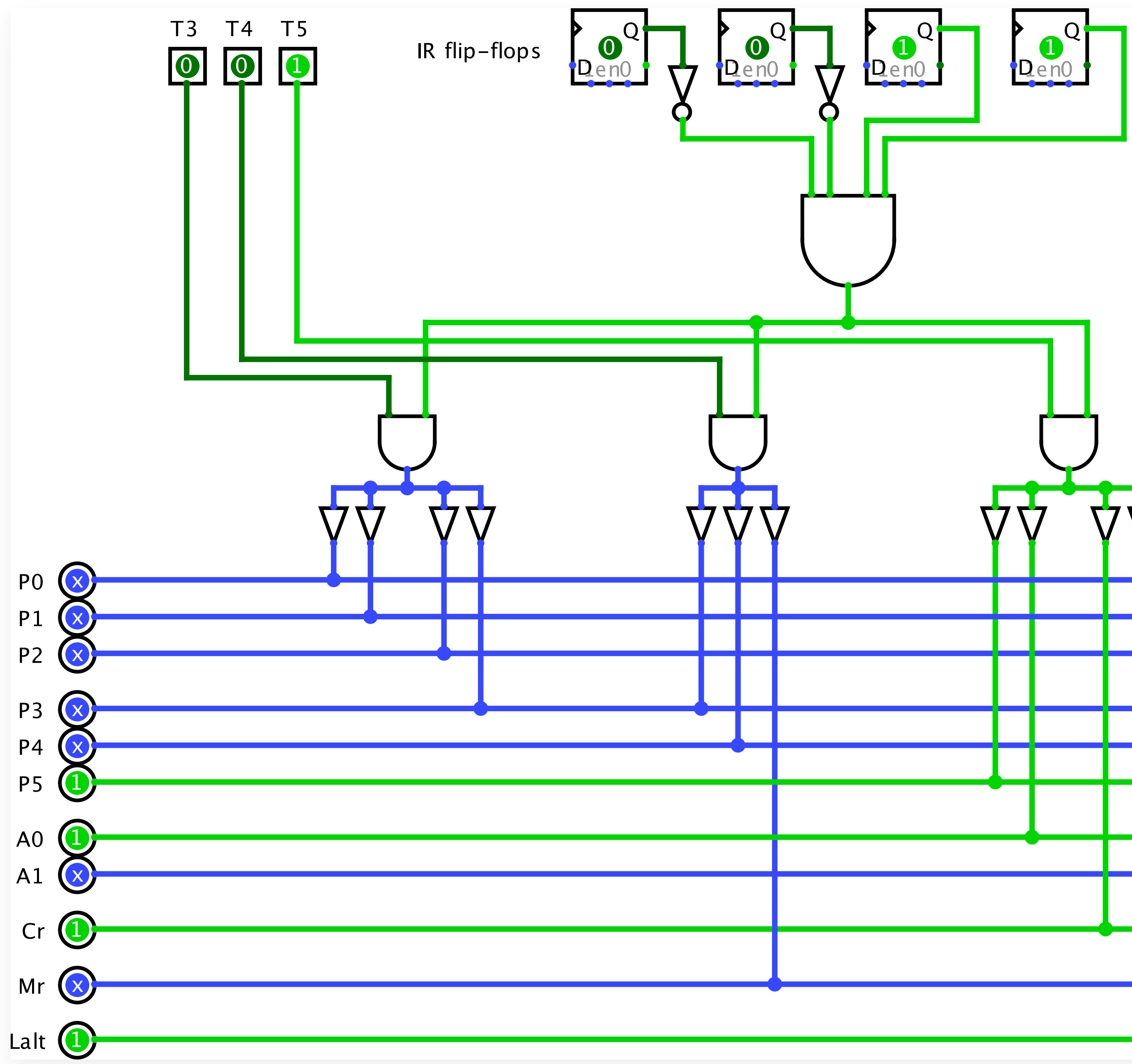
Control circuit (Add X)



