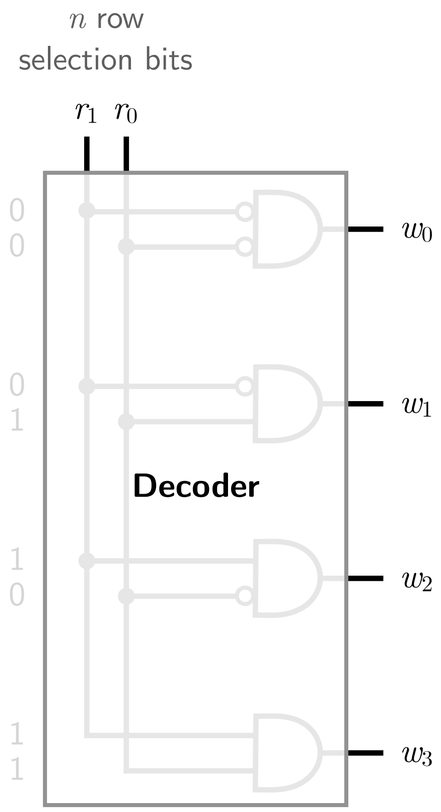
Computers

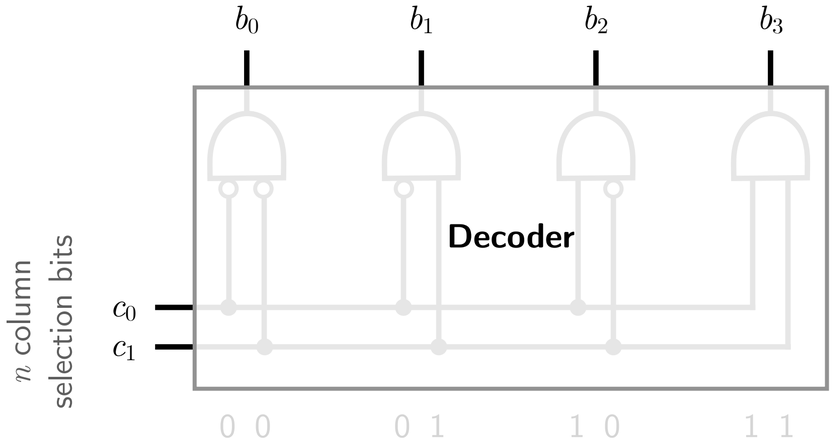
Janik Euskirchen

6

Memory

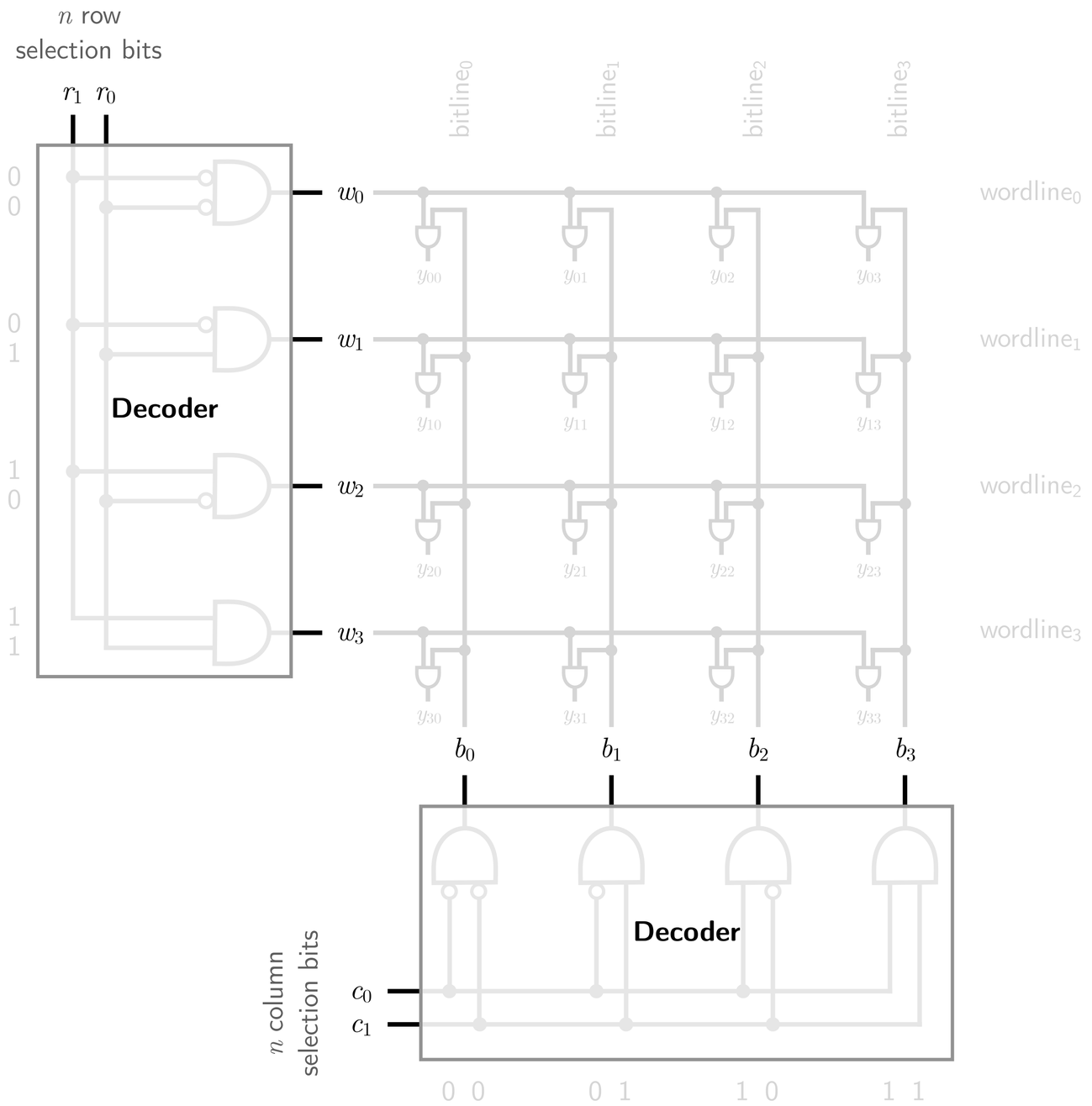
Memory (Random Access Memory, RAM)





r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

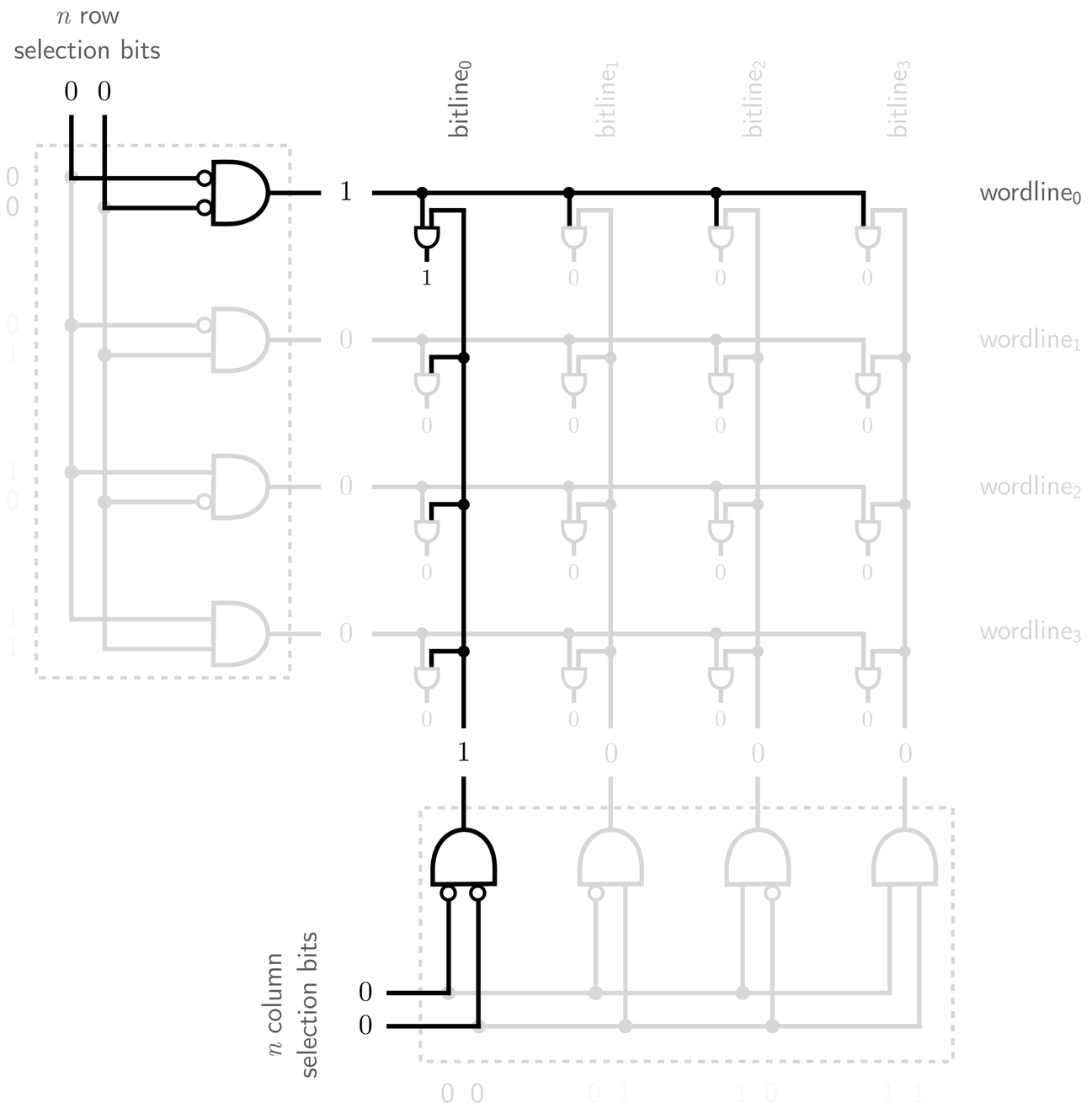
Memory (Random Access Memory, RAM)



r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

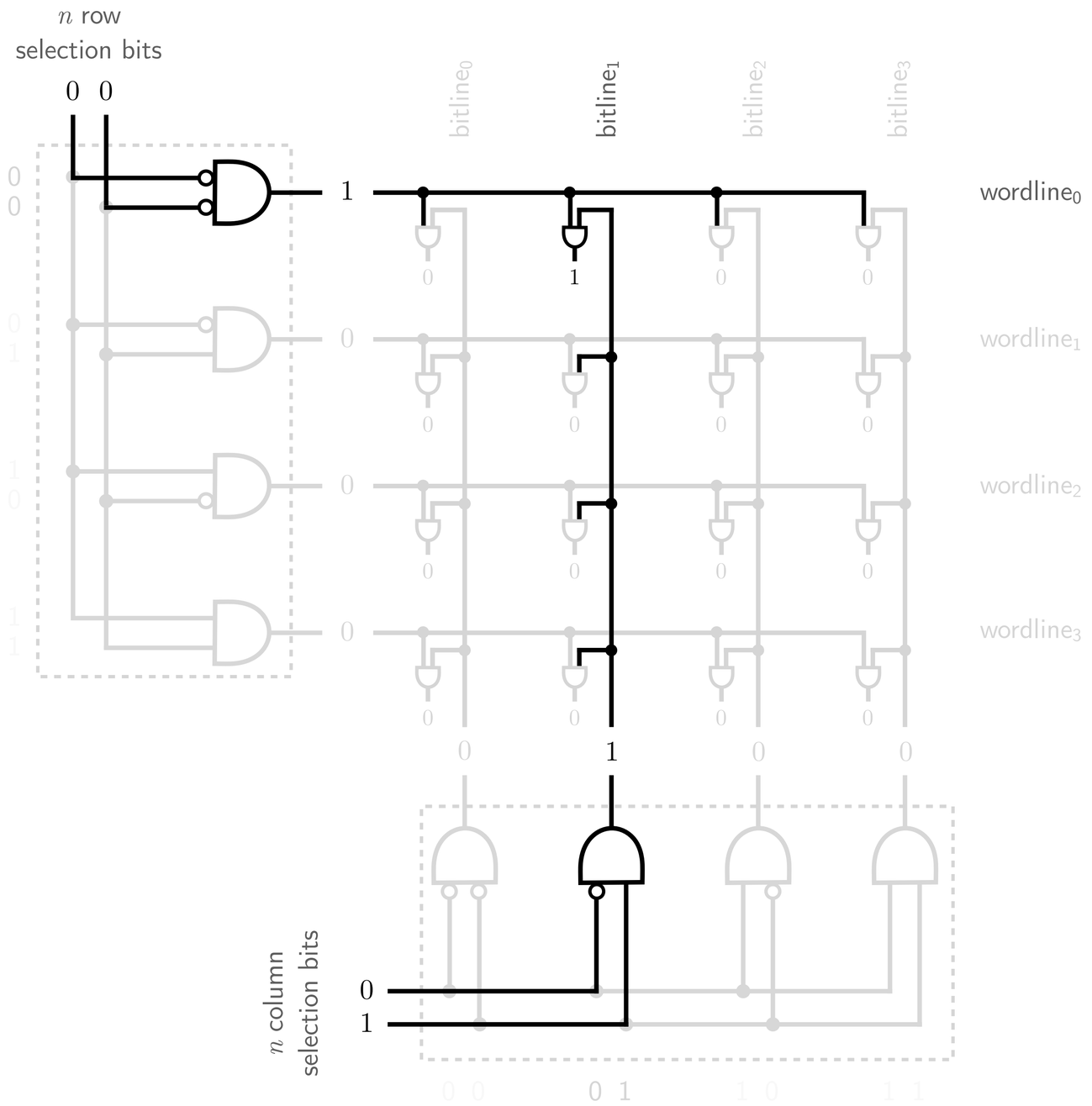
Cell at $(\underbrace{0, 0,}_{\text{row}} \underbrace{0, 0}_{\text{col}})$



r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

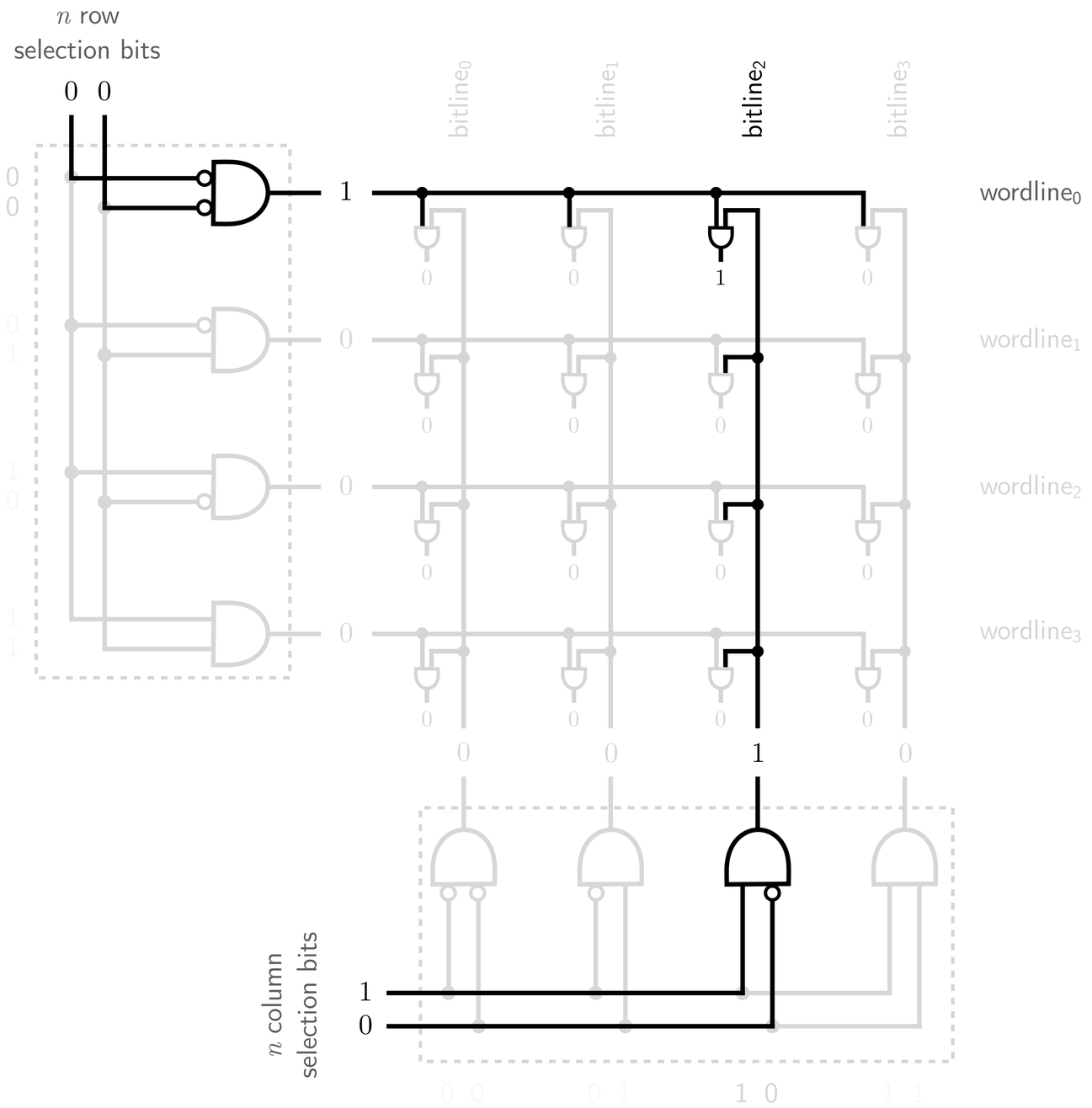
Cell at ($\underbrace{0, 0,}_{\text{row}}$ $\underbrace{0, 1}_{\text{col}}$)



r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at $(\underbrace{0, 0,}_{\text{row}} \underbrace{1, 0}_{\text{col}})$

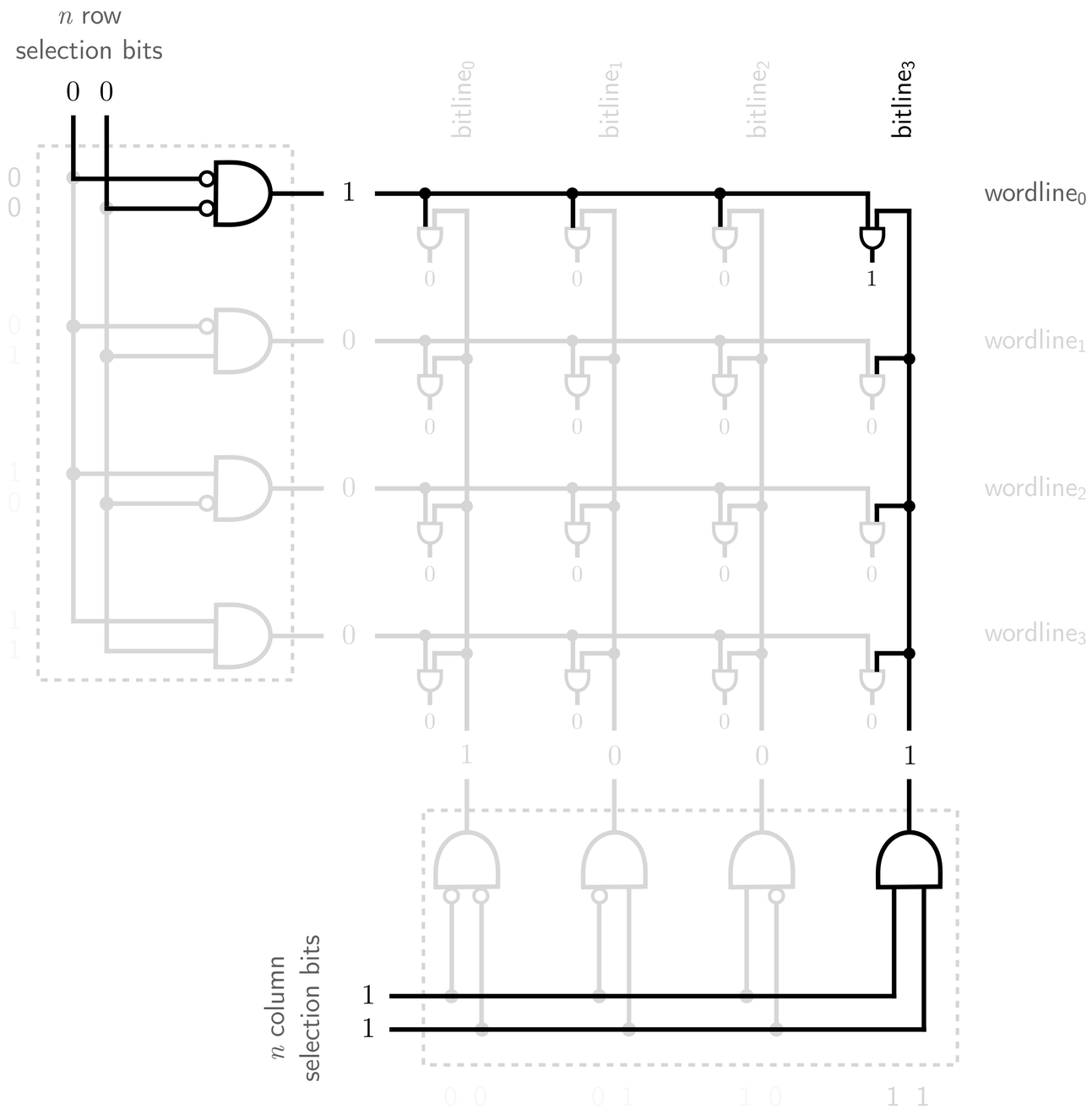


r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at ($\underbrace{0, 0,}_{\text{row}}$ $\underbrace{1, 1}_{\text{col}}$)

row: 0 col: 3

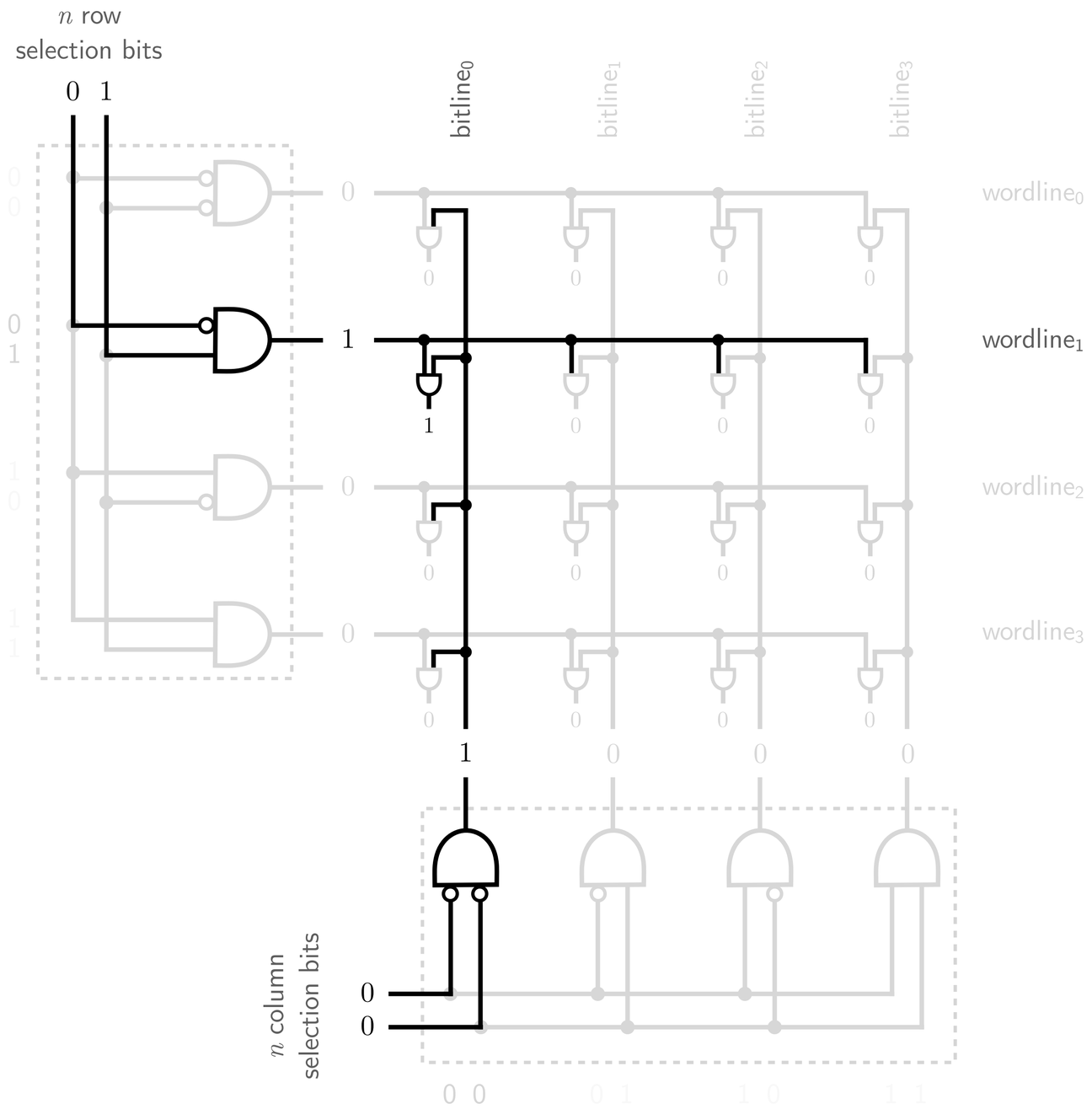


r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at $(\underbrace{0, 1,}_{\text{row}} \underbrace{0, 0}_{\text{col}})$

row: 1 col: 0

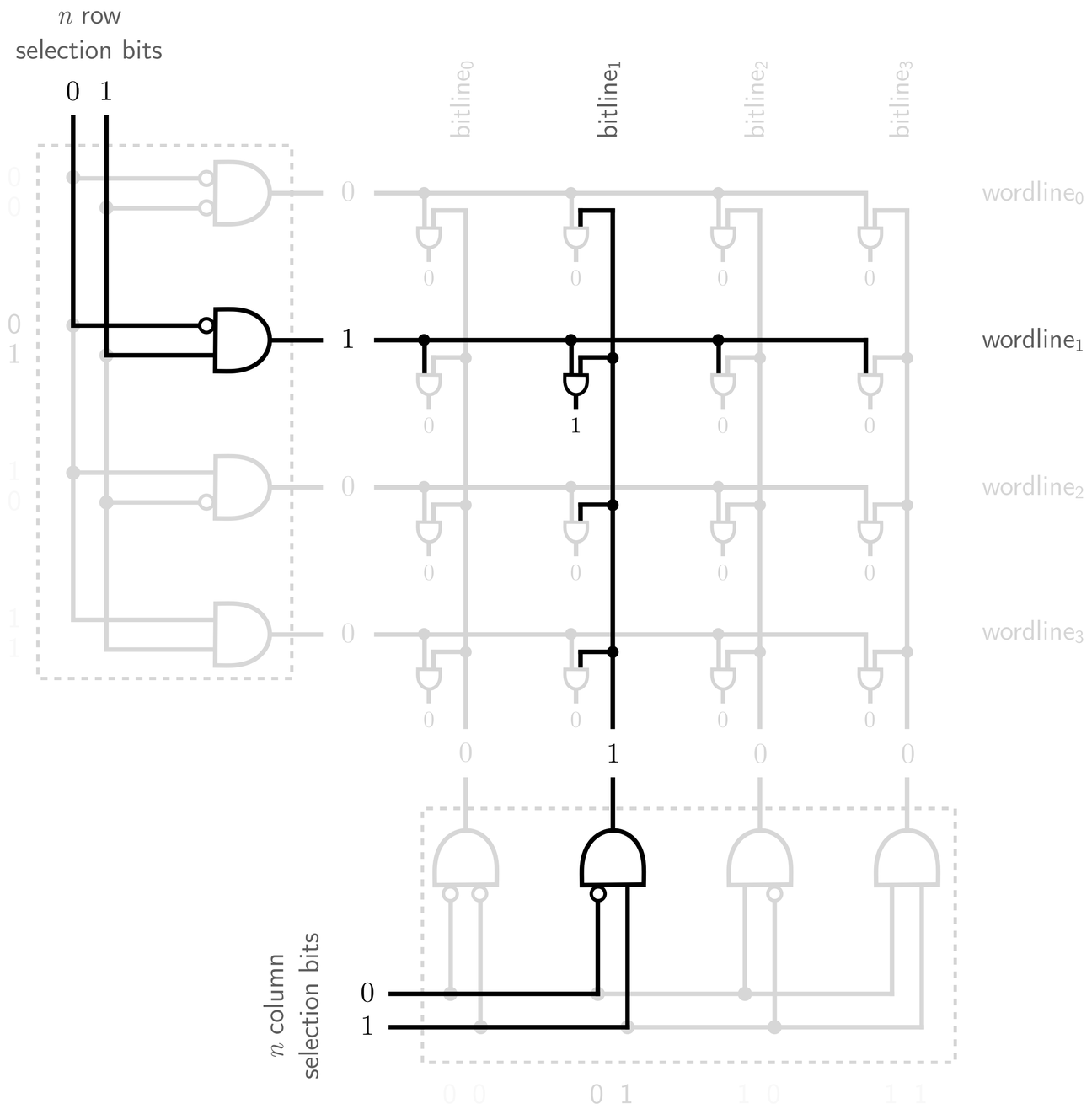


r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at ($\underbrace{0, 1,}_{\text{row}}$ $\underbrace{0, 1}_{\text{col}}$)

row: 1 col: 1

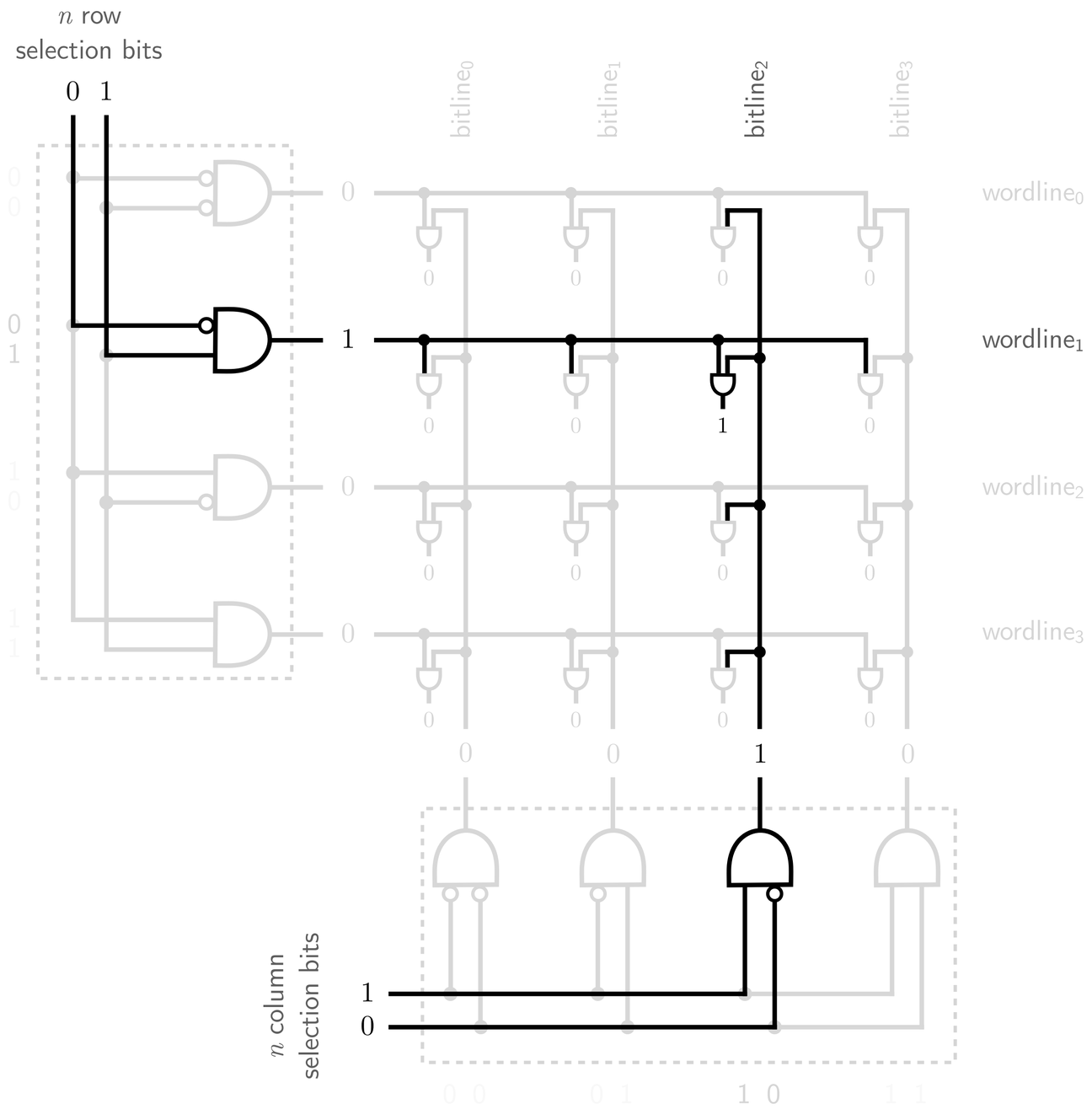


r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at $(\underbrace{0, 1,}_{\text{row}} \underbrace{1, 0}_{\text{col}})$

row: 1 col: 2



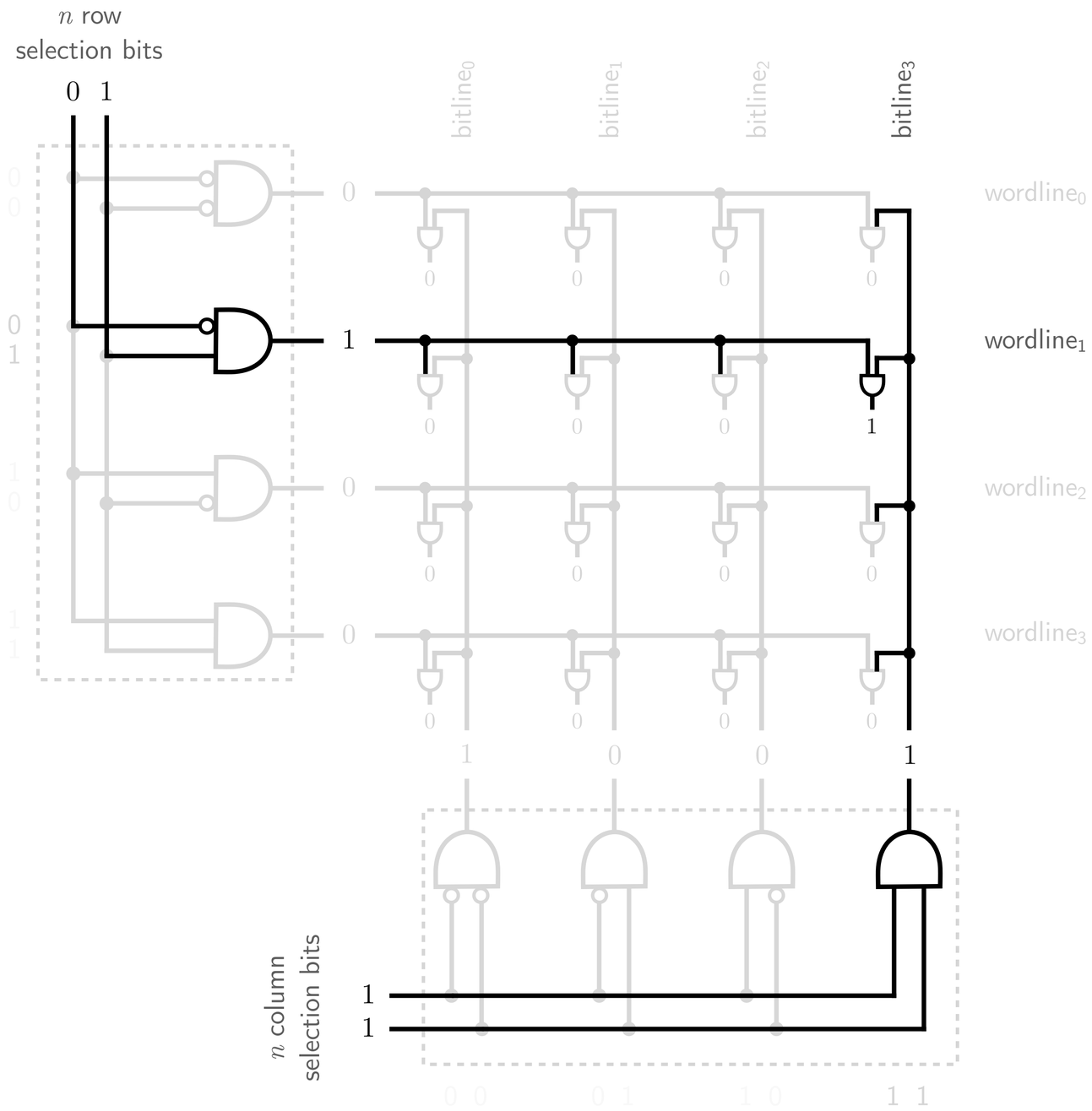
r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at ($\underbrace{0, 1,}_{\text{row}}$ $\underbrace{1, 1}_{\text{col}}$)

row: 1

col: 3

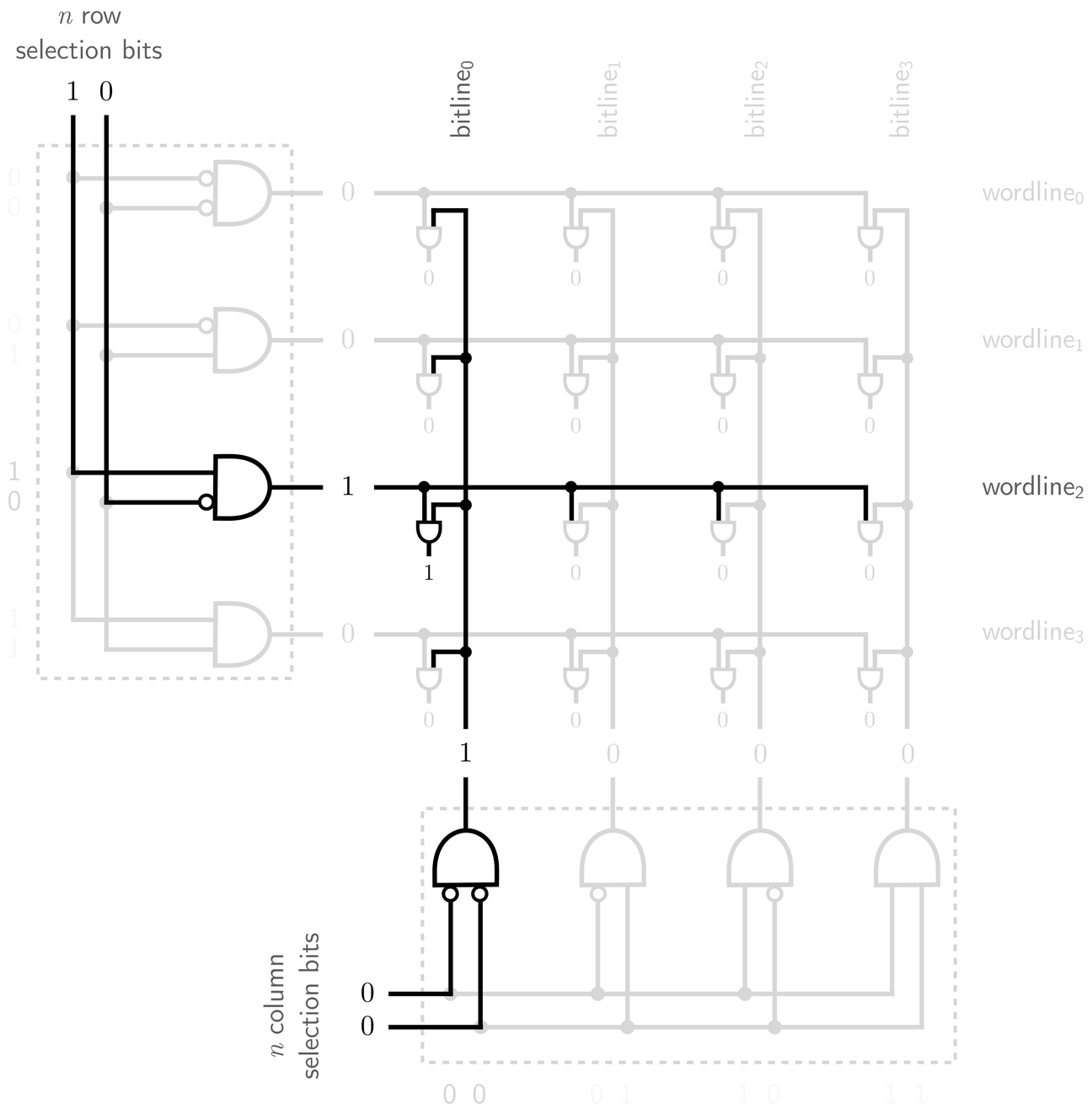


r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at $(\underbrace{1, 0,}_{\text{row}} \underbrace{0, 0}_{\text{col}})$

row: 2 col: 0

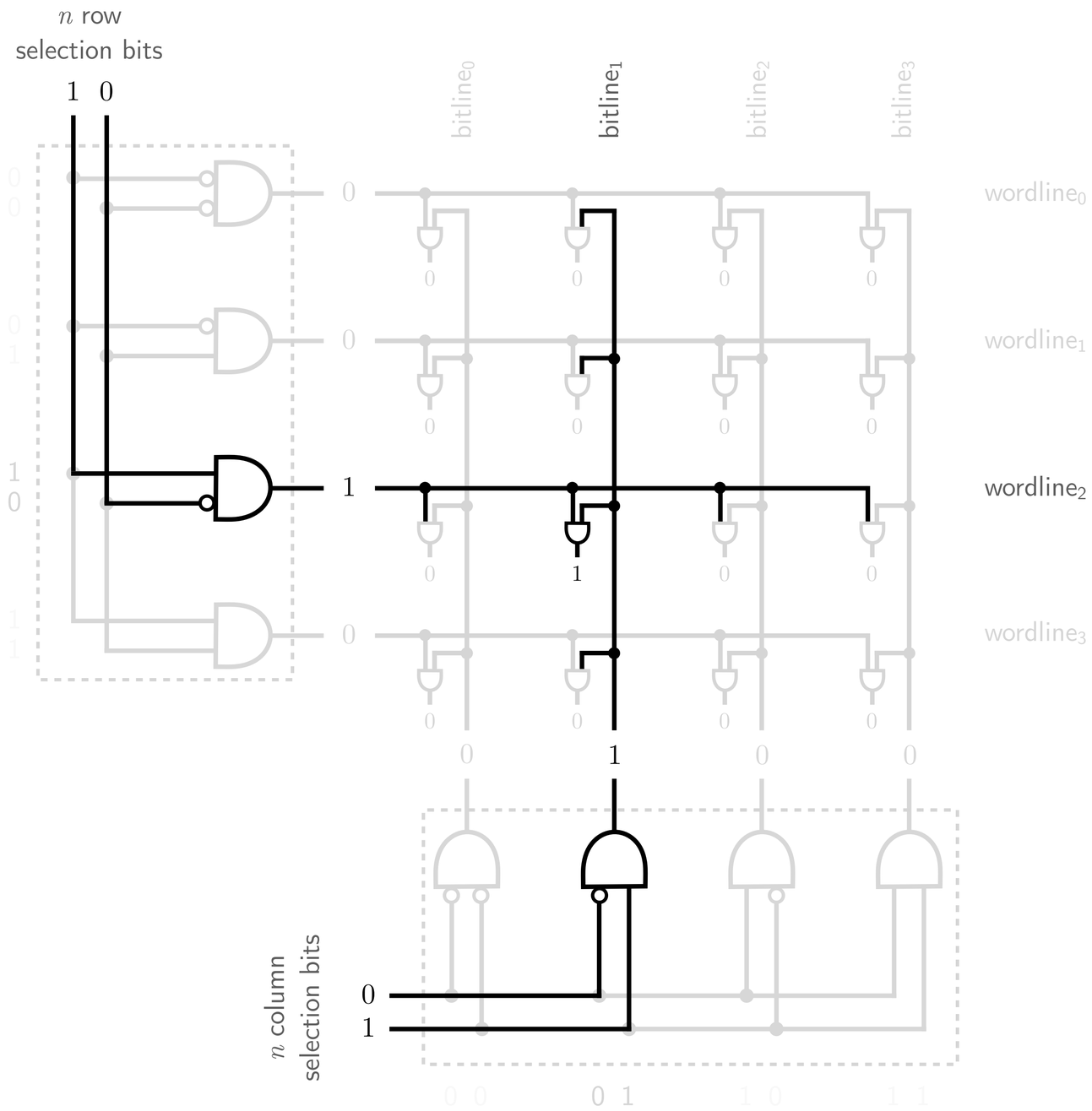


r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at $(\underbrace{1, 0,}_{\text{row}} \underbrace{0, 1}_{\text{col}})$

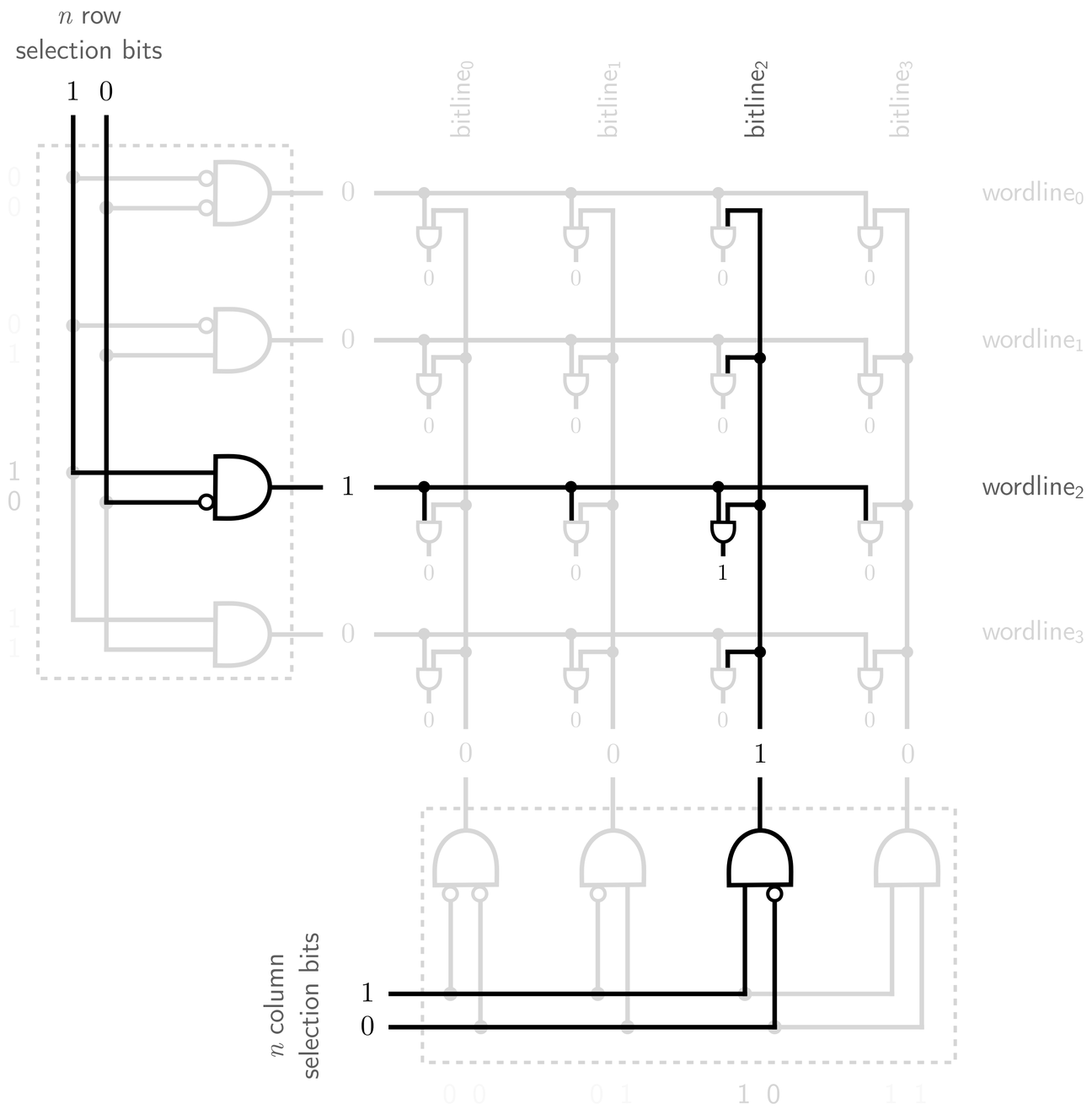
row: 2 col: 1



r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at $(\underbrace{1, 0,}_{\text{row}} \underbrace{1, 0}_{\text{col}})$

