

Moore Machines

Contents

[Definition](#)

[Differences between a Moore Machine and an FA](#)

- [No Final State](#)
- [State Output](#)
- [Producing Output from an Input String](#)
- [No Nondeterminism](#)

[Output Convention](#)

[Proceed to Moore Machine Examples](#)

Definition

JFLAP defines a Moore machine M as the sextuple $M = (Q, \Sigma, \Gamma, \delta, \omega, q_s)$ where

Q is a finite set of states $\{q_i \mid i \text{ is a nonnegative integer}\}$

Σ is the finite input alphabet

Γ is the finite output alphabet

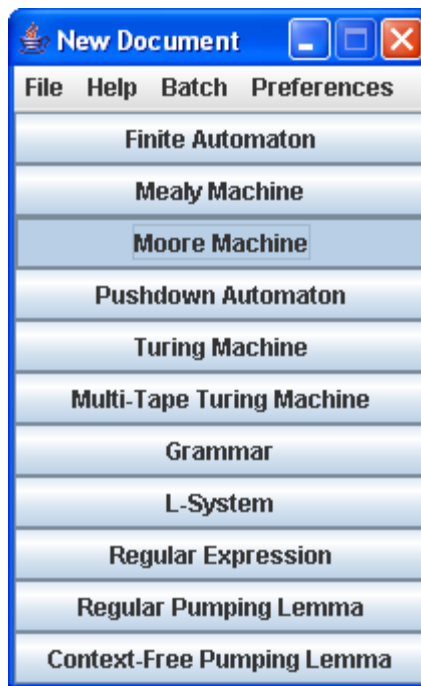
δ is the transition function, $\delta : Q \times \Sigma \rightarrow Q$

ω denotes the output function, $\omega : Q \rightarrow \Gamma$

q_s (is a member of Q) is the initial state

Moore machines are different than [Mealy machines](#) in the output function, ω . In a Moore machine, output is produced by its states, while in a Mealy machine, output is produced by its transitions.

To start a new Moore machine, select the **Moore Machine** option from the main menu.



Starting a new Moore machine


Differences between a Moore Machine and an FA

A Moore machine is very similar to a [Finite Automaton](#) (FA), with a few key differences:

- It has [no final states](#).
- Its [states produce output](#).
- It does not accept or reject input, instead, it [generates output from input](#).
- Lastly, Moore machines [cannot have nondeterministic states](#).

Let's go through these points.

No Final State

In an FA, when you have the Attribute Editor tool selected (you may do so by clicking the  button), right-clicking on a state will produce a pop-up menu that allows you to, among other things, set it to be a final state. In a Moore machine, that option is not available.



An FA state menu

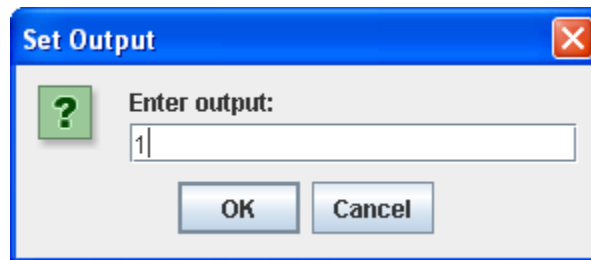
A Moore machine state menu

A Moore machine does not have final states because it does not accept or reject input. Instead, each state produces output, which will be described below.

State Output

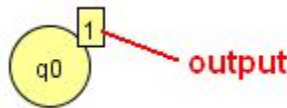
A Moore machine produces output when it is at a state.

Creating a Moore machine is the same as creating an FA with the exception of creating its states. In a Moore machine, each state produces output. When you are creating a state, a popup dialog box appears that prompts you for the output of the state.




Entering the state output

Type the state output in the dialog box and hit click **OK**. (If you click **Cancel**, the state will still be created, it will just have an empty string as its output.) When the state is created, its output will be show in its top right hand corner:



State created

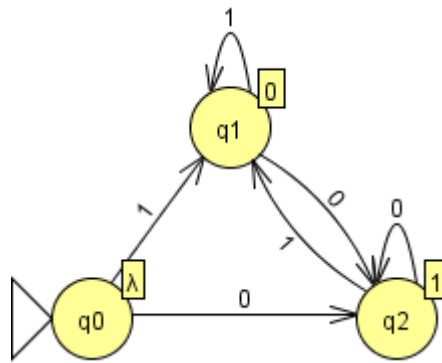
Thus, when the machine is in state q_0 , it will produce an output of "1". To change the state output, click on the state using the Arrow tool , and enter the new output in the dialog box.