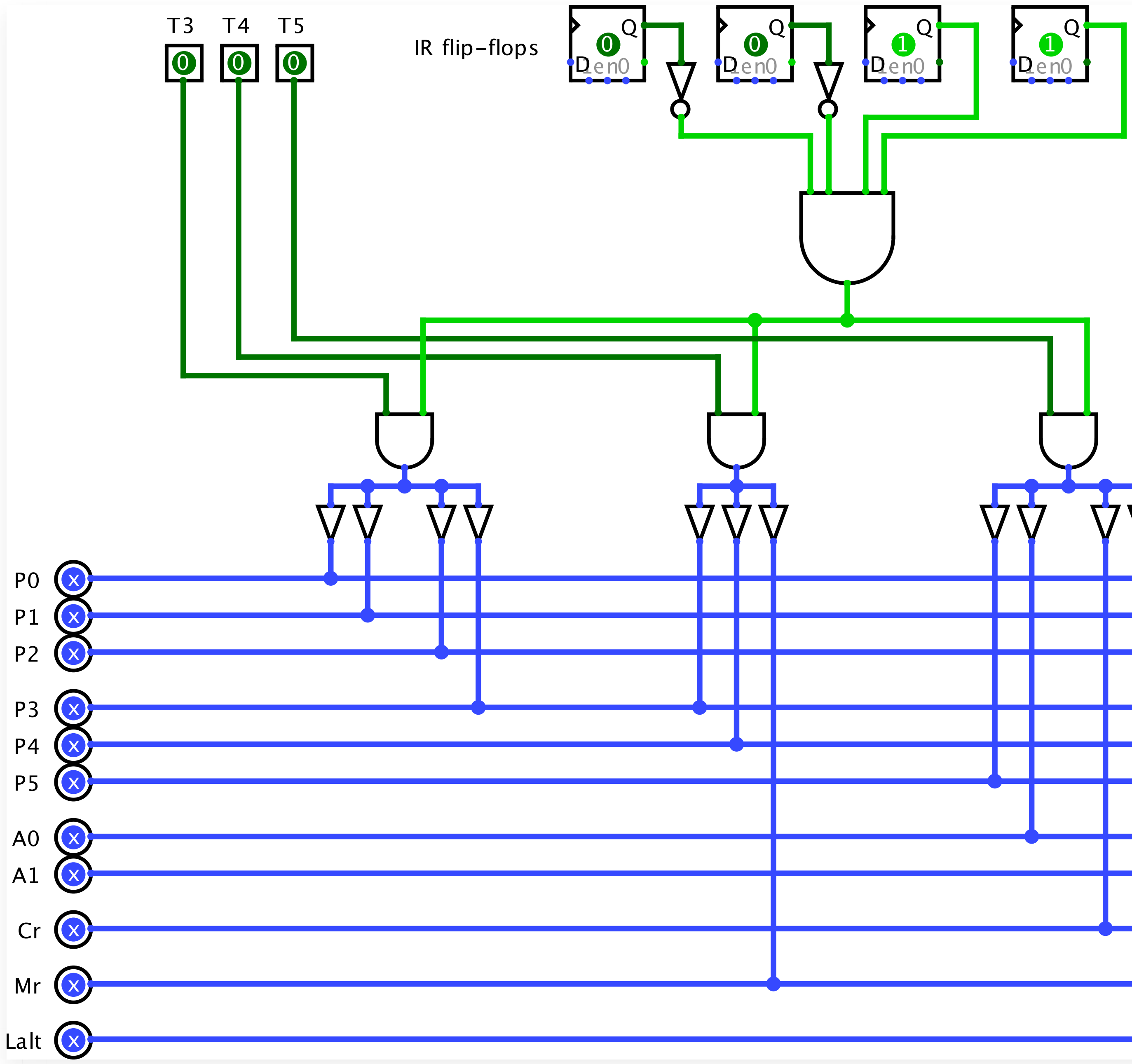
Control circuit (Add X)



Control circuit (Add X)



Control circuit (Add X)



Hardwired Control

RTL for each instruction is implemented using gates.

Hardwired Control

RTL for each instruction is implemented using gates.

- Very fast!

Hardwired Control

RTL for each instruction is implemented using gates.

- Very fast!
- But really complicated to design.

Hardwired Control

RTL for each instruction is implemented using gates.

- Very fast!
- But really complicated to design.
- What if we want to add instructions?

Hardwired Control

RTL for each instruction is implemented using gates.

- Very fast!
- But really complicated to design.
- What if we want to add instructions?
- What if we made a mistake?

Microprogrammed Control

Break up instructions into microoperations.

Microprogrammed Control

Break up instructions into microoperations.

- One microop per possible RTL instruction
  - Far fewer than instructions!

Microprogrammed Control

Break up instructions into microoperations.