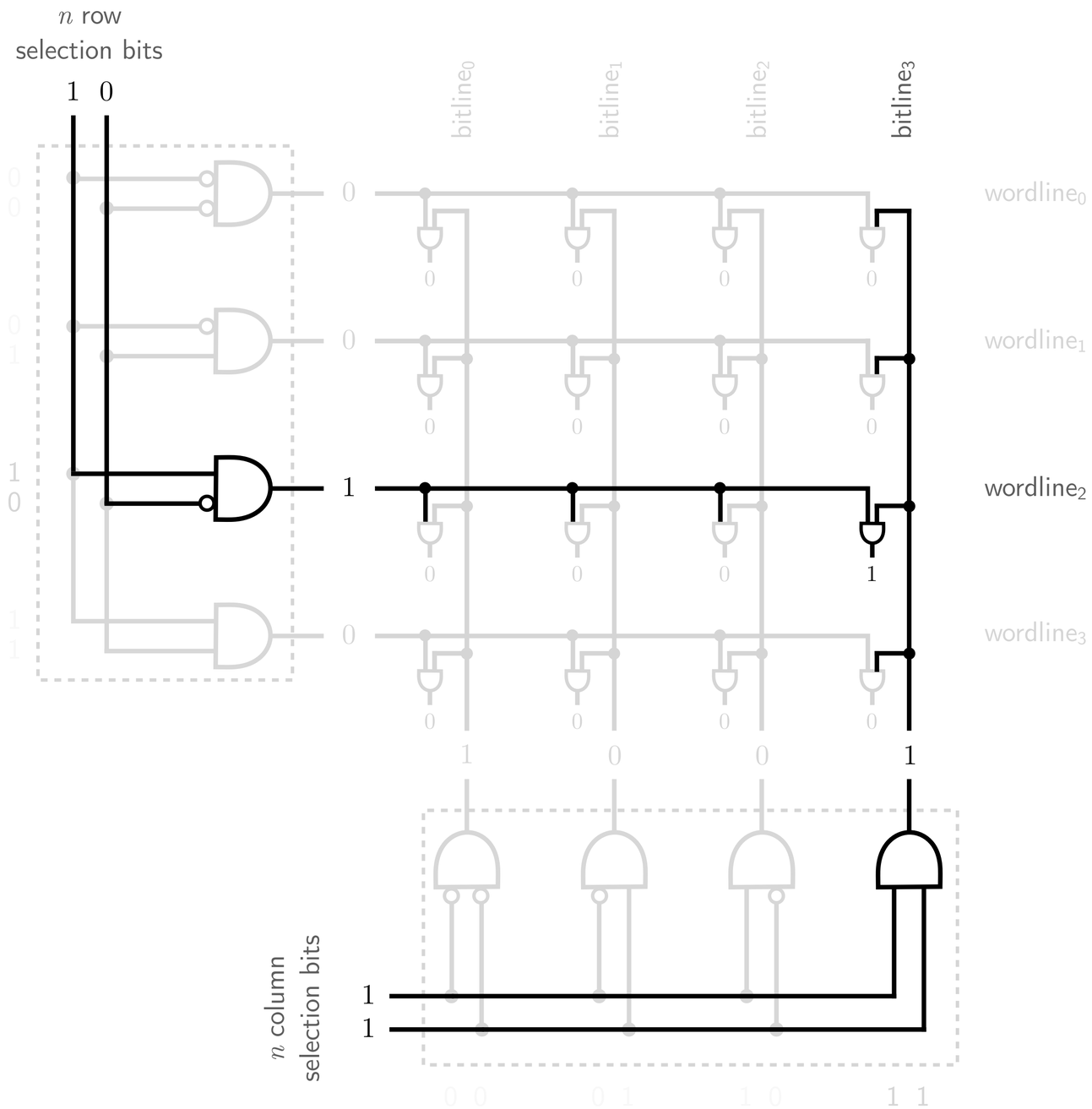


r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at ($\underbrace{1, 0,}_{\text{row}}$ $\underbrace{1, 1}_{\text{col}}$)

row 2 col 3

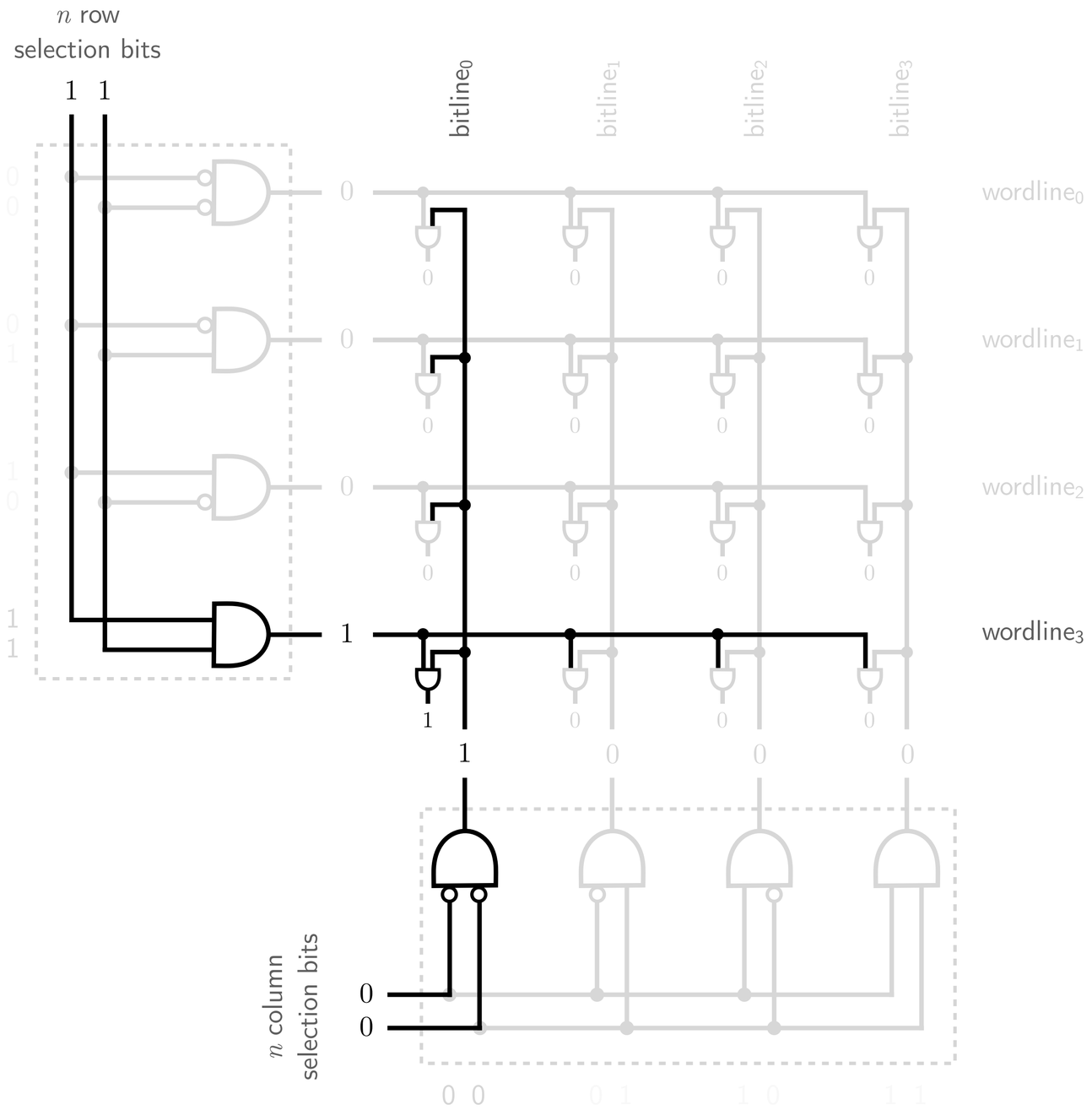


r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at $(\underbrace{1, 1,}_{\text{row}} \underbrace{0, 0}_{\text{col}})$

row: 3 col: 0

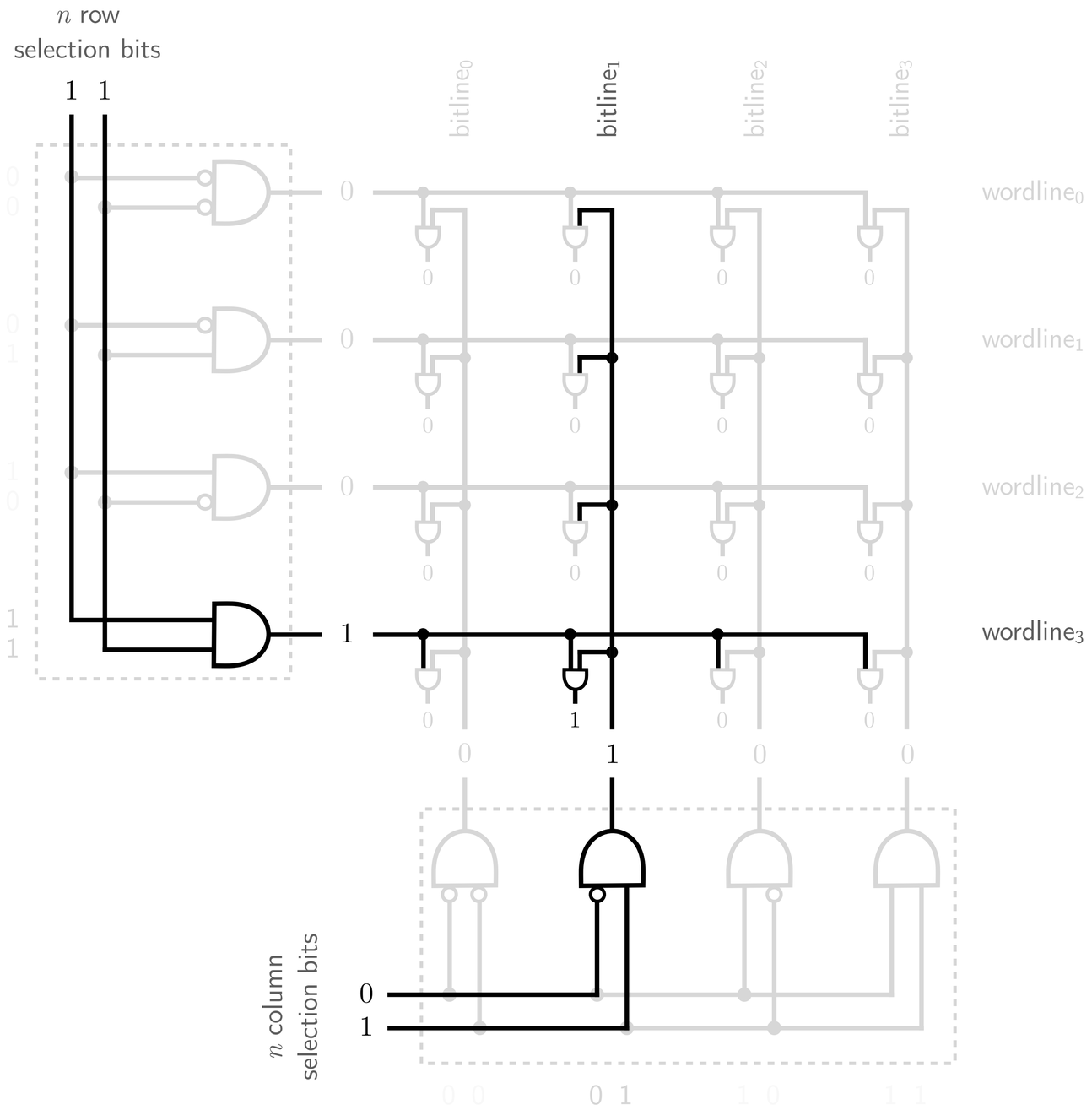


r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at ($\underbrace{1, 1,}_{\text{row}}$ $\underbrace{0, 1}_{\text{col}}$)

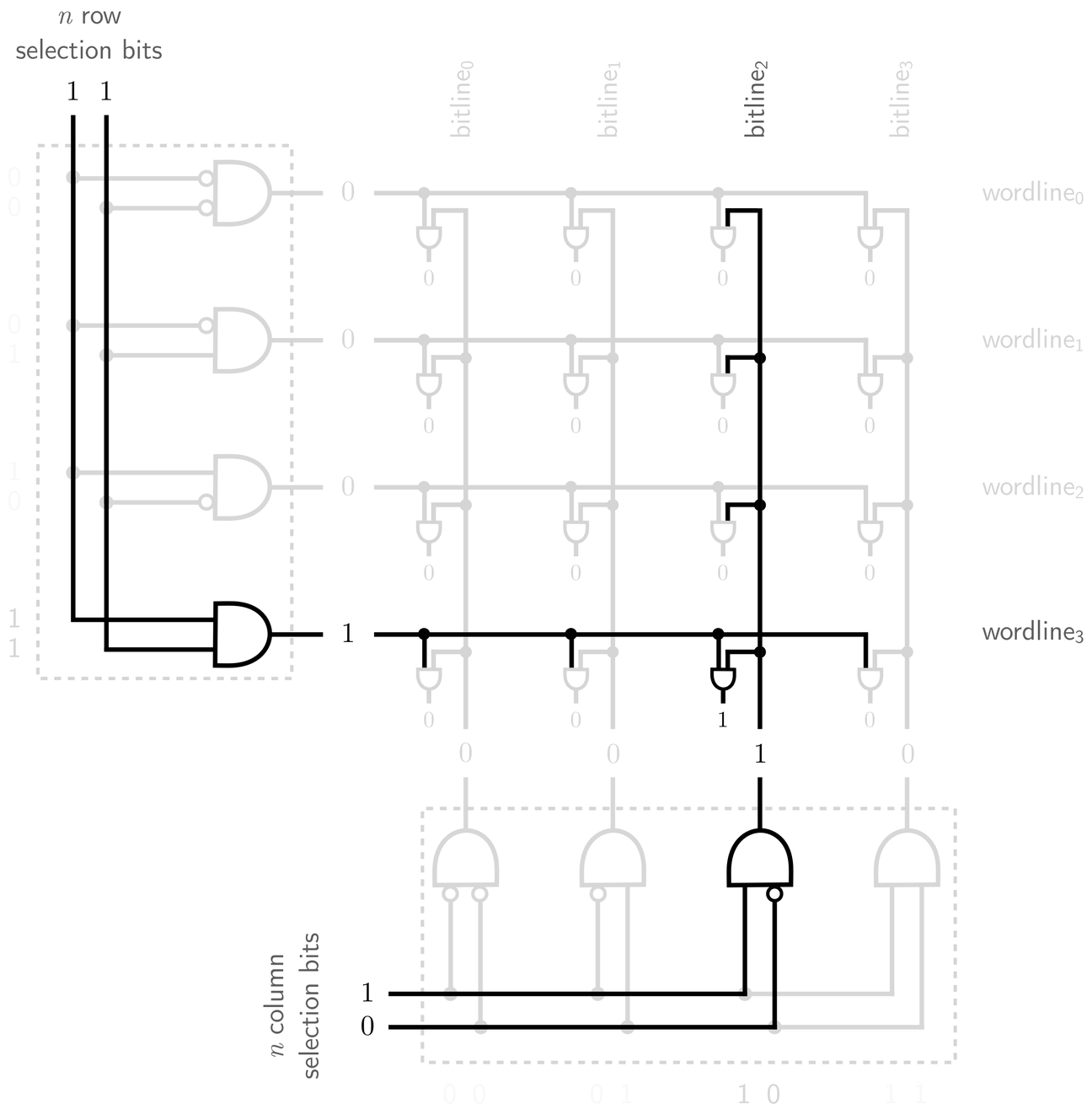
row: 3 col: 1



r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

Cell at $(\underbrace{1, 1,}_{\text{row}} \underbrace{1, 0}_{\text{col}})$

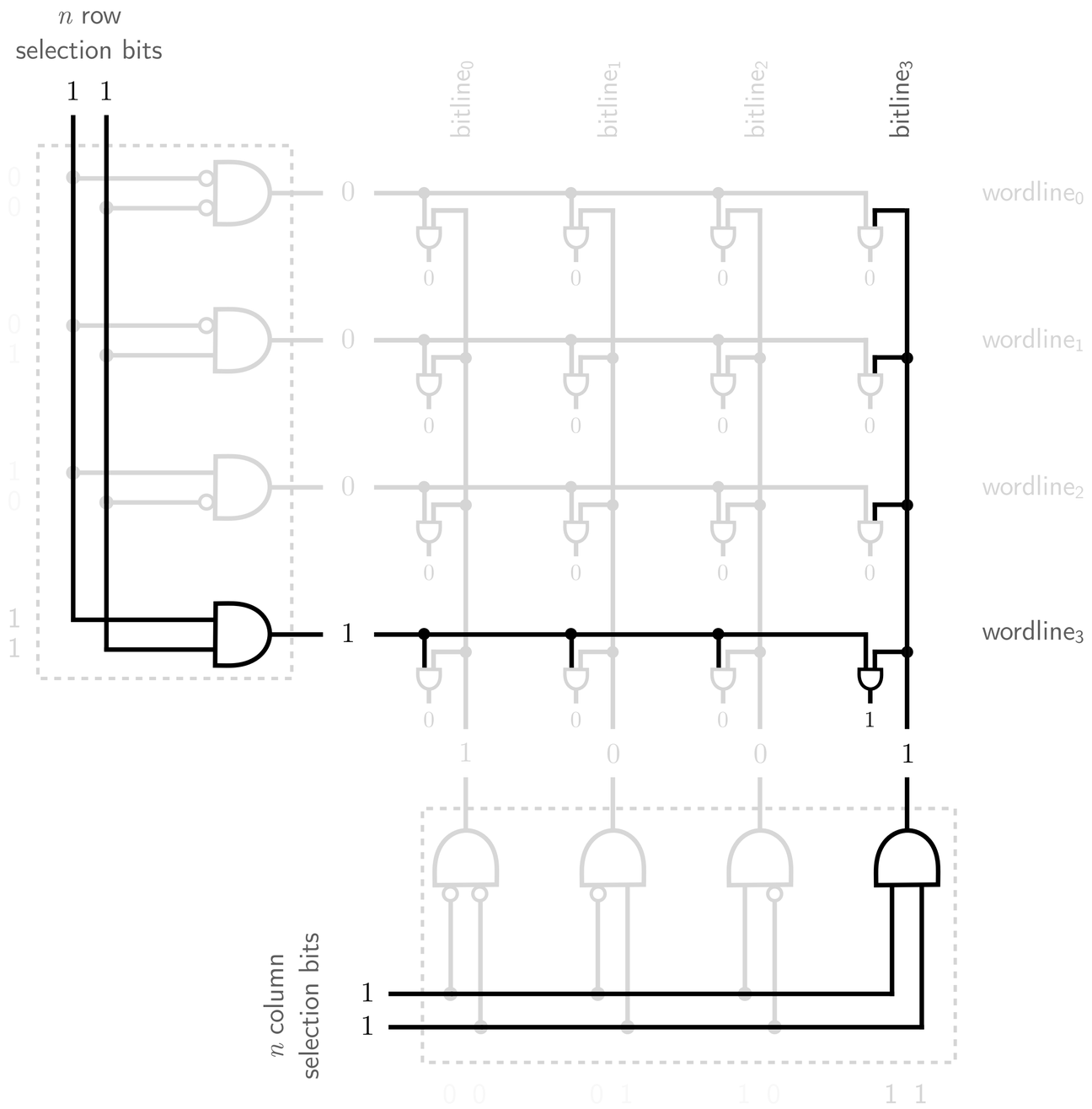


r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

Memory (Random Access Memory, RAM)

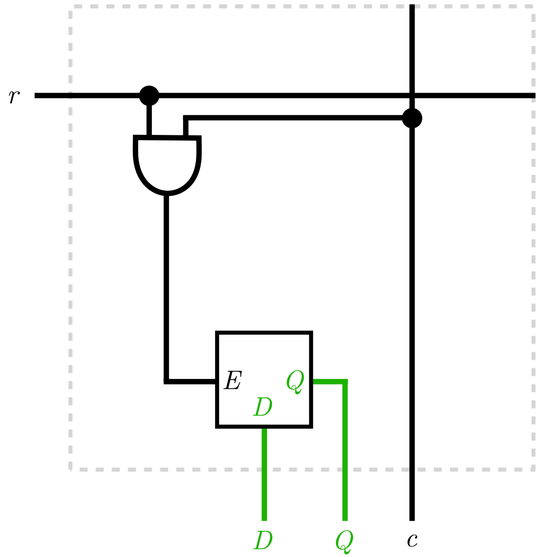
Cell at ($\underbrace{1, 1,}_{\text{row}}$ $\underbrace{1, 1}_{\text{col}}$)

row: 3 col: 3



r_1	r_0	c_1	c_0	Q
0	0	0	0	y_{00}
0	0	0	1	y_{01}
0	0	1	0	y_{02}
0	0	1	1	y_{03}
0	1	0	0	y_{10}
0	1	0	1	y_{11}
0	1	1	0	y_{12}
0	1	1	1	y_{13}
1	0	0	0	y_{20}
1	0	0	1	y_{21}
1	0	1	0	y_{22}
1	0	1	1	y_{23}
1	1	0	0	y_{30}
1	1	0	1	y_{31}
1	1	1	0	y_{32}
1	1	1	1	y_{33}

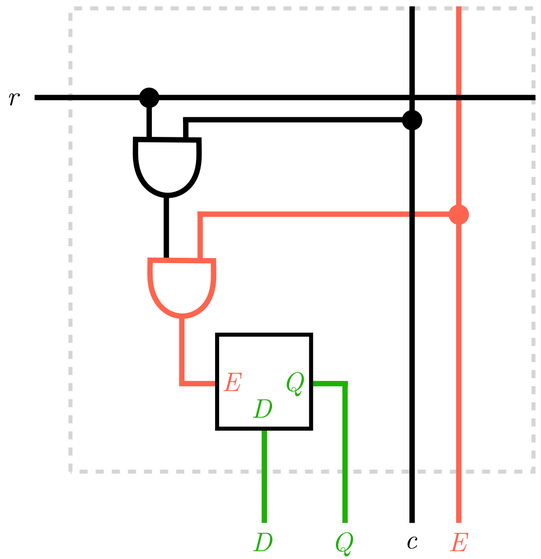
Memory (Random Access Memory, RAM)



Only memory cell at $(r,\ c)$ is active

address	4 bits (2 row bits, 2 col bits)
`ENABLE_READ`	1 bit
`ENABLE_WRITE`	1 bit
data	1 bit

Memory (Random Access Memory, RAM)

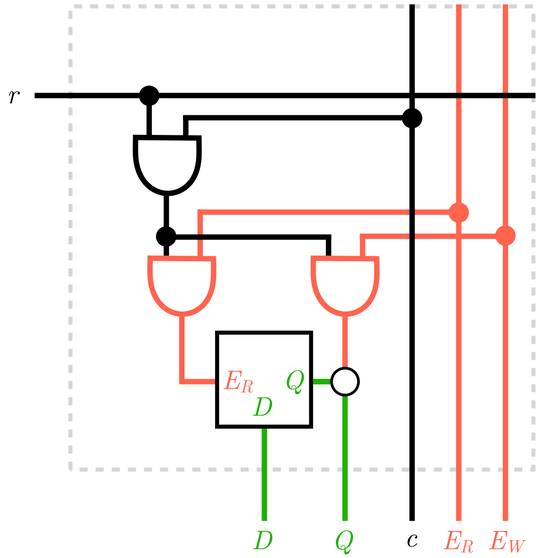


Only memory cell at (r, c) is active

Only if `ENABLE` bit is 1

address	4 bits (2 row bits, 2 col bits)
`ENABLE_READ`	1 bit
`ENABLE_WRITE`	1 bit
data	1 bit

Memory (Random Access Memory, RAM)



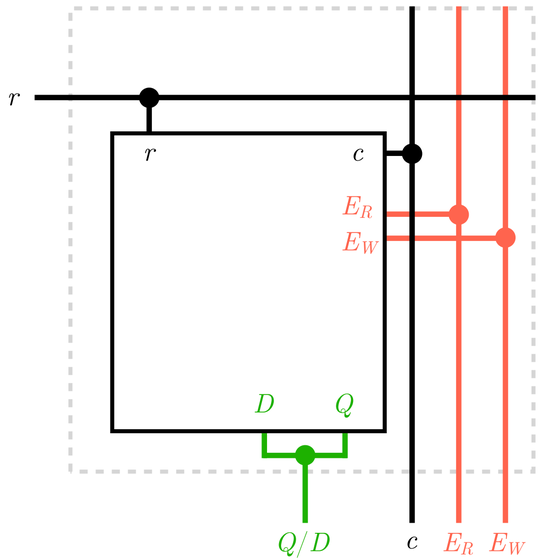
Only memory cell at (r, c) is active

Only if `ENABLE_READ` bit is 1

Only if `ENABLE_WRITE` bit is 1

address	4 bits (2 row bits, 2 col bits)
`ENABLE_READ`	1 bit
`ENABLE_WRITE`	1 bit
data	1 bit

Memory (Random Access Memory, RAM)



Only memory cell at (r, c) is active

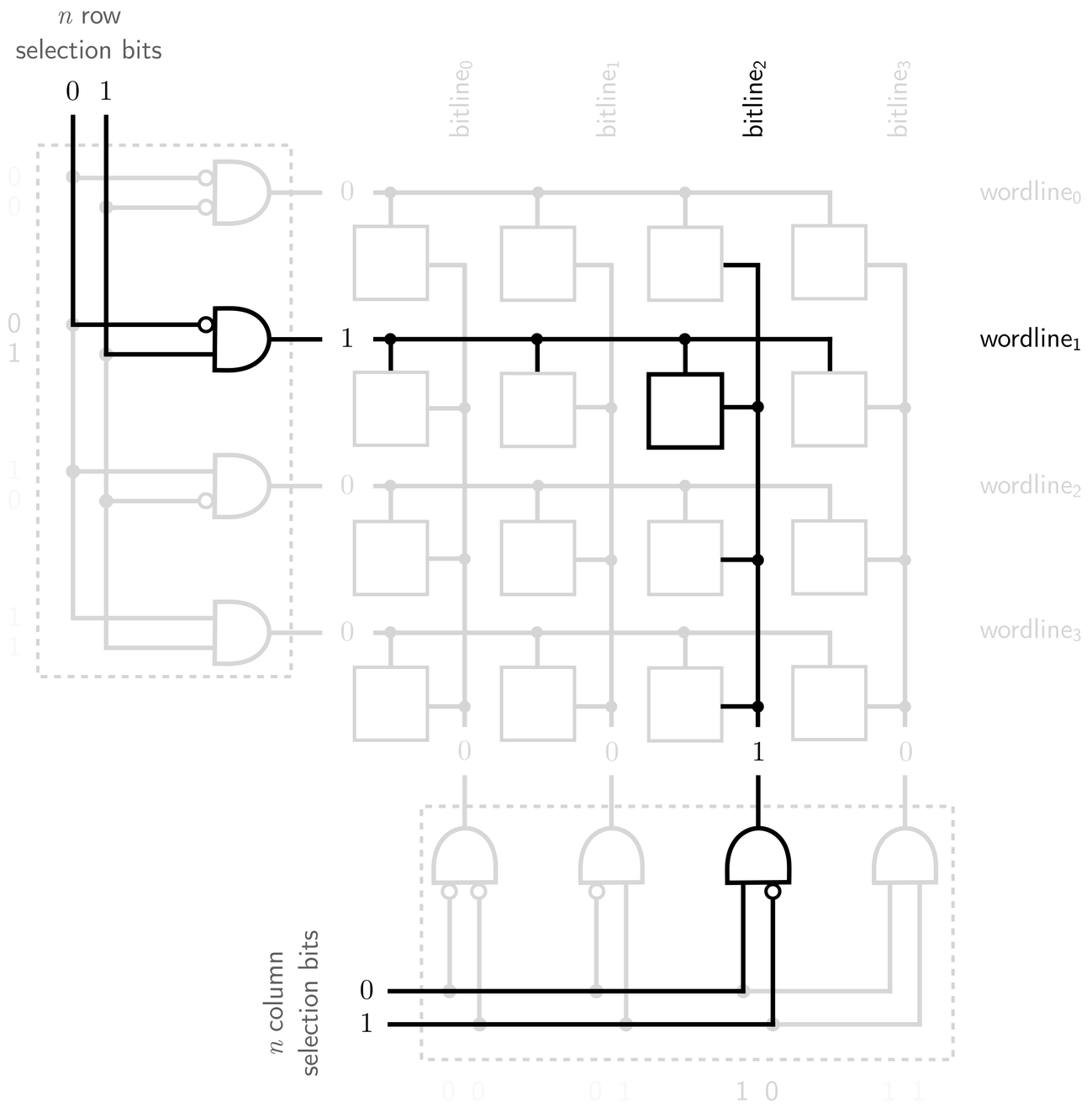
Only if `ENABLE_READ` bit is 1

Only if `ENABLE_WRITE` bit is 1

address	4 bits (2 row bits, 2 col bits)
`ENABLE_READ`	1 bit
`ENABLE_WRITE`	1 bit
data	1 bit

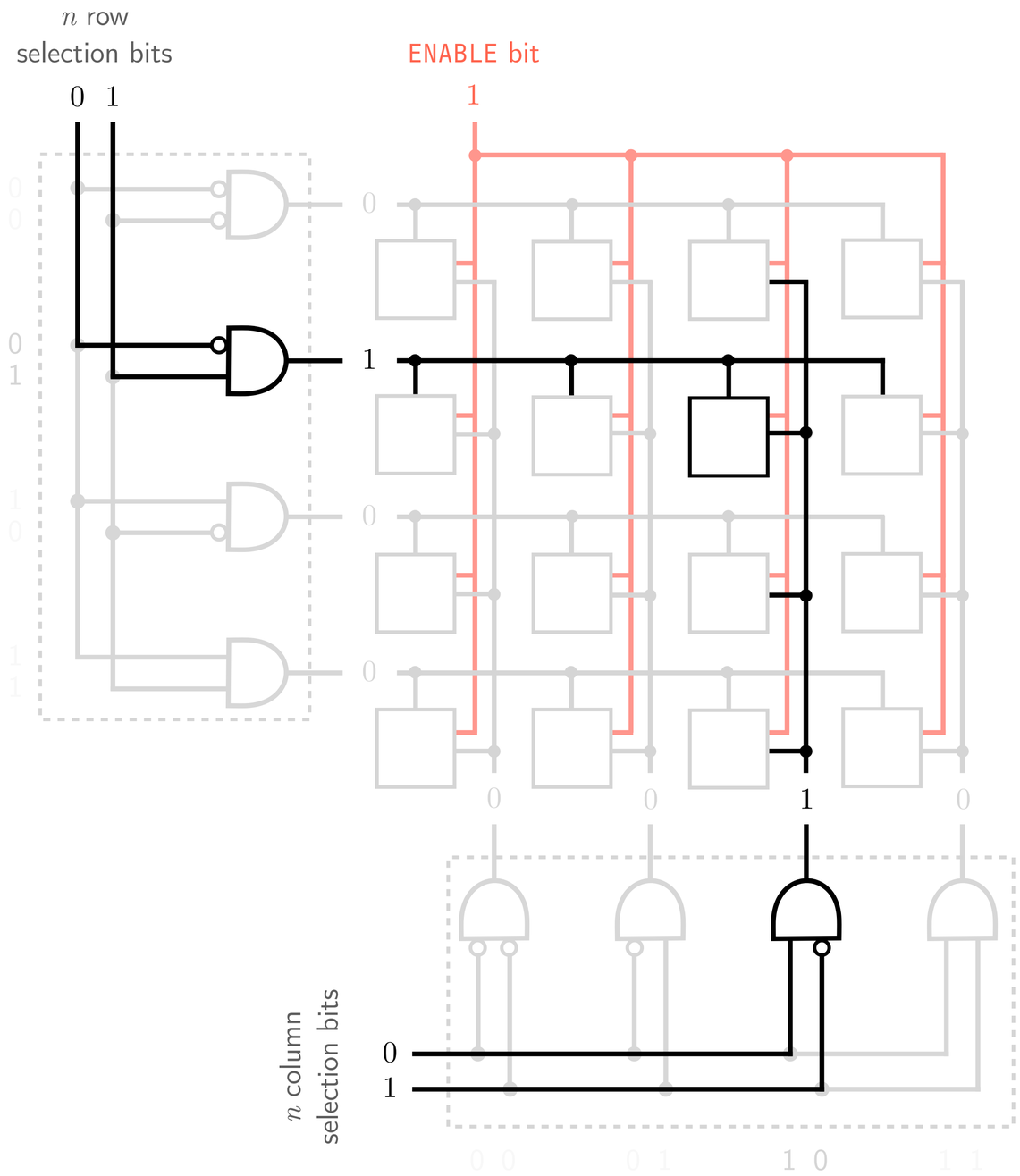
Memory (Random Access Memory, RAM)

Cell at $(\underbrace{0, 1,}_{\text{row}} \underbrace{1, 0}_{\text{col}})$

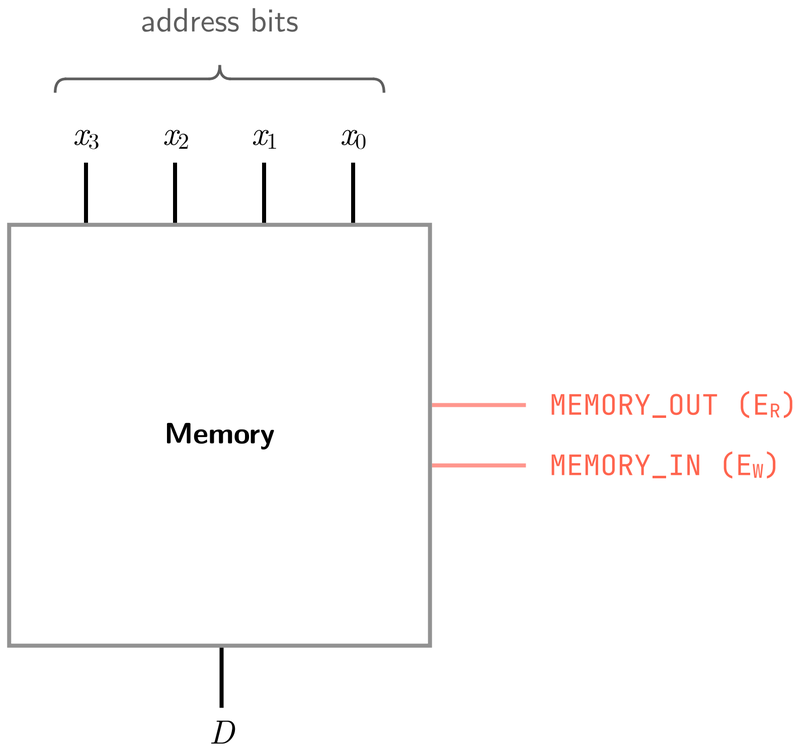


Memory (Random Access Memory, RAM)

Cell at $(\underbrace{0, 1,}_{\text{row}} \underbrace{1, 0}_{\text{col}})$

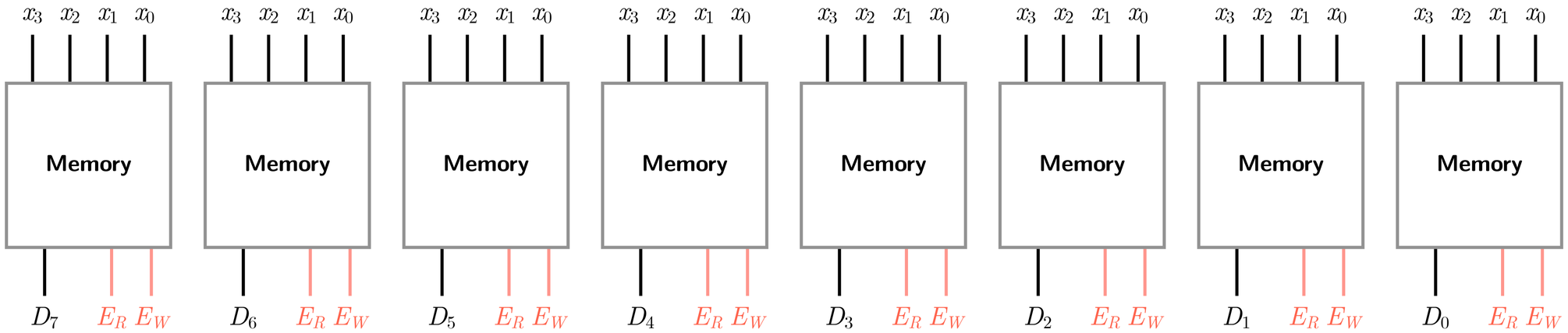


Memory (Random Access Memory, RAM)

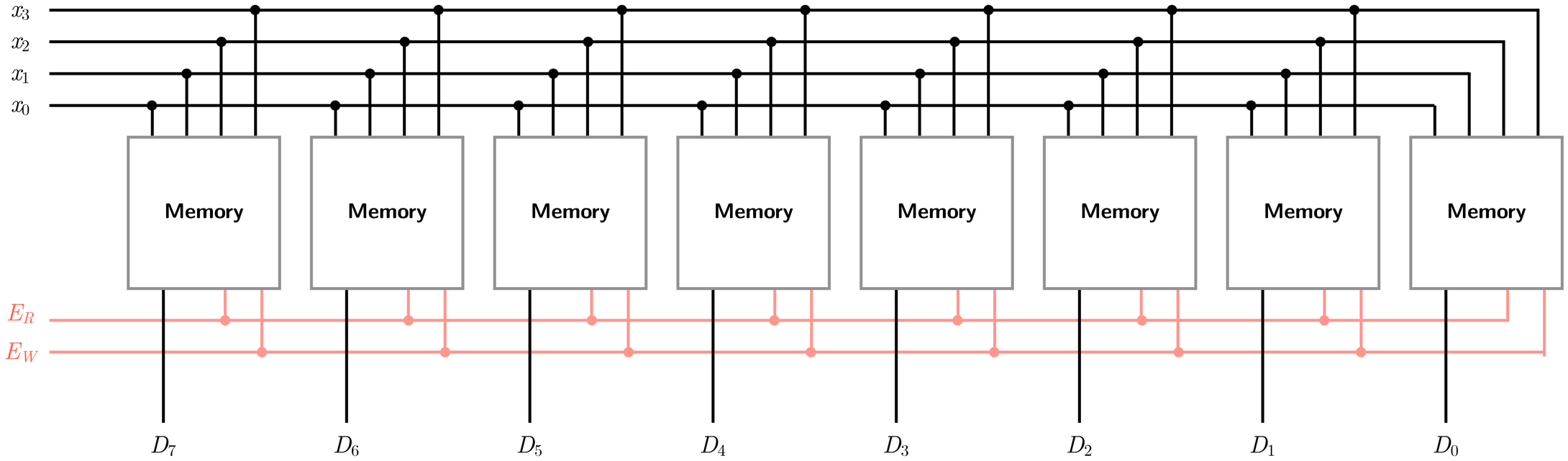


The design we have seen so far only stores 1 bit, but we can use 8 copies of this system to store 1 byte. Each bit in a byte has the same address in its respective 'copy'.

Bit interleaving



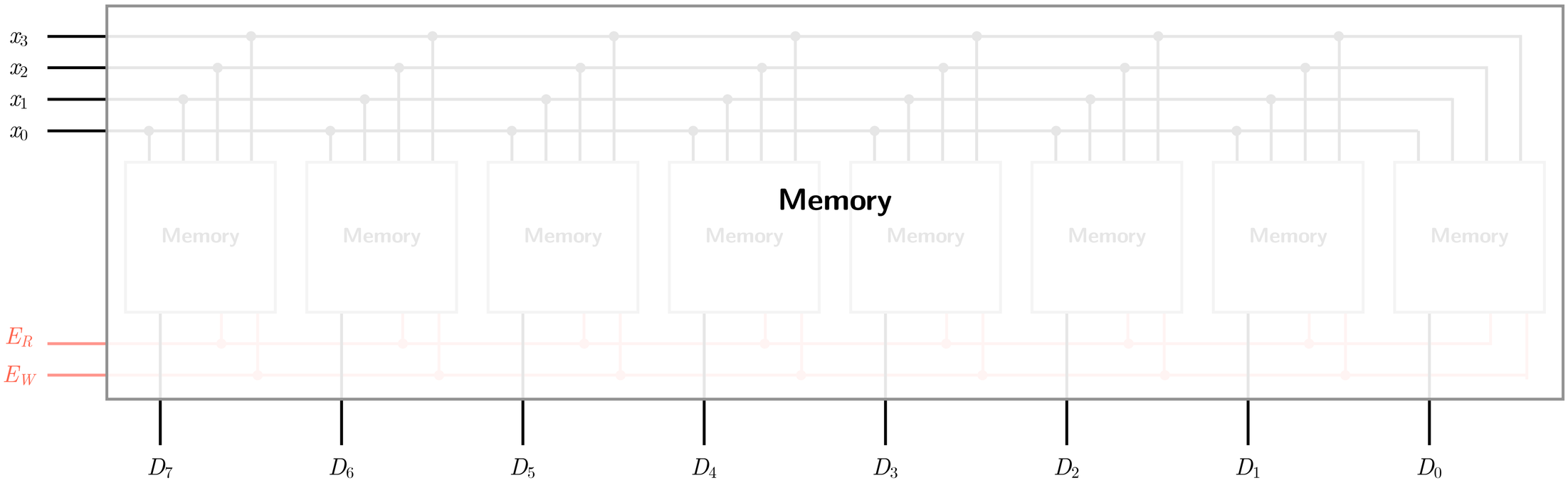
Bit interleaving



Each bit in the **byte** $D_7D_6D_5D_4D_3D_2D_1D_0$ is stored in a separate matrix at the same address.

This is just one of many possible ways of organizing memory.

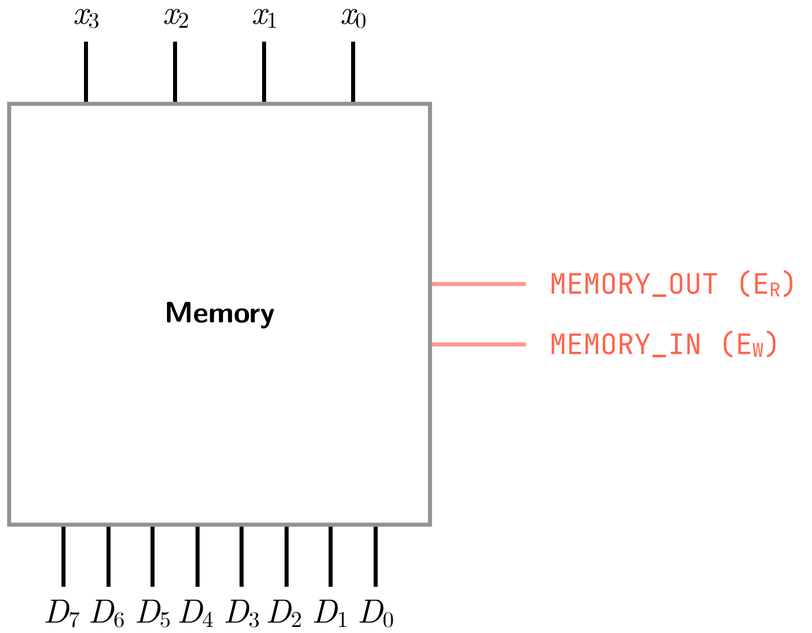
Bit interleaving



Each bit in the **byte** $D_7D_6D_5D_4D_3D_2D_1D_0$ is stored in a separate matrix at the same address.

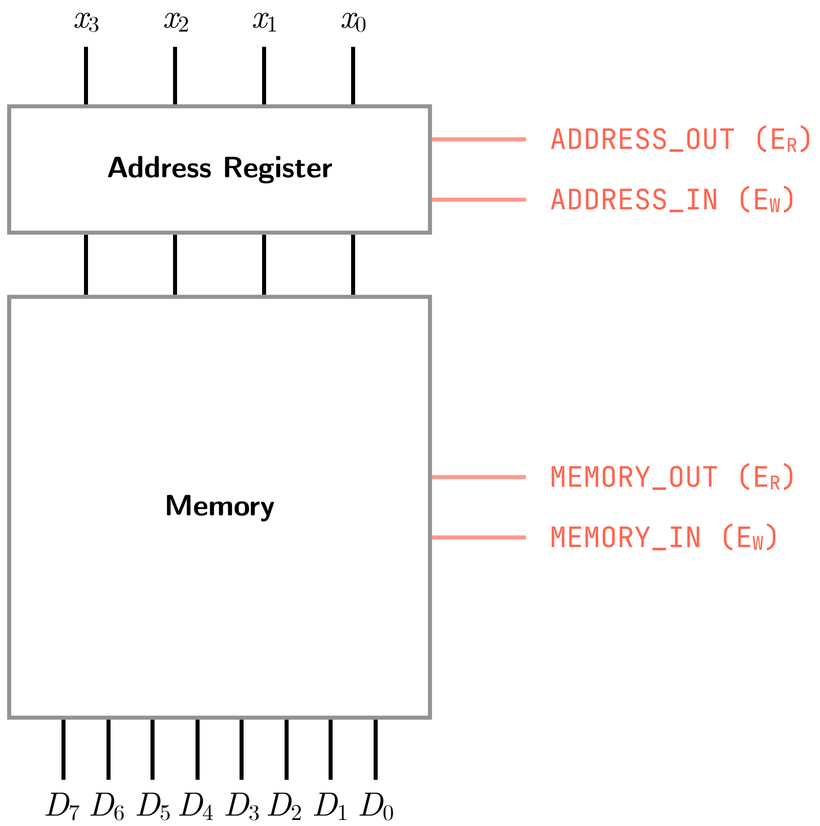
This is just one of many possible ways of organizing memory.

