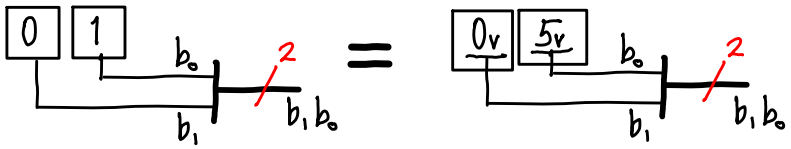
NAME _____

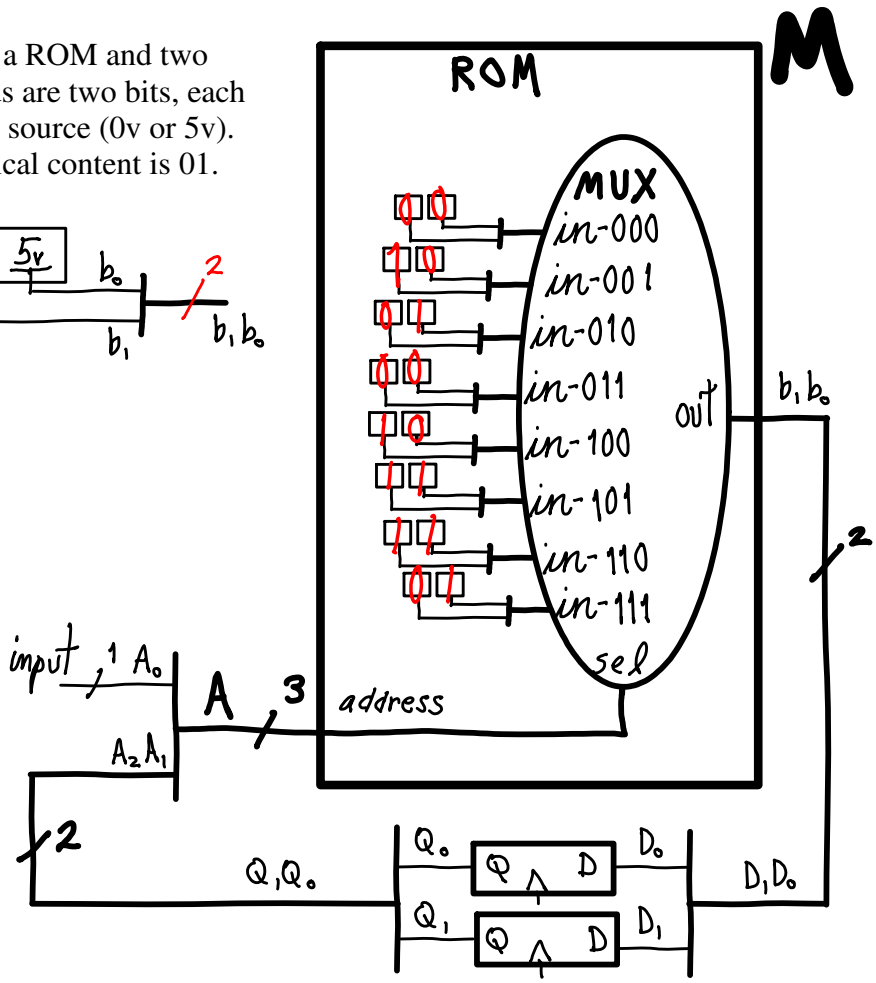
At right is an FSM, M, implemented as a ROM and two flipflops. M has no outputs. ROM words are two bits, each implemented as a connection to voltage source (0v or 5v). E.g., shown below is a word whose logical content is 01.



Bits are connected low-bit to low-bit, and so on in order. E.g., For any ROM word [b_1b_0],

b_0 connects to D_0;
b_1 connects to D_1. Also,
input connects to A_0;
Q_0 connects to A_1;
Q_1 connects to A_2.

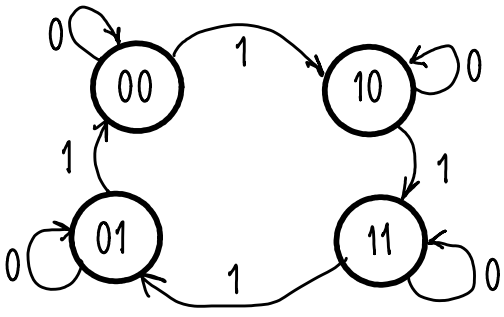
Addresses are connected so that, e.g.,
in-110 has address
A[2]A[1]A[0] = 110
Thus, A[0] = 0, is the low address bit and is coming from the *input* = 0.