- One microop per possible RTL instruction
  - Far fewer than instructions!
- Microops for each instruction stored in memory in the processor

Microprogrammed Control

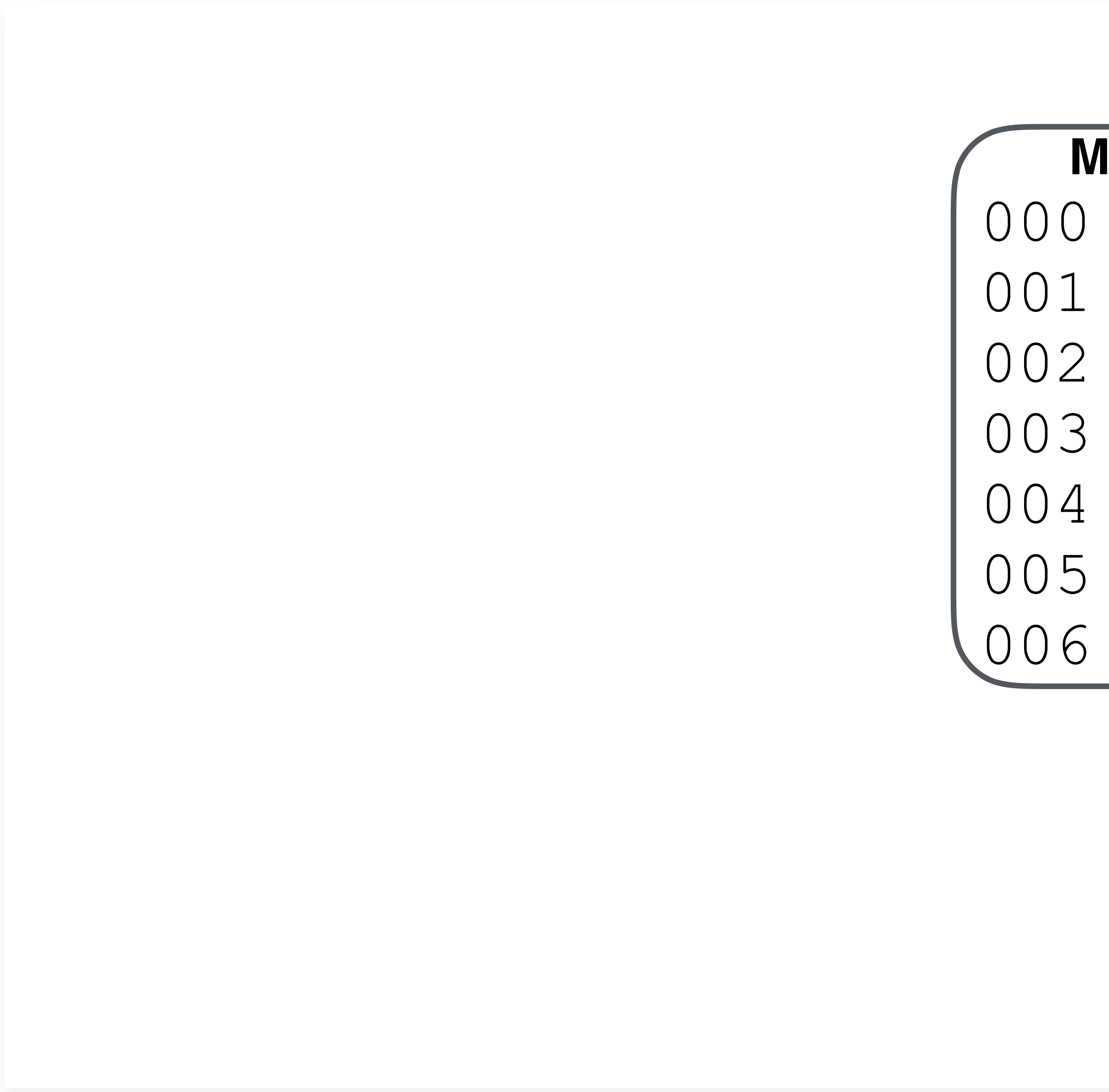
Break up instructions into microoperations.

- One microop per possible RTL instruction
  - Far fewer than instructions!
- Microops for each instruction stored in memory in the processor
- Execution slightly slower, but
  - Circuits for microops much simpler
  - Microcode can be updated