Memory (Random Access Memory, RAM)



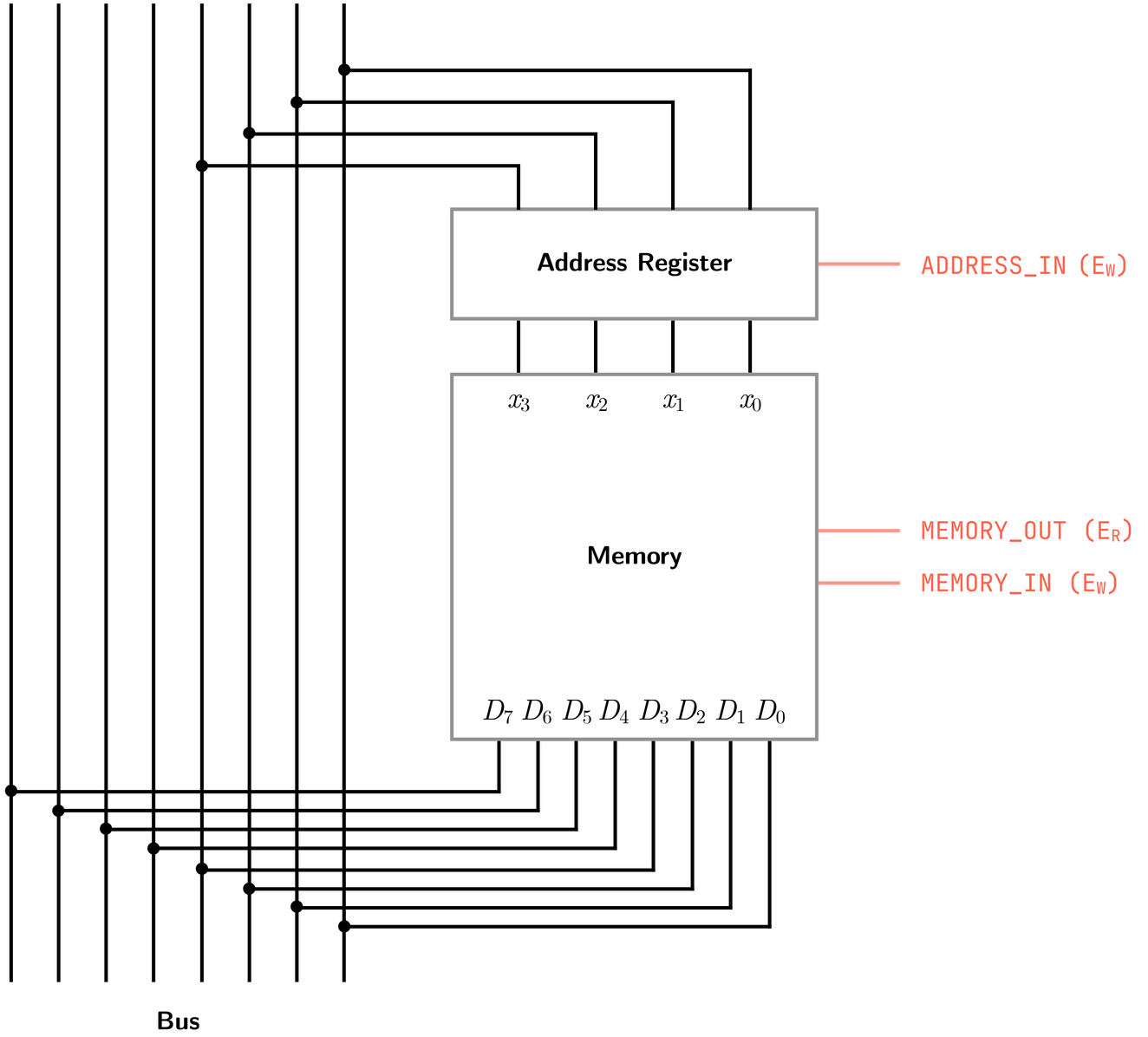
The design we have seen so far only stores 1 bit, but we can use 8 copies of this system to store 1 byte. Each bit in a byte has the same address in its respective 'copy'.

Memory (Random Access Memory, RAM)



The design we have seen so far only stores 1 bit, but we can use 8 copies of this system to store 1 byte. Each bit in a byte has the same address in its respective 'copy'.

Memory (Random Access Memory, RAM)



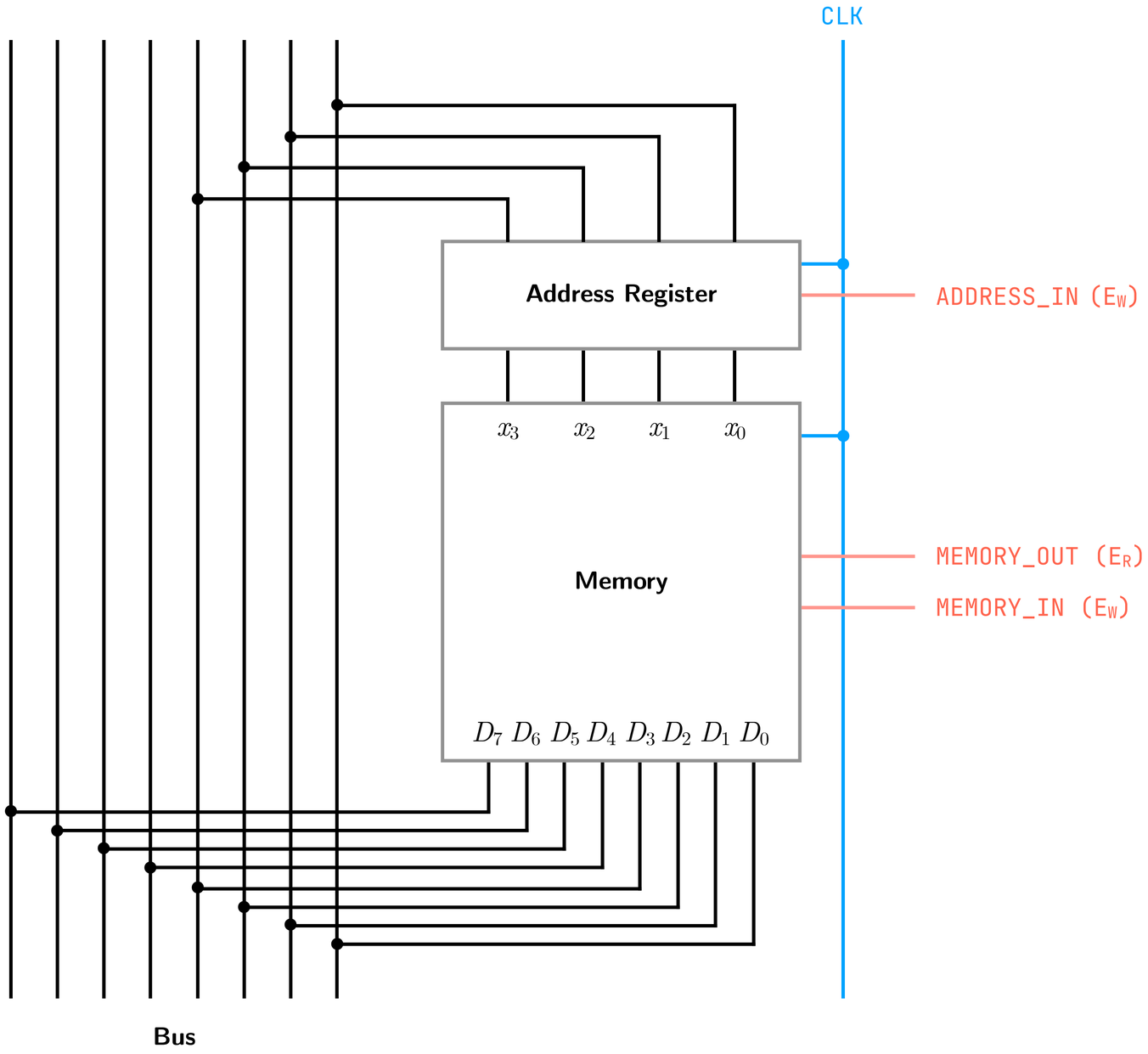
We are always "pointing" at the memory address specified in the address register.

When we do `MEMORY_OUT`, we get the data at the address specified in the address register.

The design we have seen so far only stores 1 bit, but we can use 8 copies of this system to store 1 byte. Each bit in a byte has the same address in its respective 'copy'.

By the way, we can use the same data lines for both reading and writing.

Memory (Random Access Memory, RAM)



We are always "pointing" at the memory address specified in the address register.

When we do `MEMORY_OUT`, we get the data at the address specified in the address register.

The design we have seen so far only stores 1 bit, but we can use 8 copies of this system to store 1 byte. Each bit in a byte has the same address in its respective 'copy'.

By the way, we can use the same data lines for both reading and writing.

Memory (Random Access Memory, RAM)

Random access is the ability to access an arbitrary element of a sequence in equal time from a population of addressable elements roughly as easily and efficiently as any other, no matter how many elements may be in the set.
A better name might be *arbitrary access*.

In computer science it is typically contrasted to **sequential access** which requires data to be retrieved in the order it was stored.

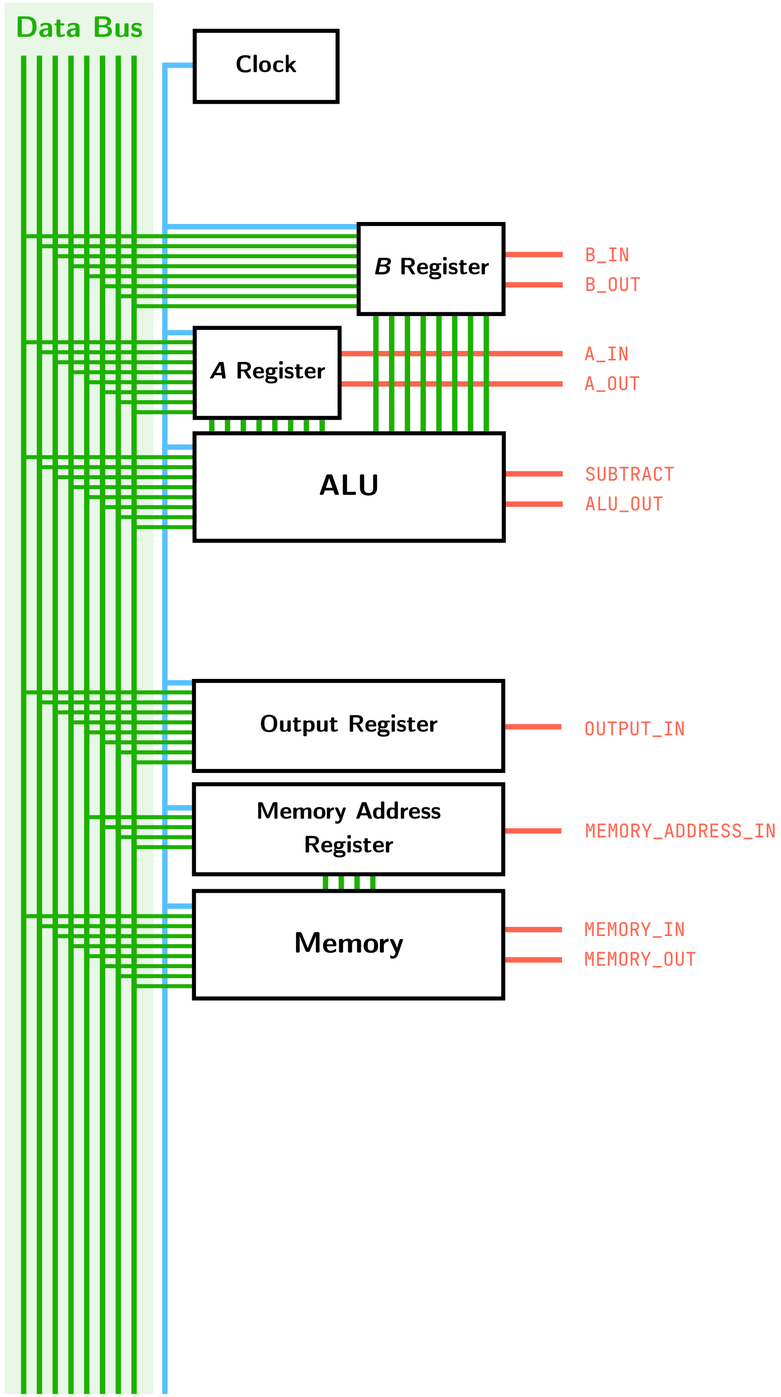
7

Control

The components that we have already talked about connected via a common bus
With control signals that we can set manually

Based on SAP-1 (also see Ben Eater)

Simplification: I'm only showing one connection between component and bus, but of course there are two: one for reading from the bus and one for writing to the bus
(We could actually use a bidirectional register though, then this is correct. But not going into that here...)



Instruction Register

We can have any number of registers that we like

Some registers might be used for specific purposes, rather than be general-purpose registers that any component can read from or write to

One specialized register is the **instruction register**

Program Counter (aka Instruction Pointer)

Other names are: **instruction pointer** and **instruction address register**, which might help make sense of what its purpose is.

We count the instructions; we start at 0, and each time an instruction is finished, we'll increment it

If addresses are, say, 4 bits, then the program counter is simply a register with 4 bits in it,

where these 4 bits constitute a counter; i.e. we start at 0000 and increment it under certain circumstances using control signals

Before going into the other details of the program counter,

let's quickly have a look at how we have to modify the binary counter that we have encountered before (T flip-flop)

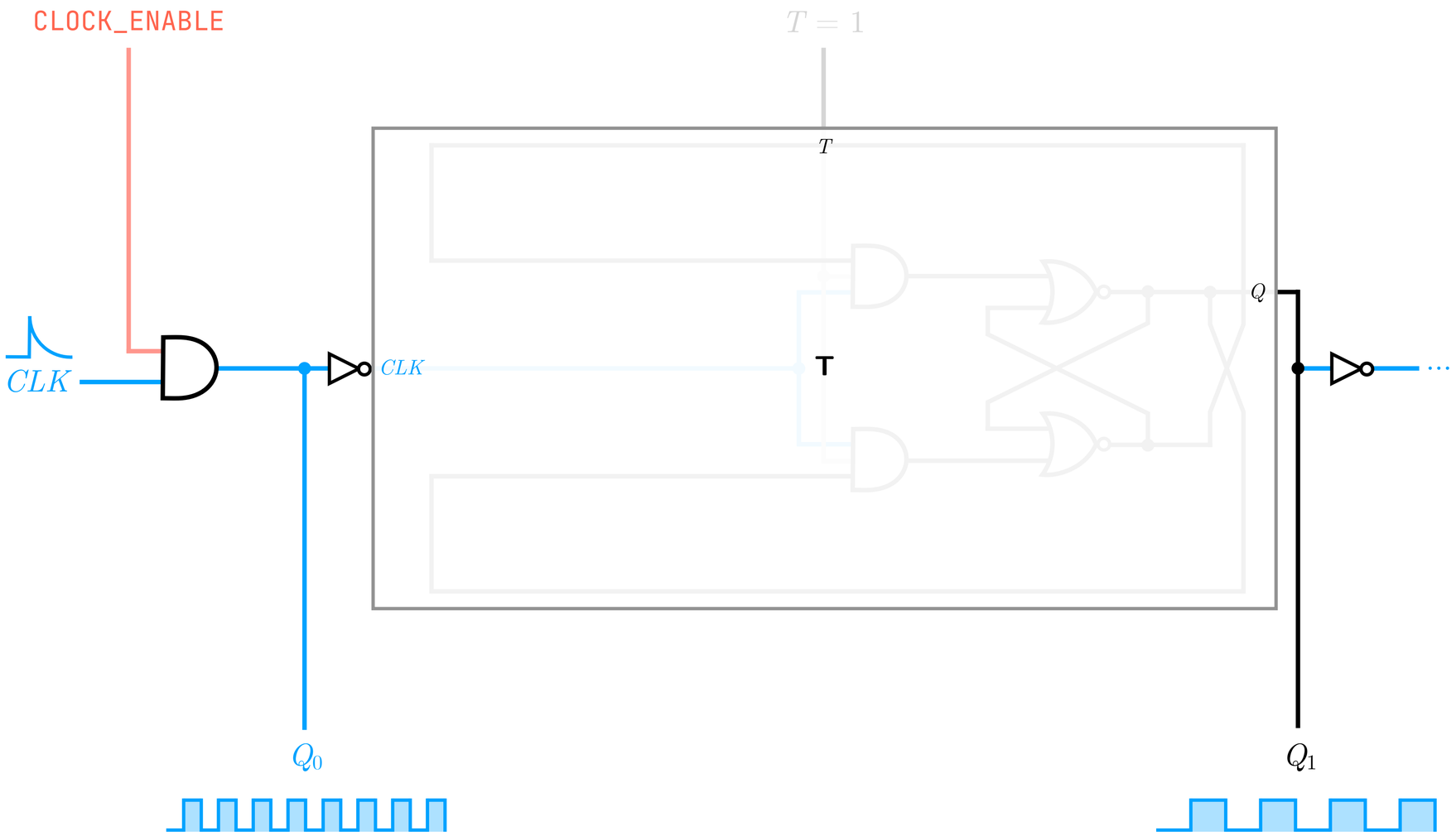
Program Counter (aka Instruction Pointer)

We already know how to make a binary counter using T flip-flops.

Unlike the ordinary binary counter, we only want the program counter to increment when an instruction is finished
(in which case we want to go to the next instruction), and an instruction might take multiple clock cycles.
I.e. it should not simply increment with each clock pulse, but only when we tell it to (by providing a control signal)

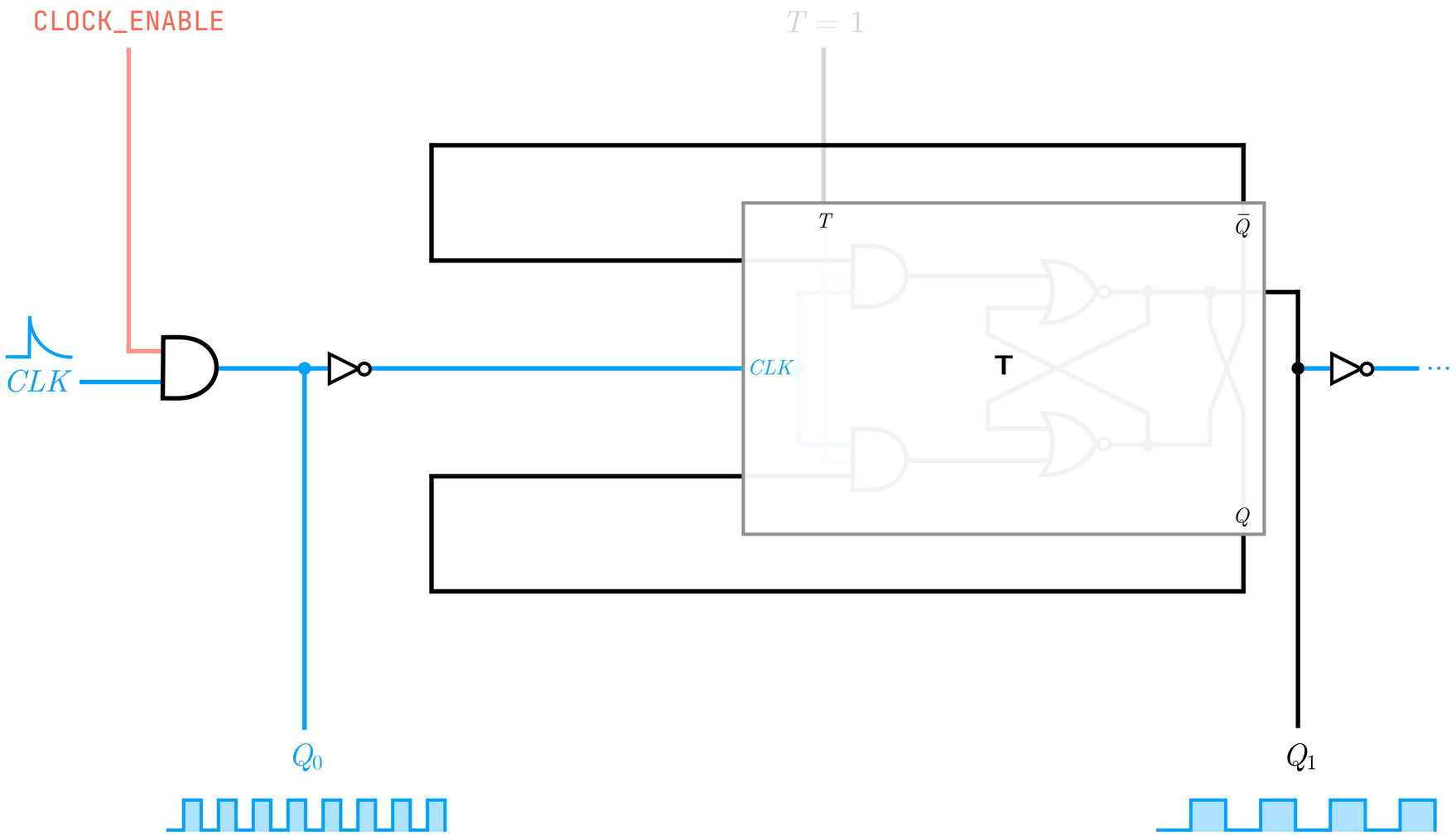
But we also want to be able to write an arbitrary instruction address into the register rather than just incrementing it.
For this, we need another control signal and some additional logic.

Program Counter (aka Instruction Pointer)

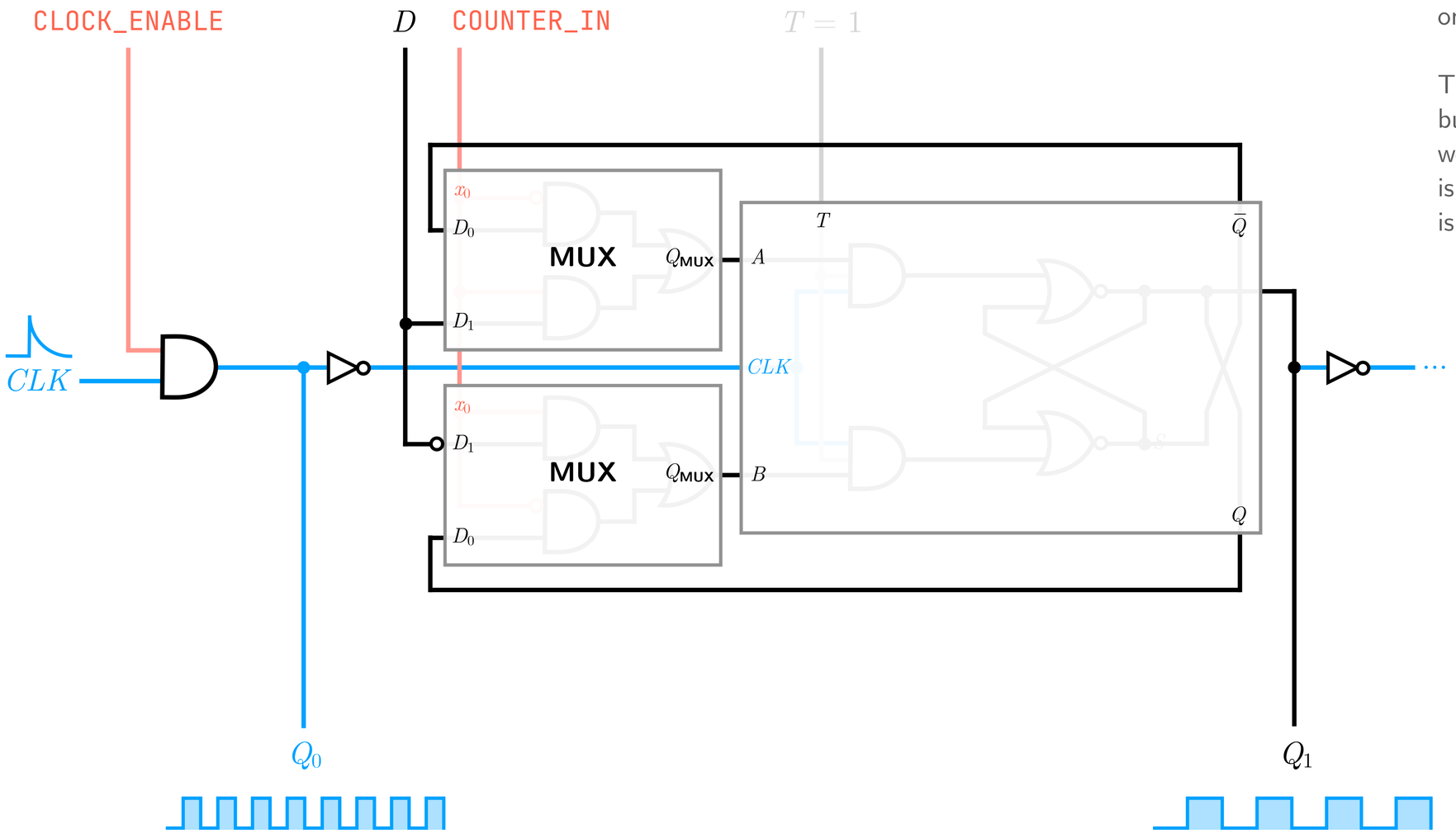


Clock gating is not recommended in more complex computers, but fine in this example.

Program Counter (aka Instruction Pointer)



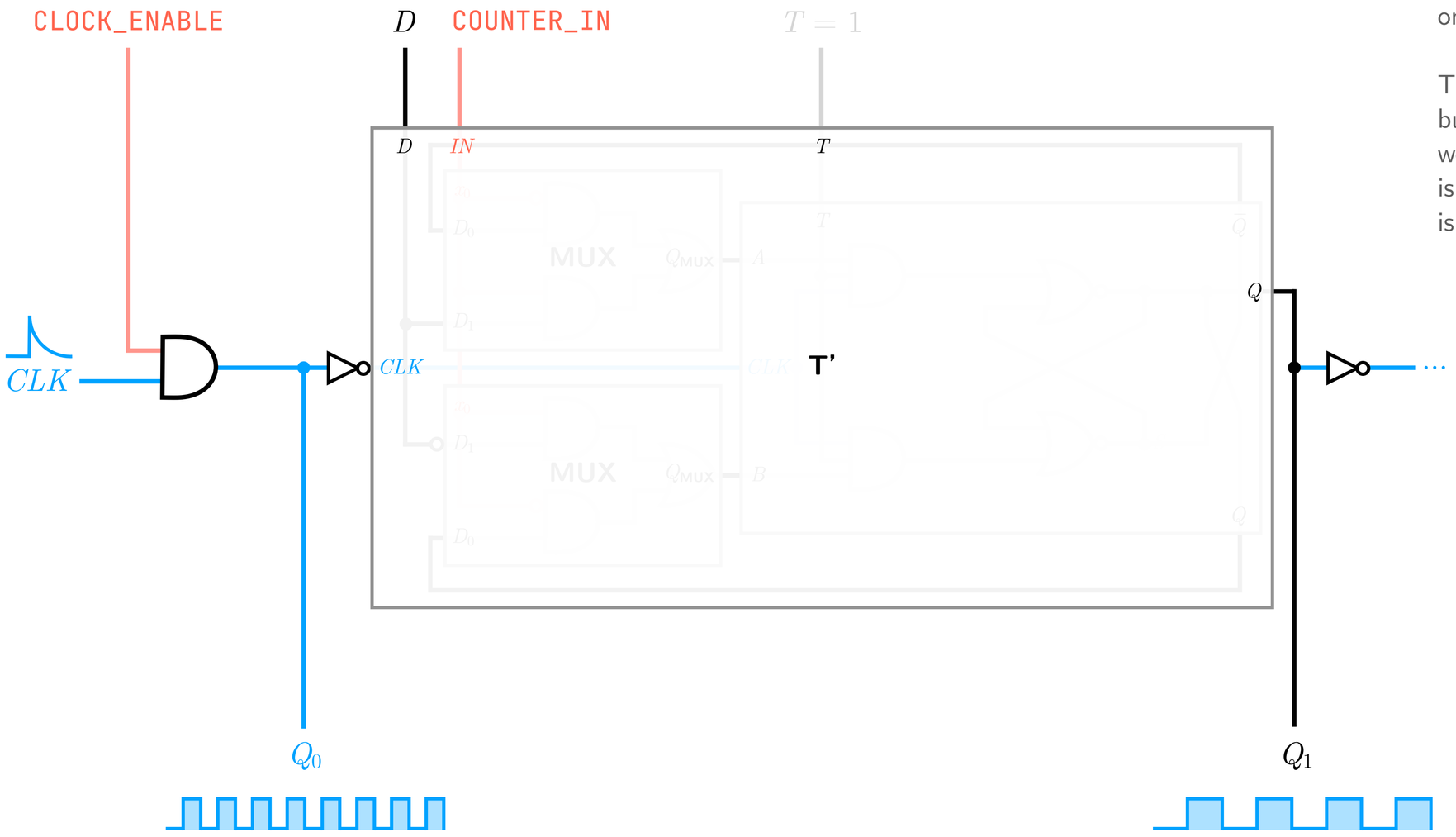
Program Counter (aka Instruction Pointer)



Same ENABLE logic as before (MUX), but we need it for each input separately, one using Q and the other notQ.

T flip flop (fixed T=J=K=1), but instead of just feeding (Q, notQ) back, we instead feed (D, D) if COUNTER_IN is active (and only if CLOCK_ENABLE is active and there's a clock pulse)

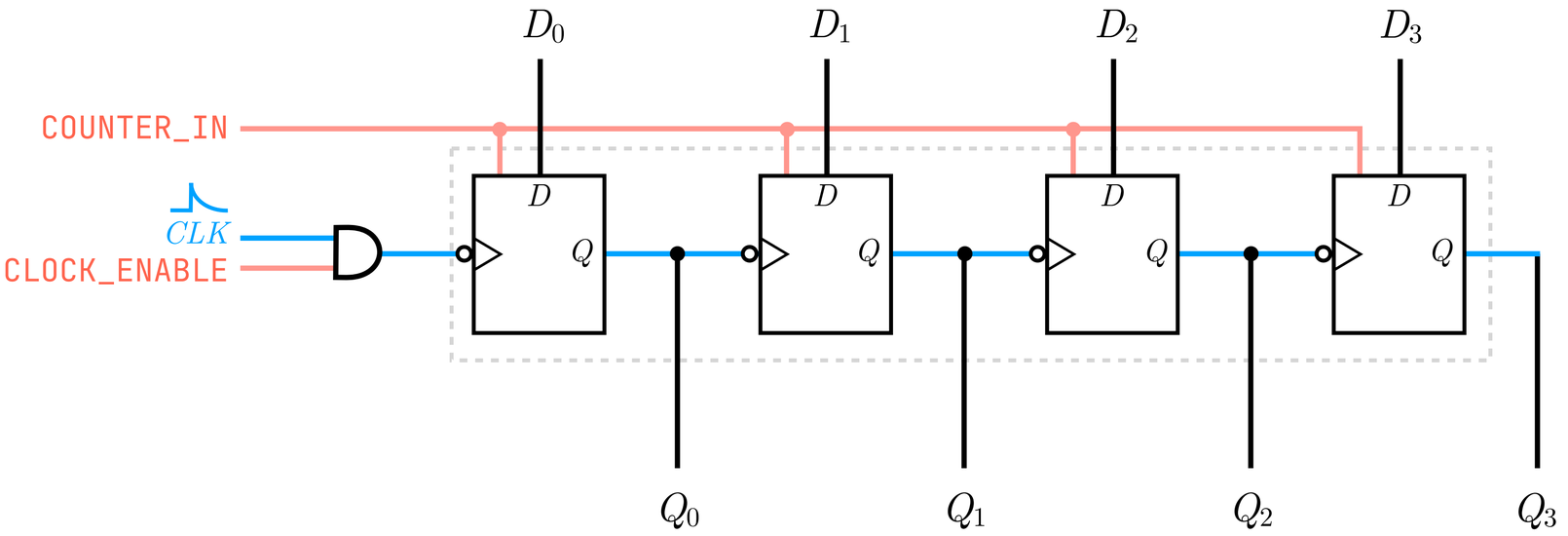
Program Counter (aka Instruction Pointer)



Same ENABLE logic as before (MUX), but we need it for each input separately, one using Q and the other notQ.

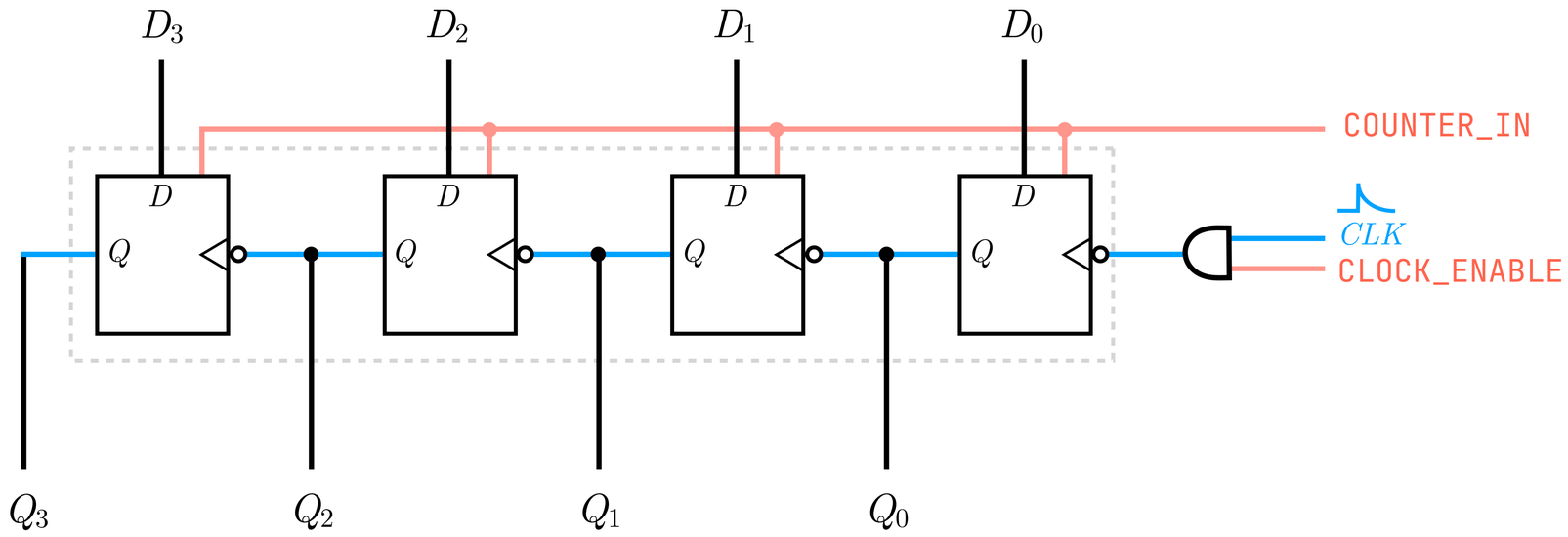
T flip flop (fixed T=J=K=1), but instead of just feeding (Q, notQ) back, we instead feed (D, D) if COUNTER_IN is active (and only if CLOCK_ENABLE is active and there's a clock pulse)

Program Counter (aka Instruction Pointer)



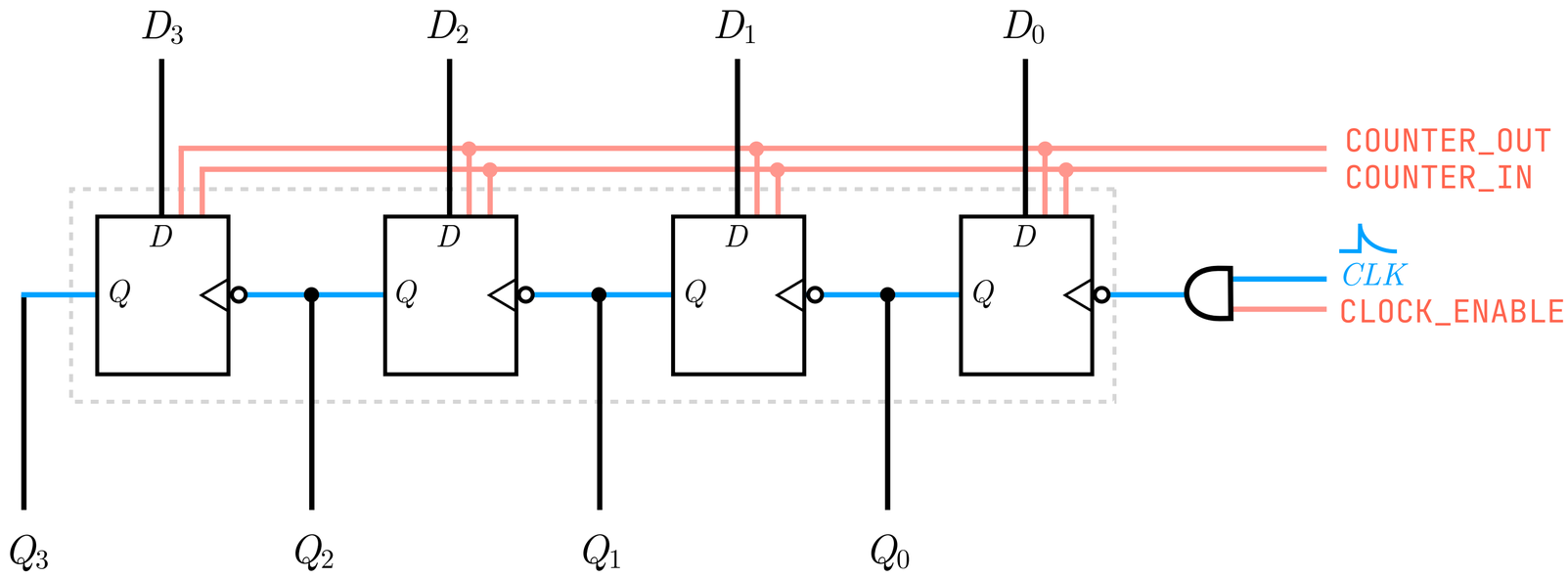
Program Counter (aka Instruction Pointer)

We can flip our diagram, so the least significant bit is on the right.

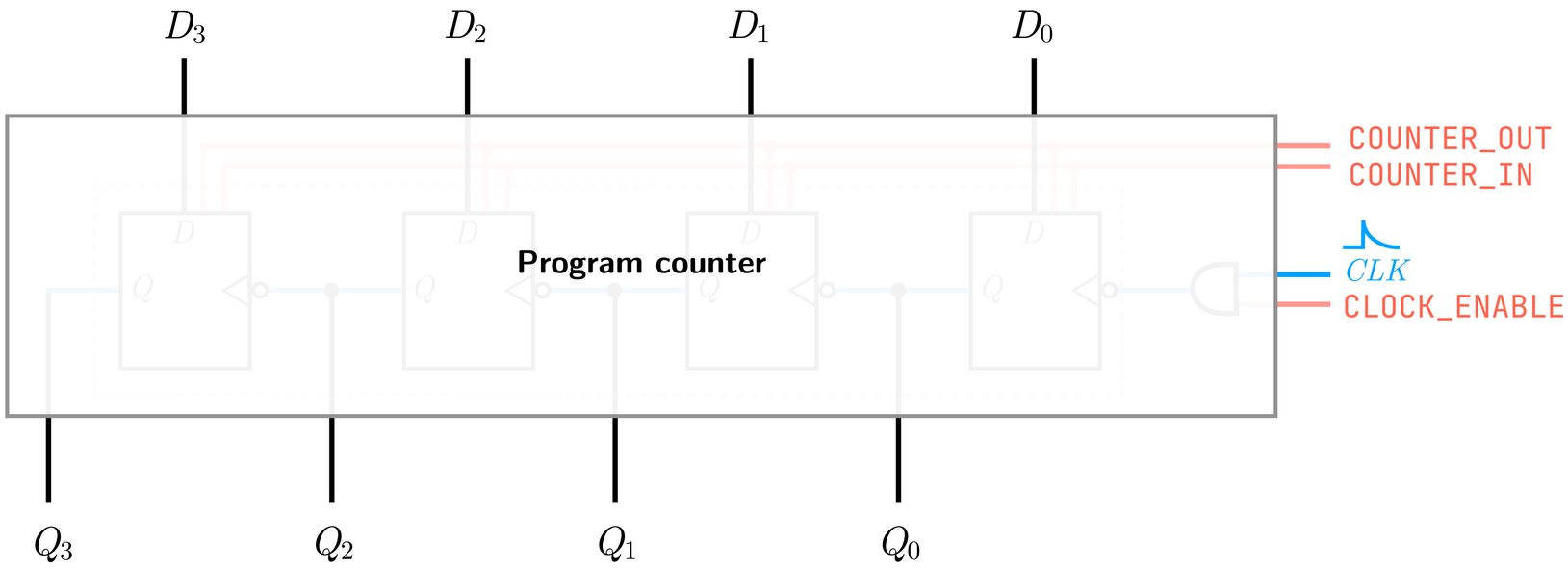


Program Counter (aka Instruction Pointer)

Same WRITE_ENABLE (COUNTER_OUT) as before using tri-state logic.

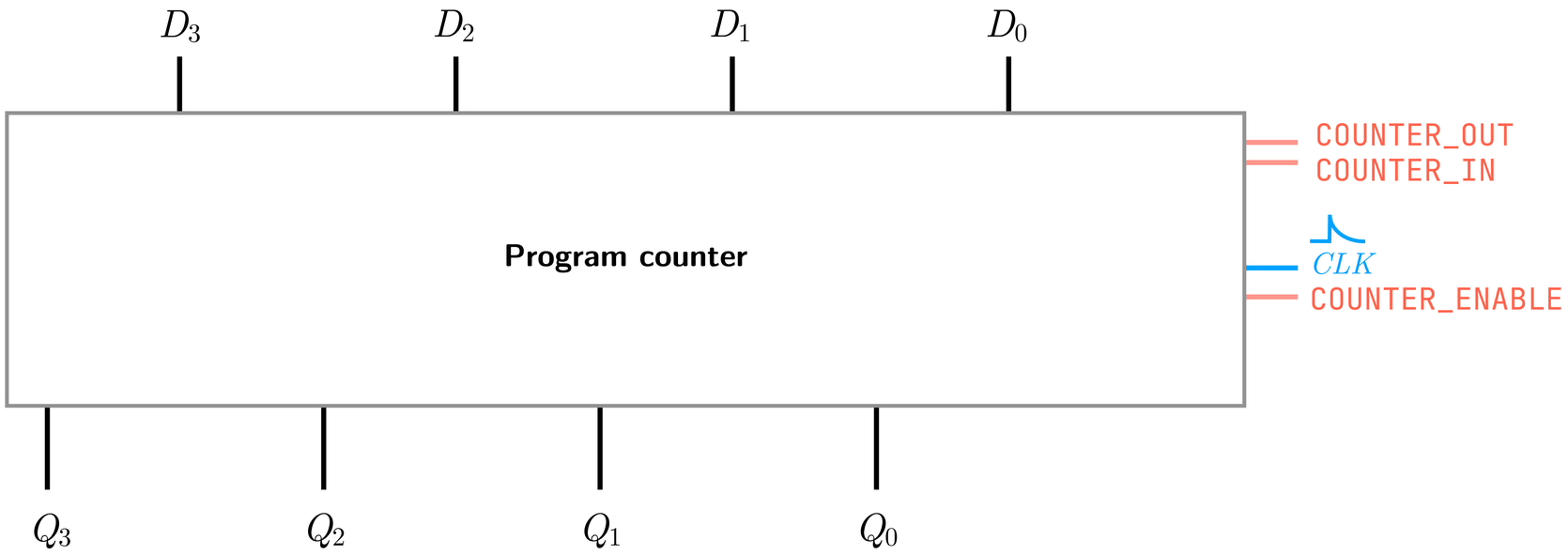


Program Counter (aka Instruction Pointer)



Program Counter (aka Instruction Pointer)

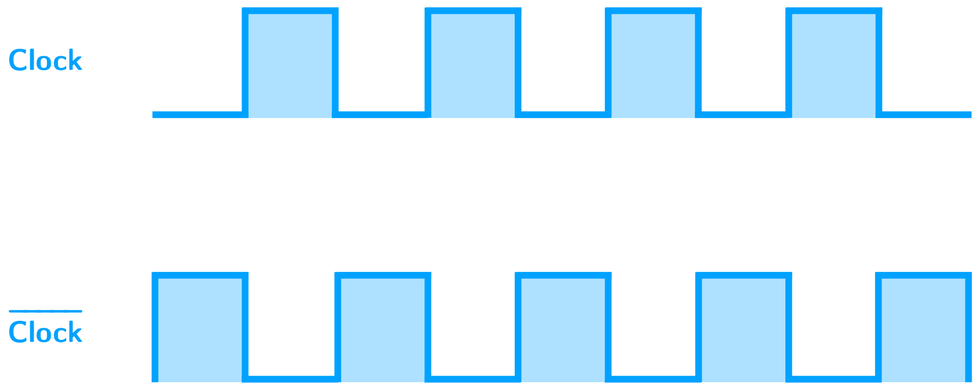
Slightly different variant
using COUNTER_ENABLE
rather then CLOCK_ENABLE.
COUNTER_IN, COUNTER_OUT
now also work independently of
CLOCK_ENABLE



Control logic

This is the coolest part
it's where everything comes together

Inverted clock!



Control logic

map `[opcode, step]` to `control_word`, where `control_word` is simply the string of all control signals and their value in that micro-step, i.e. all are 0 except for the control signals we want to execute, which are 1

ROM is used as “Instruction Decoder”

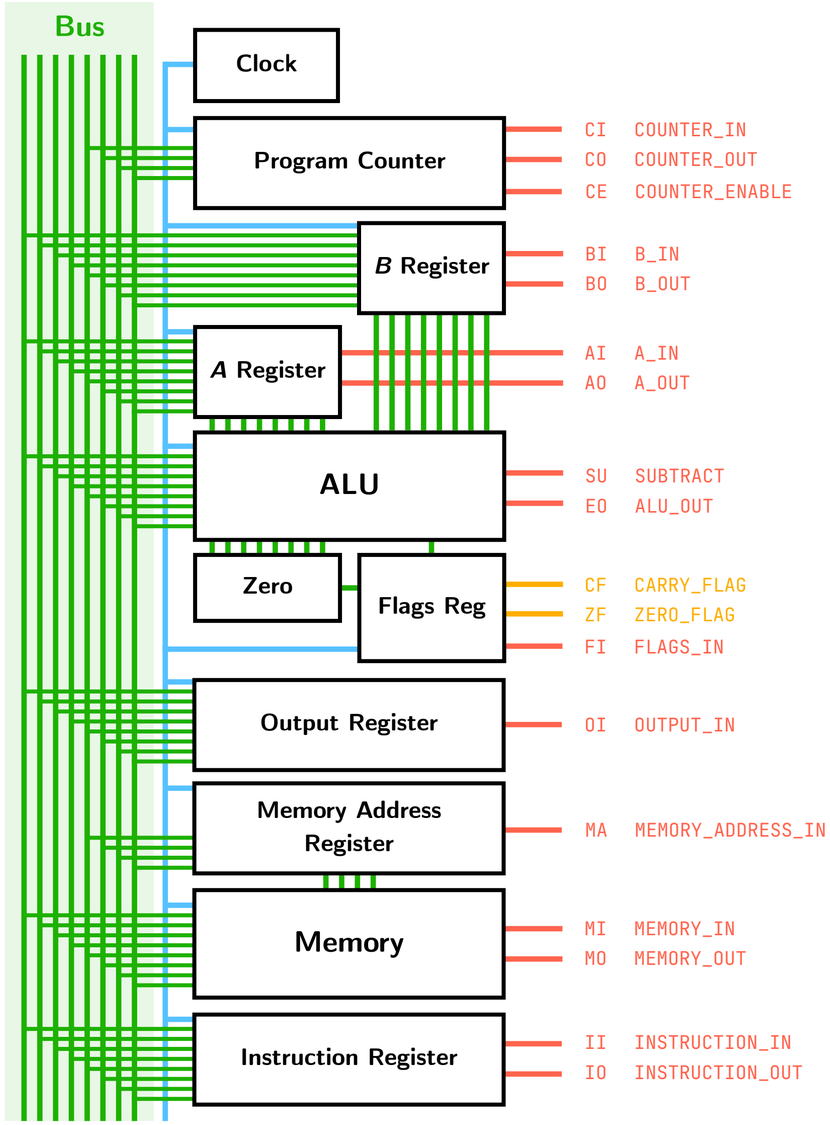
micro-step counter aka “sequencer”

Control logic

Coming up with instructions: we can use mnemonics and then come up with instruction codes

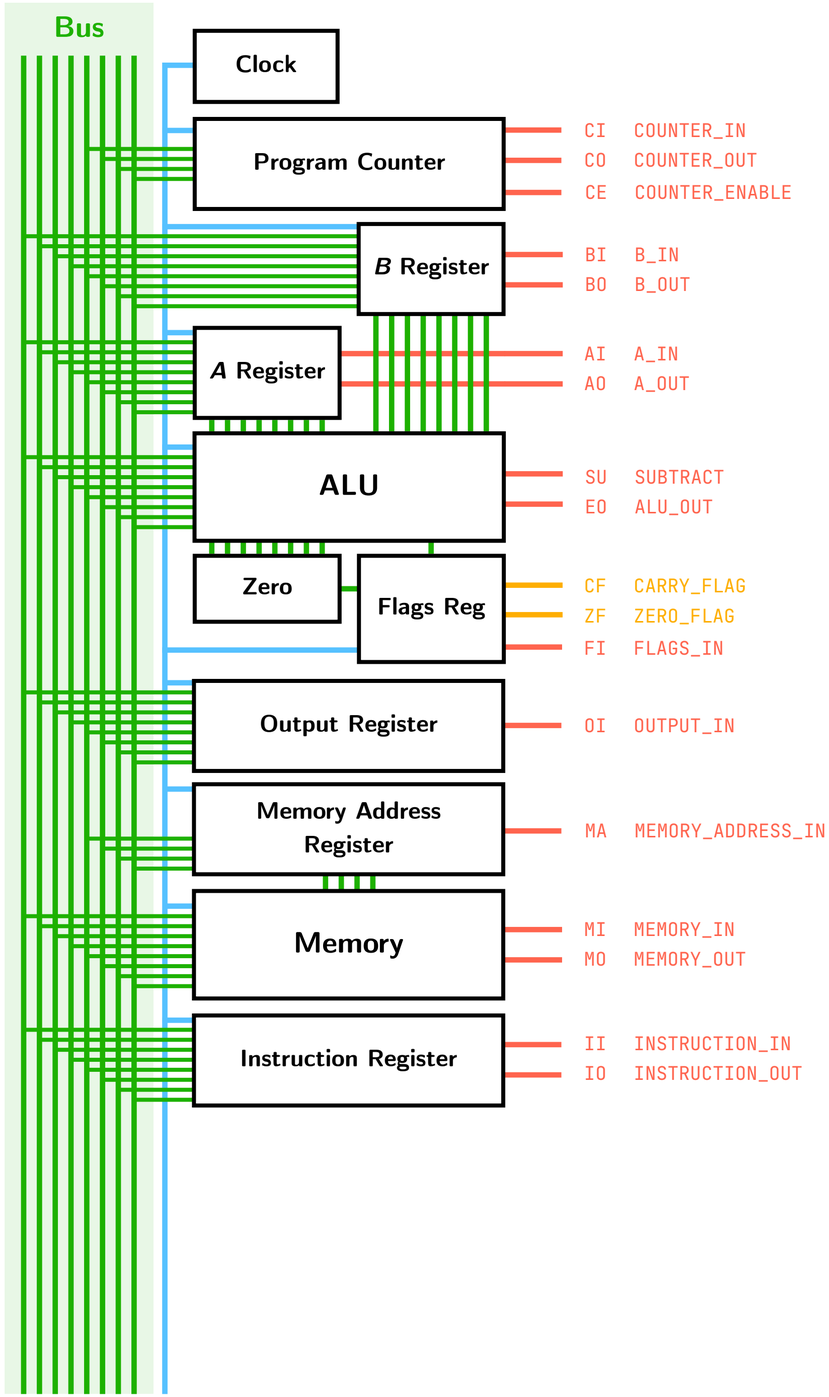
For now, imagine we activate the clock manually and in between clock highs
we set the control signals

Explain that there are steps
e.g. 6 steps, each takes one clock cycle
each instruction goes through all 6 steps
and so each instruction takes 6 clock cycles



Mnemonic	Opcode				Step			Control word																
								HALT	COUNTER_IN	COUNTER_OUT	COUNTER_ENABLE	B_IN	B_OUT	A_IN	A_OUT	SUBTRACT	ALU_OUT	FLAGS_IN	OUTPUT_IN	MEMORY_ADDRESS_IN	MEMORY_IN	MEMORY_OUT	INSTRUCTION_IN	INSTRUCTION_OUT
Fetch	*	*	*	*	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	*	*	*	*	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0
	*	*	*	*	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
LOAD	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADD	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
	0	0	1	0	1	0	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0
OUT	1	1	1	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

At this point, when explaining stuff,
there's no ROM or step counter yet !!!