With each state producing output, the Moore machine can produce output from an input string.

Producing Output from an Input String

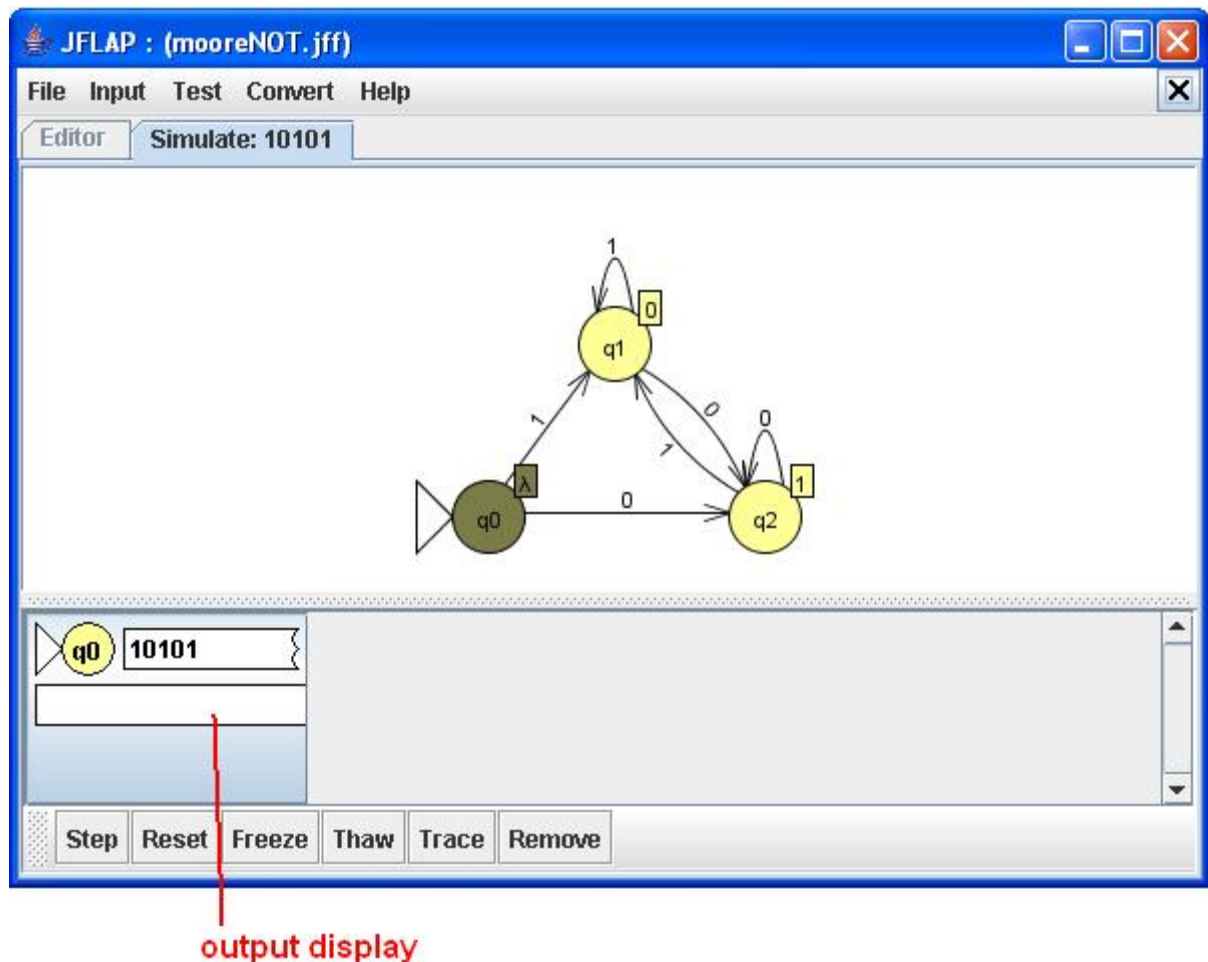
Instead of accepting or rejecting input, a Moore machine produces output from an input string.