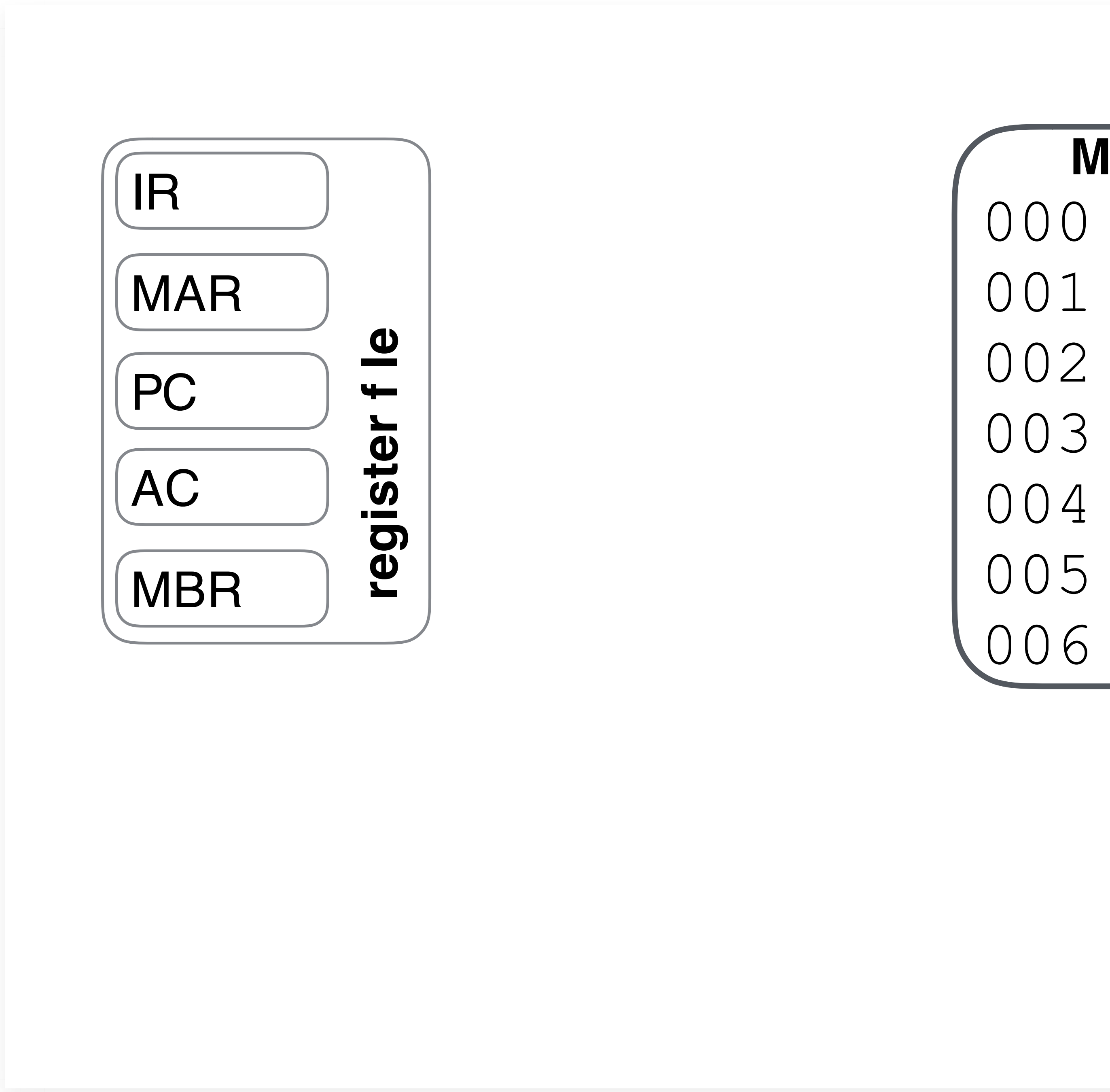
Putting it all together

MARIE Data Paths