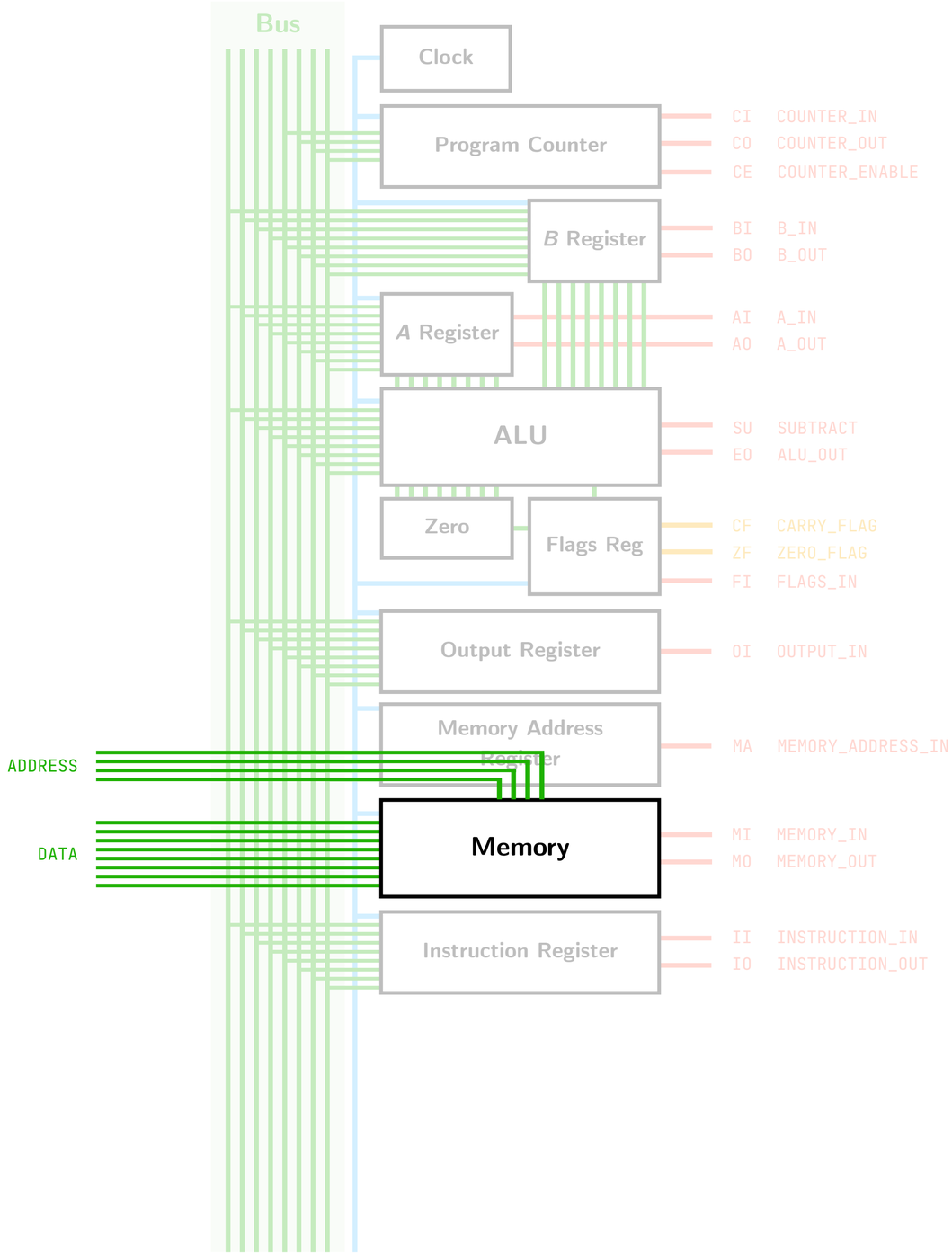
Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT


[14] 10
[15] 20
```

Not showing reset circuit

There's also connections like this
that we can use instead of the MAR
and bus in order to manually program
the computer (manually write something
to memory)
We can call this **"programming mode"**,
as opposed to the usual **"run mode"**

Manually write some data (8 bits)
at some memory address (4 bits)

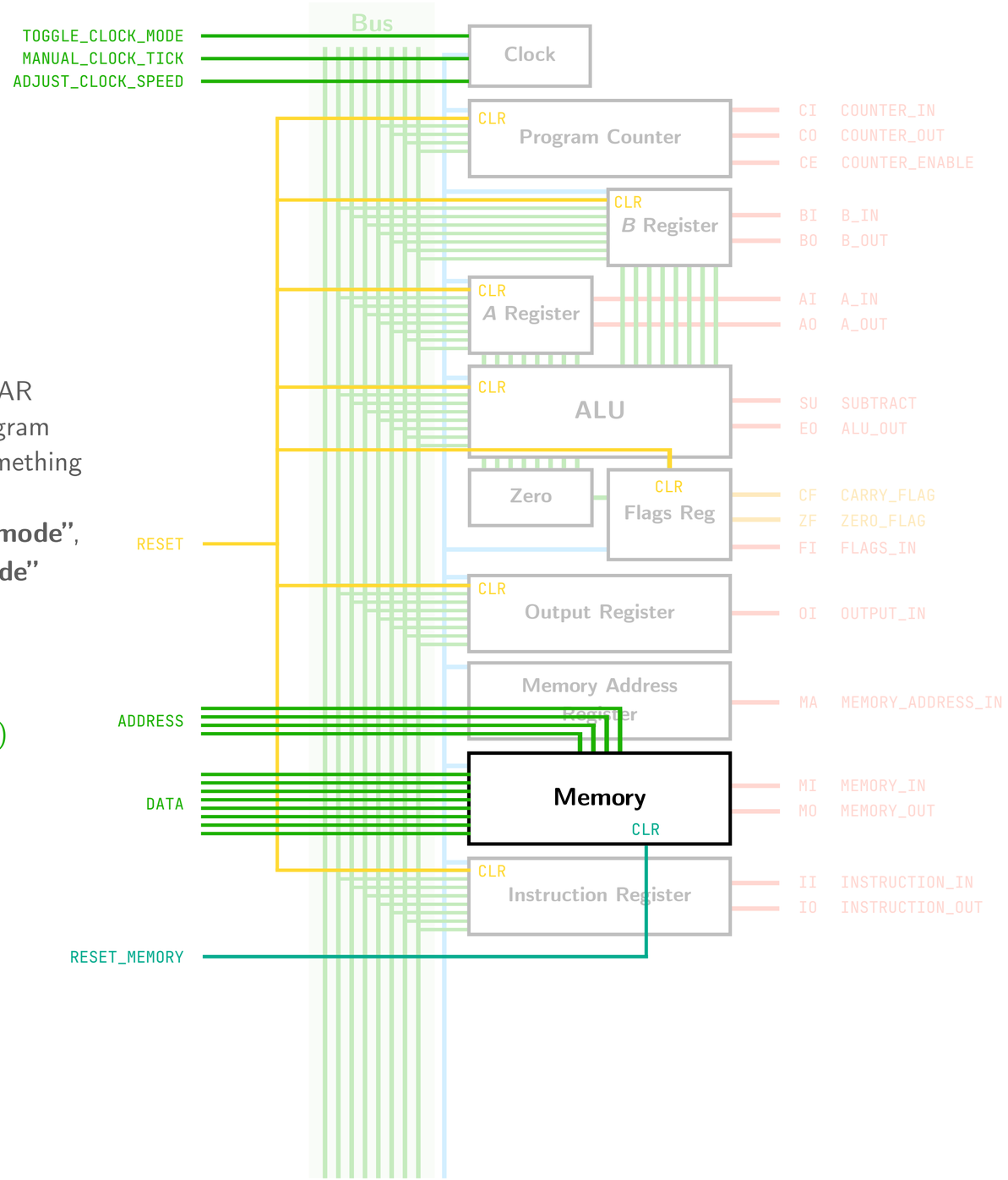


Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT



[14] 10
[15] 20
```

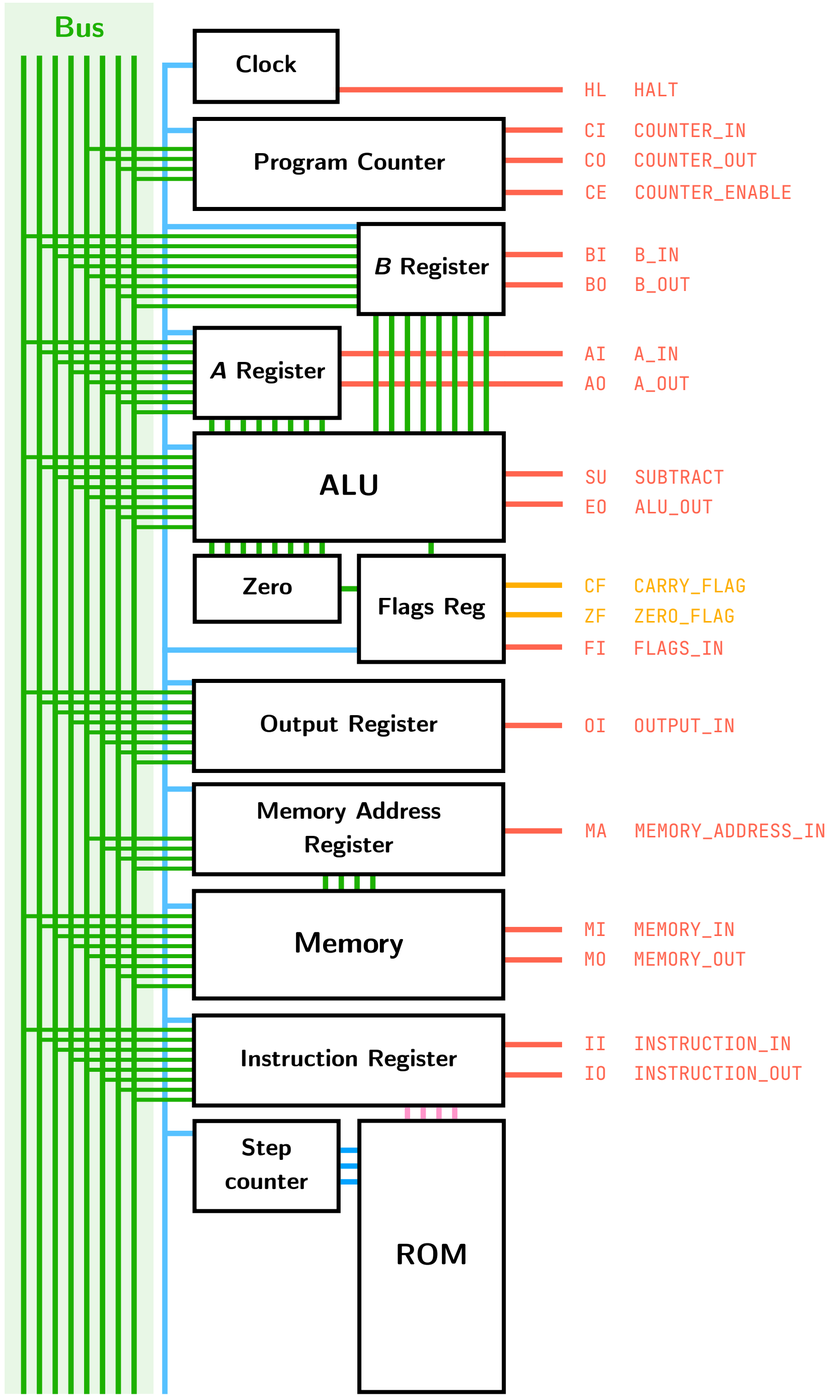
There's also connections like this that we can use instead of the MAR and bus in order to manually program the computer (manually write something to memory)
We can call this **"programming mode"**, as opposed to the usual **"run mode"**

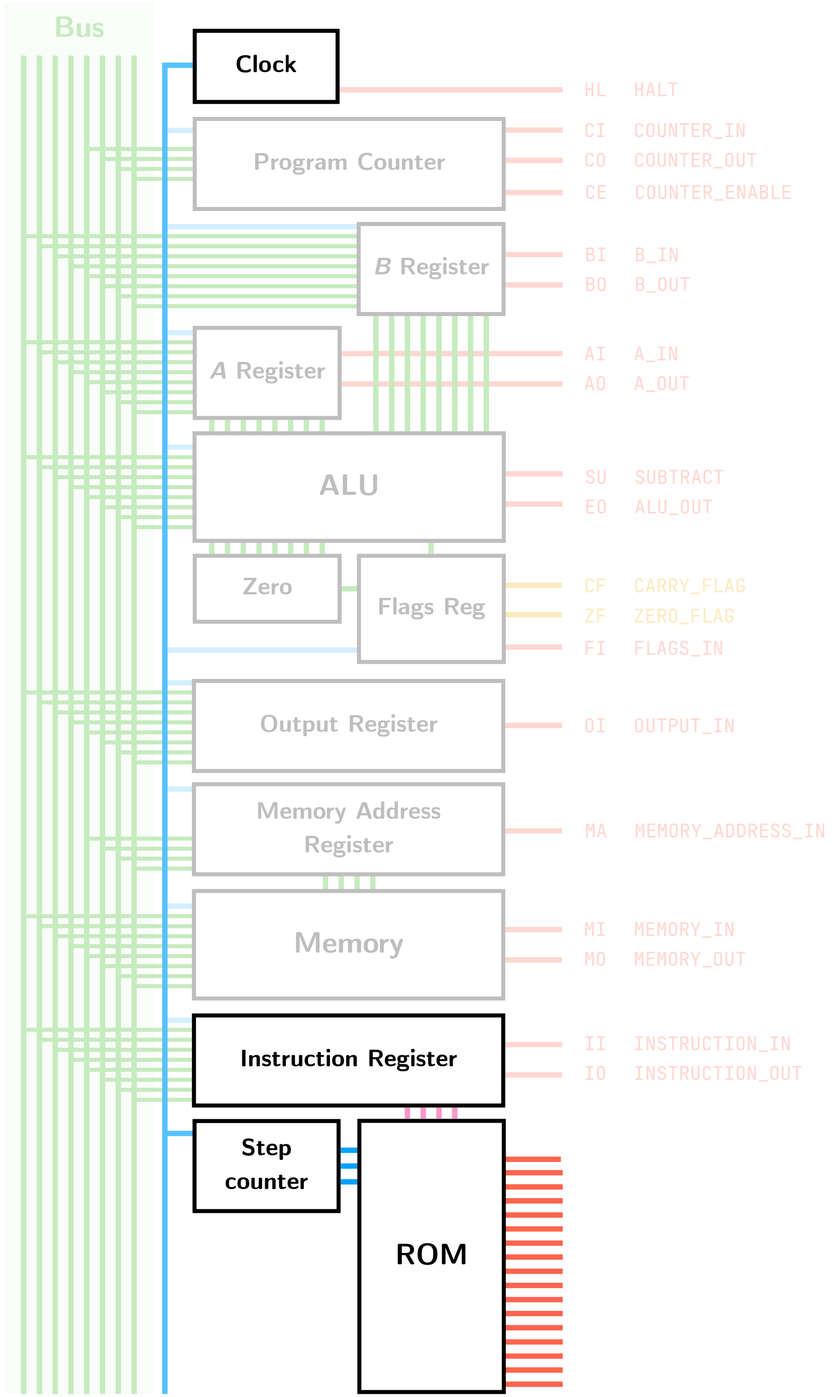
Manually write some data (8 bits) at some memory address (4 bits)

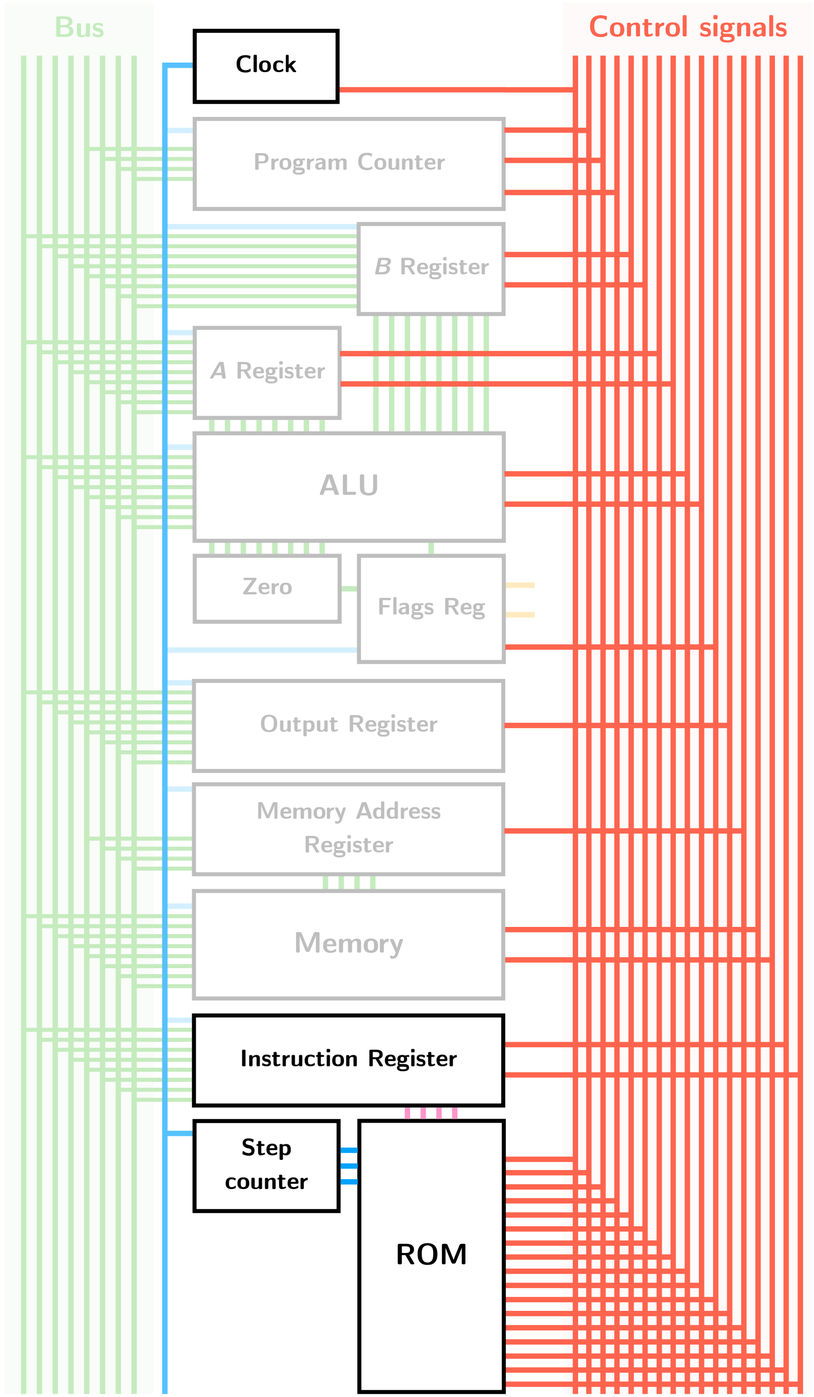


Program

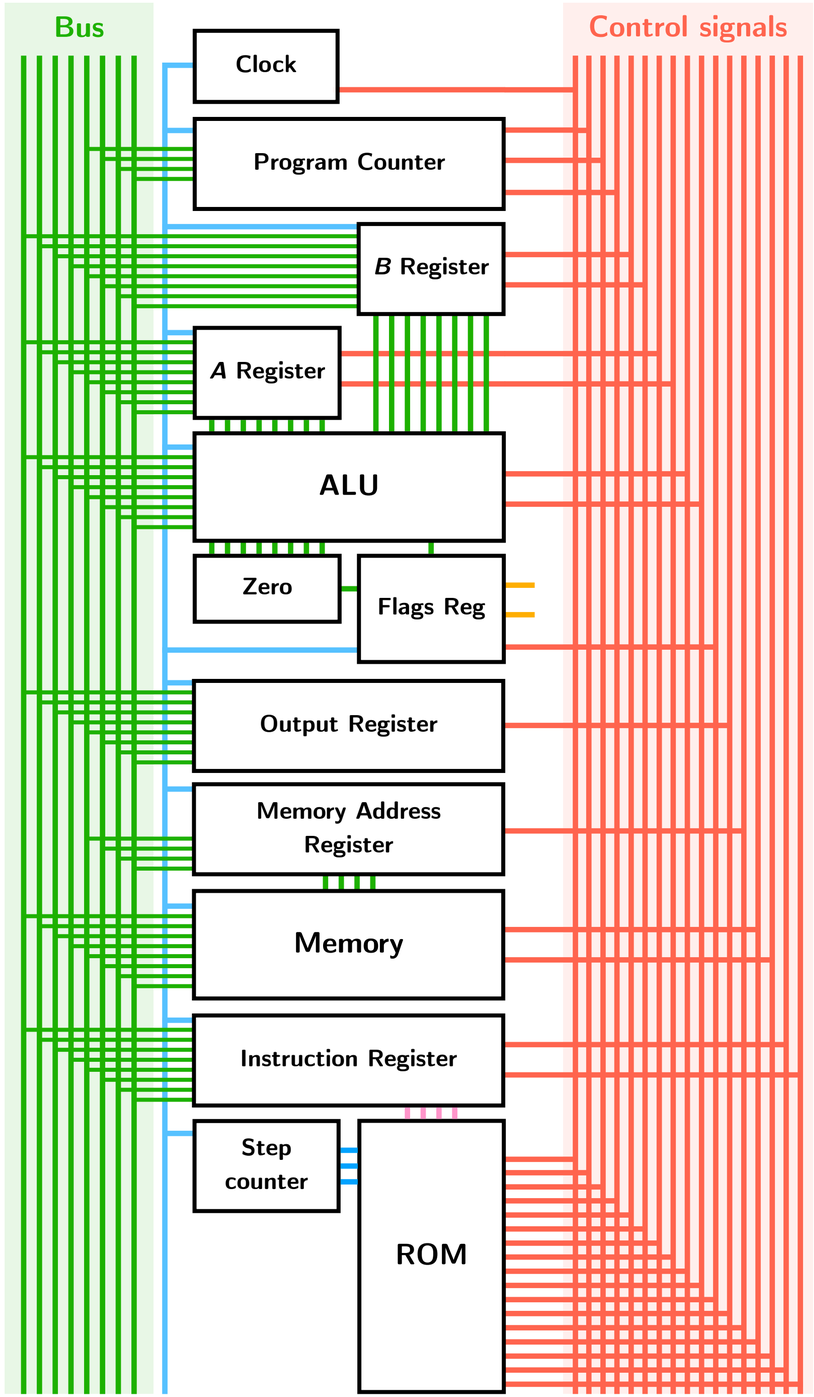
```
[0] LOAD  14
[1] ADD   15
[2] OUT


[14] 10
[15] 20
```


Bus
Clock
HL HALT
Program Counter
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
B Register
BI B_IN
BO B_OUT
A Register
AI A_IN
AO A_OUT
ALU
SU SUBTRACT
EO ALU_OUT
Zero
Flags Reg
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
Output Register
OI OUTPUT_IN
Memory Address Register
MA MEMORY_ADDRESS_IN
Memory
MI MEMORY_IN
MO MEMORY_OUT
Instruction Register
II INSTRUCTION_IN
IO INSTRUCTION_OUT
Step counter
ROM


Bus
Clock
HL HALT
Program Counter
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
B Register
BI B_IN
BO B_OUT
A Register
AI A_IN
AO A_OUT
ALU
SU SUBTRACT
EO ALU_OUT
Zero
Flags Reg
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
Output Register
OI OUTPUT_IN
Memory Address Register
MA MEMORY_ADDRESS_IN
Memory
MI MEMORY_IN
MO MEMORY_OUT
Instruction Register
II INSTRUCTION_IN
IO INSTRUCTION_OUT
Step counter
ROM

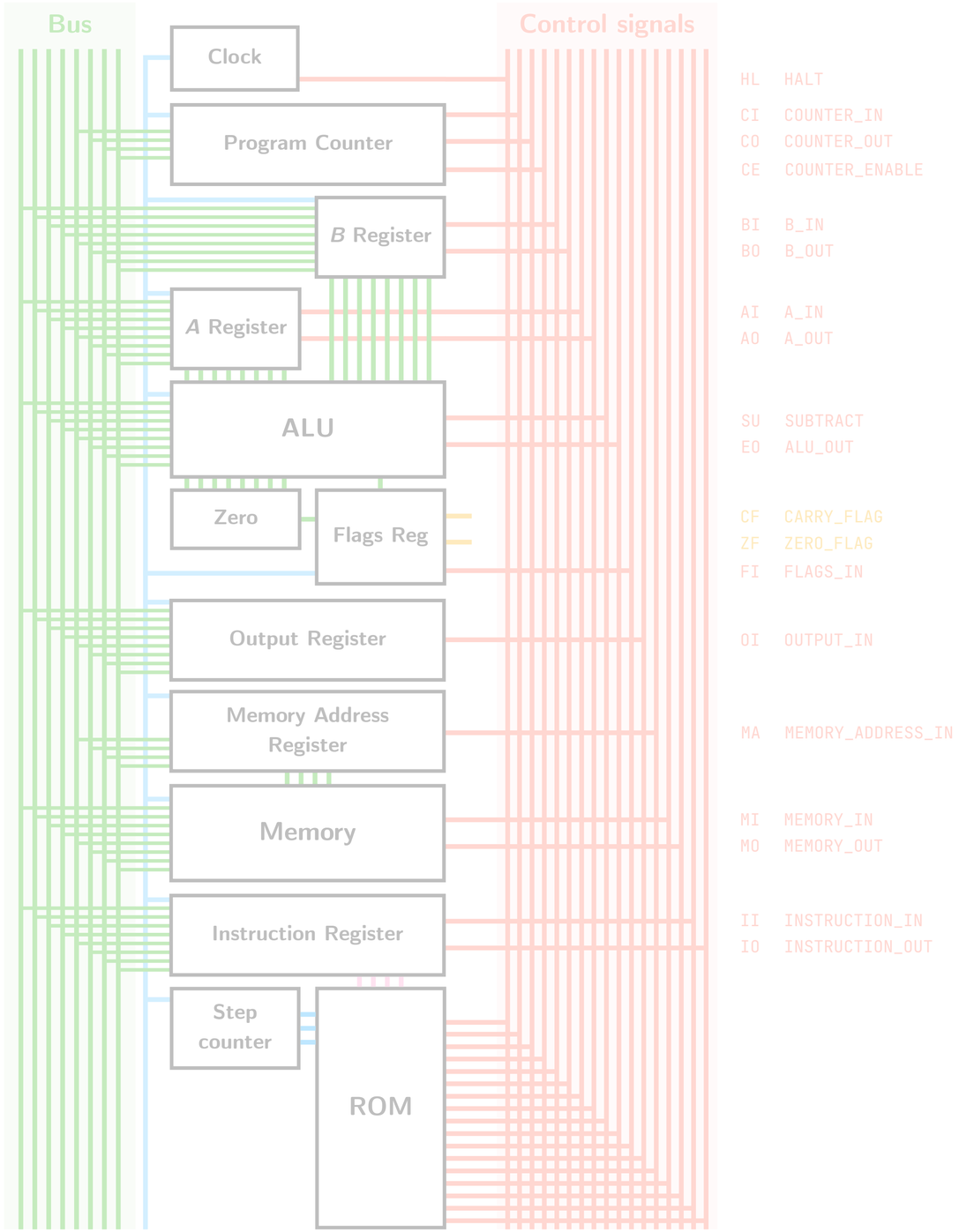


← We're not using these flags yet


Bus
Control signals
Clock
Program Counter
B Register
A Register
ALU
Zero
Flags Reg
Output Register
Memory Address Register
Memory
Instruction Register
Step counter
ROM
HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
BI B_IN
BO B_OUT
AI A_IN
AO A_OUT
SU SUBTRACT
EO ALU_OUT
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
← We're not using these flags yet
OI OUTPUT_IN
MA MEMORY_ADDRESS_IN
MI MEMORY_IN
MO MEMORY_OUT
II INSTRUCTION_IN
IO INSTRUCTION_OUT

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001**0**000
	****	001	0001000000000**11**0
	****	010	000000000000000**1**
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
LOAD	0001	011	000000000000**1**0000
LOAD	0001	100	000000**1**0000000**1**00
LOAD	0001	101	0000000000000000
ADD	0010	011	000000000000**1**0000
ADD	0010	100	0000**1**000000000**1**00
ADD	0010	101	000000**1**00**11**000000
OUT	1110	011	0000000**1**000**1**00000
OUT	1110	100	0000000000000000
OUT	1110	101	0000000000000000
HALT	1111	011	**1**000000000000000
HALT	1111	100	0000000000000000
HALT	1111	101	0000000000000000

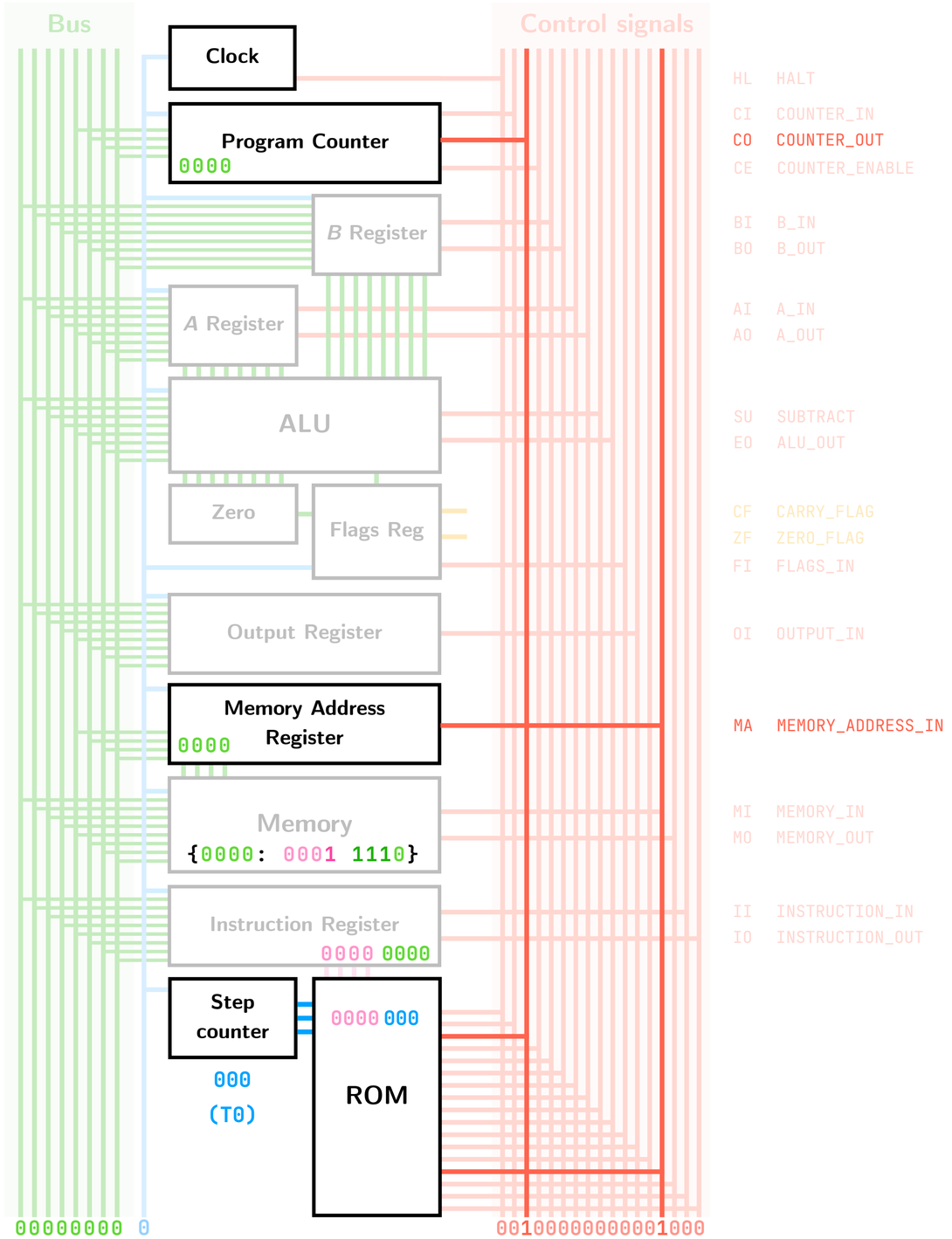


Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT



[14] 10
[15] 20
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001**1**0000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001**0**000
	0001	100	000000**1**0000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000001**0**000
	0010	100	0000**1**000000001**0**0
	0010	101	000000**1**00**1**1000000
OUT	1110	011	0000000**1**000**1**00000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	**1**000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

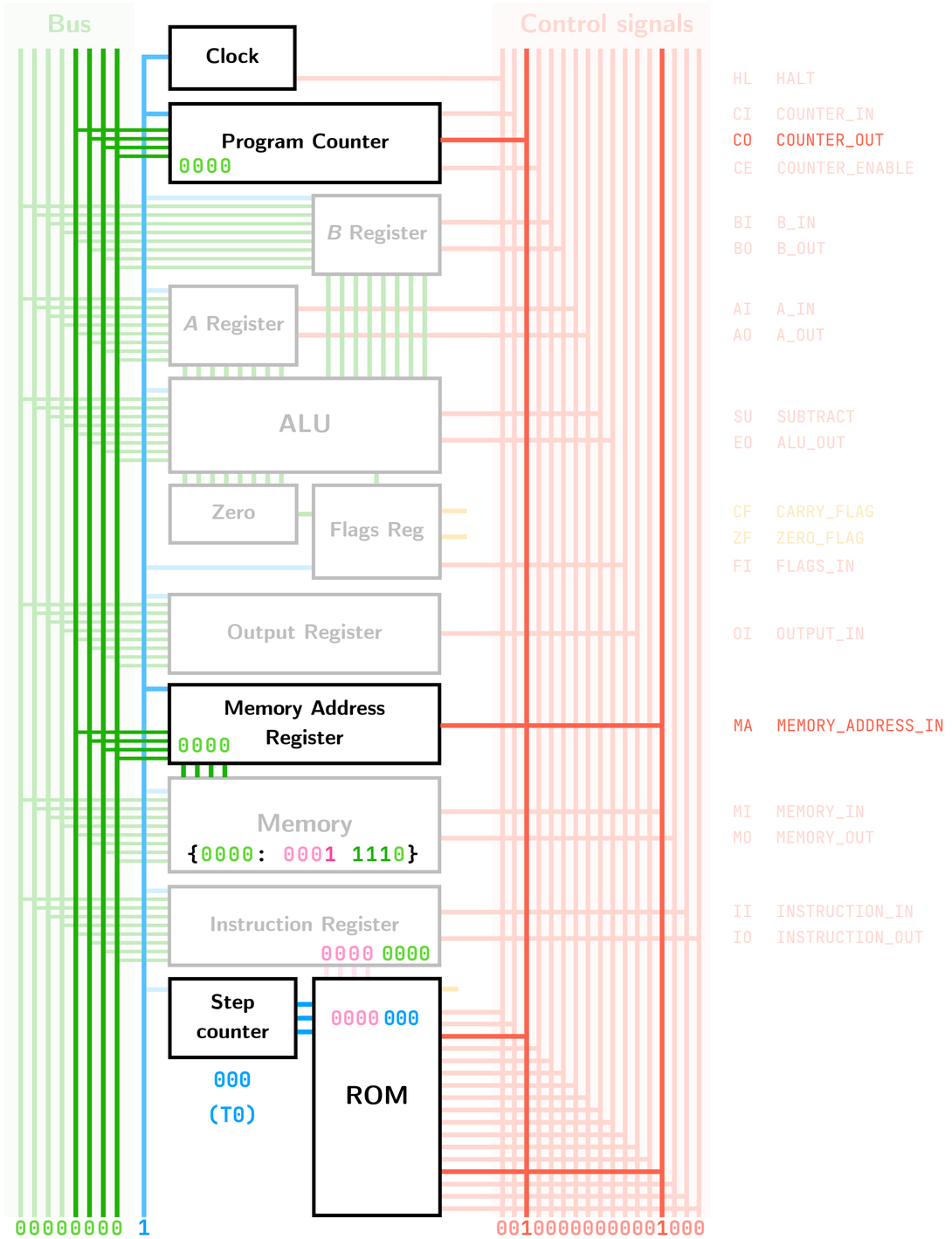
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

LOAD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001**1**0000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
LOAD	0001	011	0000000000001**0**0000
LOAD	0001	100	000000**1**0000000100
LOAD	0001	101	0000000000000000
ADD	0010	011	0000000000001**0**0000
ADD	0010	100	0000**1**000000001**0**0
ADD	0010	101	000000**1**00**1**1000000
OUT	1110	011	0000000**1**000**1**00000
OUT	1110	100	0000000000000000
OUT	1110	101	0000000000000000
HALT	1111	011	**1**000000000000000
HALT	1111	100	0000000000000000
HALT	1111	101	0000000000000000



Program

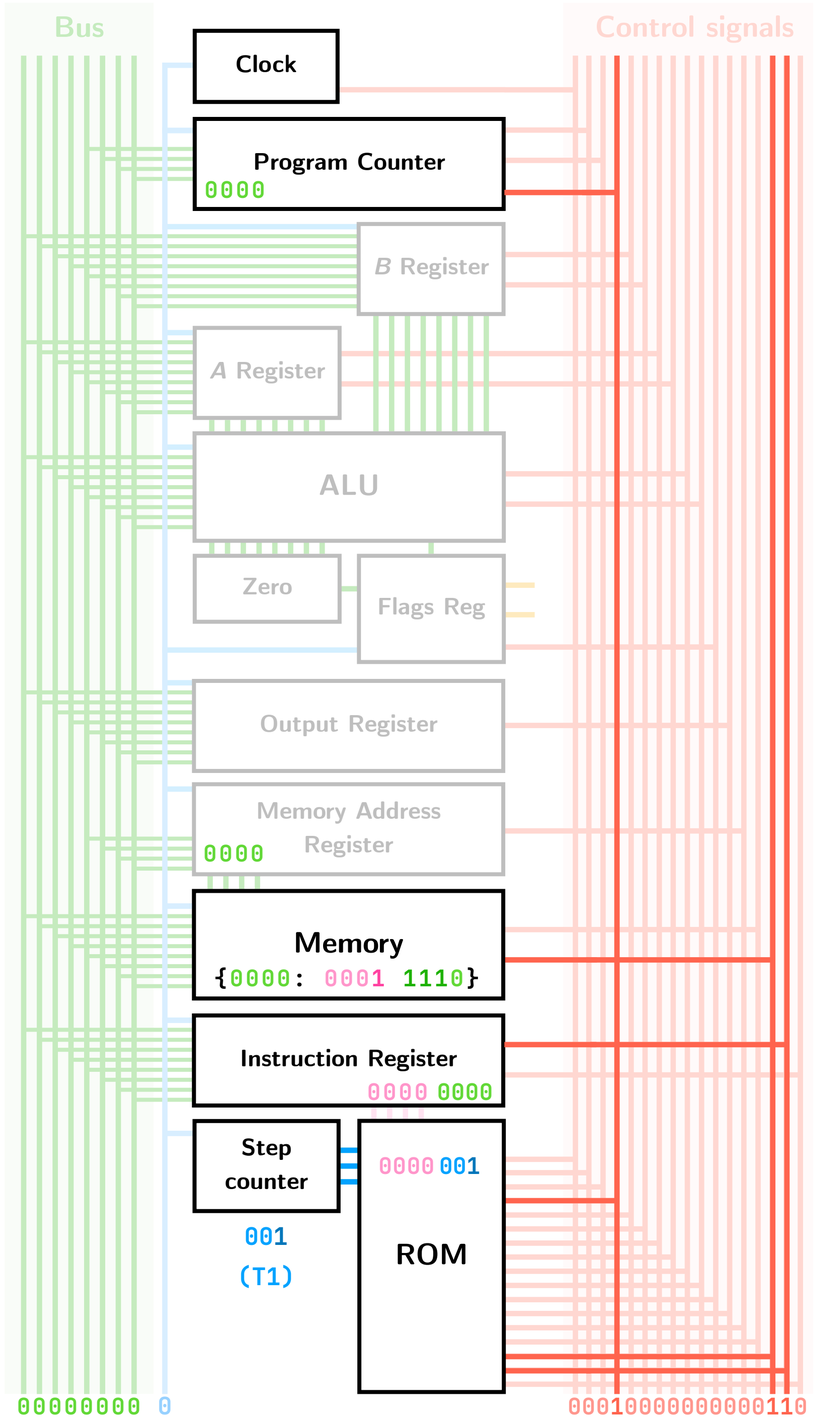
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

LOAD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000001000100000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
BI B_IN
BO B_OUT
AI A_IN
AO A_OUT
SU SUBTRACT
EO ALU_OUT
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
OI OUTPUT_IN
MA MEMORY_ADDRESS_IN
MI MEMORY_IN
MO MEMORY_OUT
II INSTRUCTION_IN
IO INSTRUCTION_OUT

Program

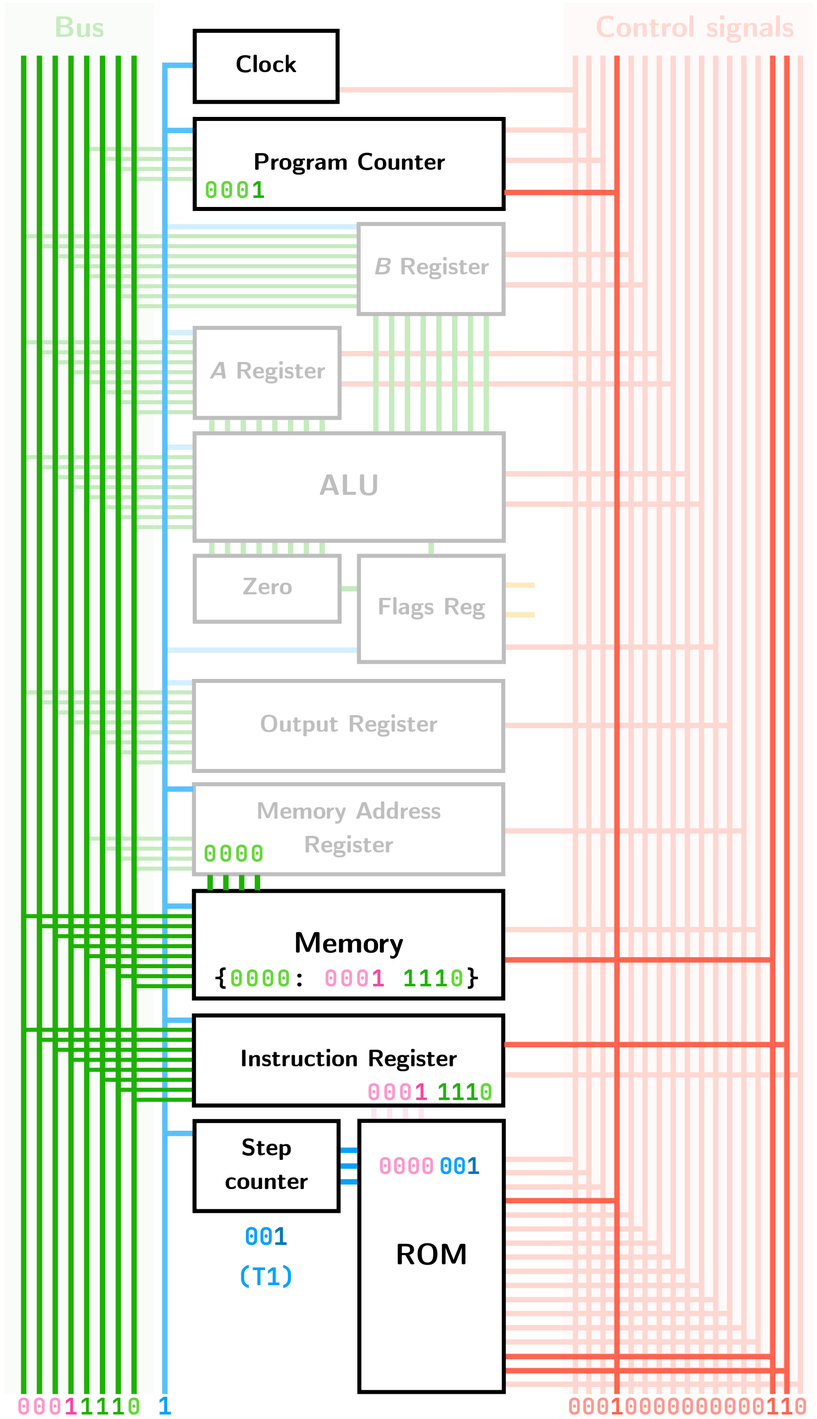
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