Q. How many different FSMs could be implemented using this hardware? That is, suppose a designer makes a table of the ROM's content for an implementer (an 8-row table of two-bit words), who then connects each memory bit to its proper voltage. How many different tables is it possible for the designer to come up with for this ROM? Would each such table implement necessarily a different machine?

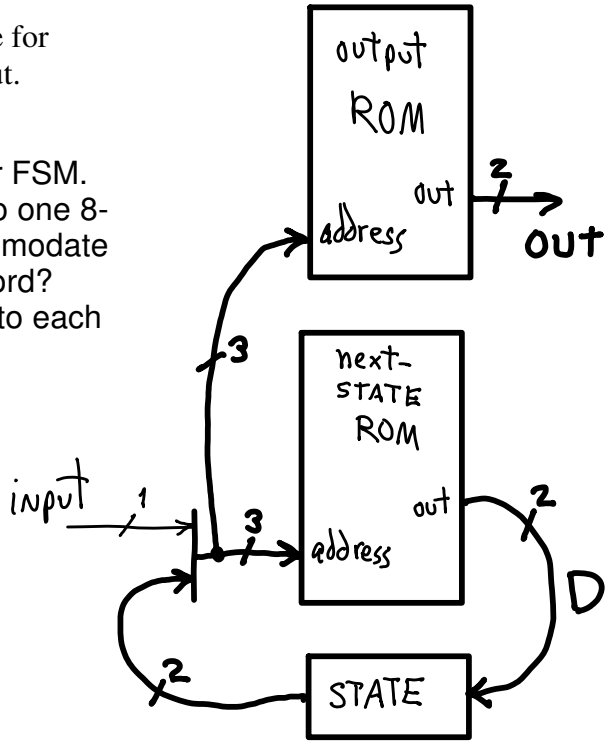
There are 16 bits in the ROM, and each can be 0 or 1. So, there are $2^{16} = 2^6 \cdot 2^{10} = 64k$ different possible tables. Each defines a unique machine.

Q. Implement such a table for the FSM, M, shown at right. That is, fill in the logical content (using 0 and 1) of the ROM above so that it implements M. Each state is labeled with its state encoding.



STATE	IN	
00	0	00
00	1	10
10	0	10
10	1	11
11	0	11
11	1	01
01	0	01
01	1	00

Shown at right is an expansion of the hardware above to provide for implementation of general FSMs. It adds a mechanism for output.



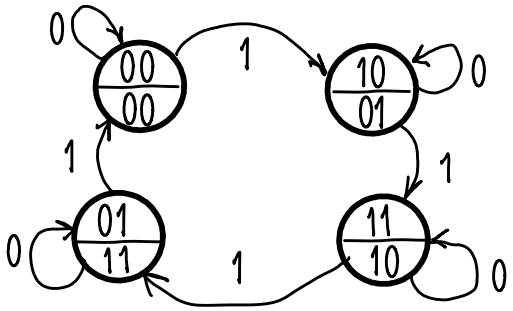
Q. We now need two 8-row tables to specify a particular FSM. Suppose the designer wanted to combine the tables into one 8-row table. How would the ROM be redesigned to accommodate this? How many bits would be needed for each ROM word? Show the assignment of the bits of a typical ROM word to each of OUT[1], OUT[0], D[1], and D[0].

Each word has 4 bits $[b_3 b_2 b_0 b_1]$

$b_0 \rightarrow D_0$ $\quad b_2 \rightarrow OUT_0$

$b_1 \rightarrow D_1$ $\quad b_3 \rightarrow OUT_1$

Q. At right is a Moore FSM (output depends only on state). Give a single table for its implementation ROM.



address	$b_1\ b_0$	$b_3\ b_2$	
000	00	00	} same output for state 00
001	10	00	
010	10	11	
011	11	11	
100	11	01	
101	01	01	
110	01	10	
111	00	10	

next-state ↑ ↖ output

words are layed out this way: $b_3 b_2 b_1 b_0$

and connect this way:

$b_3 \rightarrow OUT[1]$

$b_2 \rightarrow OUT[0]$

$b_1 \rightarrow D_1$

$b_0 \rightarrow D_0$

Q. How could the table above be recoded into a linear form suitable for simulation by a Universal Turing Machine? (Assume bit b_0 of M's output is used to implement moving its R/W head left or right, and bit b_1 is its output symbol. At the clock tick, the R/W head moves and state changes.)

Each Row would be coded ⟨current-state, input, output, move, next-state⟩

These fields come from ROM, encoded in unary. Remaining fields are encoded address bits.