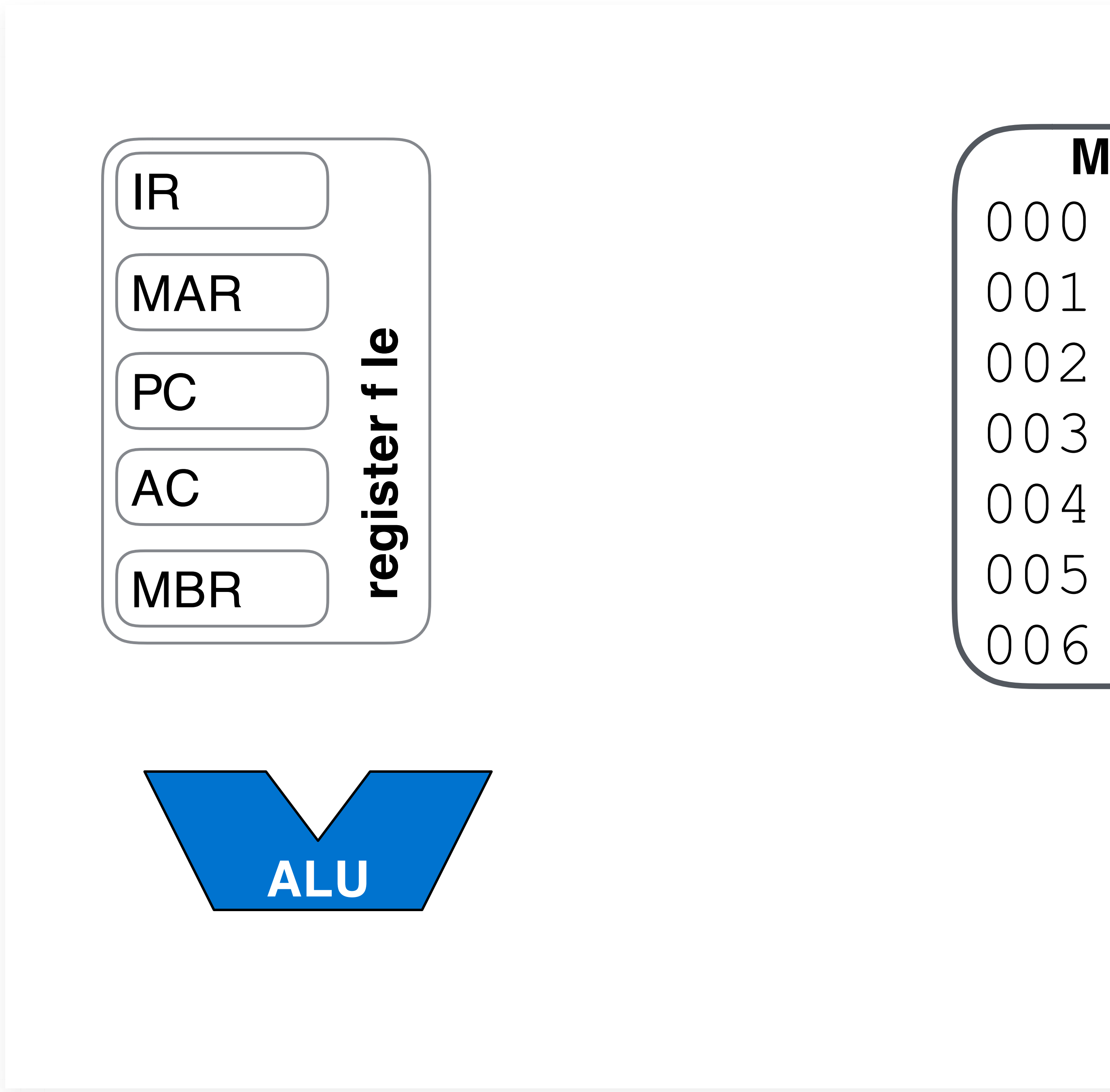
MARIE Data Paths



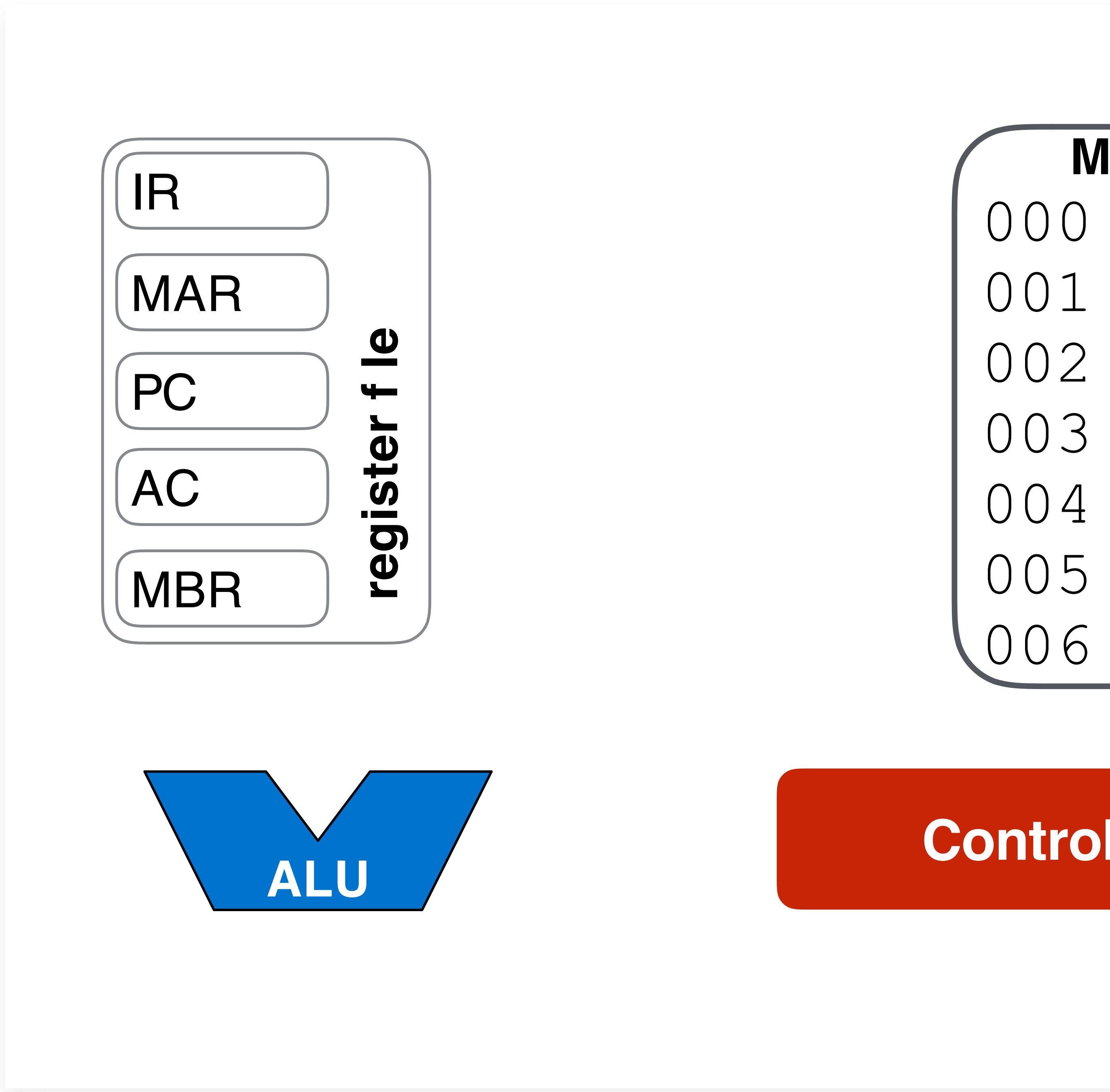