LOAD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000001000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

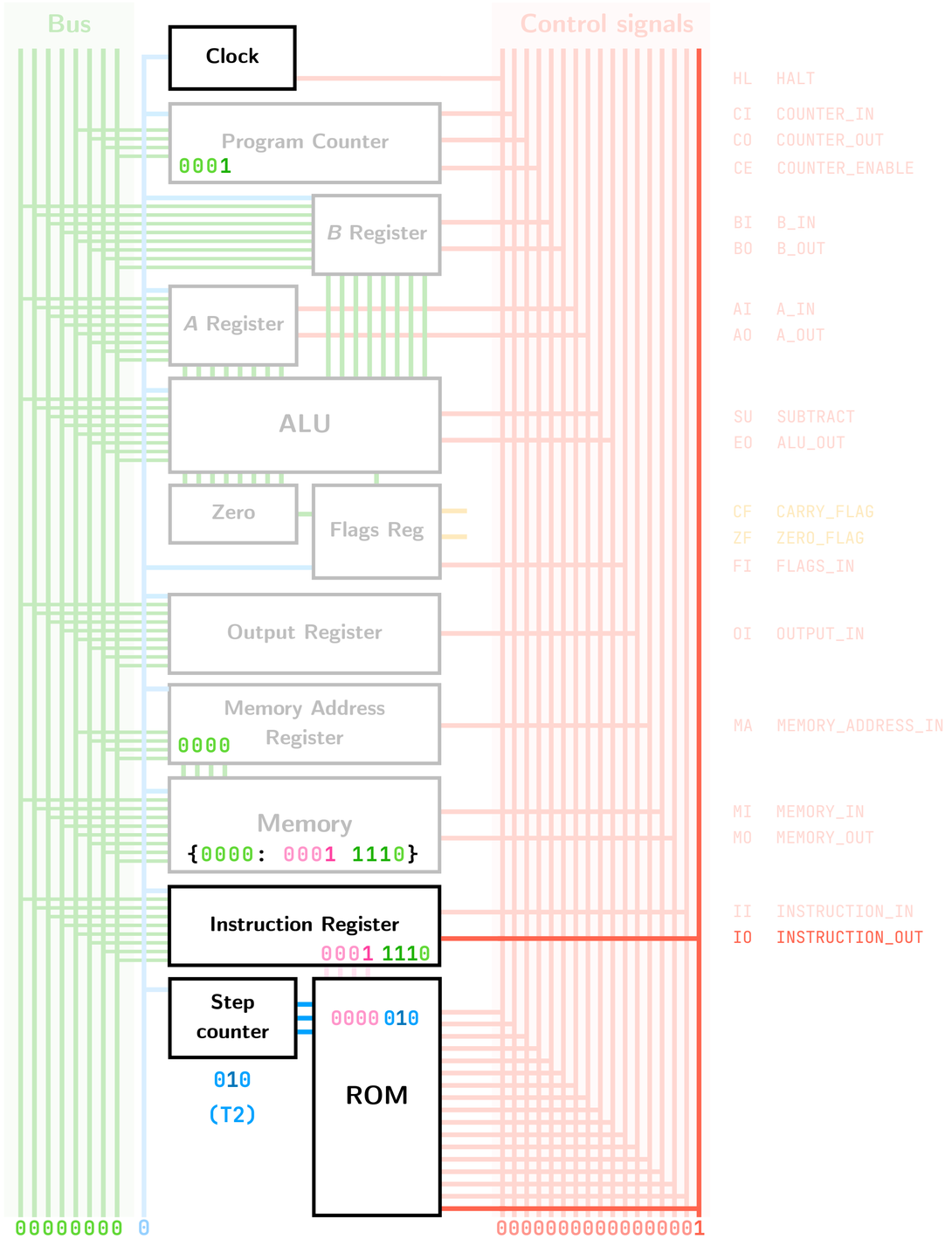
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

LOAD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMIILIOEIOIOUOIIAIOIO
	****	000	0010000000001 0000
	****	001	0001000000000 110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001 0000
	0001	100	000000100000000 100
	0001	101	0000000000000000
ADD	0010	011	0000000000001 0000
	0010	100	0000100000000 100
	0010	101	000000100110000 0
OUT	1110	011	0000000100010 0000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

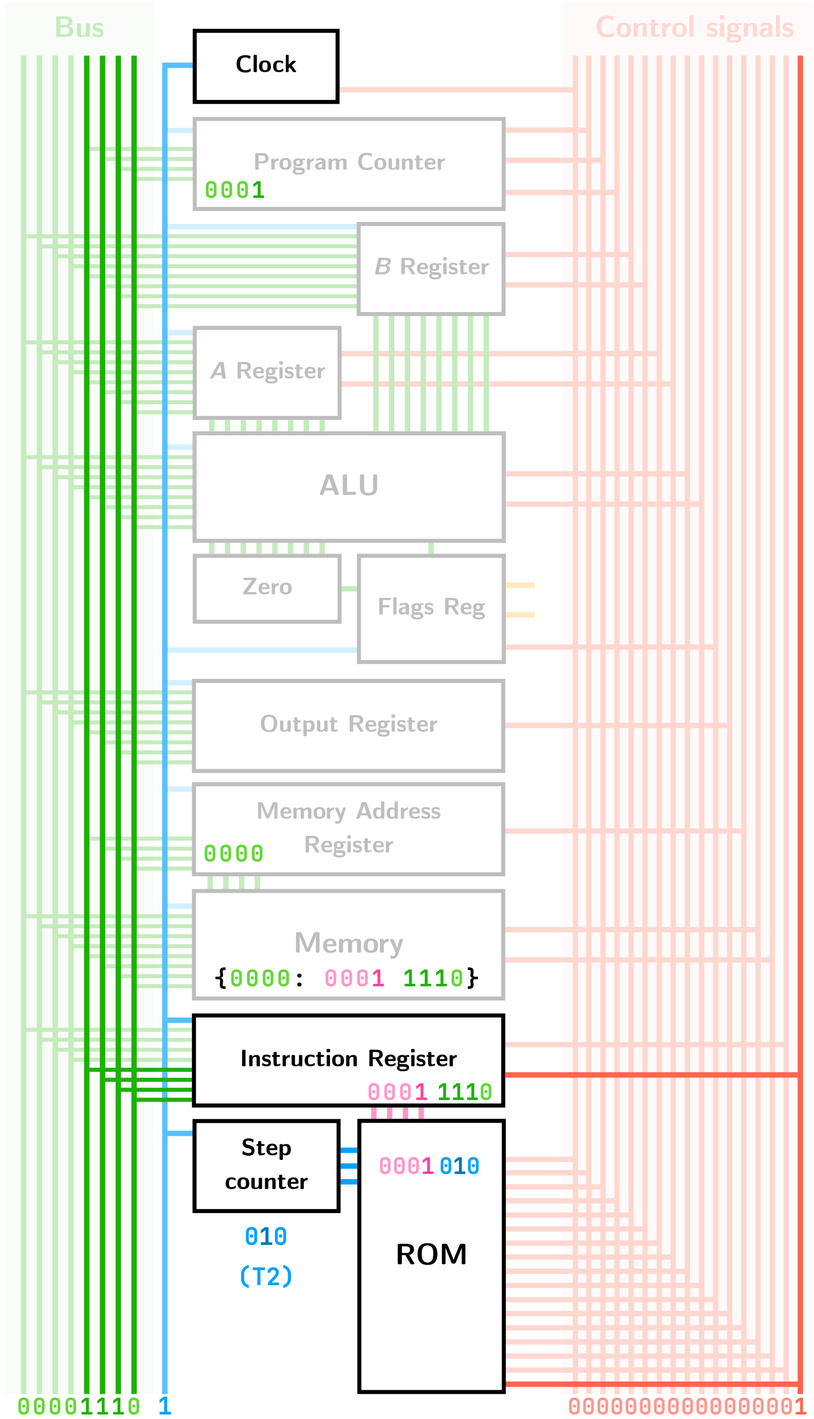
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

LOAD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000001000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

LOAD

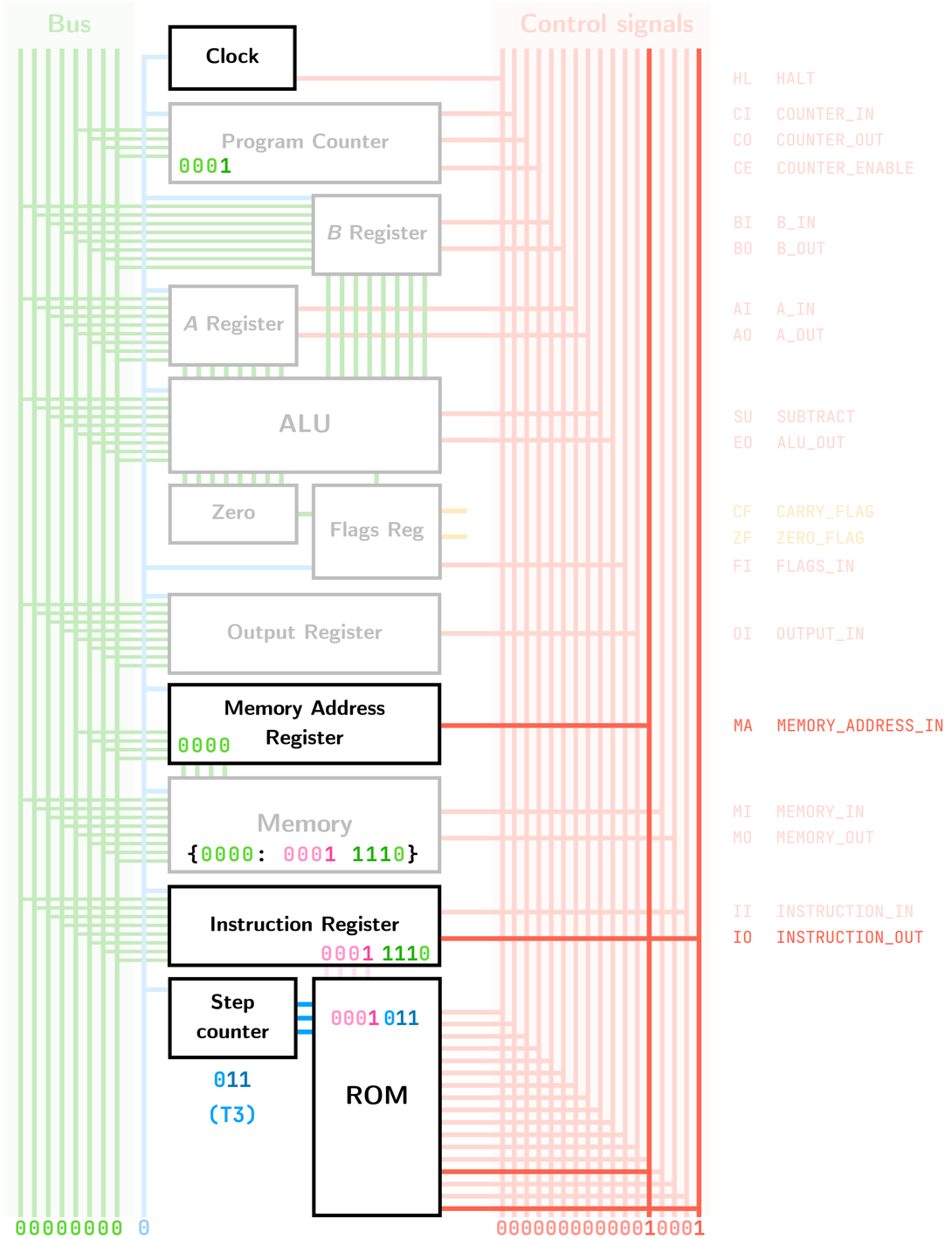
```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

← ROM is continuously addressed
Address changes mid-cycle
But it's the same control word

IR can only write last 4 bits to bus →

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000001000100000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

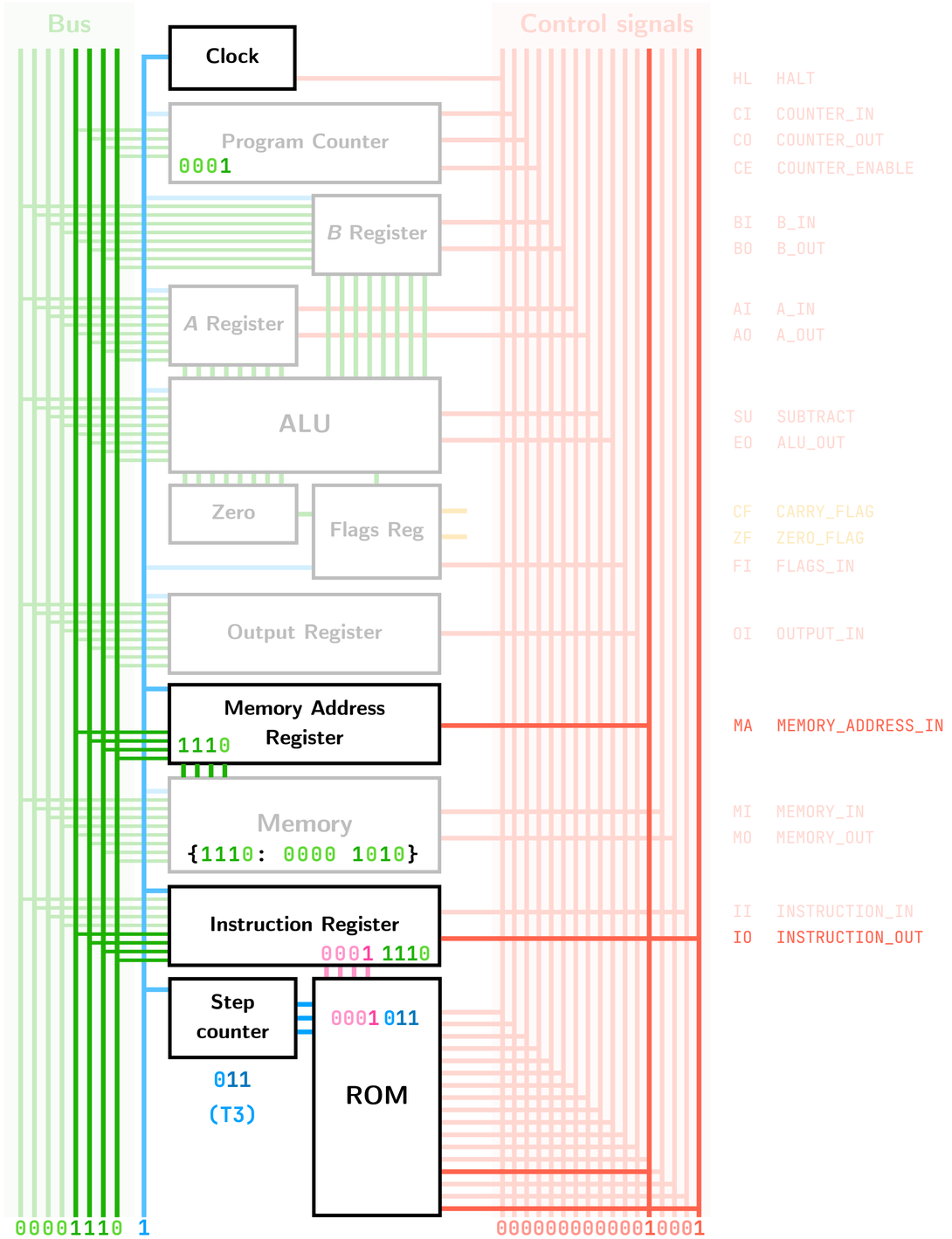
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

LOAD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	00100000000010000
	****	001	00010000000000110
	****	010	00000000000000001
NOP	0000	011	00000000000000000
NOP	0000	100	00000000000000000
NOP	0000	101	00000000000000000
LOAD	0001	011	00000000000010000
LOAD	0001	100	00000100000000100
LOAD	0001	101	00000000000000000
ADD	0010	011	00000000000010000
ADD	0010	100	00001000000000100
ADD	0010	101	00000010011000000
OUT	1110	011	00000001000100000
OUT	1110	100	00000000000000000
OUT	1110	101	00000000000000000
HALT	1111	011	10000000000000000
HALT	1111	100	00000000000000000
HALT	1111	101	00000000000000000



Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

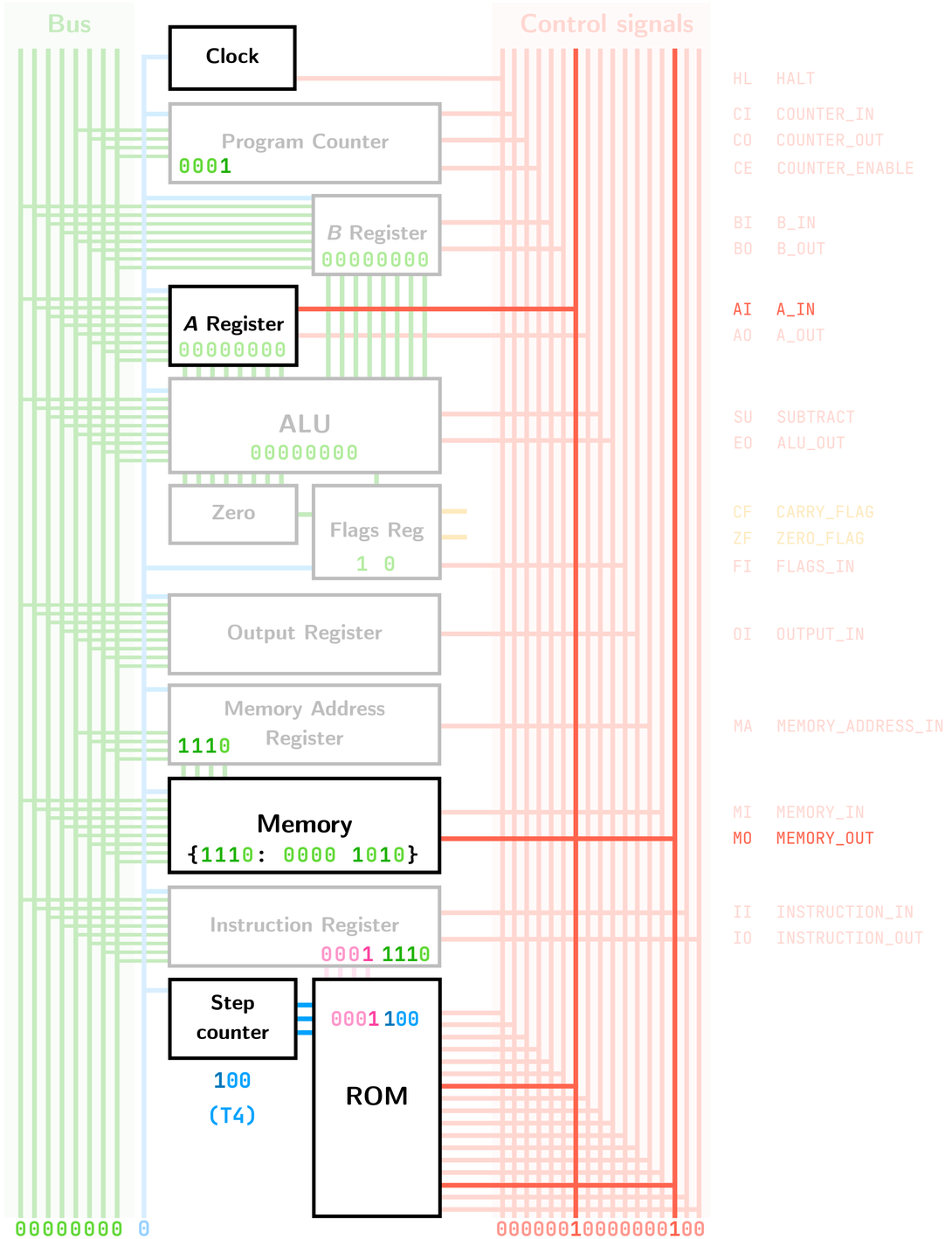
LOAD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

IR can only write last 4 bits to bus →

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
	0001	011	0000000000001000
LOAD	0001	100	0000010000000100
	0001	101	0000000000000000
	0010	011	0000000000001000
ADD	0010	100	0000100000000100
	0010	101	0000010011000000
	1110	011	0000000100010000
OUT	1110	100	0000000000000000
	1110	101	0000000000000000
	1111	011	1000000000000000
HALT	1111	100	0000000000000000
	1111	101	0000000000000000



Program

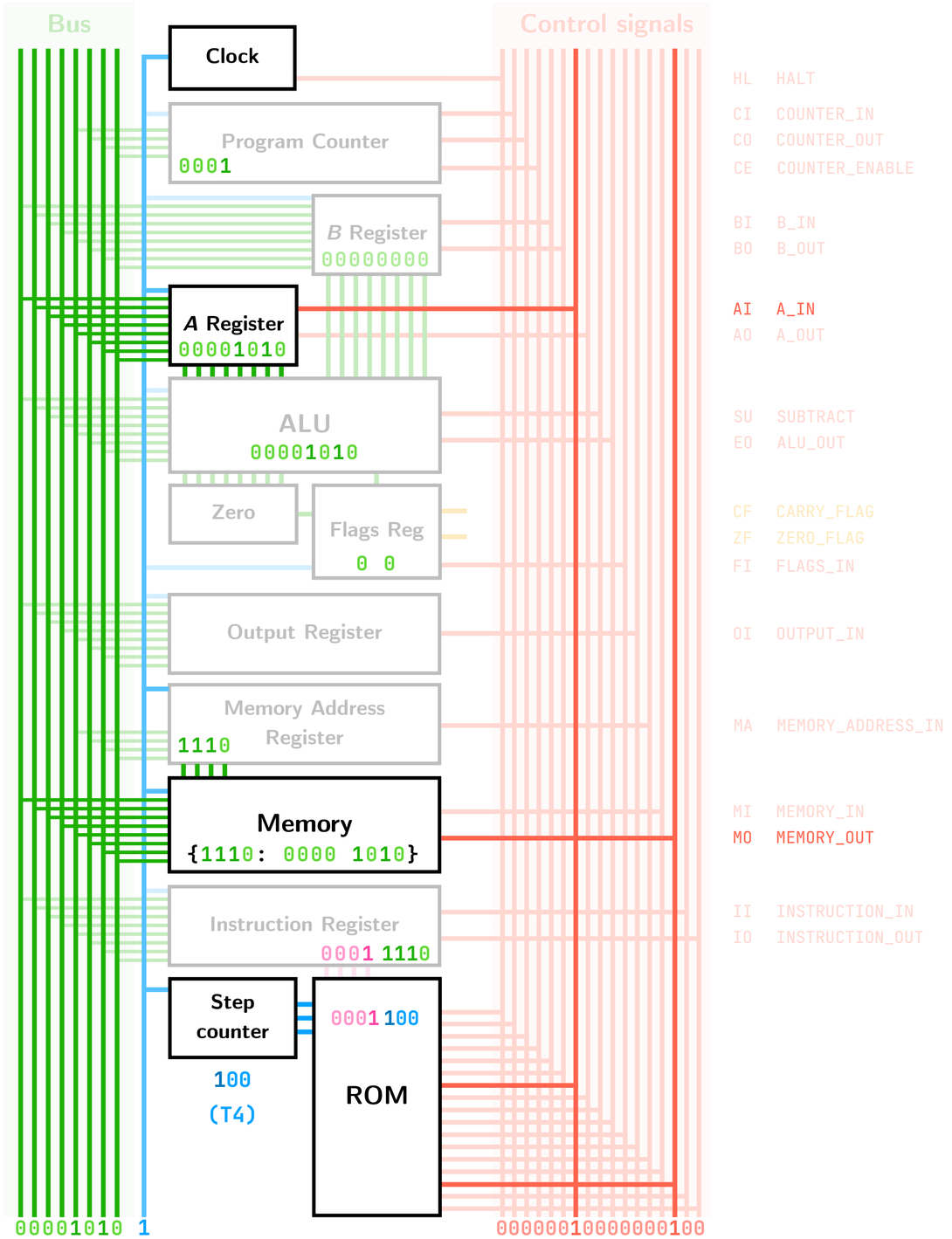
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

LOAD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
	0001	011	0000000000010000
LOAD	0001	100	0000010000000100
	0001	101	0000000000000000
	0010	011	0000000000010000
ADD	0010	100	0000100000000100
	0010	101	0000010011000000
	1110	011	0000001000100000
OUT	1110	100	0000000000000000
	1110	101	0000000000000000
	1111	011	1000000000000000
HALT	1111	100	0000000000000000
	1111	101	0000000000000000



Program

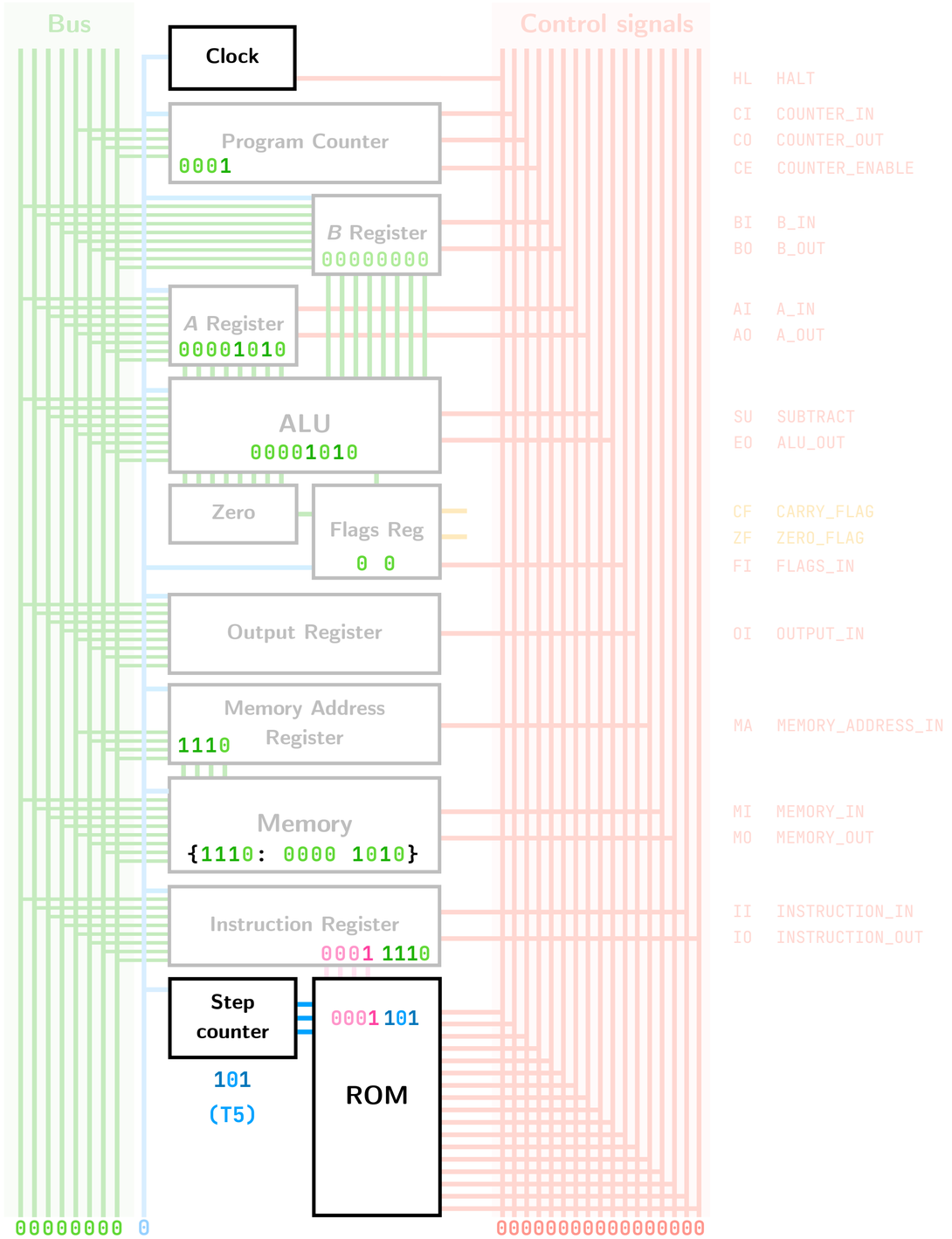
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

LOAD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000001000100000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

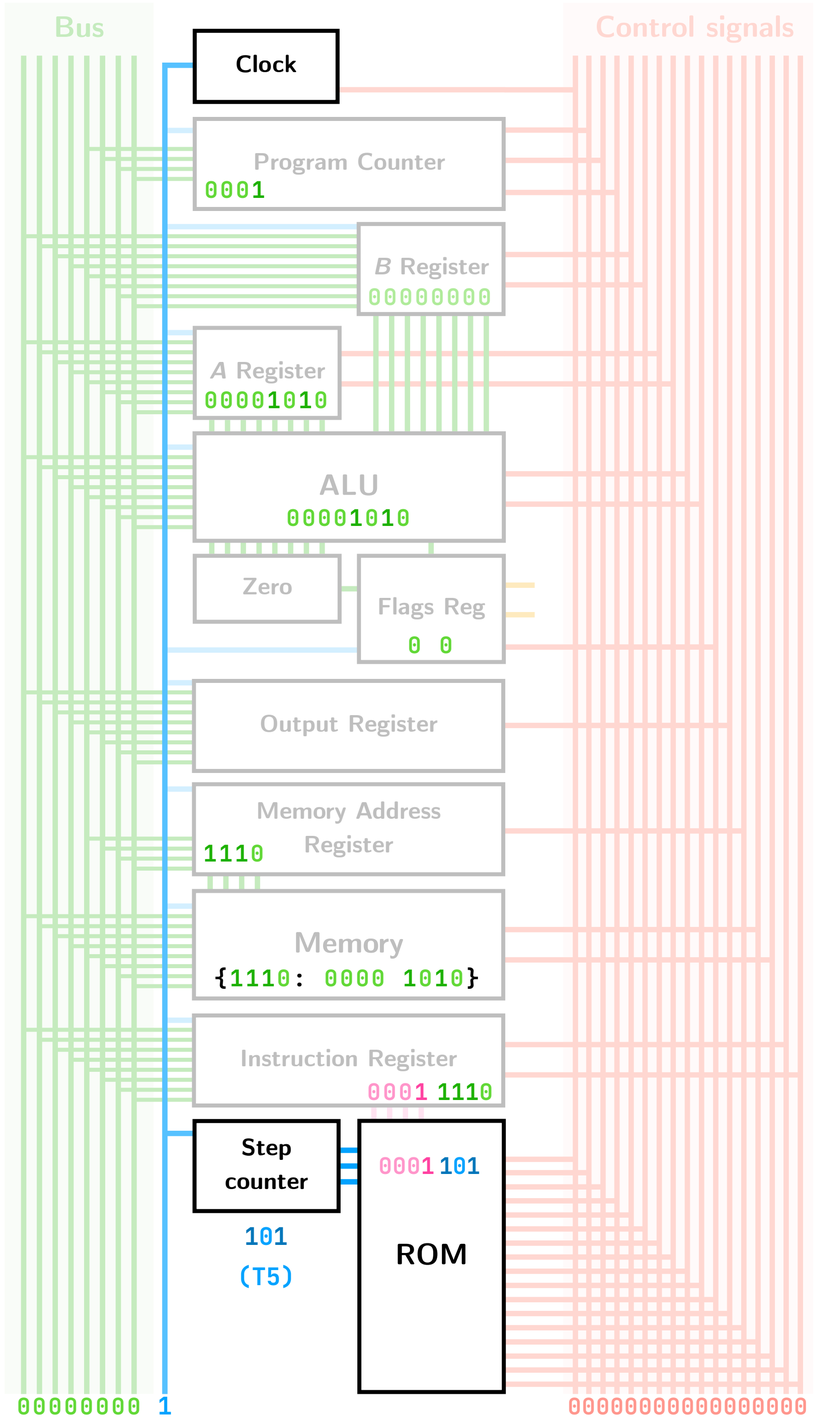
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

LOAD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
LOAD	0001	100	0000010000000100
LOAD	0001	101	0000000000000000
ADD	0010	011	0000000000010000
ADD	0010	100	0000100000000100
ADD	0010	101	0000010011000000
OUT	1110	011	0000001000100000
OUT	1110	100	0000000000000000
OUT	1110	101	0000000000000000
HALT	1111	011	1000000000000000
HALT	1111	100	0000000000000000
HALT	1111	101	0000000000000000



Program

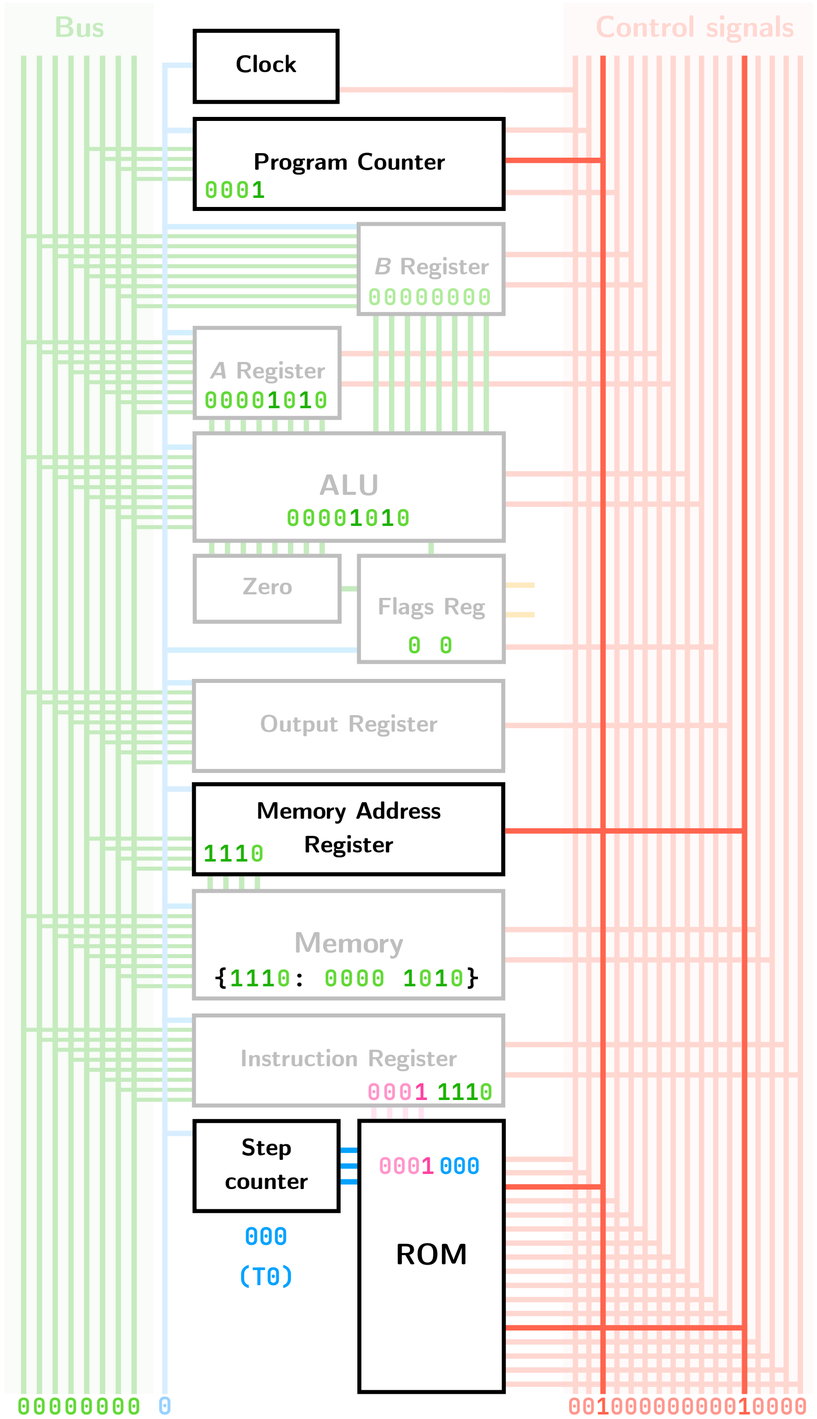
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

LOAD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    A_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001**0**0000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
LOAD	0001	011	0000000000001**0**000
LOAD	0001	100	000000**1**0000000100
LOAD	0001	101	0000000000000000
ADD	0010	011	0000000000001**0**000
ADD	0010	100	0000**1**000000001**0**0
ADD	0010	101	000000**1**00**1**1000000
OUT	1110	011	0000000**1**000**1**00000
OUT	1110	100	0000000000000000
OUT	1110	101	0000000000000000
HALT	1111	011	**1**000000000000000
HALT	1111	100	0000000000000000
HALT	1111	101	0000000000000000



Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

ADD

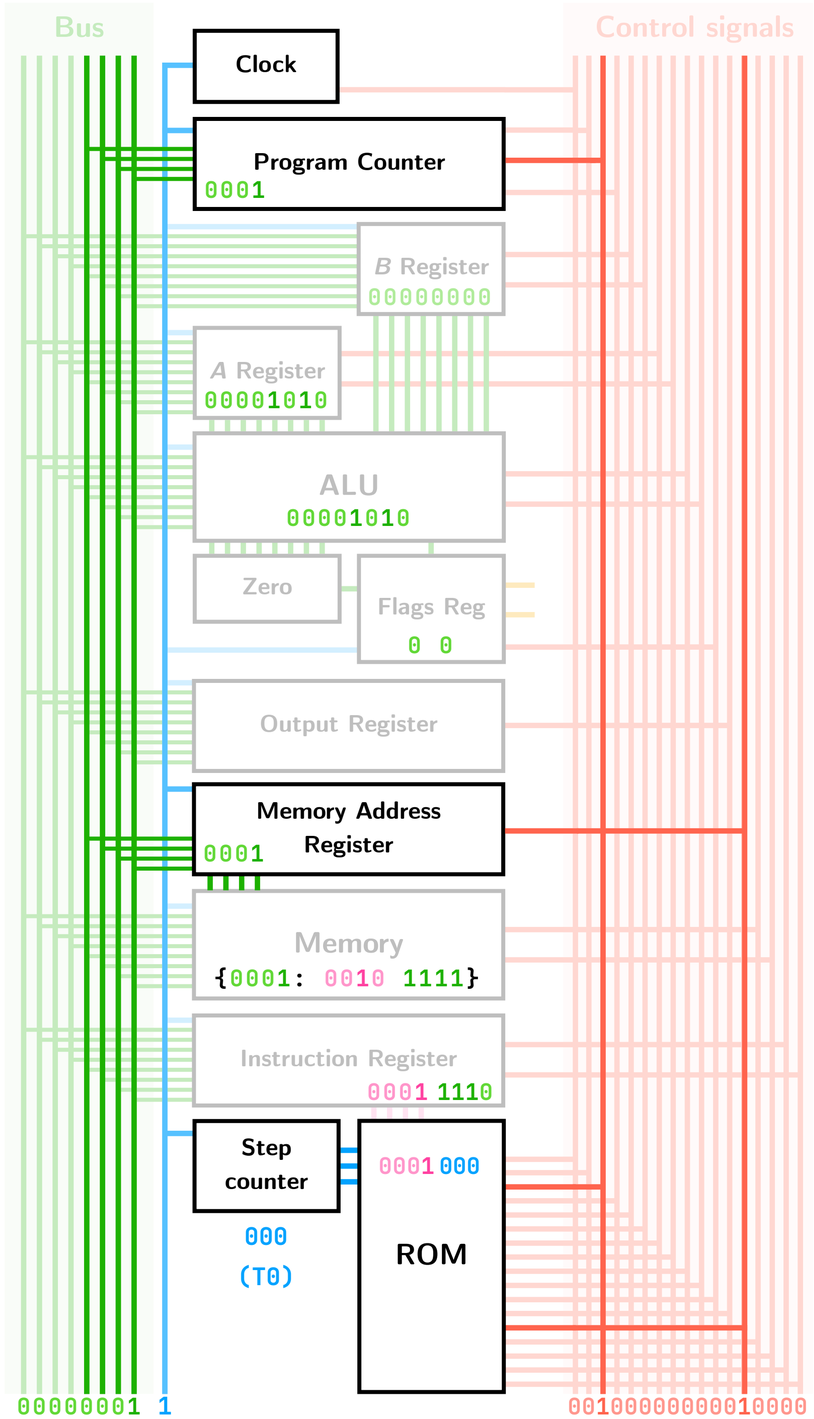
```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    B_IN
[5] ALU_OUT
    A_IN
    FLAGS_IN
```

← Micro-step counter only counts to 101

So it wraps back to 000

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001**1**0000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
LOAD	0001	011	0000000000001**1**0000
LOAD	0001	100	000000**1**0000000100
LOAD	0001	101	0000000000000000
ADD	0010	011	0000000000001**1**0000
ADD	0010	100	0000**1**000000000100
ADD	0010	101	000000**1**00**11**000000
OUT	1110	011	0000000**1**000**1**00000
OUT	1110	100	0000000000000000
OUT	1110	101	0000000000000000
HALT	1111	011	**1**000000000000000
HALT	1111	100	0000000000000000
HALT	1111	101	0000000000000000



HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE

BI B_IN
BO B_OUT

AI A_IN
AO A_OUT

SU SUBTRACT
EO ALU_OUT

CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN

OI OUTPUT_IN

MA MEMORY_ADDRESS_IN

MI MEMORY_IN
MO MEMORY_OUT

II INSTRUCTION_IN
IO INSTRUCTION_OUT

Program

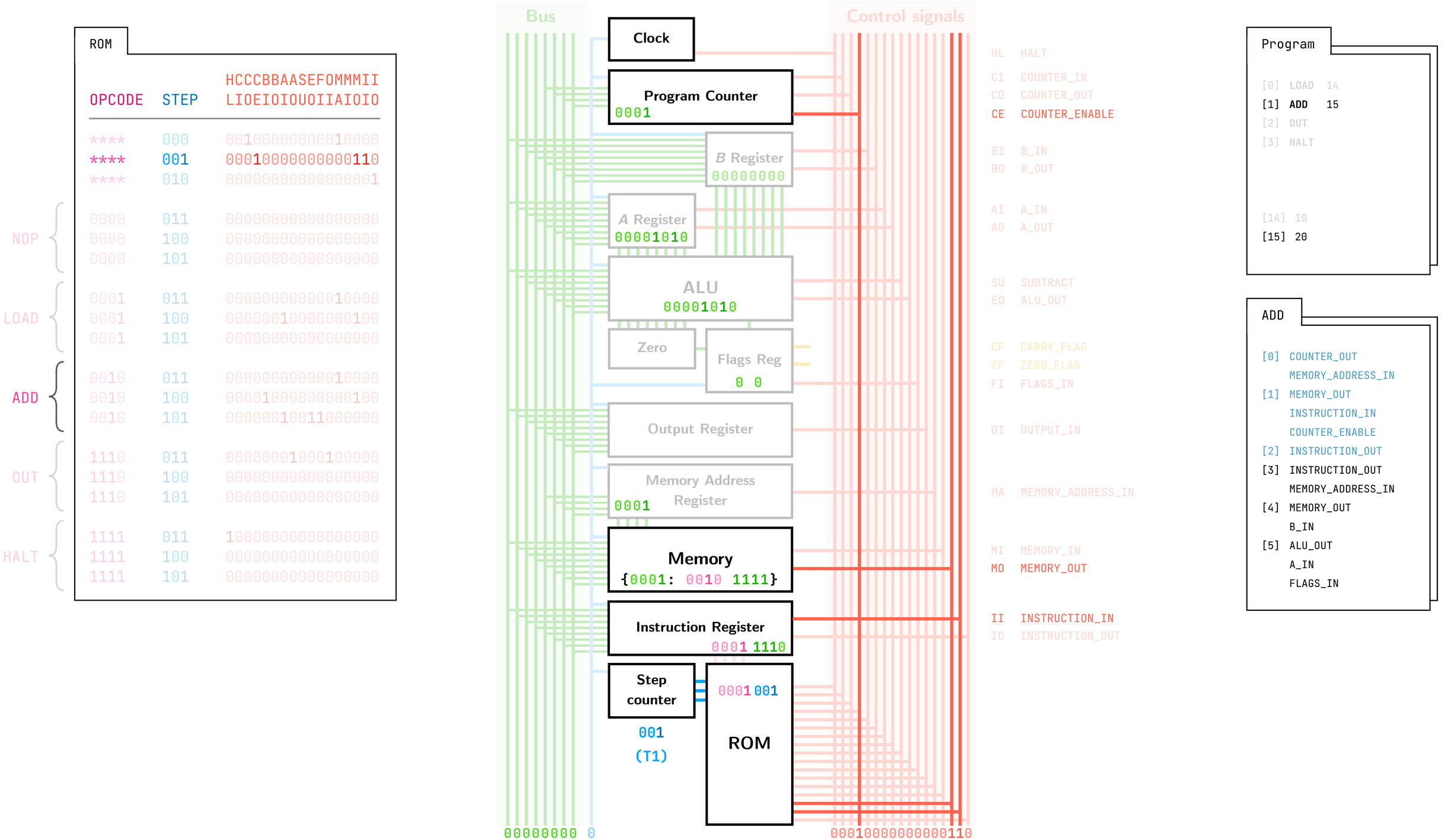
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

ADD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    B_IN
[5] ALU_OUT
    A_IN
    FLAGS_IN
```

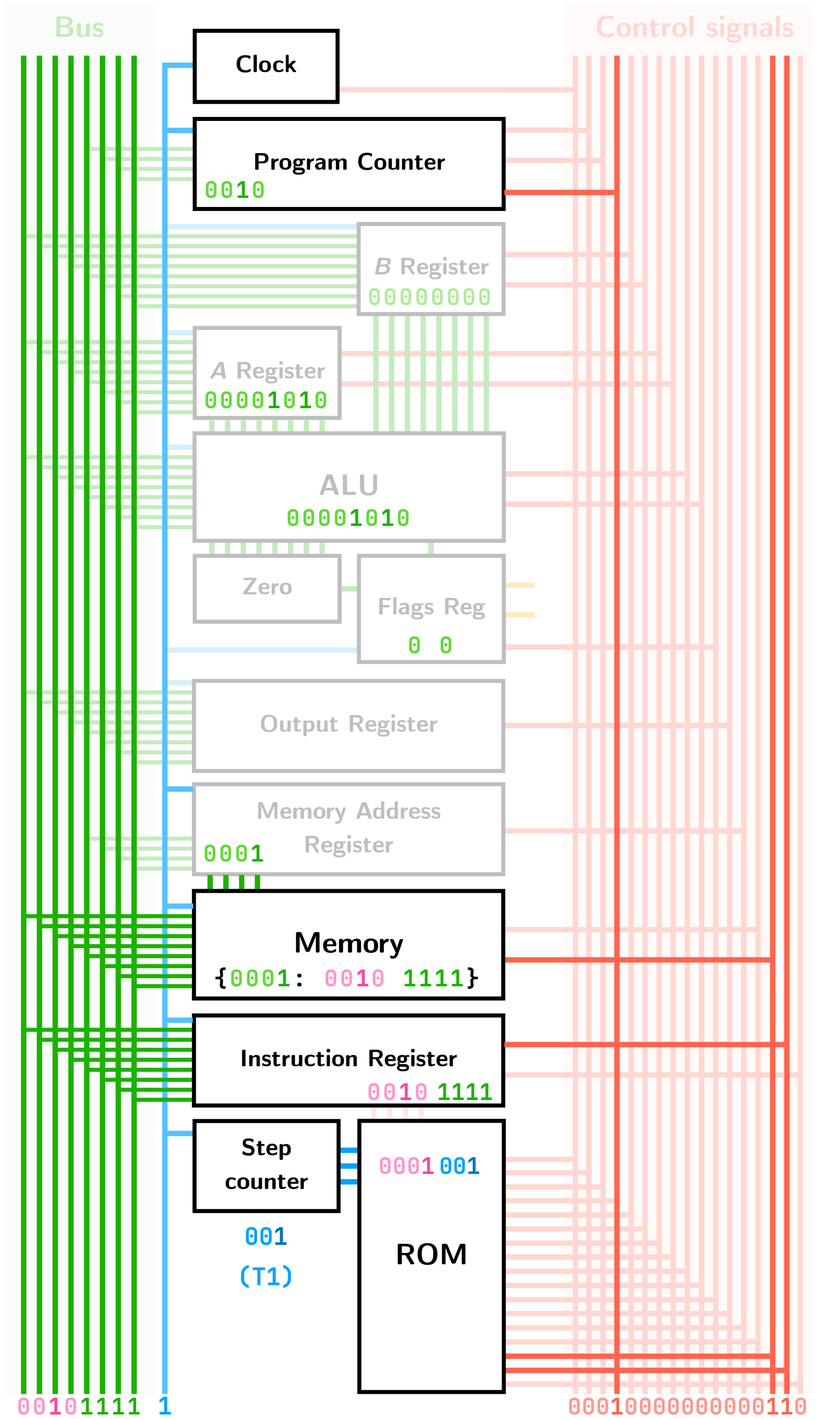
← Micro-step counter only counts to 101

So it wraps back to 000


ROM
OPCODE STEP HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
**** 000 0010000000010000
**** 001 0001000000000110
**** 010 0000000000000001
NOP
0000 011 0000000000000000
0000 100 0000000000000000
0000 101 0000000000000000
LOAD
0001 011 0000000000010000
0001 100 0000010000000100
0001 101 0000000000000000
ADD
0010 011 0000000000010000
0010 100 0000100000000100
0010 101 0000010011000000
OUT
1110 011 0000000100010000
1110 100 0000000000000000
1110 101 0000000000000000
HALT
1111 011 1000000000000000
1111 100 0000000000000000
1111 101 0000000000000000
Bus
Control signals
Clock
Program Counter
0001
B Register
00000000
A Register
00001010
ALU
00001010
Zero
Flags Reg
0 0
Output Register
Memory Address Register
0001
Memory
{0001: 0010 1111}
Instruction Register
0001 1110
Step counter
001
(T1)
0001 001
ROM
00000000 0
0001000000000110
HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
BI B_IN
BO B_OUT
AI A_IN
AO A_OUT
SU SUBTRACT
EO ALU_OUT
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
OI OUTPUT_IN
MA MEMORY_ADDRESS_IN
MI MEMORY_IN
MO MEMORY_OUT
II INSTRUCTION_IN
IO INSTRUCTION_OUT
Program
[0] LOAD 14
[1] ADD 15
[2] OUT
[3] HALT
[14] 10
[15] 20
ADD
[0] COUNTER_OUT
MEMORY_ADDRESS_IN
[1] MEMORY_OUT
INSTRUCTION_IN
COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
MEMORY_ADDRESS_IN
[4] MEMORY_OUT
B_IN
[5] ALU_OUT
A_IN
FLAGS_IN

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
LOAD	0001	100	0000010000000100
LOAD	0001	101	0000000000000000
ADD	0010	011	0000000000010000
ADD	0010	100	0000100000000100
ADD	0010	101	0000010011000000
OUT	1110	011	0000000100010000
OUT	1110	100	0000000000000000
OUT	1110	101	0000000000000000
HALT	1111	011	1000000000000000
HALT	1111	100	0000000000000000
HALT	1111	101	0000000000000000



HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
BI B_IN
BO B_OUT
AI A_IN
AO A_OUT
SU SUBTRACT
EO ALU_OUT
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
OI OUTPUT_IN
MA MEMORY_ADDRESS_IN
MI MEMORY_IN
MO MEMORY_OUT
II INSTRUCTION_IN
IO INSTRUCTION_OUT

Program

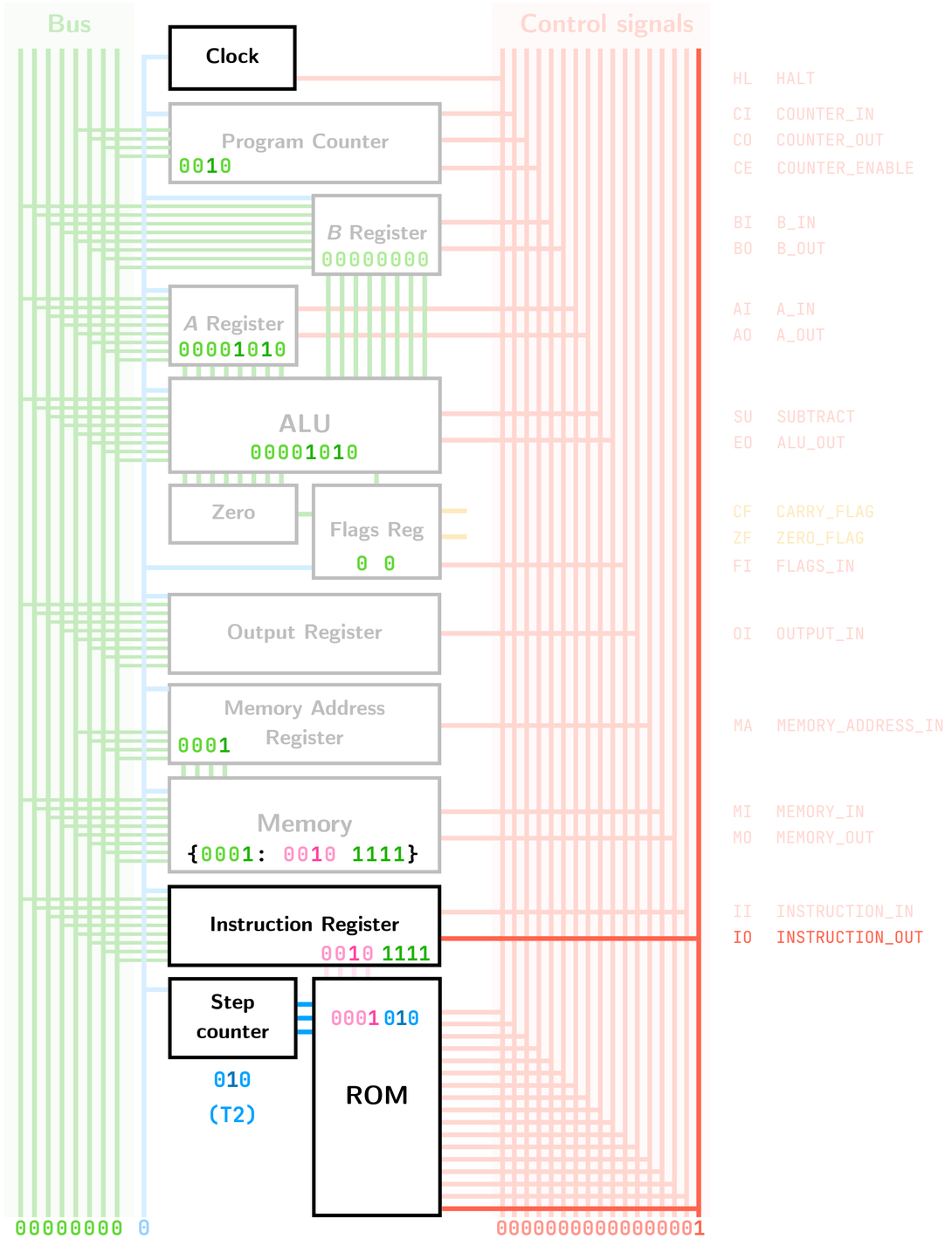
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

ADD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    B_IN
[5] ALU_OUT
    A_IN
    FLAGS_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMIILIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

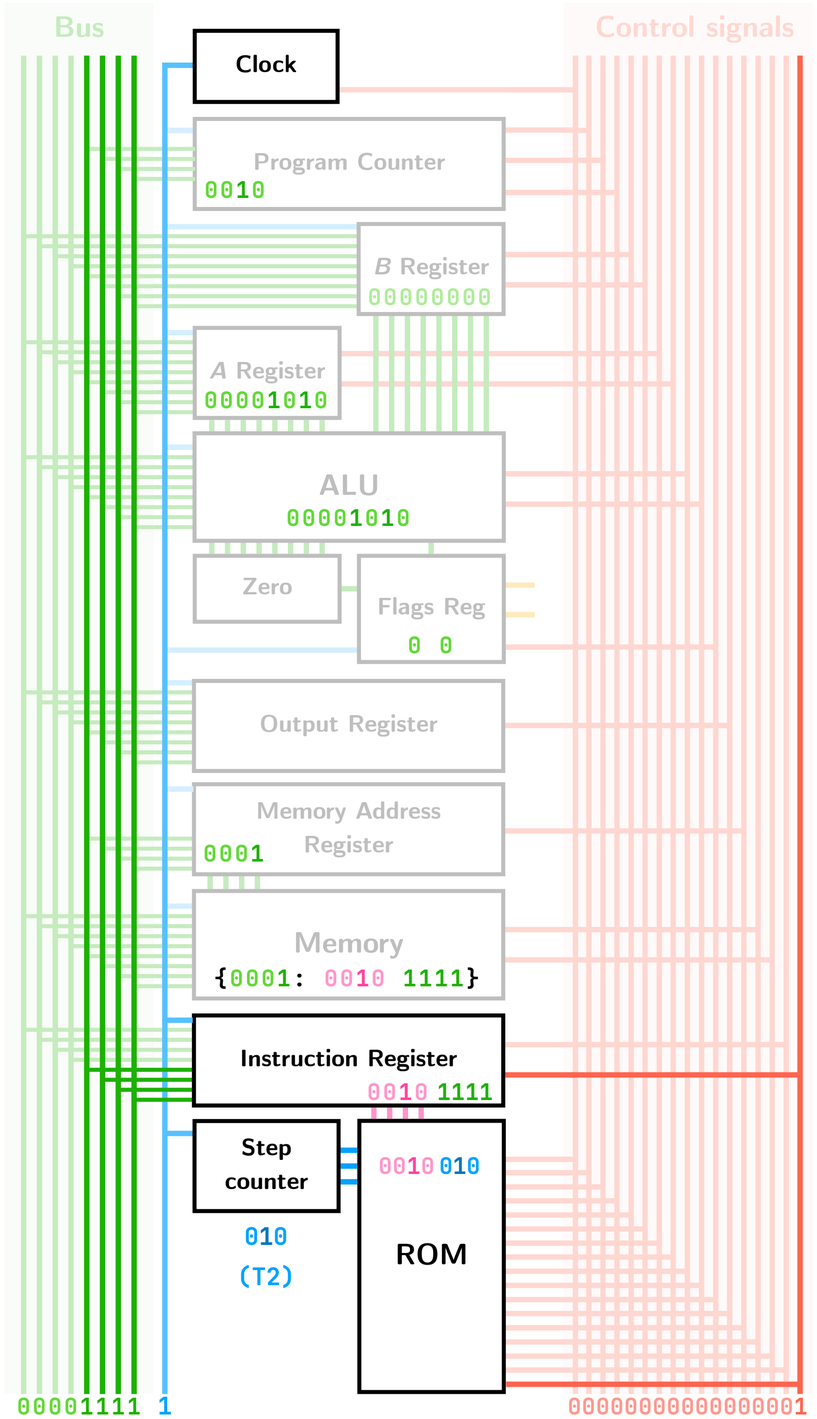
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

ADD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    B_IN
[5] ALU_OUT
    A_IN
    FLAGS_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMIILIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
BI B_IN
BO B_OUT
AI A_IN
AO A_OUT
SU SUBTRACT
EO ALU_OUT
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
OI OUTPUT_IN
MA MEMORY_ADDRESS_IN
MI MEMORY_IN
MO MEMORY_OUT
II INSTRUCTION_IN
IO INSTRUCTION_OUT

Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT

[14] 10
[15] 20
```

ADD

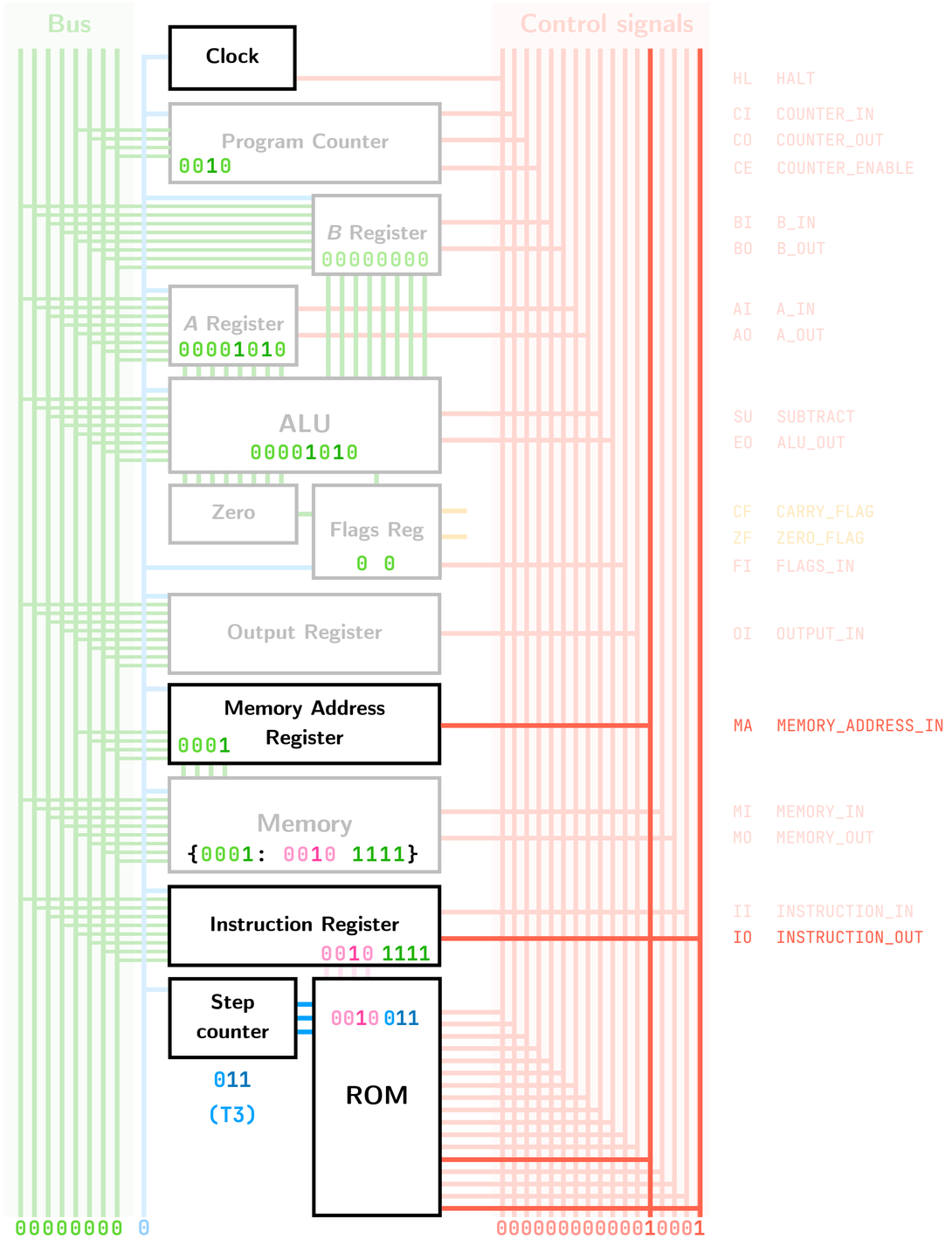
```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    B_IN
[5] ALU_OUT
    A_IN
    FLAGS_IN
```

← ROM is continuously addressed
Address changes mid-cycle
But it's the same control word

IR can only write last 4 bits to bus →

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000001001000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

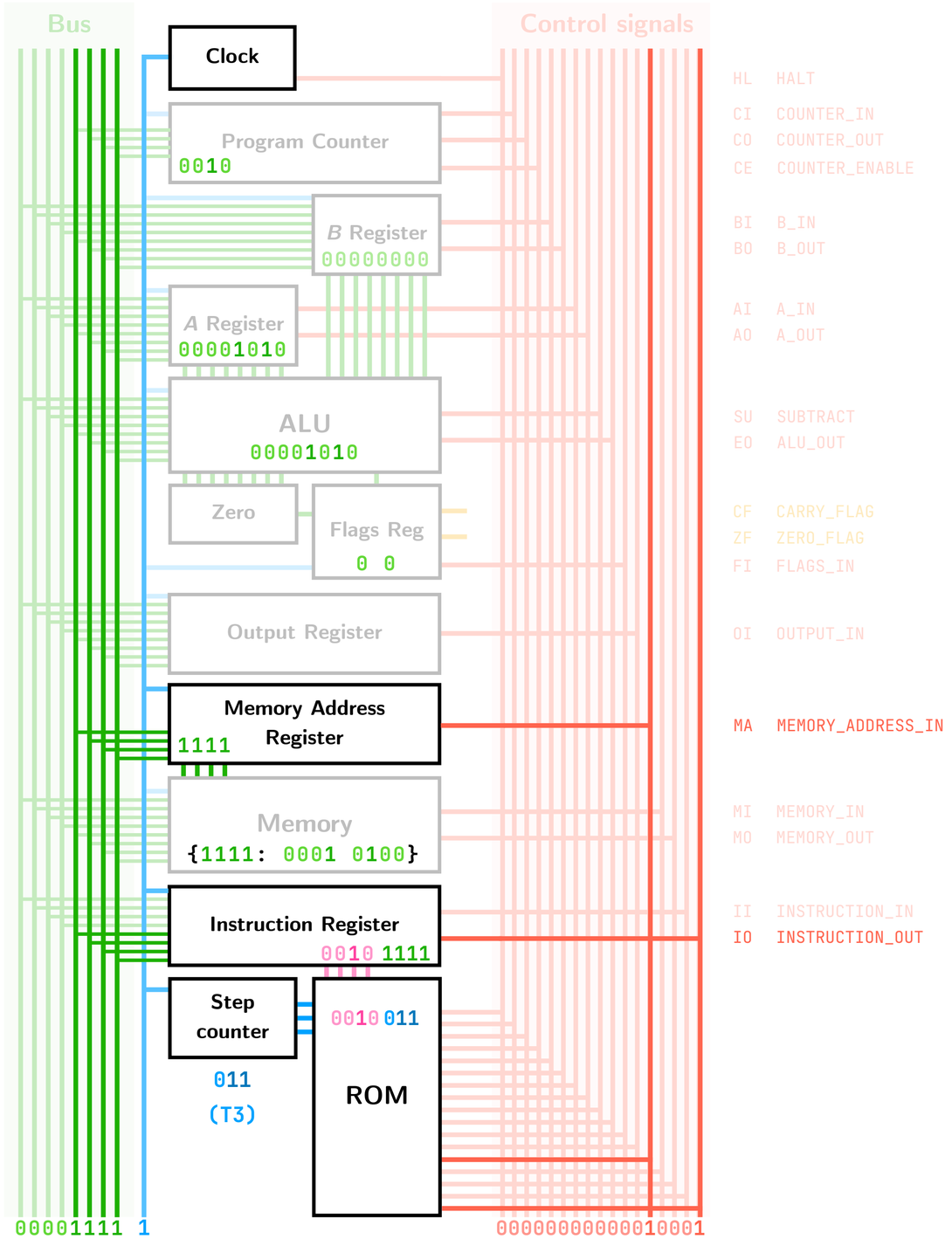
ADD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    B_IN
[5] ALU_OUT
    A_IN
    FLAGS_IN
```

IR can only write last 4 bits to bus →

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	00100000000010000
	****	001	00010000000000110
	****	010	00000000000000001
NOP	0000	011	00000000000000000
NOP	0000	100	00000000000000000
NOP	0000	101	00000000000000000
LOAD	0001	011	00000000000010000
LOAD	0001	100	00000010000000100
LOAD	0001	101	00000000000000000
ADD	0010	011	00000000000010000
ADD	0010	100	00001000000000100
ADD	0010	101	00000010011000000
OUT	1110	011	00000001000100000
OUT	1110	100	00000000000000000
OUT	1110	101	00000000000000000
HALT	1111	011	10000000000000000
HALT	1111	100	00000000000000000
HALT	1111	101	00000000000000000



IR can only write last 4 bits to bus →

Program

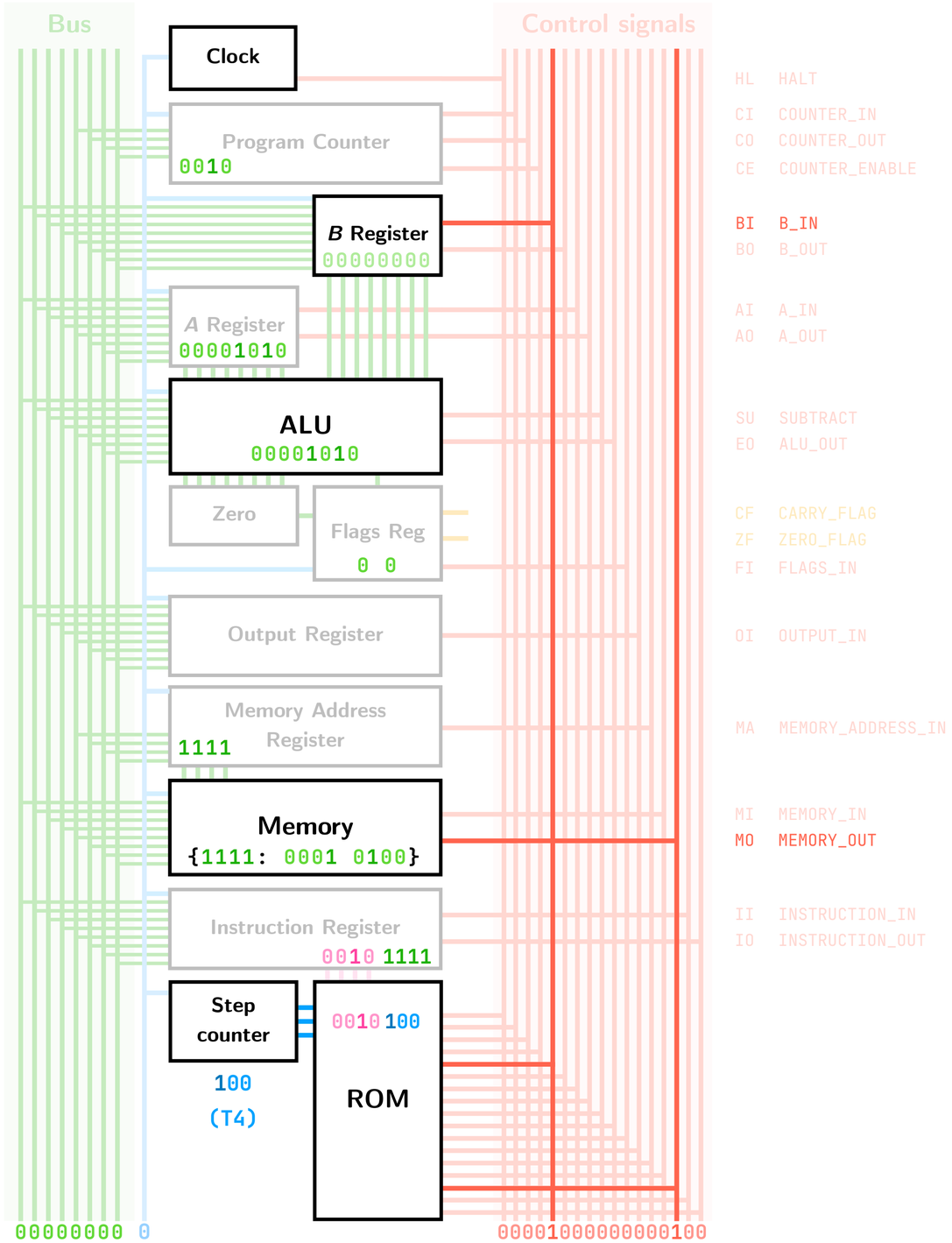
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

ADD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    B_IN
[5] ALU_OUT
    A_IN
    FLAGS_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMIILIOEIOIOUOIIAIOIO
	****	000	00100000000010000
	****	001	00010000000000110
	****	010	00000000000000001
NOP	0000	011	00000000000000000
NOP	0000	100	00000000000000000
NOP	0000	101	00000000000000000
LOAD	0001	011	00000000000010000
LOAD	0001	100	00000100000000100
LOAD	0001	101	00000000000000000
ADD	0010	011	00000000000010000
ADD	0010	100	00001000000000100
ADD	0010	101	00000010011000000
OUT	1110	011	00000001000100000
OUT	1110	100	00000000000000000
OUT	1110	101	00000000000000000
HALT	1111	011	10000000000000000
HALT	1111	100	00000000000000000
HALT	1111	101	00000000000000000



Program

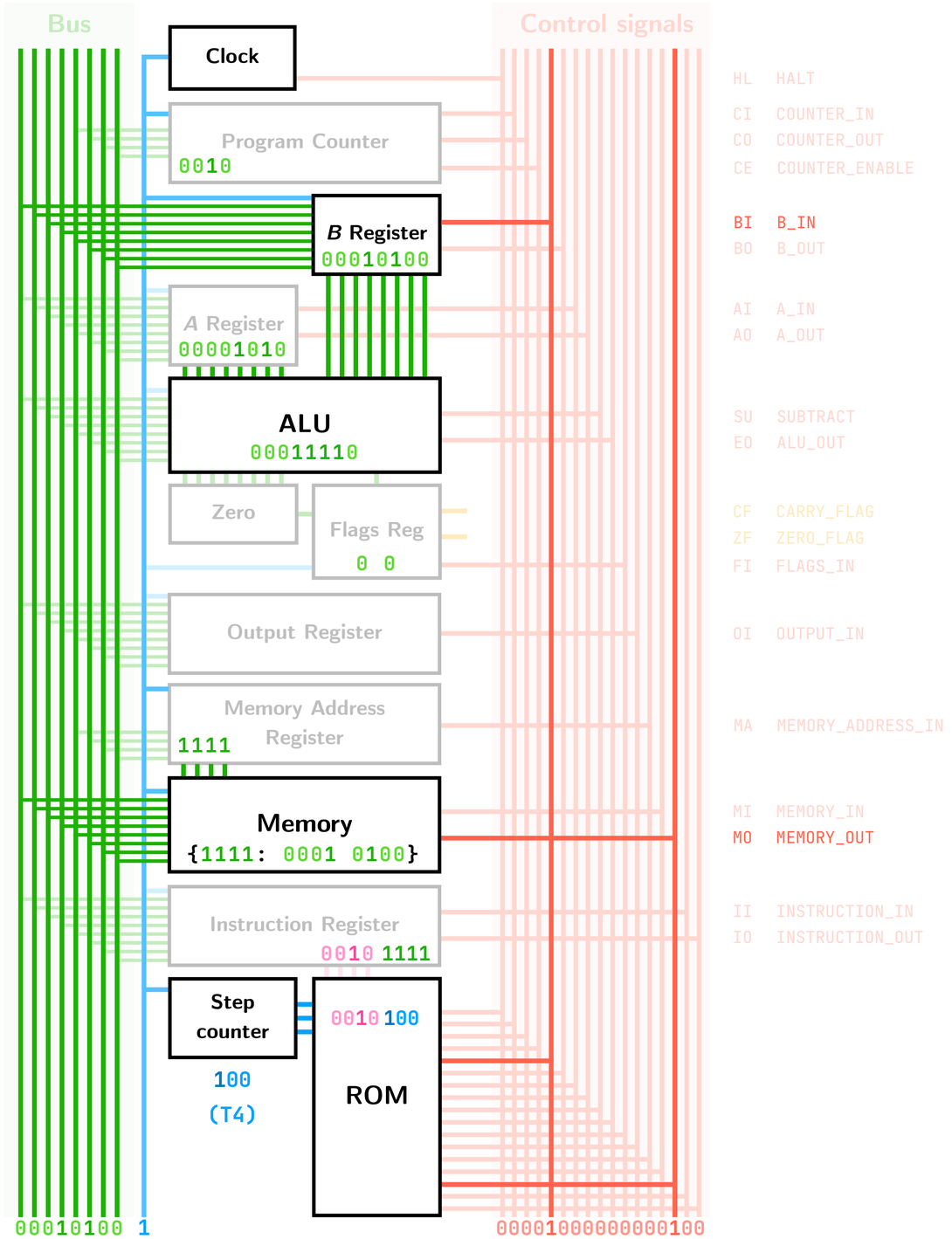
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

ADD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    B_IN
[5] ALU_OUT
    A_IN
    FLAGS_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

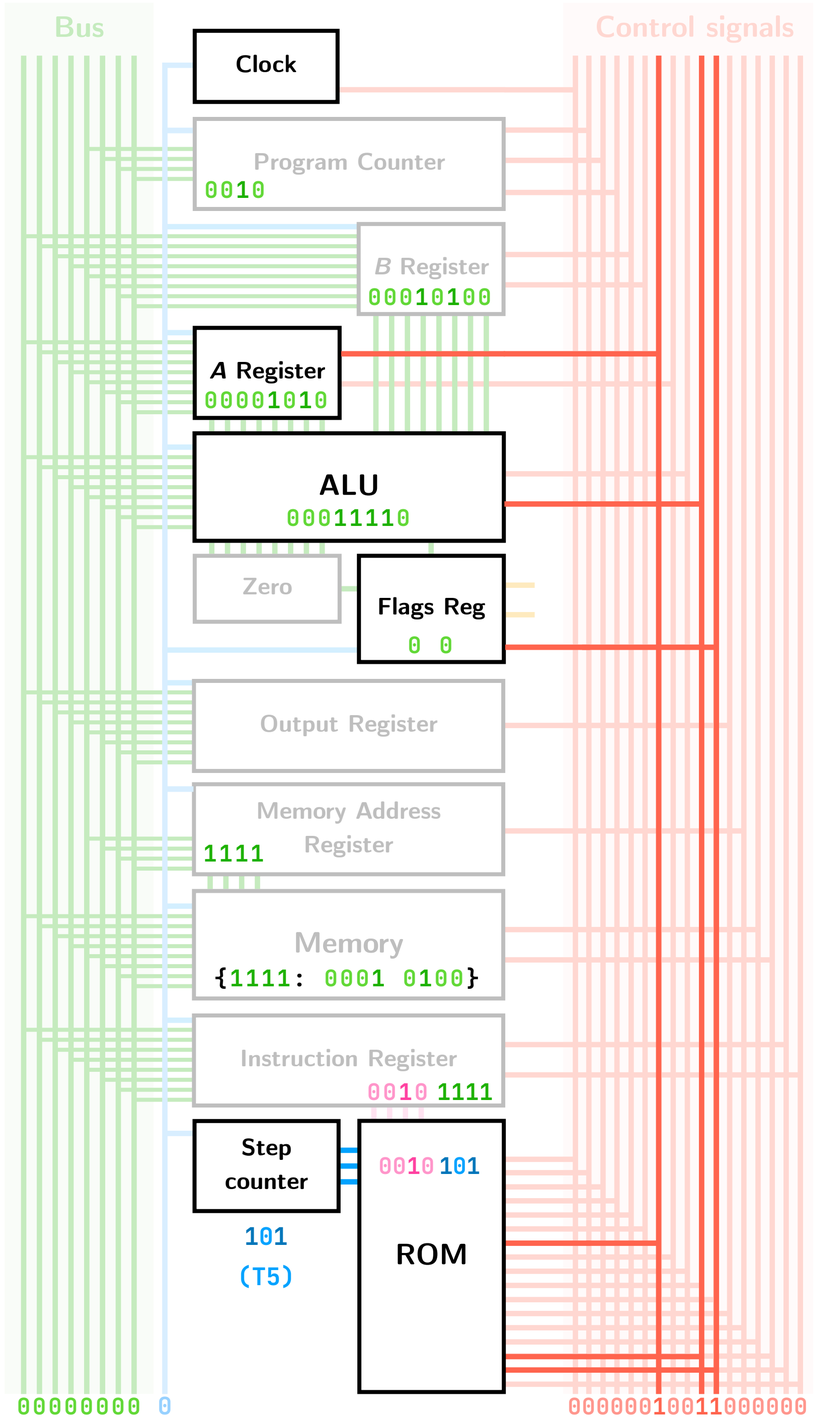
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

ADD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    B_IN
[5] ALU_OUT
    A_IN
    FLAGS_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
LOAD	0001	100	0000010000000100
LOAD	0001	101	0000000000000000
ADD	0010	011	0000000000010000
ADD	0010	100	0000100000000100
ADD	0010	101	0000010011000000
OUT	1110	011	0000000100010000
OUT	1110	100	0000000000000000
OUT	1110	101	0000000000000000
HALT	1111	011	1000000000000000
HALT	1111	100	0000000000000000
HALT	1111	101	0000000000000000



HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
BI B_IN
BO B_OUT
AI A_IN
AO A_OUT
SU SUBTRACT
EO ALU_OUT
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
OI OUTPUT_IN
MA MEMORY_ADDRESS_IN
MI MEMORY_IN
MO MEMORY_OUT
II INSTRUCTION_IN
IO INSTRUCTION_OUT

Program

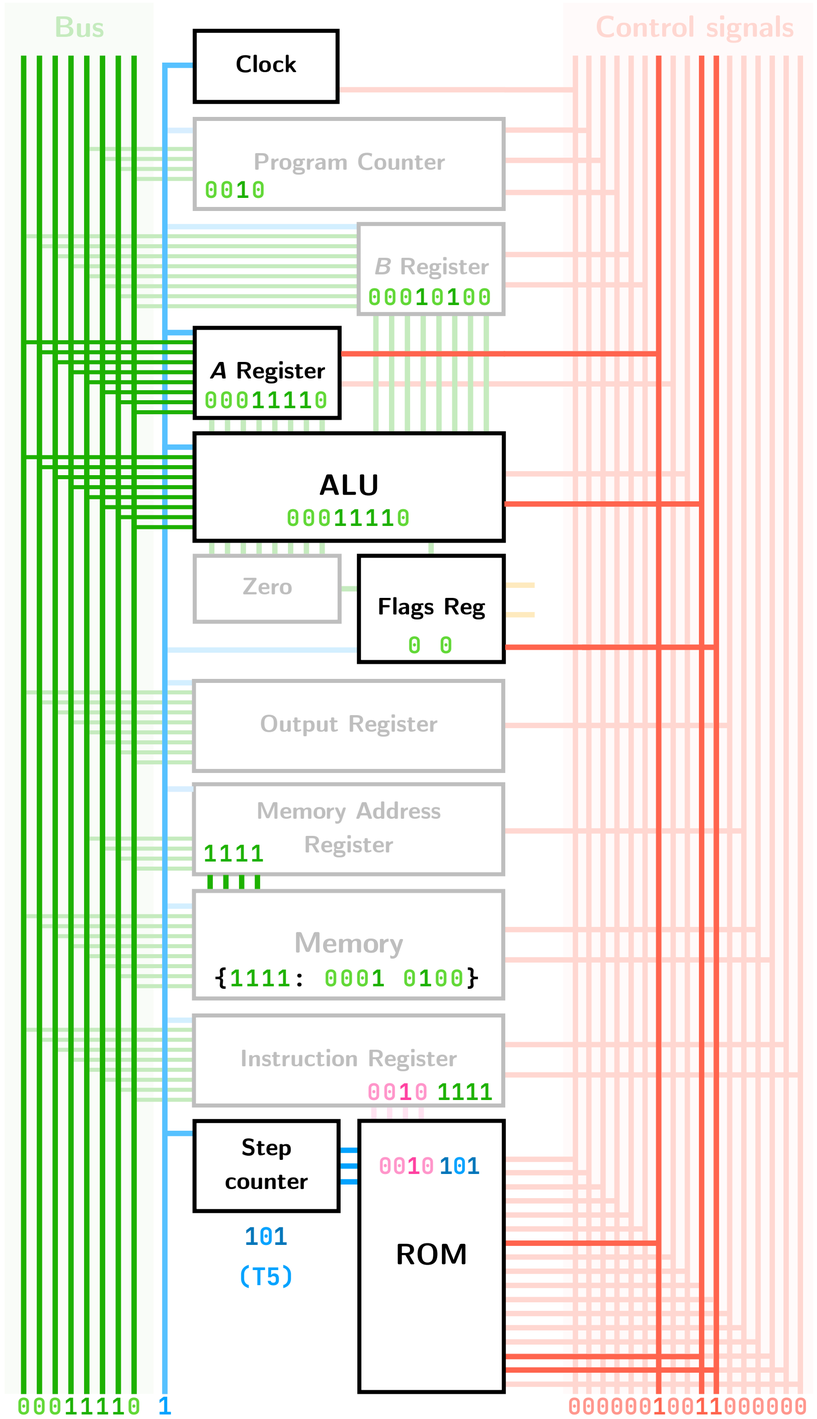
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

ADD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    B_IN
[5] ALU_OUT
    A_IN
    FLAGS_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	00100000000010000
	****	001	00010000000000110
	****	010	00000000000000001
NOP	0000	011	00000000000000000
NOP	0000	100	00000000000000000
NOP	0000	101	00000000000000000
LOAD	0001	011	00000000000010000
LOAD	0001	100	00000100000000100
LOAD	0001	101	00000000000000000
ADD	0010	011	00000000000010000
ADD	0010	100	00001000000000100
ADD	0010	101	00000010011000000
OUT	1110	011	00000001000100000
OUT	1110	100	00000000000000000
OUT	1110	101	00000000000000000
HALT	1111	011	10000000000000000
HALT	1111	100	00000000000000000
HALT	1111	101	00000000000000000



HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE

BI B_IN
BO B_OUT

AI A_IN
AO A_OUT

SU SUBTRACT
EO ALU_OUT

CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN

OI OUTPUT_IN

MA MEMORY_ADDRESS_IN

MI MEMORY_IN
MO MEMORY_OUT

II INSTRUCTION_IN
IO INSTRUCTION_OUT

Program

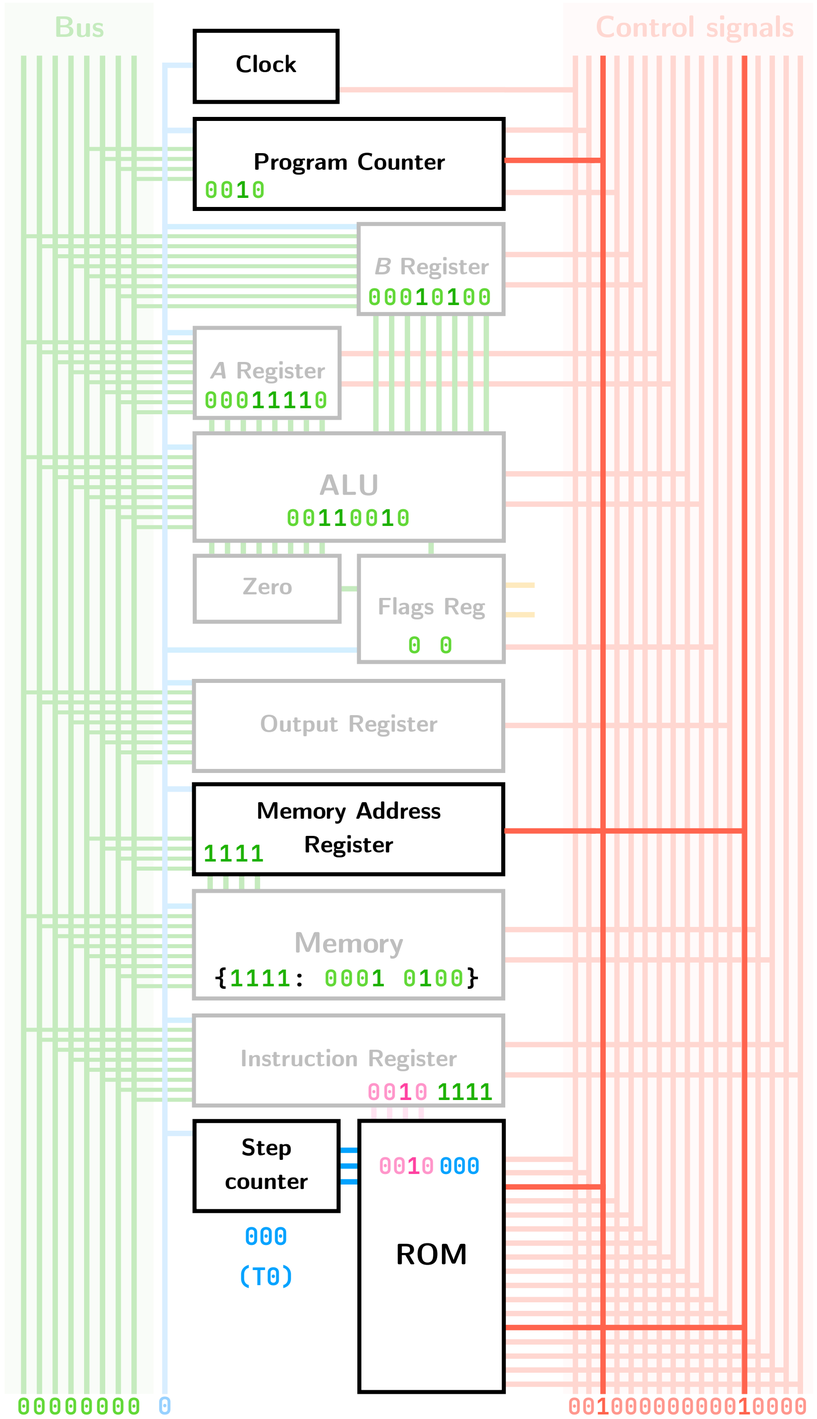
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

ADD

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] INSTRUCTION_OUT
    MEMORY_ADDRESS_IN
[4] MEMORY_OUT
    B_IN
[5] ALU_OUT
    A_IN
    FLAGS_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
LOAD	0001	100	0000010000000100
LOAD	0001	101	0000000000000000
ADD	0010	011	0000000000010000
ADD	0010	100	0000100000000100
ADD	0010	101	0000010011000000
OUT	1110	011	0000000100010000
OUT	1110	100	0000000000000000
OUT	1110	101	0000000000000000
HALT	1111	011	1000000000000000
HALT	1111	100	0000000000000000
HALT	1111	101	0000000000000000



HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE

BI B_IN
BO B_OUT

AI A_IN
AO A_OUT

SU SUBTRACT
EO ALU_OUT

CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN

OI OUTPUT_IN

MA MEMORY_ADDRESS_IN

MI MEMORY_IN
MO MEMORY_OUT

II INSTRUCTION_IN
IO INSTRUCTION_OUT

Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

OUT

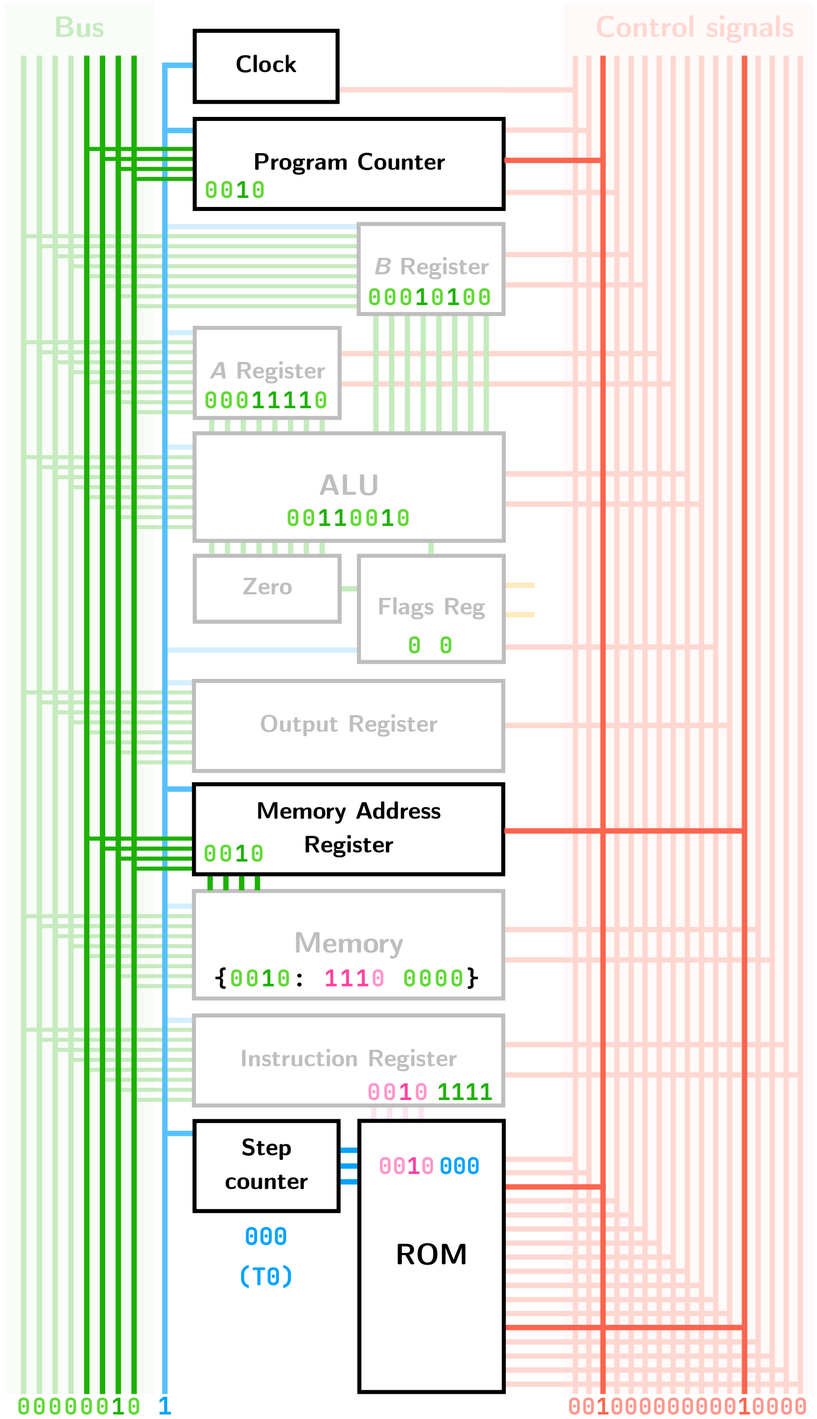
```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
    O_IN
```

← Micro-step counter only counts to 101

So it wraps back to 000

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
BI B_IN
BO B_OUT
AI A_IN
AO A_OUT
SU SUBTRACT
EO ALU_OUT
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
OI OUTPUT_IN
MA MEMORY_ADDRESS_IN
MI MEMORY_IN
MO MEMORY_OUT
II INSTRUCTION_IN
IO INSTRUCTION_OUT