MARIE Data Paths



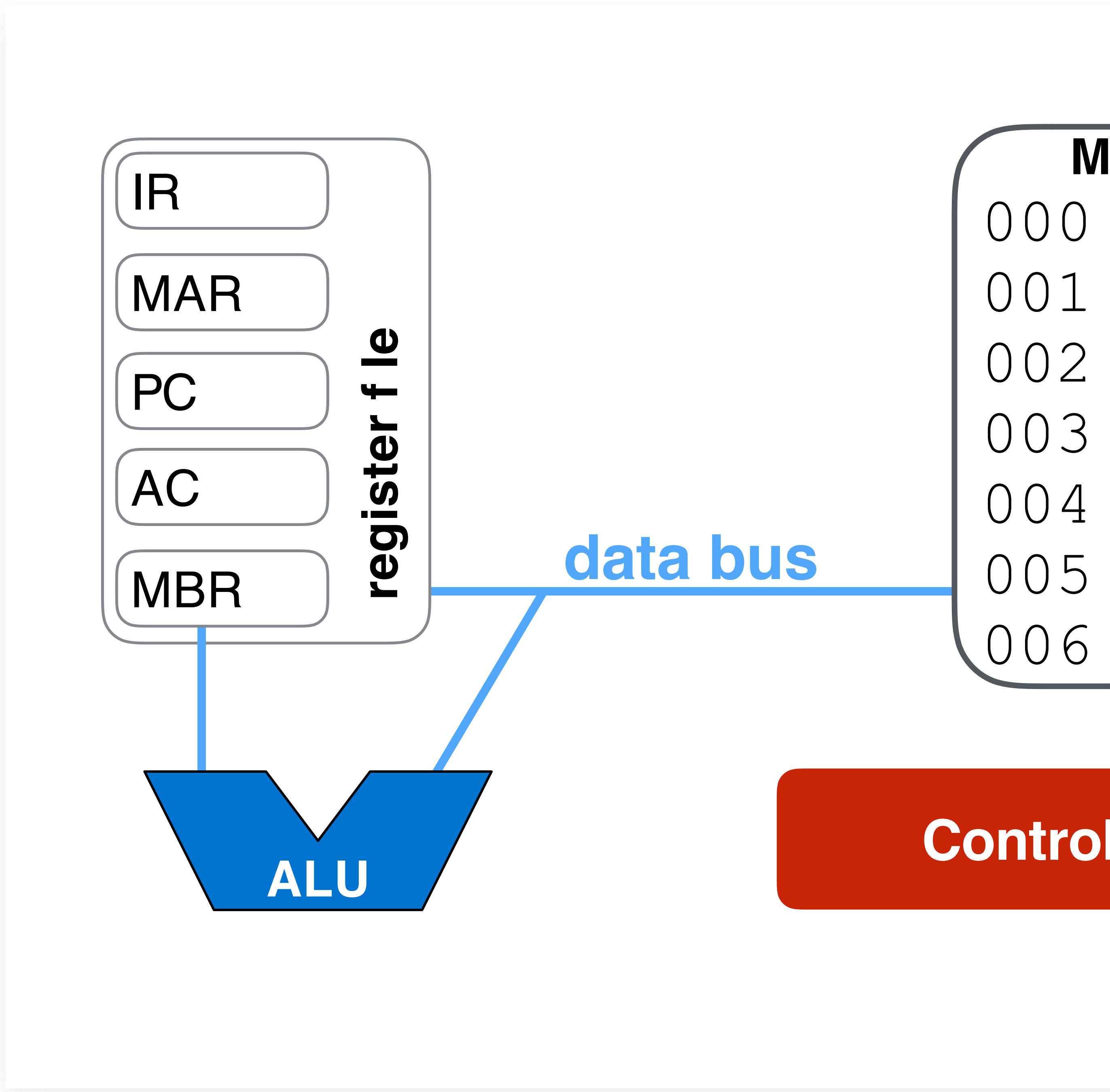
MARIE Data Paths