Program

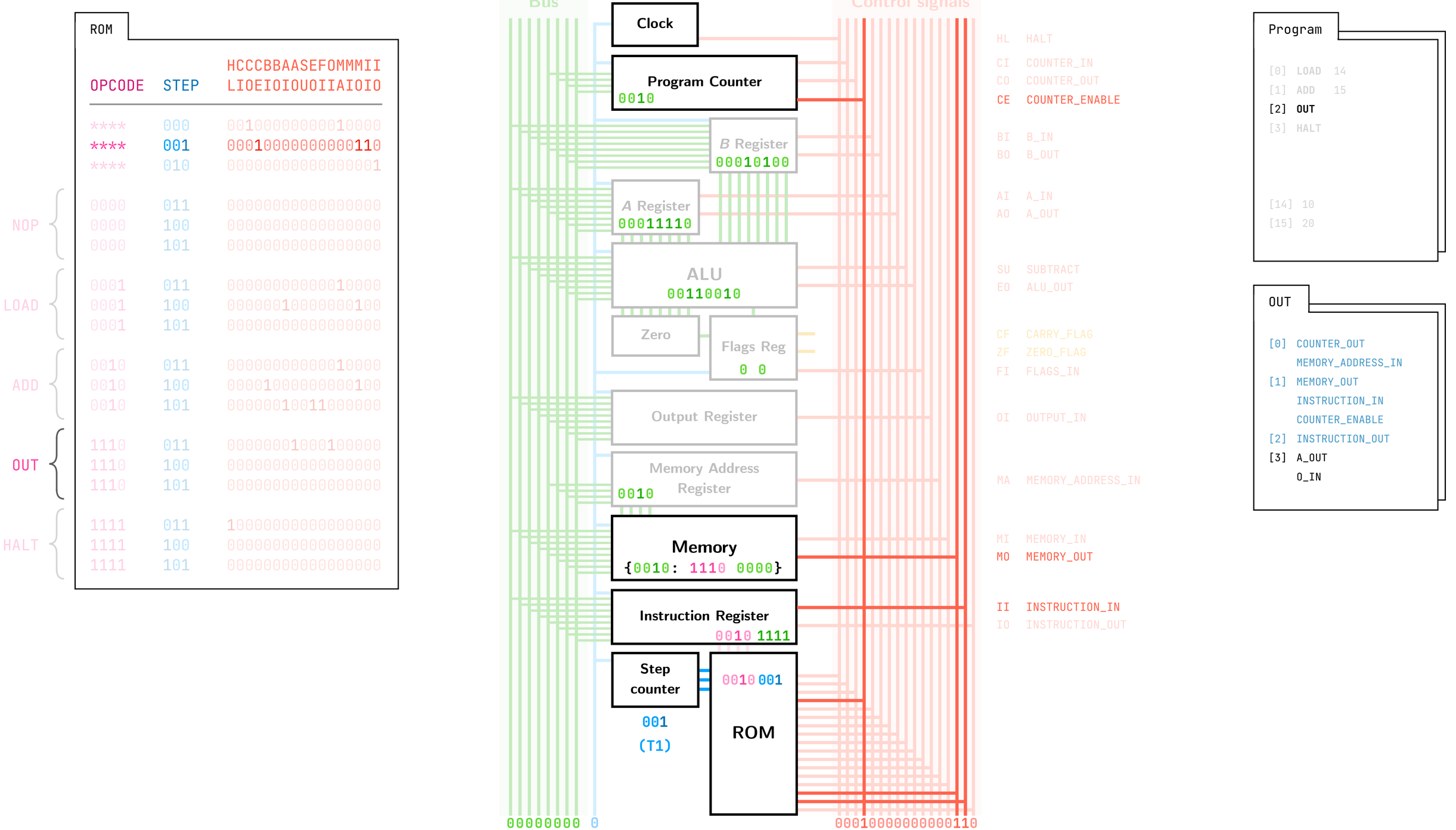
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

OUT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
    O_IN
```

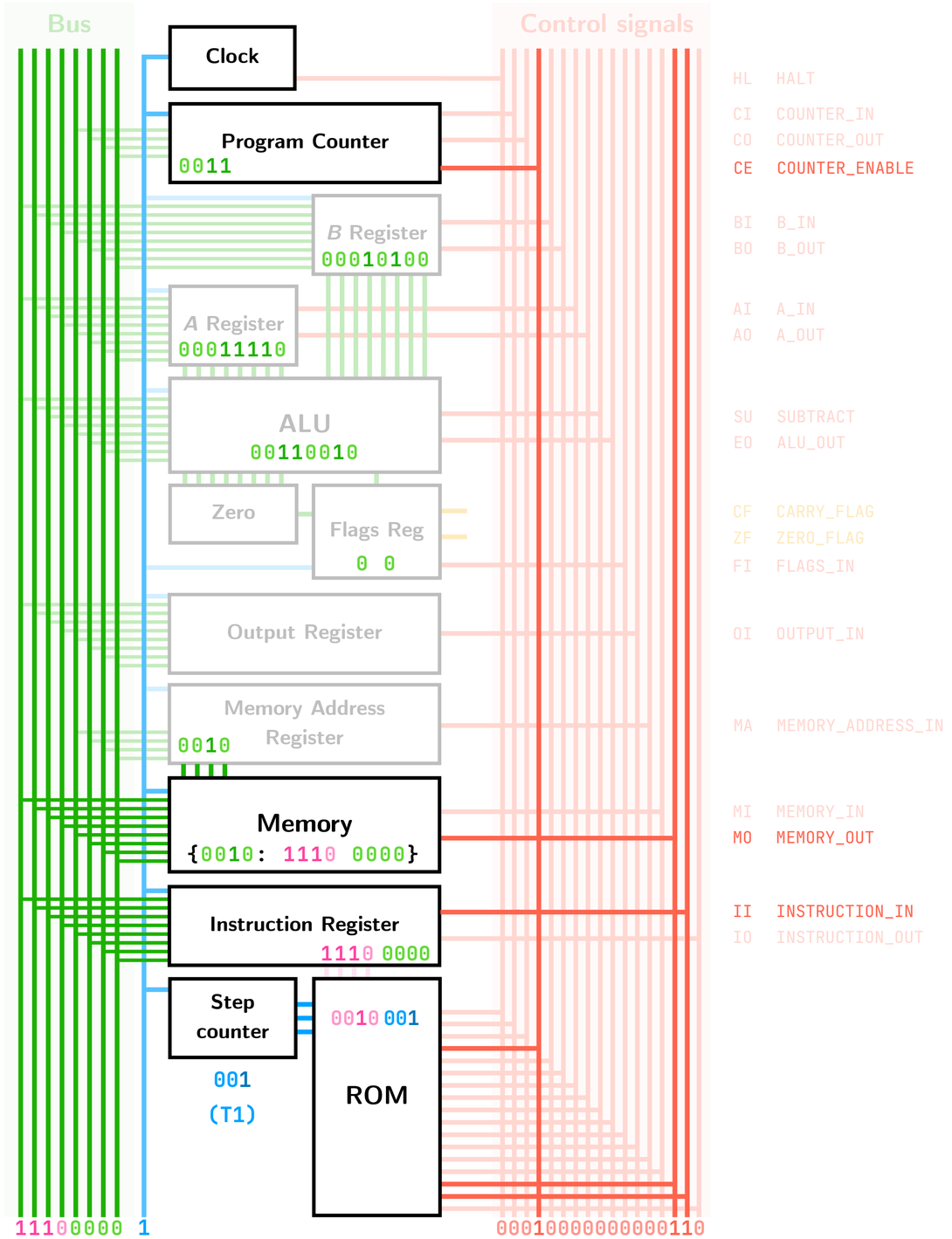
← Micro-step counter only counts to 101

So it wraps back to 000


ROM
OPCODE STEP HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
**** 000 0010000000010000
**** 001 0001000000000110
**** 010 0000000000000001
NOP
0000 011 0000000000000000
0000 100 0000000000000000
0000 101 0000000000000000
LOAD
0001 011 0000000000010000
0001 100 0000010000000100
0001 101 0000000000000000
ADD
0010 011 0000000000010000
0010 100 0000100000000100
0010 101 0000010011000000
OUT
1110 011 0000000100010000
1110 100 0000000000000000
1110 101 0000000000000000
HALT
1111 011 1000000000000000
1111 100 0000000000000000
1111 101 0000000000000000
Bus
Control signals
Clock
Program Counter
0010
B Register
00010100
A Register
00011110
ALU
00110010
Zero
Flags Reg
0 0
Output Register
Memory Address Register
0010
Memory
{0010: 1110 0000}
Instruction Register
0010 1111
Step counter
001
(T1)
0010 001
ROM
00000000 0
0001000000000110
HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
BI B_IN
BO B_OUT
AI A_IN
AO A_OUT
SU SUBTRACT
EO ALU_OUT
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
OI OUTPUT_IN
MA MEMORY_ADDRESS_IN
MI MEMORY_IN
MO MEMORY_OUT
II INSTRUCTION_IN
IO INSTRUCTION_OUT
Program
[0] LOAD 14
[1] ADD 15
[2] OUT
[3] HALT
[14] 10
[15] 20
OUT
[0] COUNTER_OUT
MEMORY_ADDRESS_IN
[1] MEMORY_OUT
INSTRUCTION_IN
COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
O_IN

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMIILIOEIOIOUOIIAIOIO
	****	000	0010000000001000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000001000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

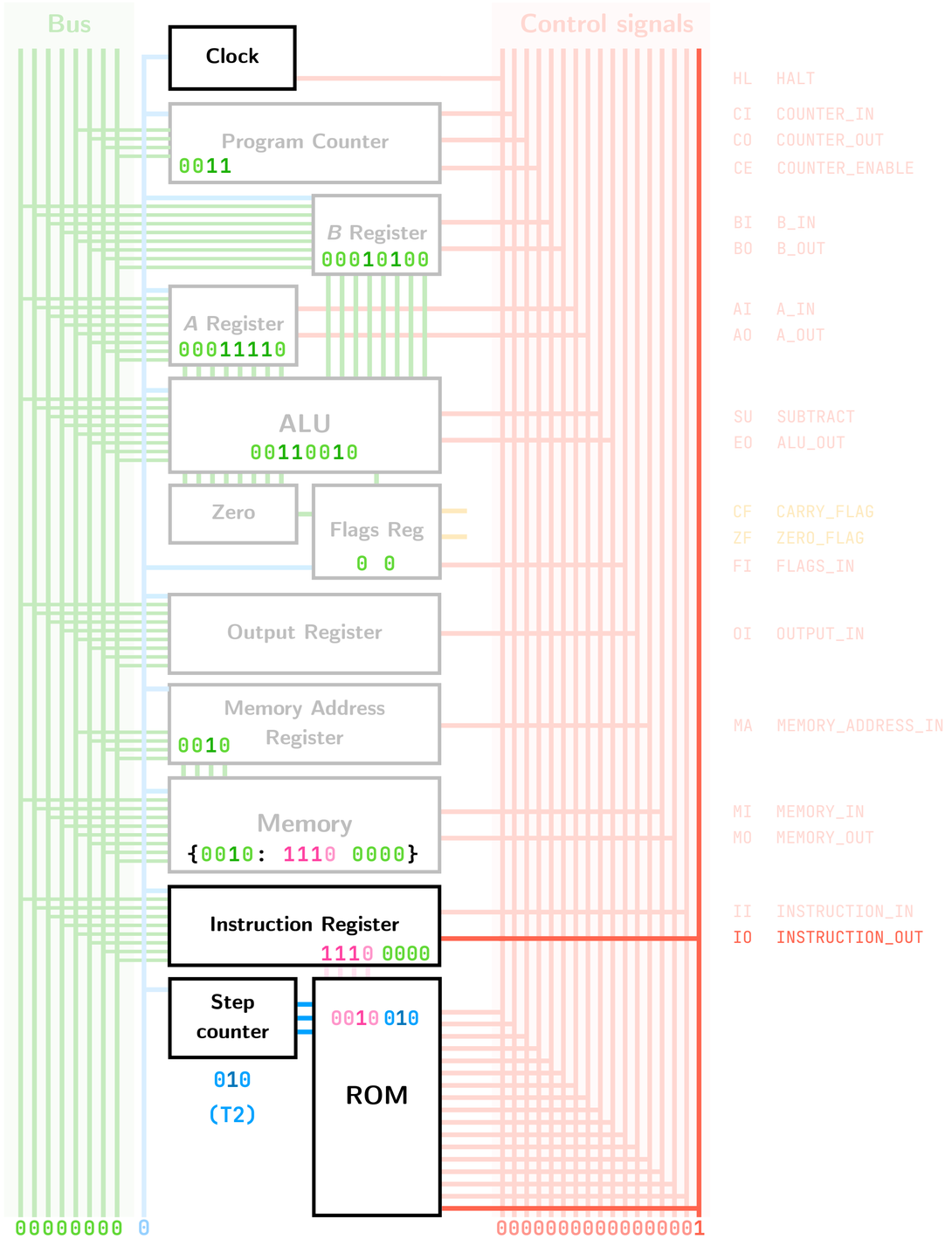
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

OUT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
    O_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMIILIOEIOIOUOIIAIOIO
	****	000	0010000000001 0000
	****	001	0001000000000 110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001 0000
	0001	100	000000 1000000 100
	0001	101	0000000000000000
ADD	0010	011	0000000000001 0000
	0010	100	0000 1000000000 100
	0010	101	000000 1001 1000000
OUT	1110	011	0000000 1000 1 00000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

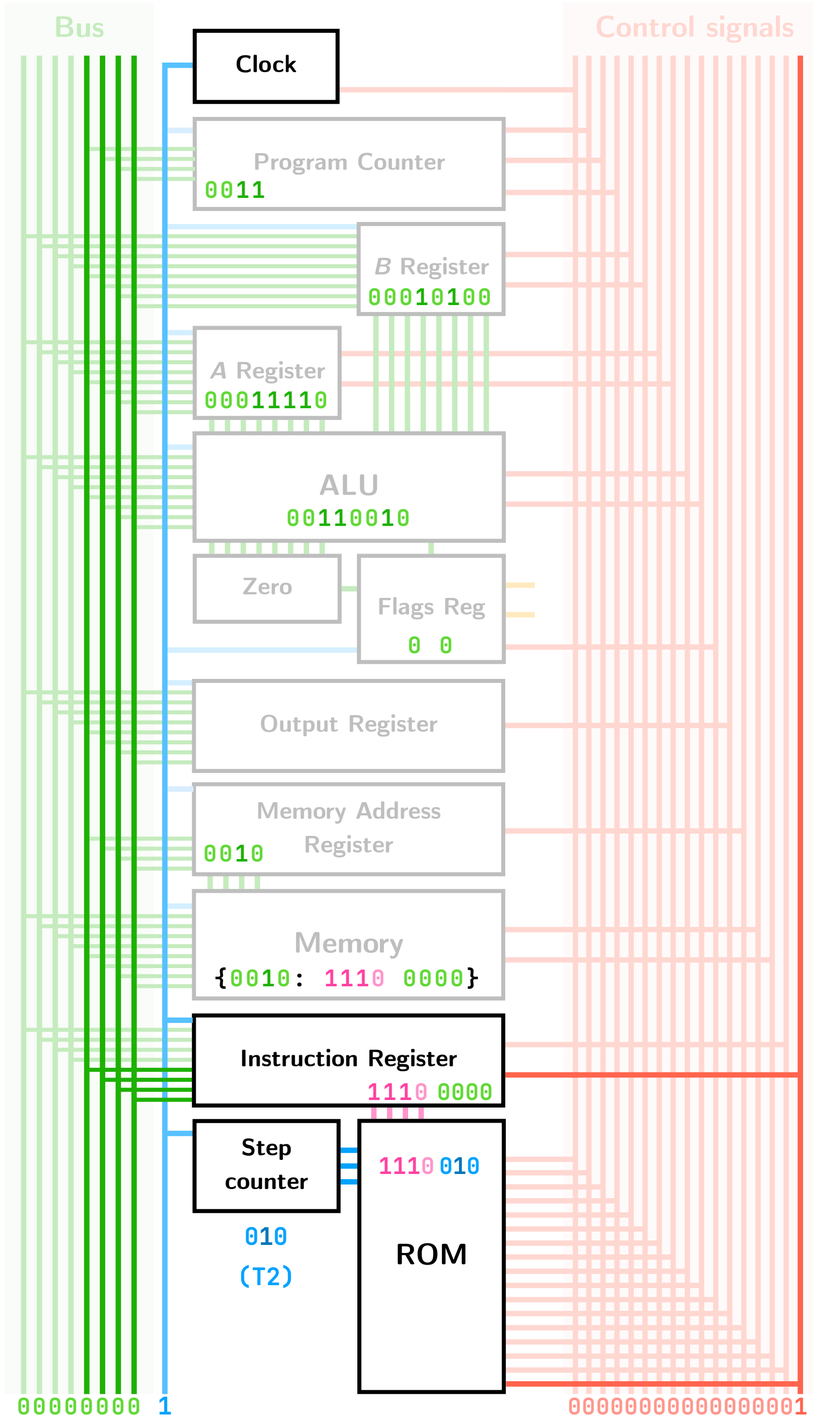
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

OUT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
    O_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000001000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

OUT

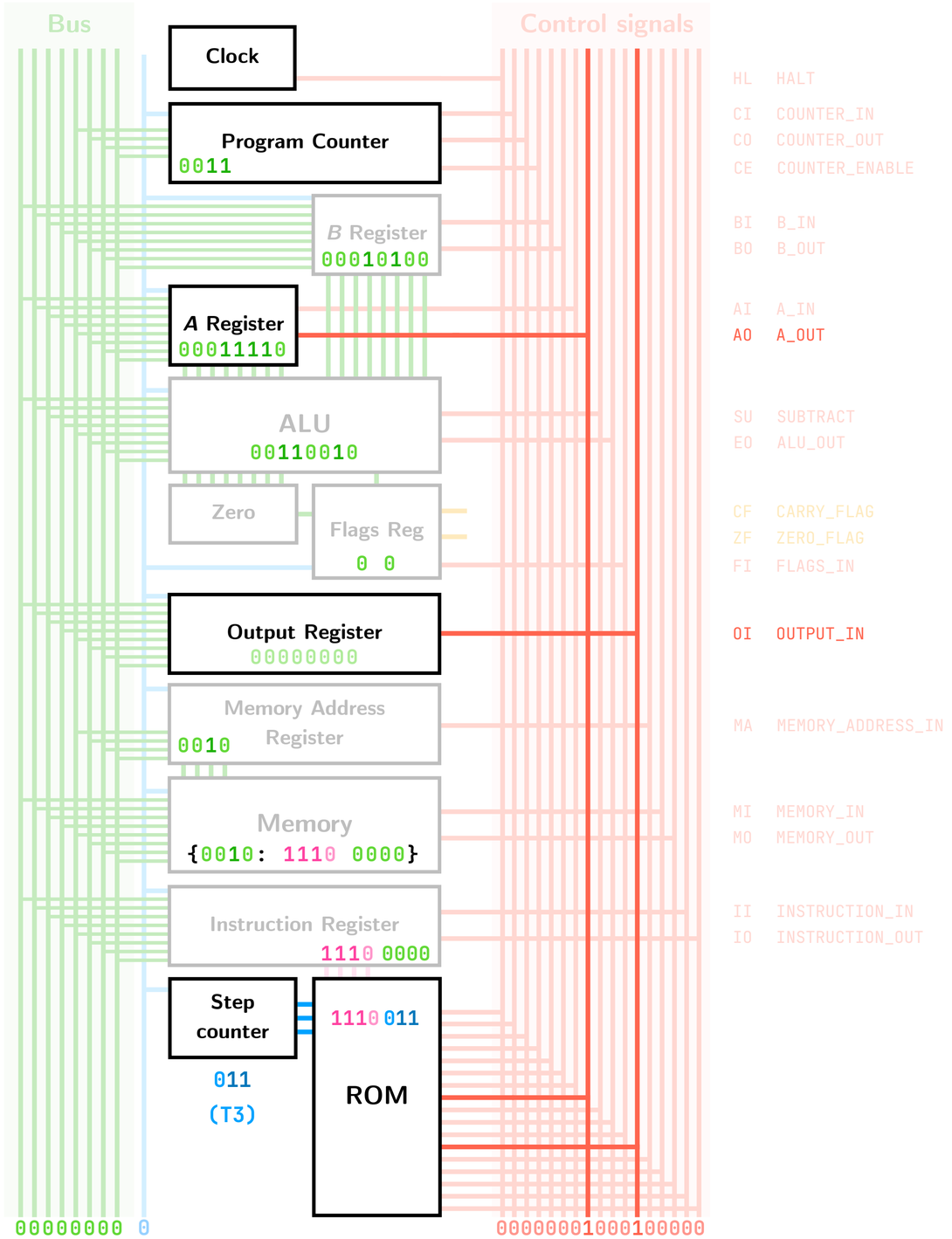
```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
    O_IN
```

← ROM is continuously addressed
Address changes mid-cycle
But it's the same control word

IR can only write last 4 bits to bus →

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMIILIOEIOIOUOIIAIOIO
	****	000	0010000000001000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000001000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



HL	HALT
CI	COUNTER_IN
CO	COUNTER_OUT
CE	COUNTER_ENABLE
BI	B_IN
BO	B_OUT
AI	A_IN
AO	A_OUT
SU	SUBTRACT
EO	ALU_OUT
CF	CARRY_FLAG
ZF	ZERO_FLAG
FI	FLAGS_IN
OI	OUTPUT_IN
MA	MEMORY_ADDRESS_IN
MI	MEMORY_IN
MO	MEMORY_OUT
II	INSTRUCTION_IN
IO	INSTRUCTION_OUT

Program

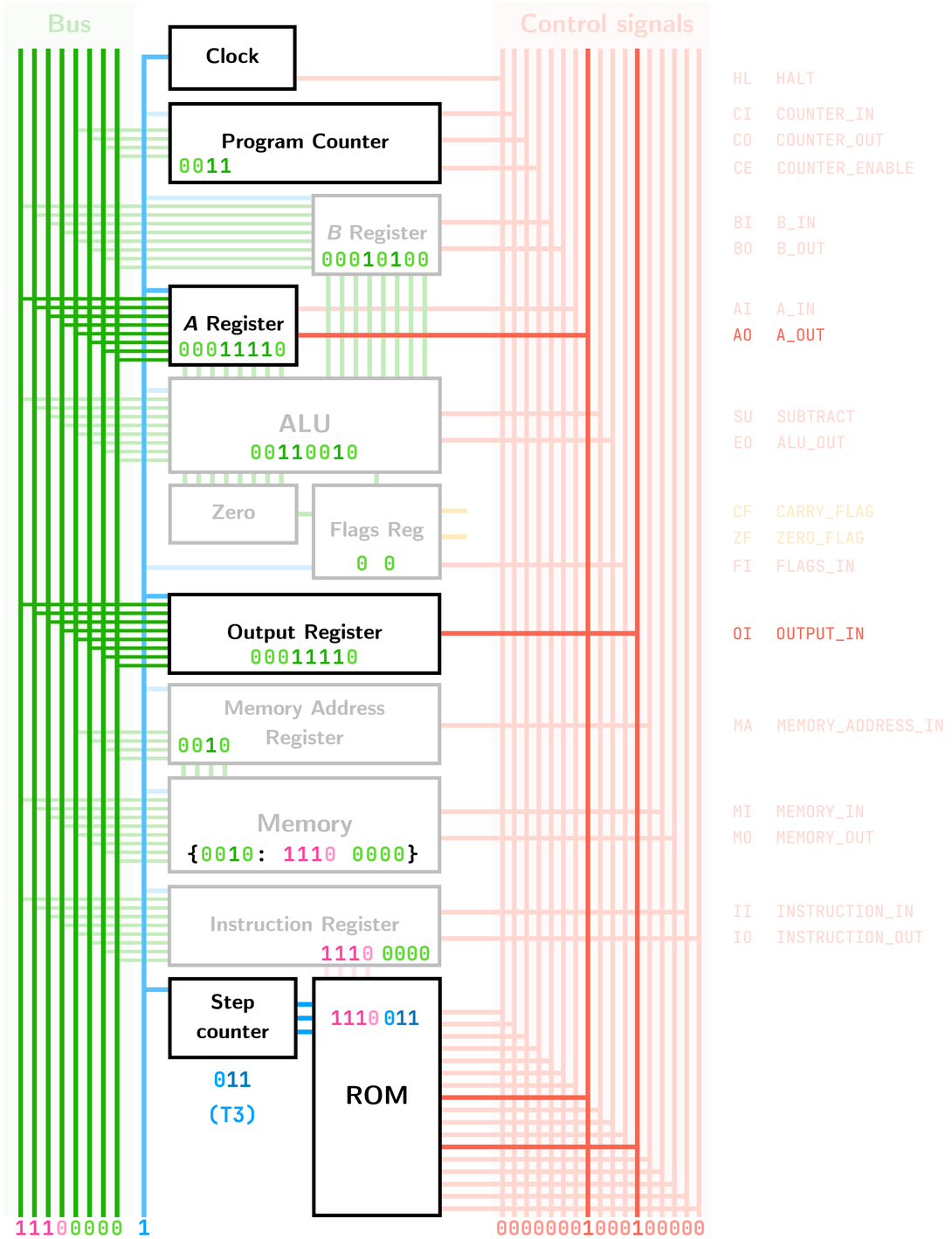
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

OUT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
    O_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMIILIOEIOIOUOIIAIOIO
	****	000	0010000000001000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000001000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

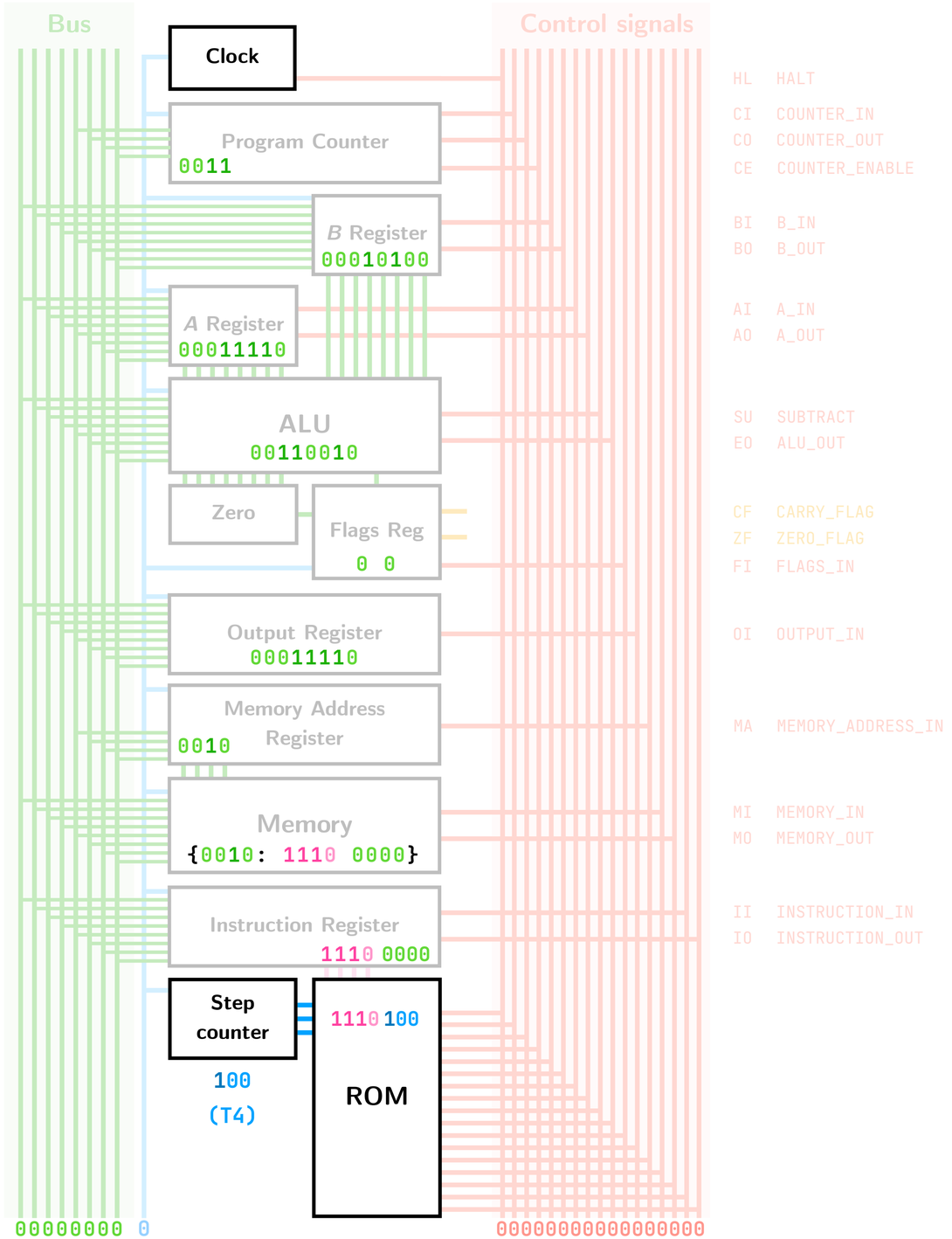
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

OUT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
    O_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMIILIOEIOIOUOIIAIOIO
	****	000	0010000000001000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000001000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

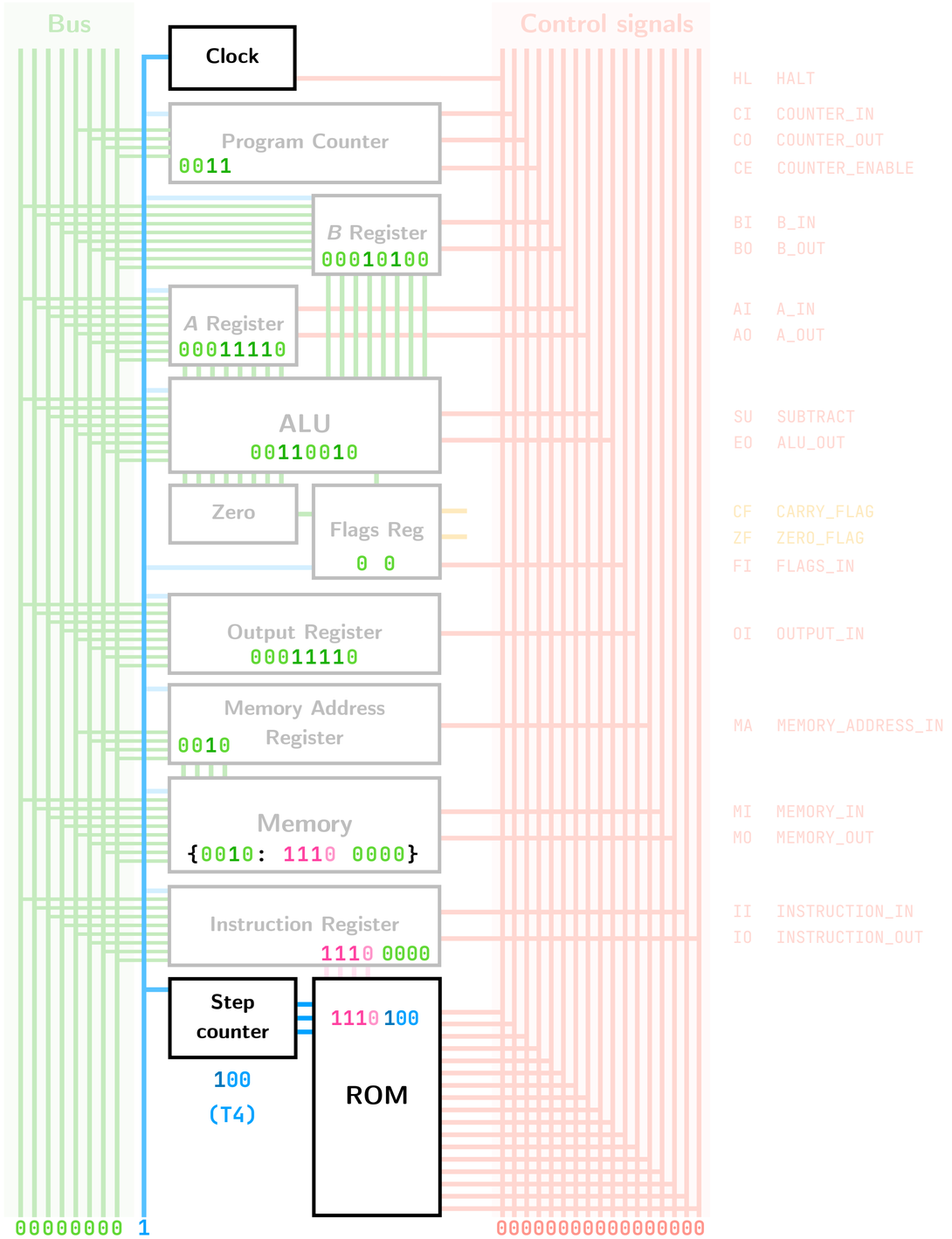
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

OUT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
    O_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMIILIOEIOIOUOIIAIOIO
	****	000	00100000000010000
	****	001	00010000000000110
	****	010	00000000000000001
NOP	0000	011	00000000000000000
NOP	0000	100	00000000000000000
NOP	0000	101	00000000000000000
LOAD	0001	011	00000000000010000
LOAD	0001	100	00000100000000100
LOAD	0001	101	00000000000000000
ADD	0010	011	00000000000010000
ADD	0010	100	00001000000000100
ADD	0010	101	00000010011000000
	1110	011	00000001000100000
OUT	1110	100	00000000000000000
	1110	101	00000000000000000
HALT	1111	011	10000000000000000
HALT	1111	100	00000000000000000
HALT	1111	101	00000000000000000



Program

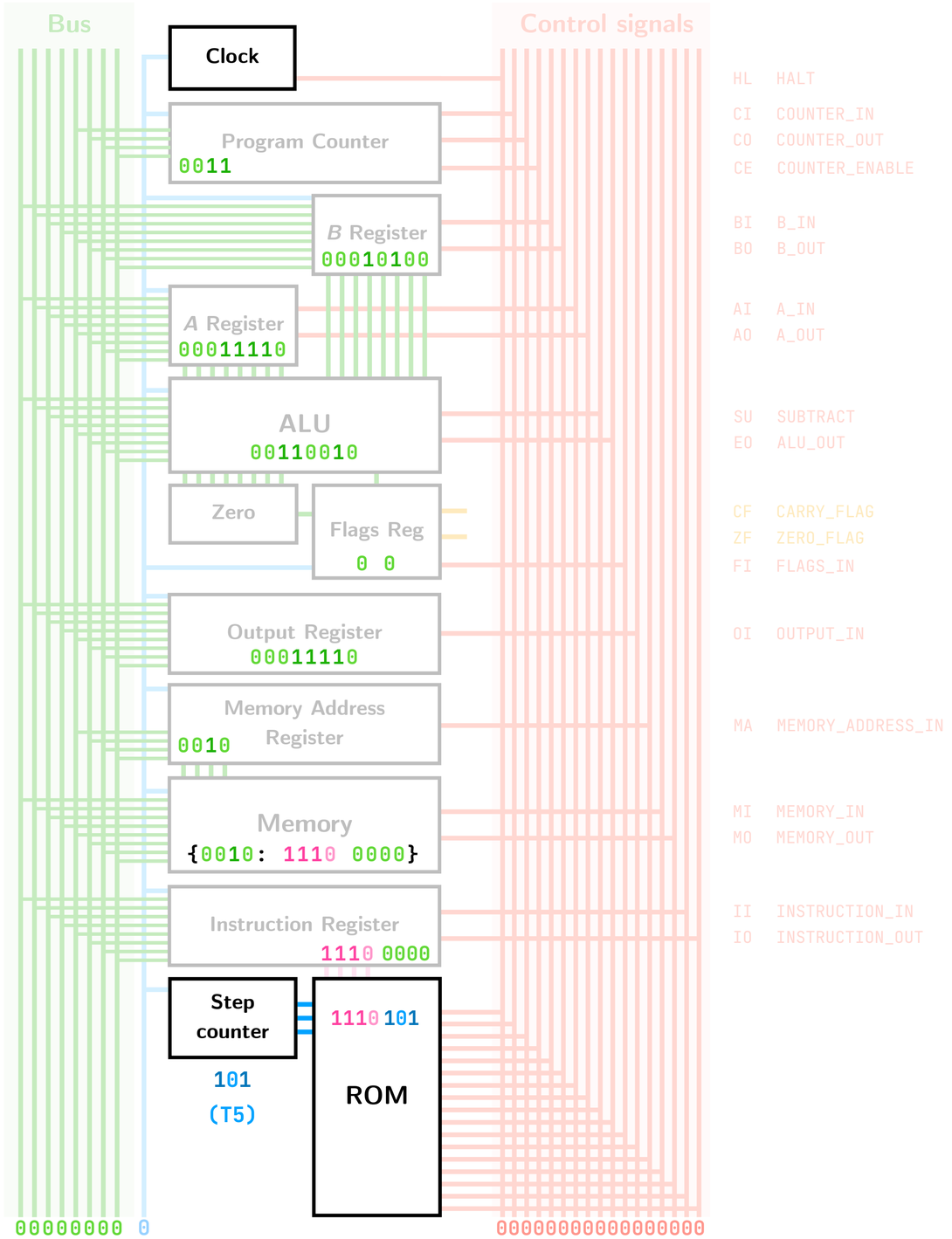
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

OUT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
    O_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
LOAD	0001	100	0000010000000100
LOAD	0001	101	0000000000000000
ADD	0010	011	0000000000010000
ADD	0010	100	0000100000000100
ADD	0010	101	0000010011000000
OUT	1110	011	0000001000100000
OUT	1110	100	0000000000000000
OUT	1110	101	0000000000000000
HALT	1111	011	1000000000000000
HALT	1111	100	0000000000000000
HALT	1111	101	0000000000000000



Program

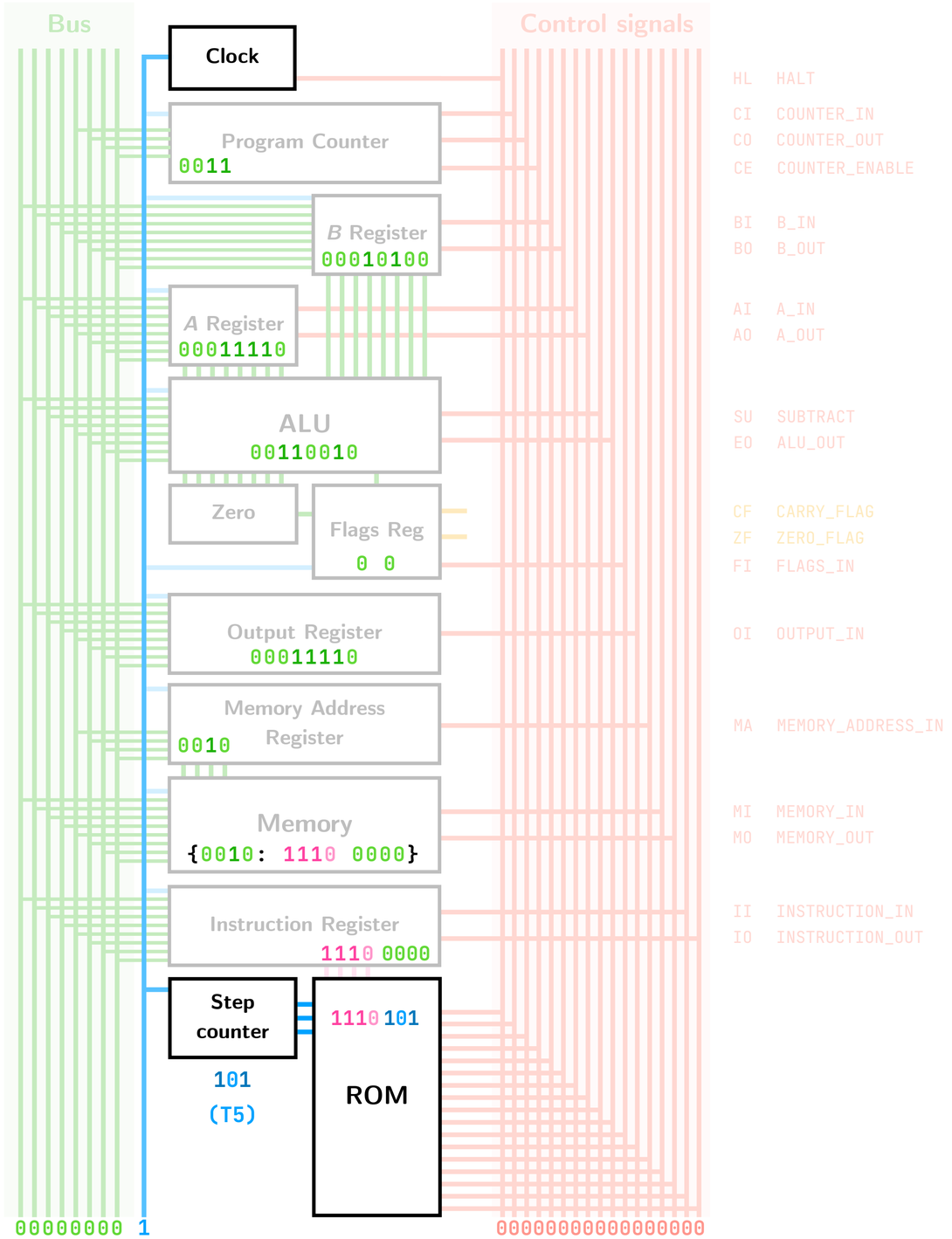
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

OUT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
    O_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001 0000
	****	001	0001000000000 110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
NOP	0000	100	0000000000000000
NOP	0000	101	0000000000000000
LOAD	0001	011	0000000000001 0000
LOAD	0001	100	000000100000000 100
LOAD	0001	101	0000000000000000
ADD	0010	011	0000000000001 0000
ADD	0010	100	0000100000000 100
ADD	0010	101	000000100110000 0
OUT	1110	011	0000000100010 0000
OUT	1110	100	0000000000000000
OUT	1110	101	0000000000000000
HALT	1111	011	1000000000000000
HALT	1111	100	0000000000000000
HALT	1111	101	0000000000000000



Program

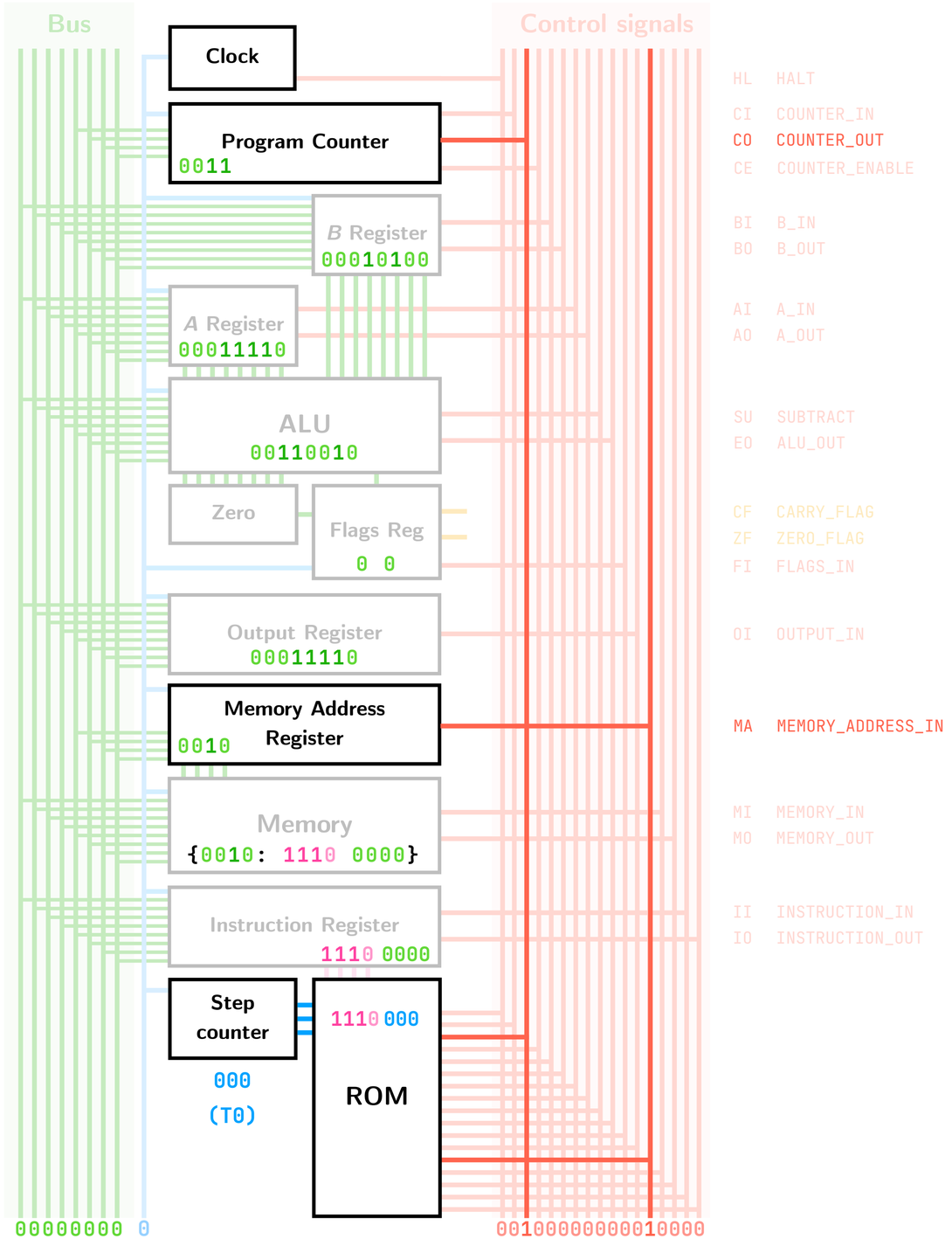
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

OUT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] A_OUT
    O_IN
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000001000100000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
BI B_IN
BO B_OUT
AI A_IN
AO A_OUT
SU SUBTRACT
EO ALU_OUT
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
OI OUTPUT_IN
MA MEMORY_ADDRESS_IN
MI MEMORY_IN
MO MEMORY_OUT
II INSTRUCTION_IN
IO INSTRUCTION_OUT

Program

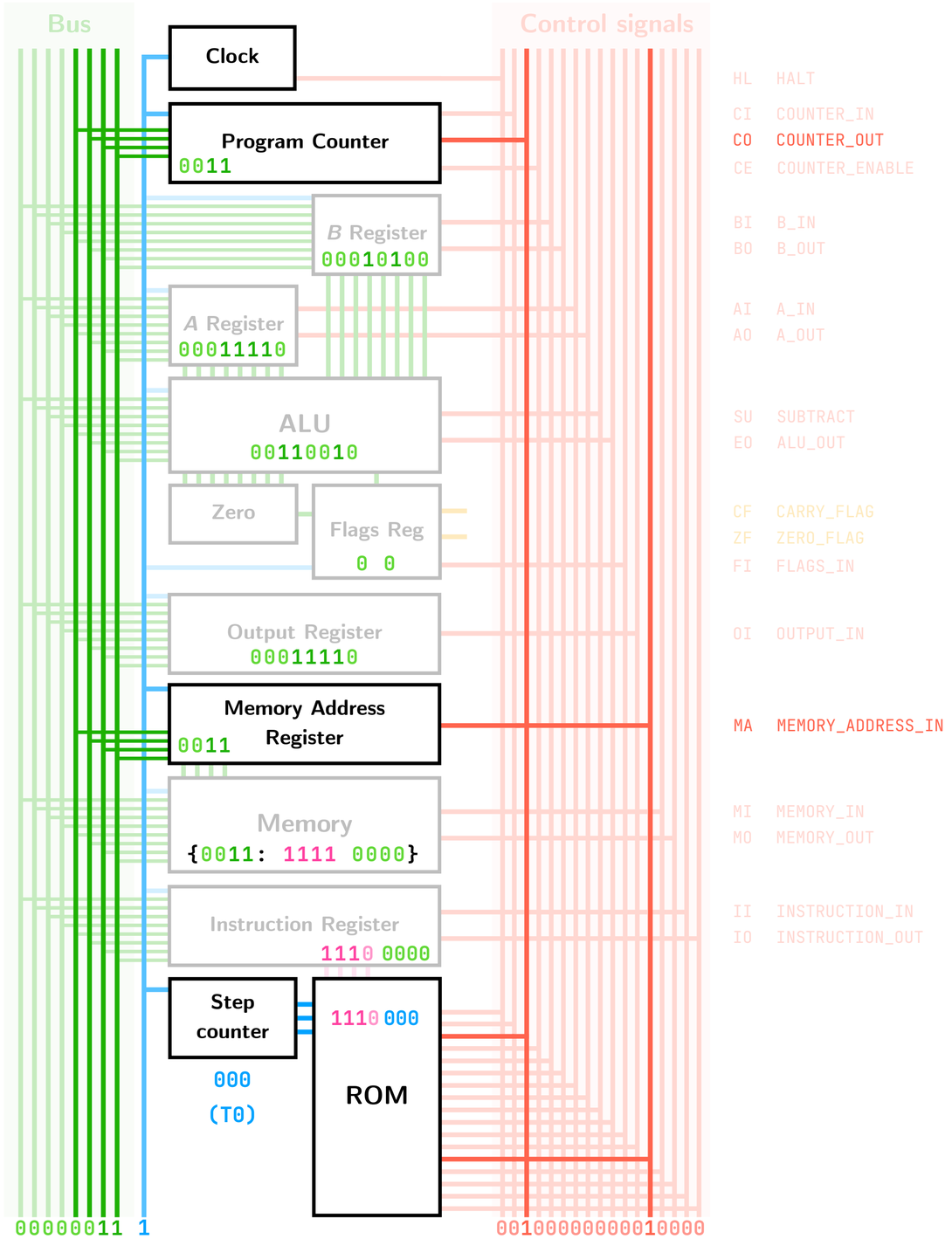
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

HALT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] HALT
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000001000100000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE

BI B_IN
BO B_OUT

AI A_IN
AO A_OUT

SU SUBTRACT
EO ALU_OUT

CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN

OI OUTPUT_IN

MA MEMORY_ADDRESS_IN

MI MEMORY_IN
MO MEMORY_OUT

II INSTRUCTION_IN
IO INSTRUCTION_OUT

Program

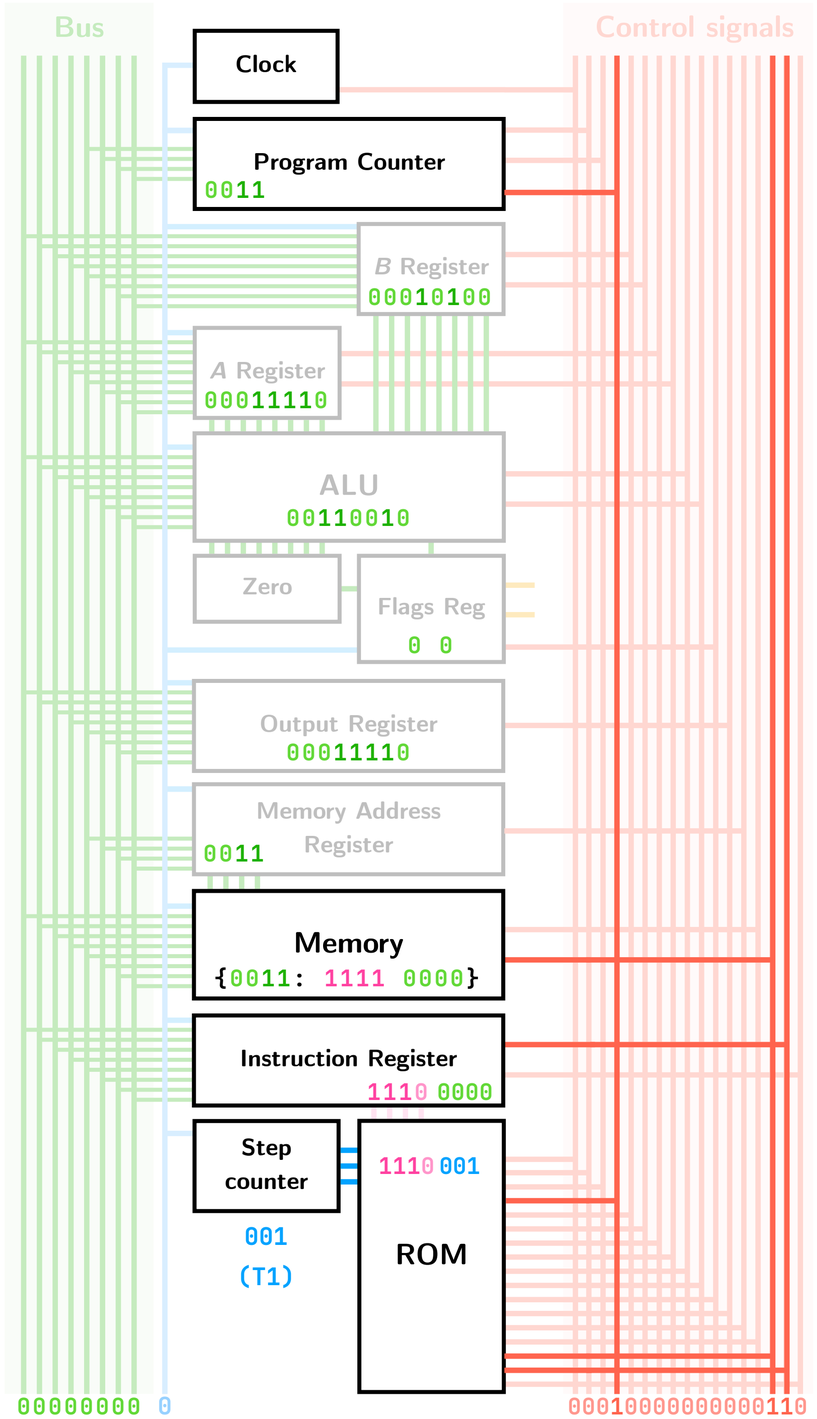
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

HALT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] HALT
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



HL HALT
CI COUNTER_IN
CO COUNTER_OUT
CE COUNTER_ENABLE
BI B_IN
BO B_OUT
AI A_IN
AO A_OUT
SU SUBTRACT
EO ALU_OUT
CF CARRY_FLAG
ZF ZERO_FLAG
FI FLAGS_IN
OI OUTPUT_IN
MA MEMORY_ADDRESS_IN
MI MEMORY_IN
MO MEMORY_OUT
II INSTRUCTION_IN
IO INSTRUCTION_OUT

Program

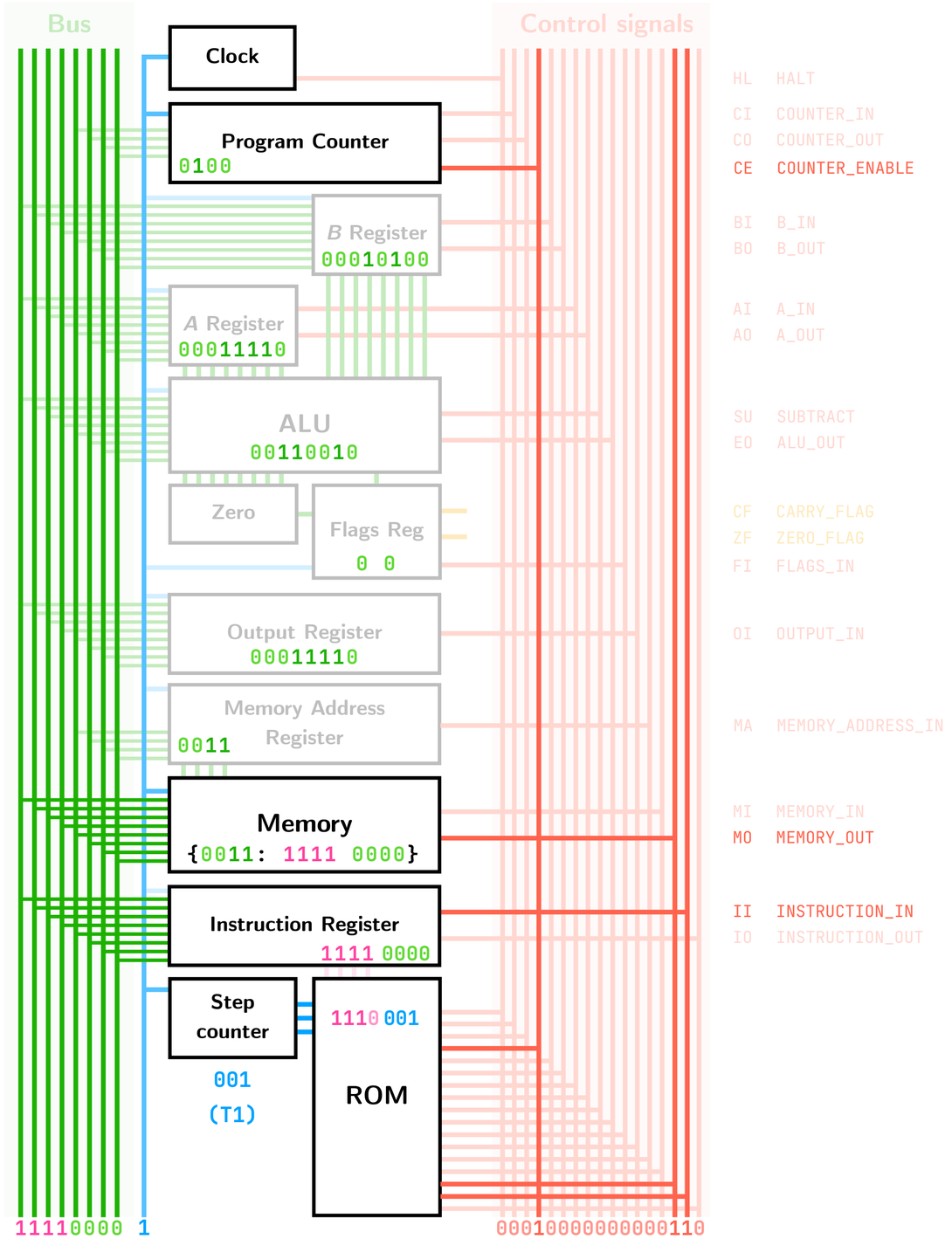
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

HALT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] HALT
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

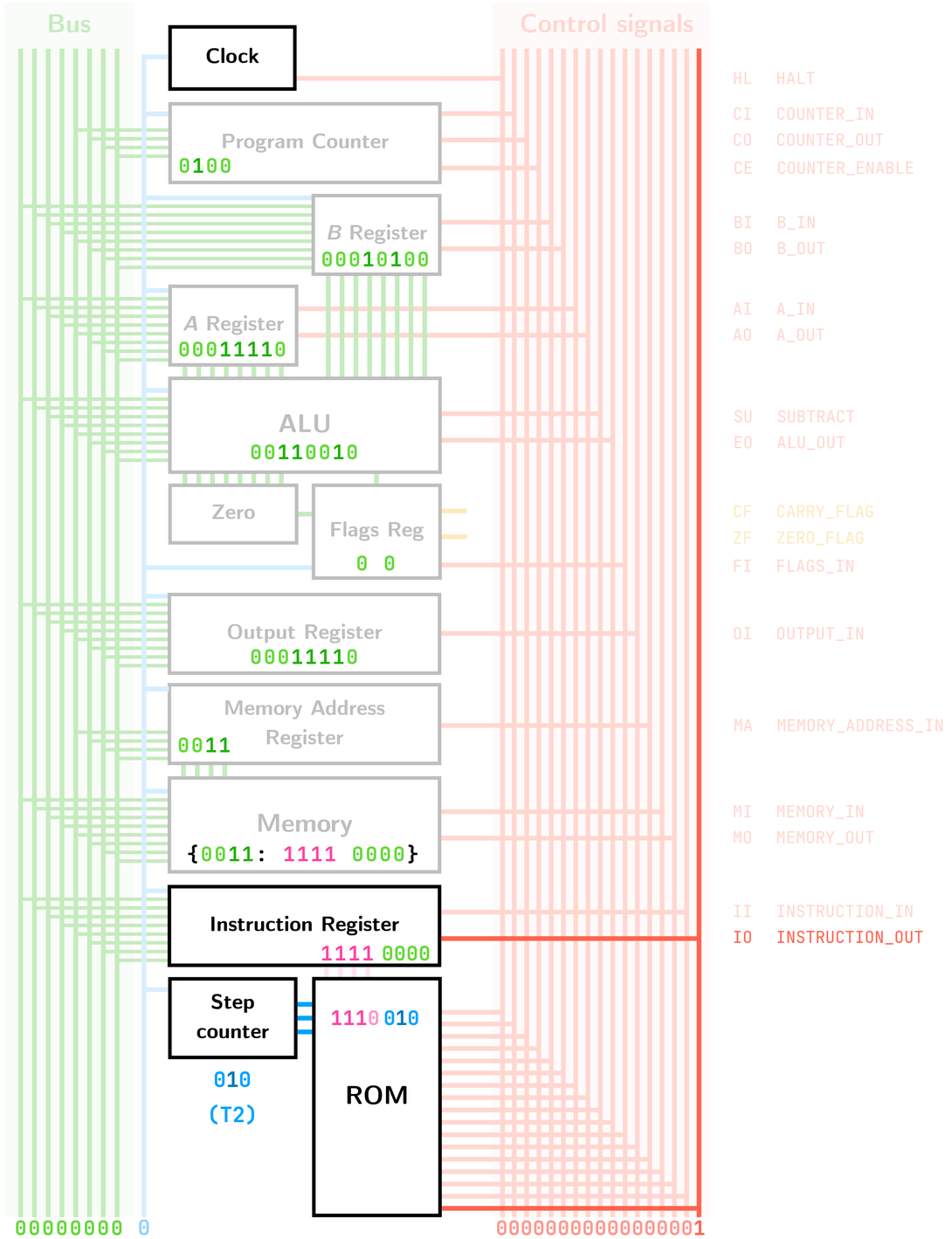
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

HALT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] HALT
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMIILIOEIOIOUOIIAIOIO
	****	000	0010000000001000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000001000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

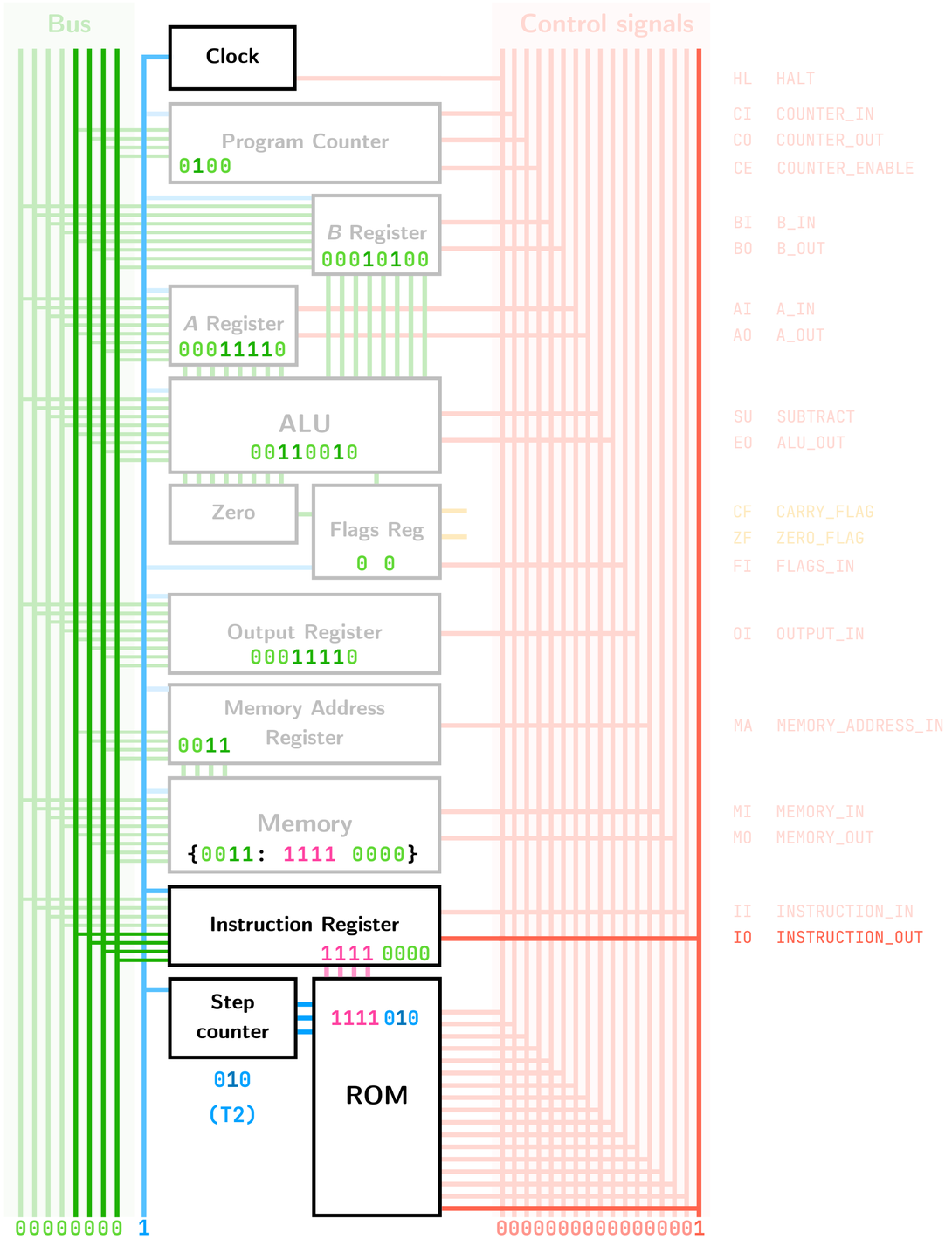
```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