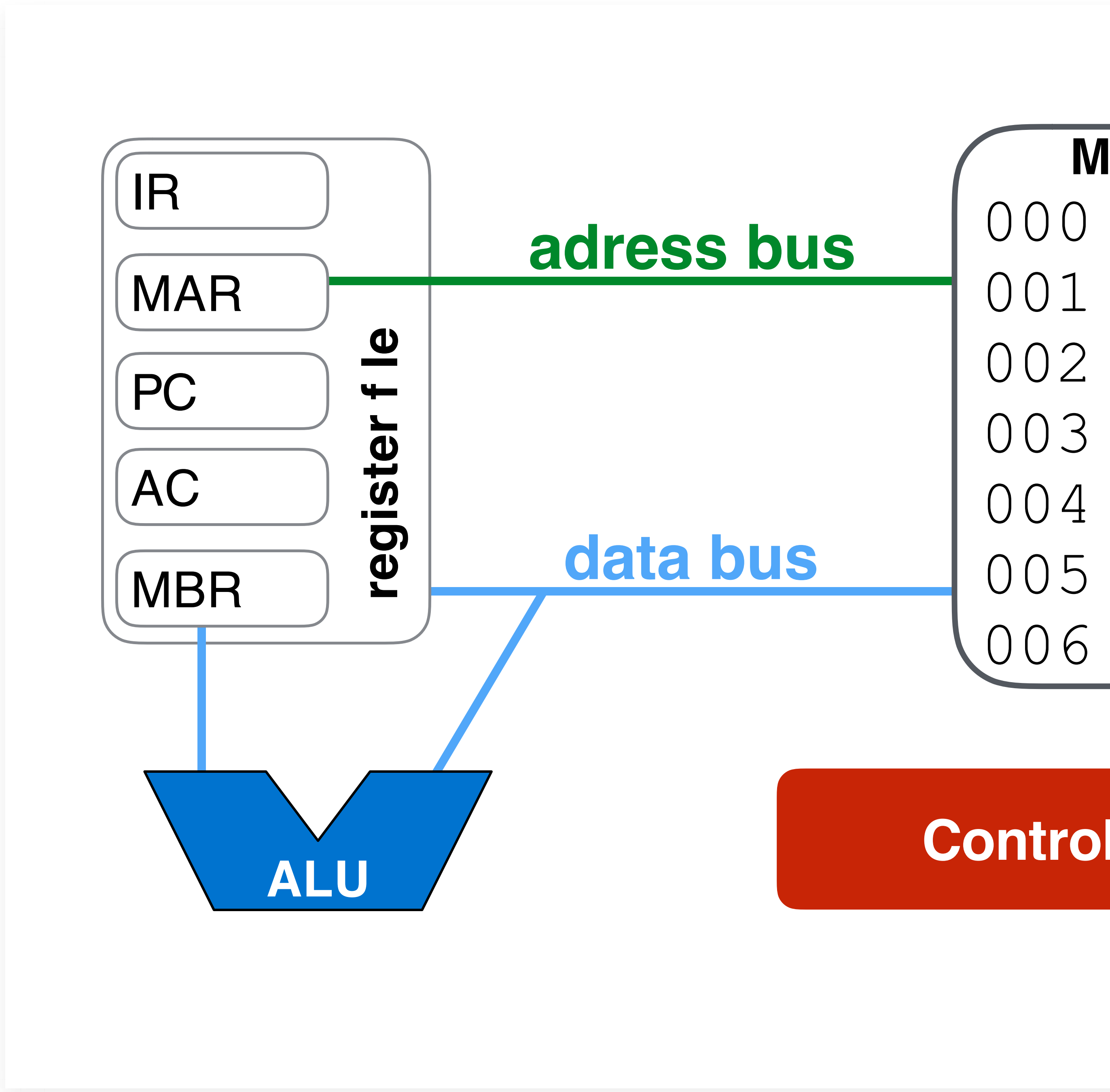
MARIE Data Paths



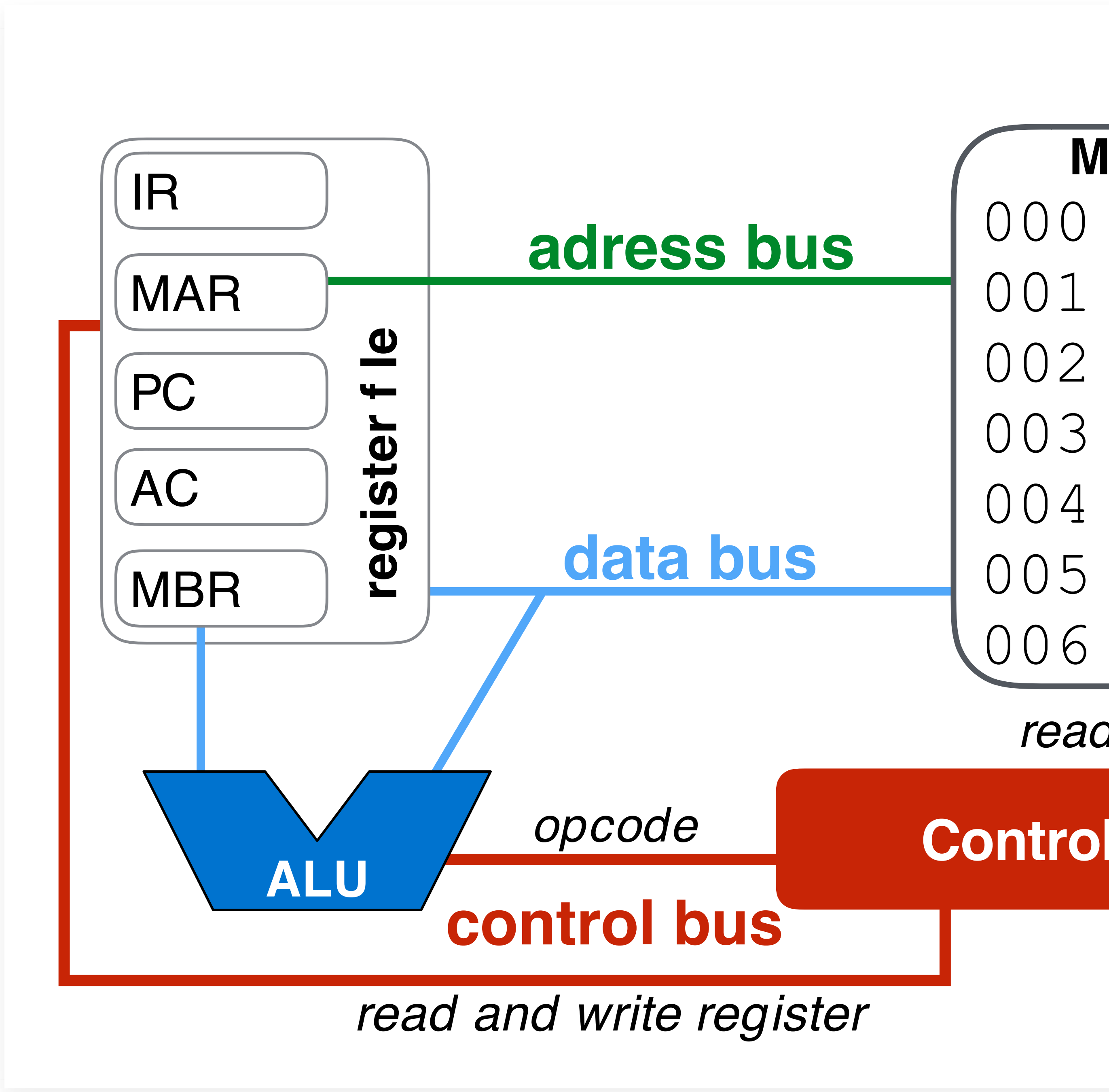
MARIE Data Paths



MARIE Data Paths



MARIE Data Paths



Next lectures