HALT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] HALT
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000001000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

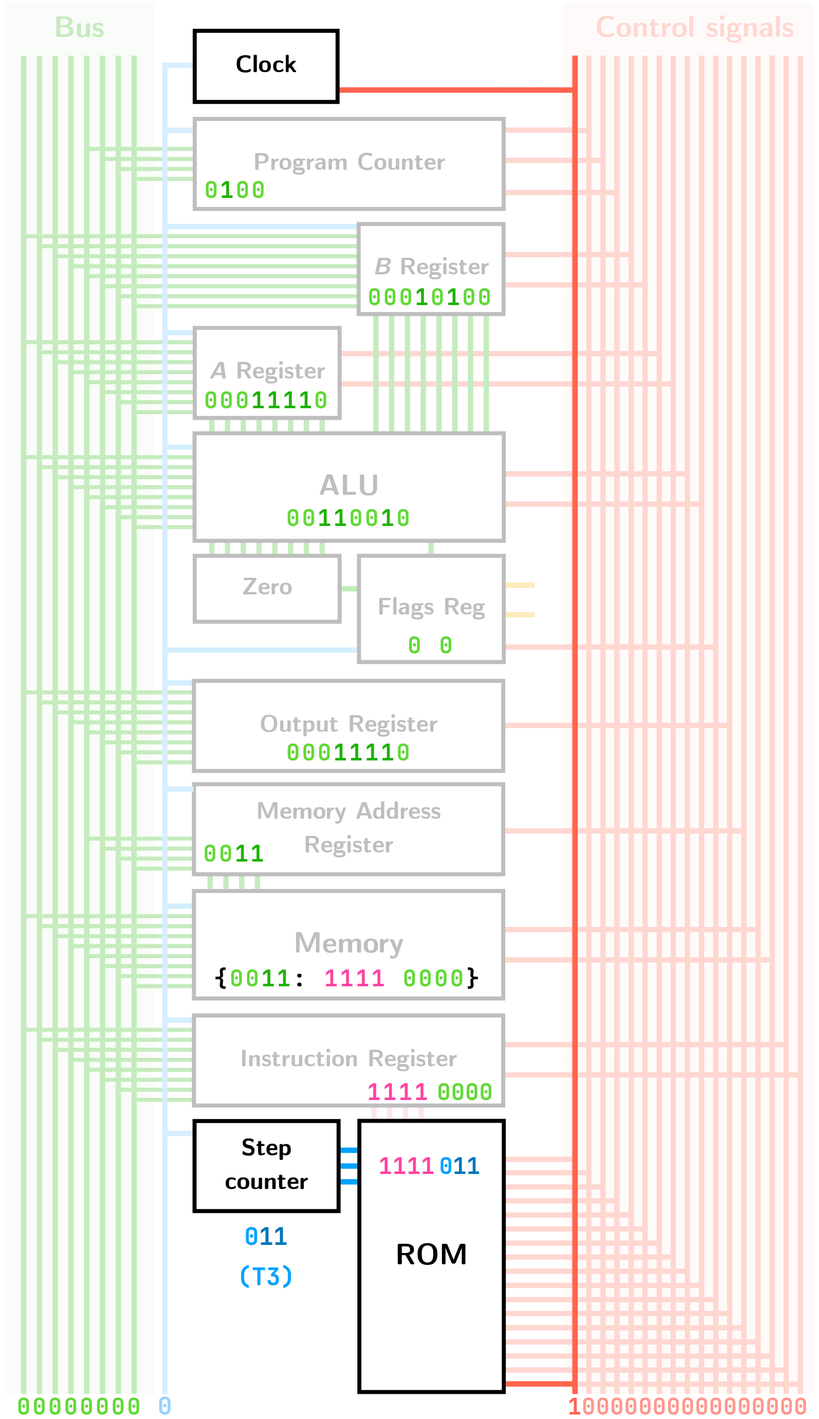
HALT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] HALT
```

IR can only write last 4 bits to bus →

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000001000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000001000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000001000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000000100010000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000



HL	HALT
CI	COUNTER_IN
CO	COUNTER_OUT
CE	COUNTER_ENABLE
BI	B_IN
BO	B_OUT
AI	A_IN
AO	A_OUT
SU	SUBTRACT
EO	ALU_OUT
CF	CARRY_FLAG
ZF	ZERO_FLAG
FI	FLAGS_IN
OI	OUTPUT_IN
MA	MEMORY_ADDRESS_IN
MI	MEMORY_IN
MO	MEMORY_OUT
II	INSTRUCTION_IN
IO	INSTRUCTION_OUT

Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

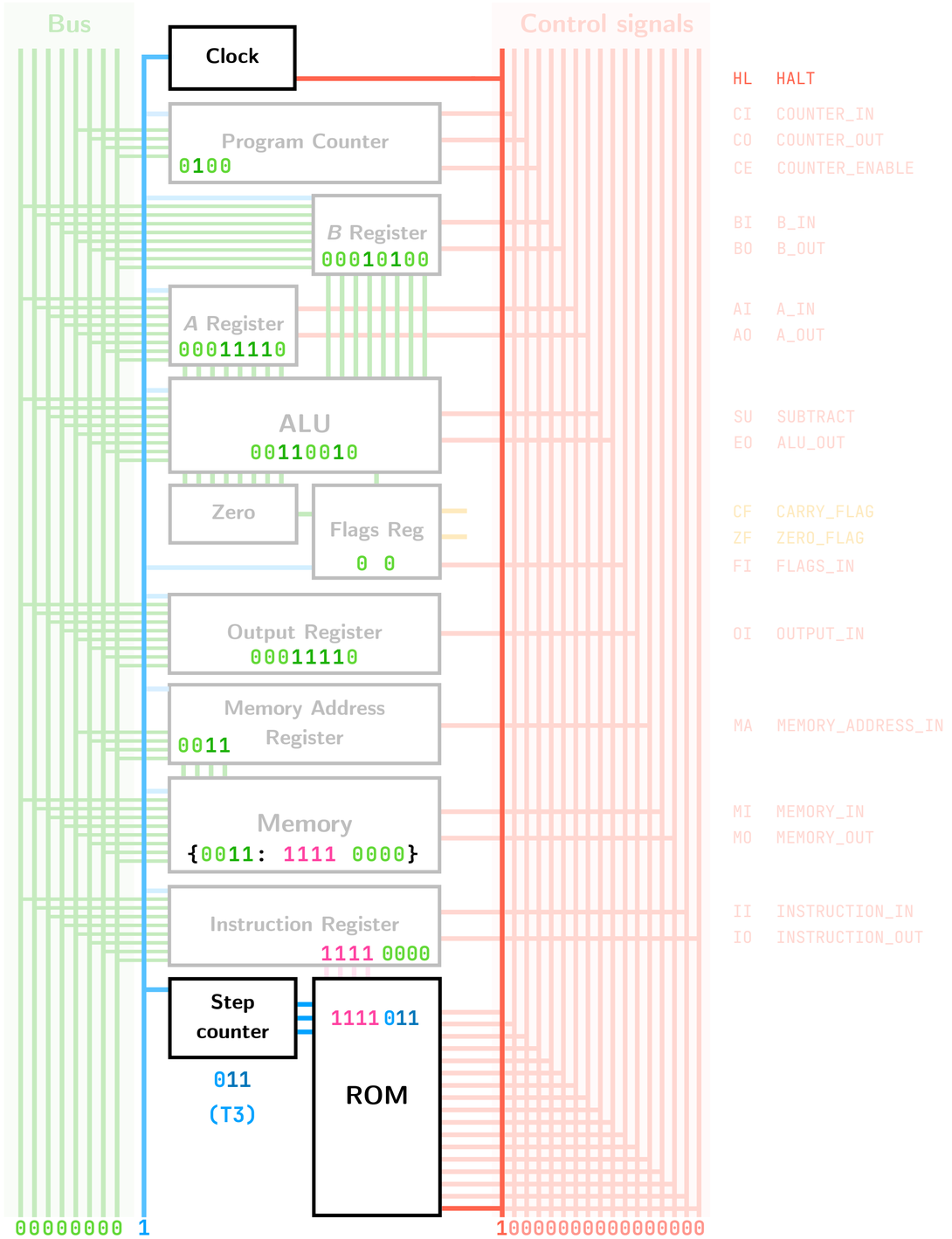
HALT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] HALT
```

ROM

	OPCODE	STEP	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	0010000000010000
	****	001	0001000000000110
	****	010	0000000000000001
NOP	0000	011	0000000000000000
	0000	100	0000000000000000
	0000	101	0000000000000000
LOAD	0001	011	0000000000010000
	0001	100	0000010000000100
	0001	101	0000000000000000
ADD	0010	011	0000000000010000
	0010	100	0000100000000100
	0010	101	0000010011000000
OUT	1110	011	0000001000100000
	1110	100	0000000000000000
	1110	101	0000000000000000
HALT	1111	011	1000000000000000
	1111	100	0000000000000000
	1111	101	0000000000000000

Next steps won't happen
because there is no clock signal
unless the clock is unhalted



Program

```
[0] LOAD  14
[1] ADD   15
[2] OUT
[3] HALT


[14] 10
[15] 20
```

HALT

```
[0] COUNTER_OUT
    MEMORY_ADDRESS_IN
[1] MEMORY_OUT
    INSTRUCTION_IN
    COUNTER_ENABLE
[2] INSTRUCTION_OUT
[3] HALT
```

Control logic

Turing-completeness, conditional jumps and call/return
(SAP-2 and SAP-3)

Conditional jumps

Input to ROM will now be not just `[opcode, step]` but `[flags, opcode, step]`

so that the control word of a particular instruction at a given step also depends on the flags (e.g. `CARRY_FLAG`, `ZERO_FLAG`)

most control words will look the same, no matter the flags

but, importantly, not our new instructions:

- `JZ` (jump-if-zero)
- `JC` (jump-if-carry)

you can also add more flags to the control word, e.g. `OVERFLOW_FLAG`, `NEGATIVE_FLAG`, `PARITY_FLAG`,

and define more conditional jump instructions, e.g. other common ones are:

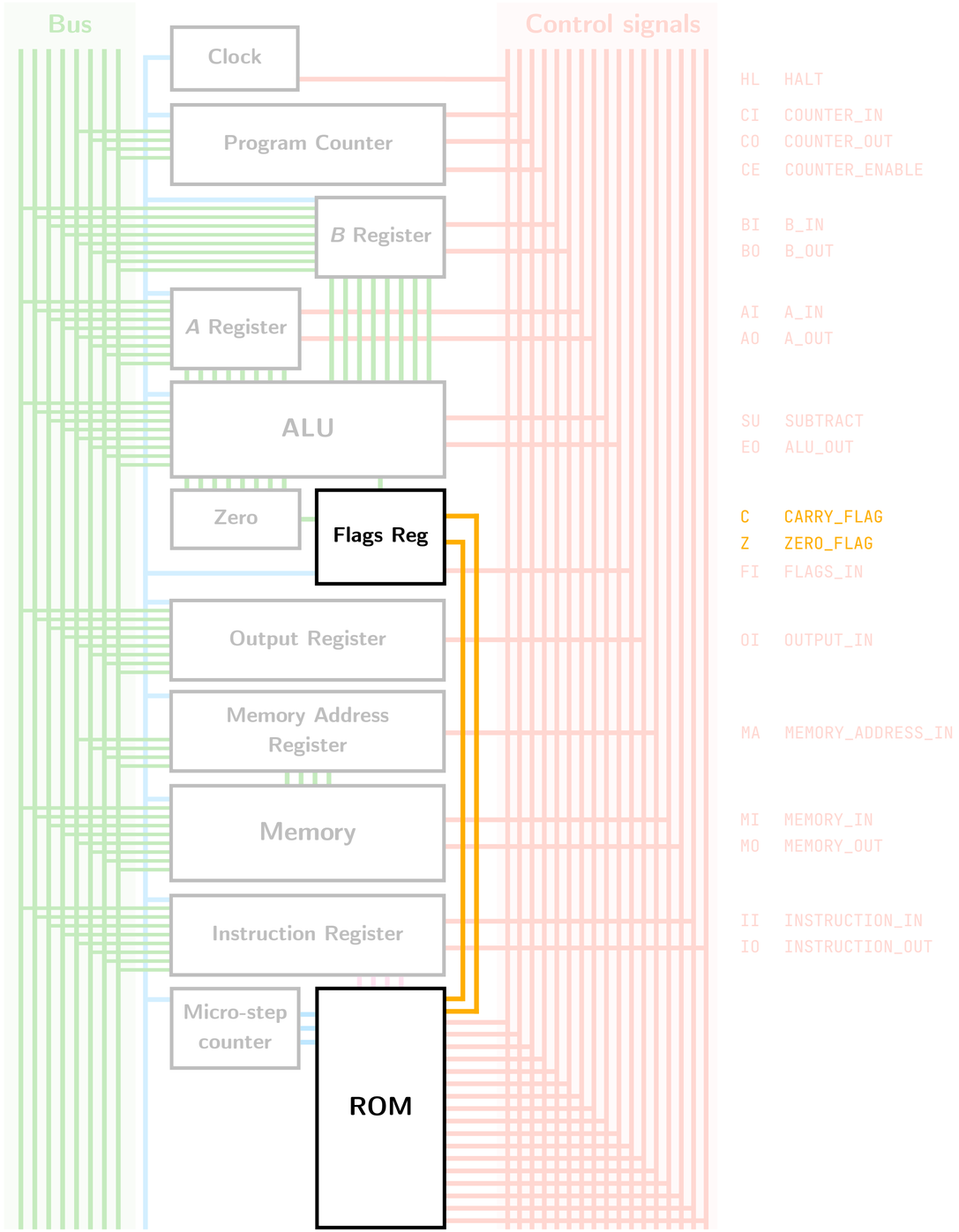
- `JNZ` (jump-if-not-zero)
- `JNC` (jump-if-no-carry)
- `JE` (jump-if-equal)
- `JNE` (jump-if-not-equal)
- `JG` (jump-if-greater)
- `JGE` (jump-if-greater-or-equal)
- `JL` (jump-if-less)
- `JLE` (jump-if-less-or-equal)
- `JG` (jump-if-parity)

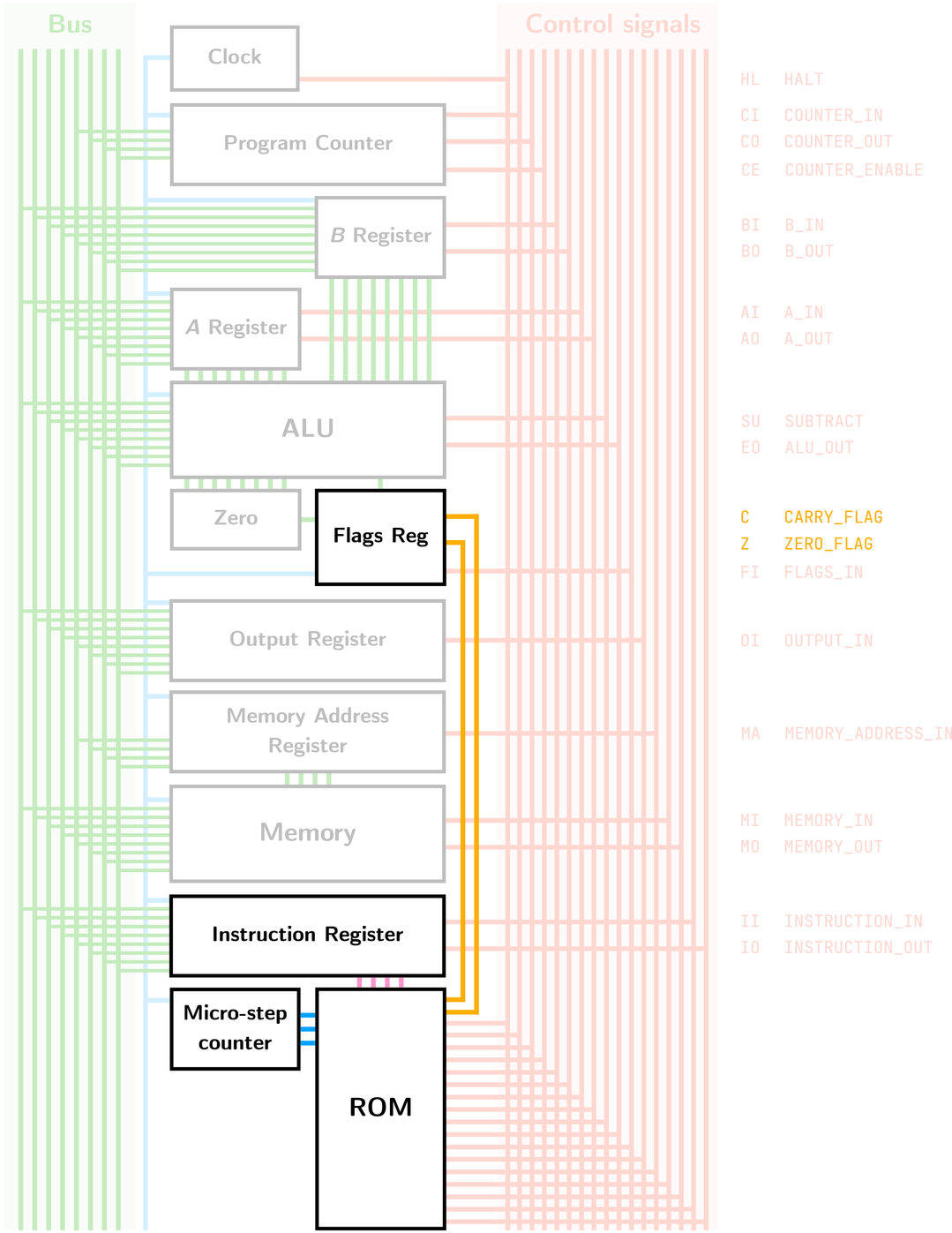
conditional jumps are often used right after a `CMP` (comparison) instruction, in order to set the flag value based on the result of the comparison

the `CMP` is simply a subtraction without storing the result anywhere, but crucially—like an ordinary subtraction—setting the flags

the result of a subtraction tells us if the first argument is less than, equal to or greater than the second argument

Flags are now connected to ROM!





Flags are now connected to ROM!

`[flags, opcode, step] → control_word`

Control logic

!!

Mnemonic	Opcode				Step			Flags		Control word																	
								CARRY_FLAG	ZERO_FLAG	HALT	COUNTER_IN	COUNTER_OUT	COUNTER_ENABLE	B_IN	B_OUT	A_IN	A_OUT	SUBTRACT	ALU_OUT	FLAGS_IN	OUTPUT_IN	MEMORY_ADDRESS_IN	MEMORY_IN	MEMORY_OUT	INSTRUCTION_IN	INSTRUCTION_OUT	
Fetch	*	*	*	*	0	0	0	*	*	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
	*	*	*	*	0	0	1	*	*	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	
	*	*	*	*	0	1	0	*	*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
LOAD	0	0	0	1	0	1	1	*	*	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	
	0	0	0	1	1	0	0	*	*	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	
	0	0	0	1	1	0	1	*	*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
JMP	0	1	1	0	0	1	1	*	*	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
	0	1	1	0	1	0	0	*	*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	0	1	1	0	1	0	1	*	*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
JC	0	1	1	1	0	1	1	0	*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	!!
	0	1	1	1	0	1	1	1	*	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	!!
	0	1	1	1	1	0	0	*	*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	0	1	1	1	1	0	1	*	*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

if/else using a conditional jump

E.g. if the number at memory address 15 is non-negative,
do something (`code1`),
otherwise, do something else (`code2`),
afterwards, continue with the rest of the program (`code3`)

.asm

```
[0]  load  15
[1]  cmp   0
[2]  jc    5
[3]  <code1>
[4]  jmp   6
[5]  <code2>
[6]  <code3>

[15] 5
```

} check condition and set flags

} jump to else block if carry flag

} jump outside of if/else block

Pseudocode

```
[15] ← 5
A ← [15]
if CF == 0 then
    ; true if A >= 0
    <code1>
else
    ; true if A < 0
    <code2>
end if
<code3>
```

Pseudocode

```
let x ← 5
if x >= 0 then
    <code1>
else
    <code2>
end if
<code3>
```

do...while using a conditional jump

E.g. Integer multiplication using addition, subtraction and a conditional jump
We simply use the first number as a counter,
and then add the second number to itself and subtract 1 from the counter
repeatedly until the counter hits 0.

.asm

```
        ; start loop
  [0]   load  13
  [1]   add   15
  [2]   store 13
  [3]   load  14
  [4]   sub   1
  [5]   store 14
  [6]   jz    8         ; jump out of loop
  [7]   jmp   0         ; jump to start
        ; end loop
  [8]   <more code>

 [13]   0               ; intermediate result
 [14]   5               ; mul1 (counter)
 [15]   20              ; mul2
```

Pseudocode

```
do
    ; add mul2 to intermed.
    ;  result
    A ← [13]
    A ← A + [15]
    [13] ← A
    ; subtract counter:
    A ← [14]
    A ← A - 1
    [14] ← A
while ZF != 0
<more code>
```

Pseudocode

```
let result ← 0
let counter ← 5
do
    result ← result + x
    counter ← counter - 1
while counter != 0
<more code>
```

do...while using a conditional jump

General pattern

.asm

```
        ; start loop
[0]     <code>
        ; by this point, flag must
        ;  reflect the desired condition
[1]     jz      3       ; jump out of loop
[2]     jmp     0       ; jump to start
        ; end loop
[3]     <more code>
```

Pseudocode

```
do
    <code>
while <flag condition>
<more code>
```

Pseudocode

```
do
    <code>
while <condition>
<more code>
```

while using a conditional jump

General pattern

.asm

```
[0]  <code before loop>
     ; by this point, flag must
     ;  reflect the desired condition
     ; start loop
[1]  jz    4        ; jump out of loop
[2]  <code>
     ; by this point, flag must
     ;  reflect the desired condition
[3]  jmp   1        ; jump to start
     ; end loop
[4]  <more code>
```

Pseudocode

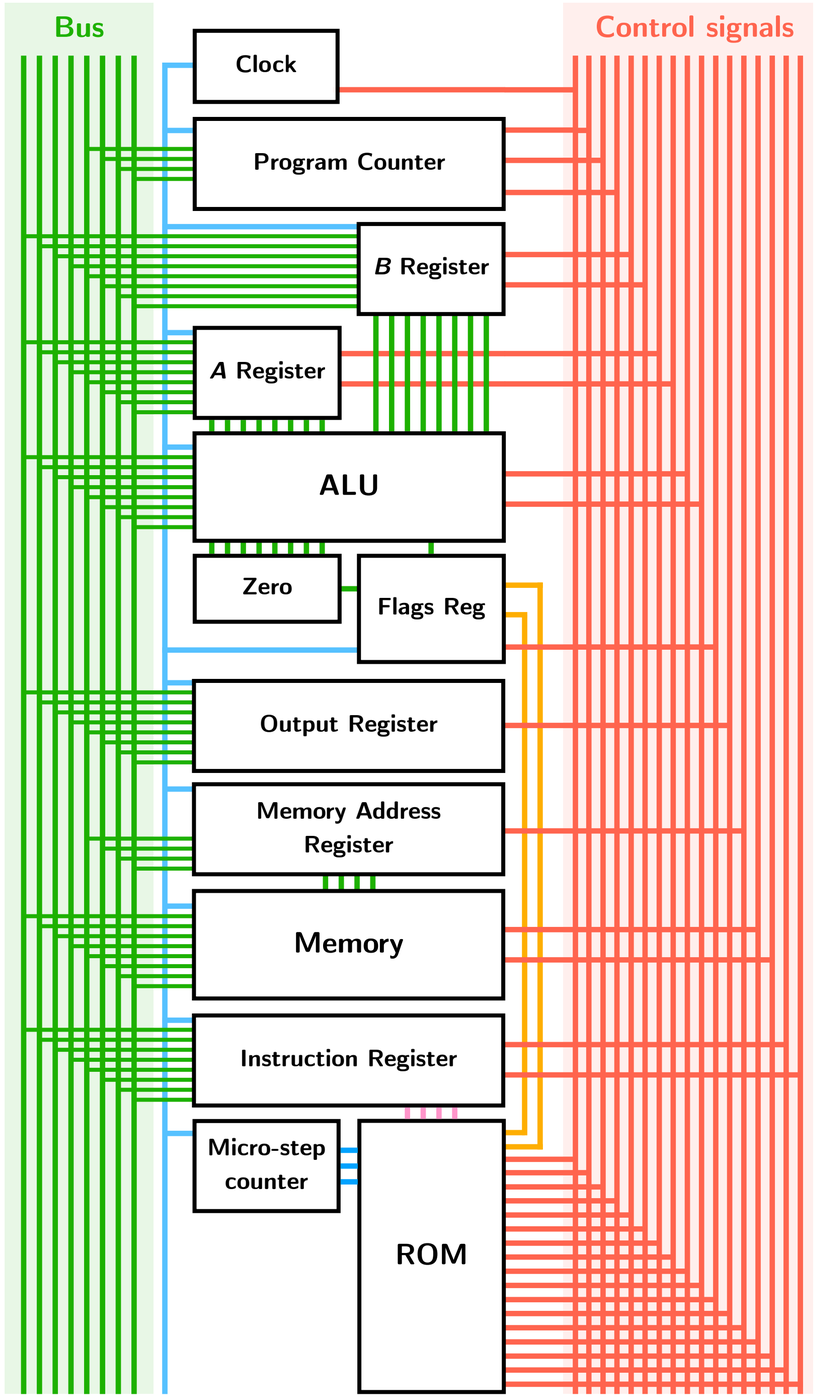
```
<code before loop>
while <flag condition>
    <code>
end while
<more code>
```

Pseudocode

```
<code before loop>
while <condition>
    <code>
end while
<more code>
```

ROM

	OPCODE	STEP	CF FF	HCCCBBAASEFOMMMII LIOEIOIOUOIIAIOIO
	****	000	**	0010000000001**0**000
	****	001	**	0001000000000110
	****	010	**	0000000000000001
LOAD	0001	011	**	000000000000**1**0000
	0001	100	**	000000**1**0000000100
	0001	101	**	0000000000000000
ADD	0010	011	**	000000000000**1**0000
	0010	100	**	0000**1**000000000100
	0010	101	**	000000**1**00**11**000000
STORE	0100	011	**	00000000000**1**0001
	0100	100	**	0000000**1**000001000
	0100	101	**	0000000000000000
OUT	1110	011	**	0000000**1**000**1**00000
	1110	100	**	0000000000000000
	1110	101	**	0000000000000000
HALT	1111	011	**	**1**000000000000000
	1111	100	**	0000000000000000
	1111	101	**	0000000000000000
JMP	0110	011	**	0**1**00000000000001
	0110	100	**	0000000000000000
	0110	101	**	0000000000000000
JC	0111	011	0*	0000000000000000
	0111	011	1*	0**1**00000000000001
	0111	100	**	0000000000000000
	0111	101	**	0000000000000000



Program

```
[0]  LOAD   13
[1]  ADD    15
[2]  STORE  13
[3]  LOAD   14
[4]  SUB    1
[5]  STORE  14
[6]  JZ     8
[7]  JMP    0
[8]  OUT    4
[9]  HALT

[13] 0
[14] 5
[15] 20
```

Assembly

"Assembly" is not a single programming language like C or Python

Each CPU has its own specific assembly language corresponding to how that CPU works

and the corresponding assembler assembles that assembly code into the machine code for that particular CPU

*.asm

```
[0]  loadi  3
[1]  store  15
[2]  loadi  0
[3]  addi   15
[4]  out
[5]  jmp    3
```

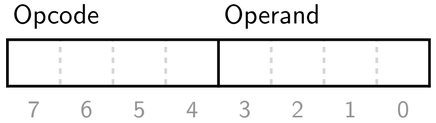
Assembler

*

```
0101  0011
0110  1111
0101  0000
0010  1111
1110  0000
0110  0011
```

Executable!

Our instruction format:



Assembly

There are many more ways to make things easier for us by writing a better assembler:

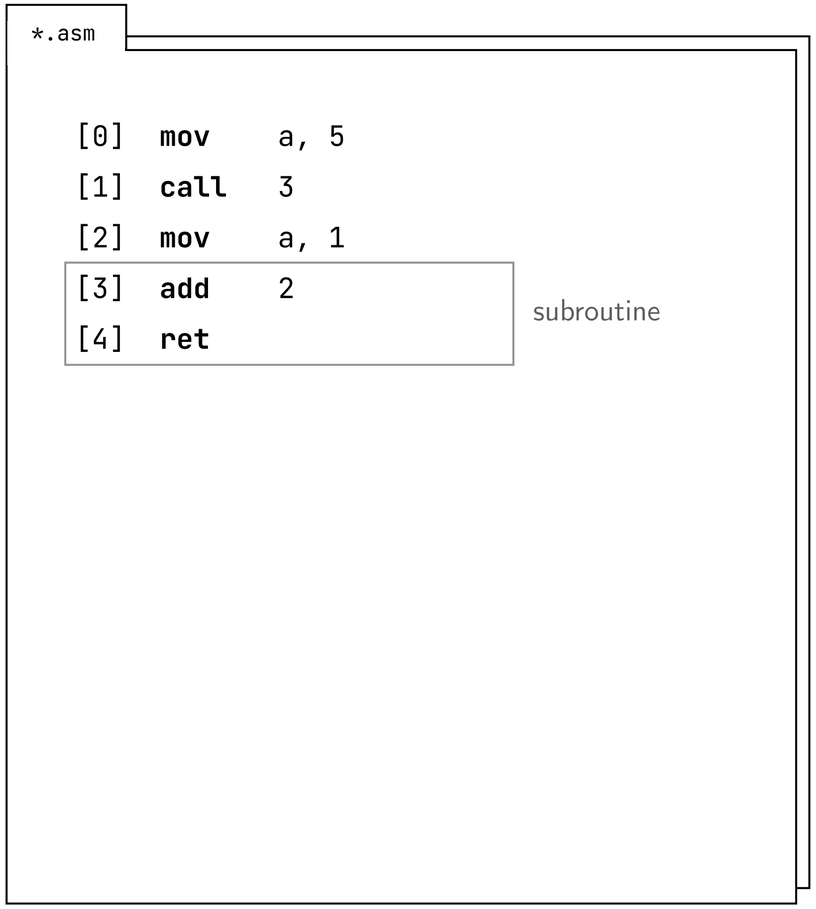
- **Comments:** they make our assembly scripts more understandable, and can simply be ignored by the assembler, and so it doesn't affect the executable binary at all.
- **Labels:** instead of using specific memory addresses in our scripts, we can use labels for positions in our code, and then have the assembler figure out appropriate memory addresses.

Assembly

Let's quickly look at **labels**.

We now longer care about specific addresses in memory.

We simply give a line in our code a name ("label") and then we use the label to, say, jump to whatever address corresponds to it, or to get data from memory.



Assembly

Let's quickly look at **labels**.

We now longer care about specific addresses in memory.

We simply give a line in our code a name ("label") and then we use the label to, say, jump to whatever address corresponds to it, or to get data from memory.

*.asm

```
[0]  mov   a, 5
[1]  call  3          wrong address!
[2]  call  3          wrong address!
[3]  mov   a, 1
[4]  add   2          subroutine
[5]  ret
```

Assembly

Let's quickly look at **labels**.

We now longer care about specific addresses in memory.

We simply give a line in our code a name ("label") and then we use the label to, say, jump to whatever address corresponds to it, or to get data from memory.

*.asm

```
[0]  mov   a, 5
[1]  call  4          fix manually!
[2]  call  4          fix manually!
[3]  mov   a, 1
[4]  add   2          subroutine
[5]  ret
```

Assembly

Let's quickly look at **labels**.

We now longer care about specific addresses in memory.

We simply give a line in our code a name ("label") and then we use the label to, say, jump to whatever address corresponds to it, or to get data from memory.