Let's look at this simple Moore machine (which is provided in [mooreNOT.jff](#)):



A simple Moore machine

When we select the menu option **Input : Step...** and enter your input, we get a display similar to that of an FA, except with another piece of tape below the input tape. This displays the output of the machine at the current step.



The output display

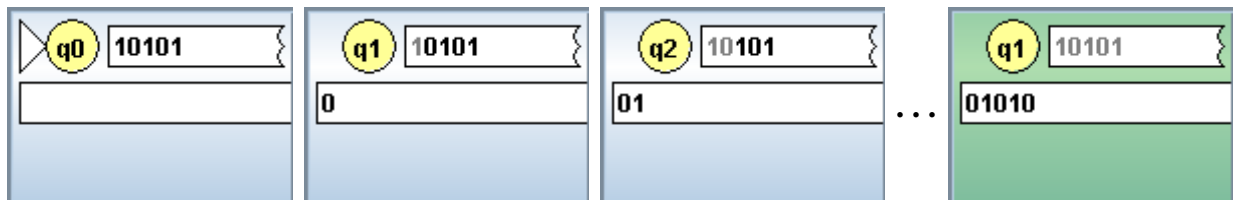
At each step. The output tape updates itself to display the total output that has been produced so far:

Initial:

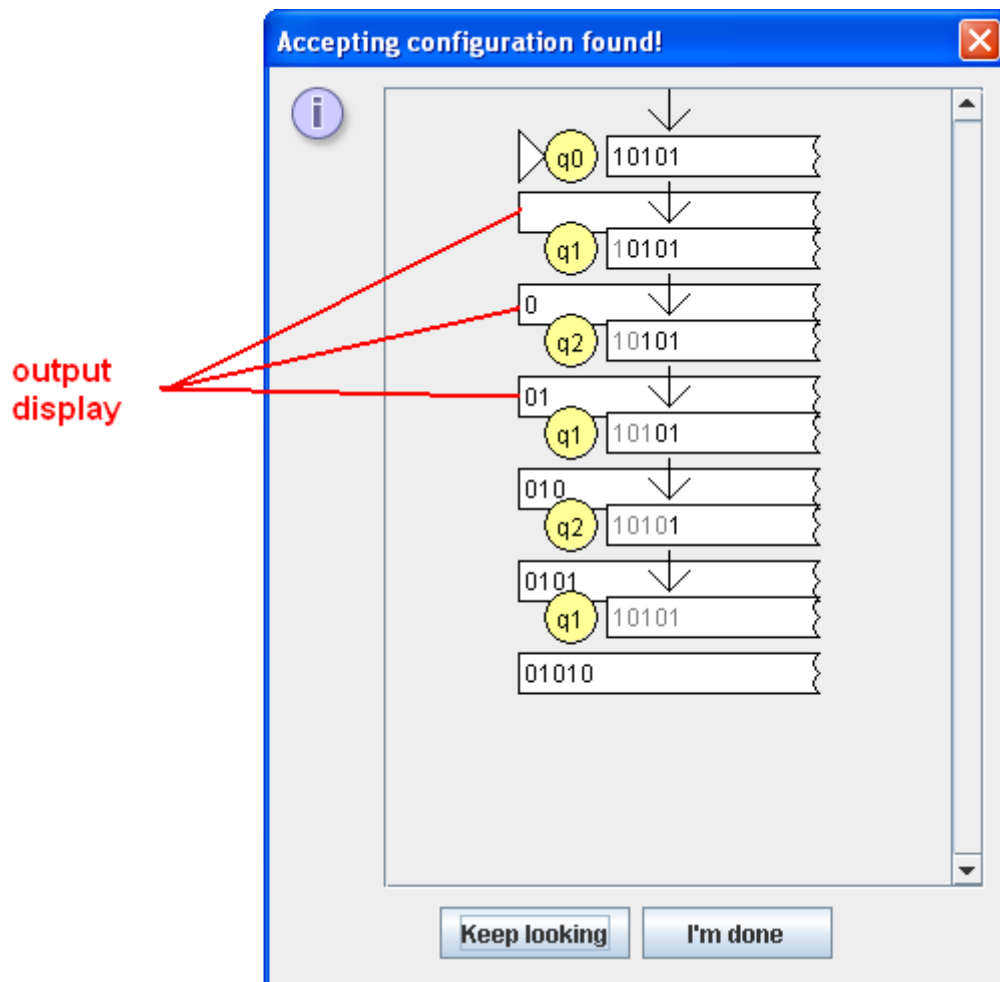
Step 1:

Step 2:

... Final:

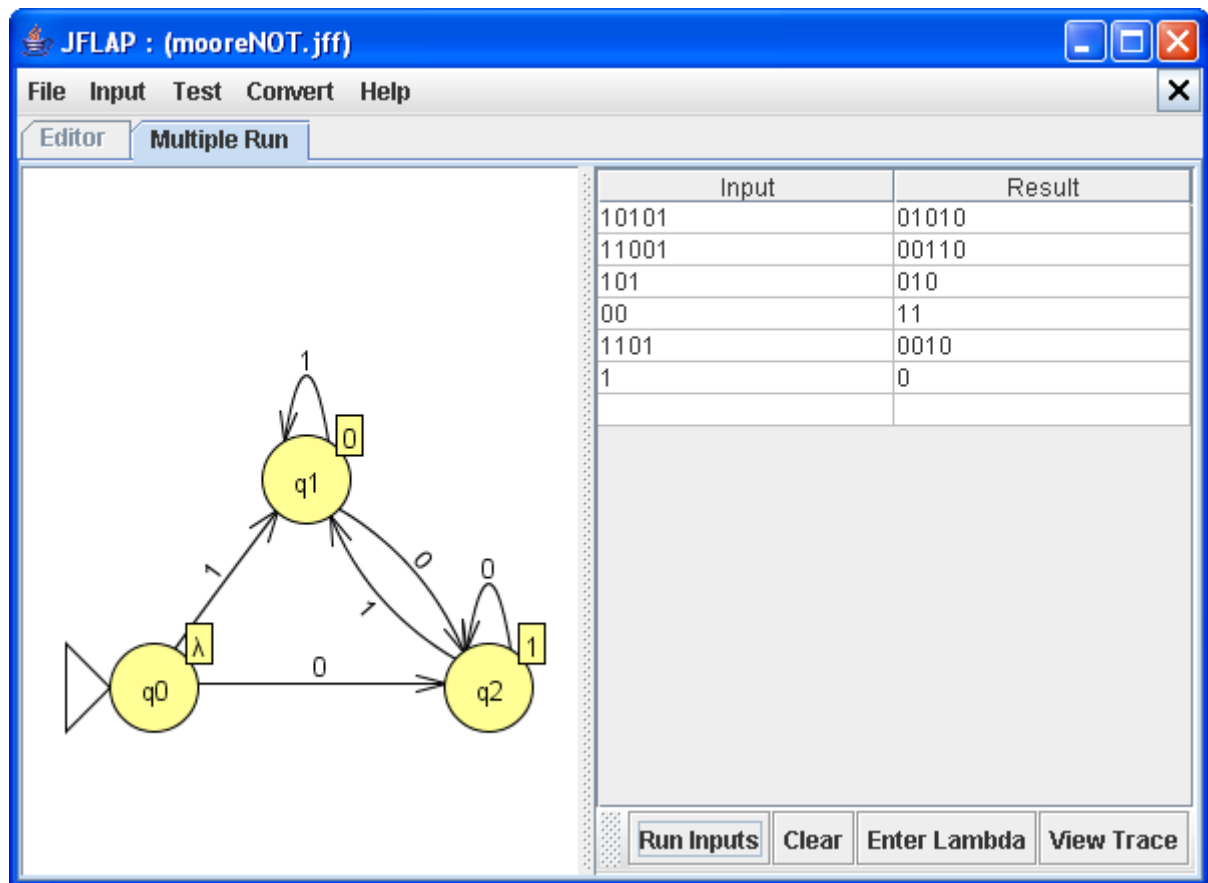


The menu option **Input : Fast Run...** works similarly, with the output being displayed in a tape under the input tape:



Fast run output display

Similarly, when we select **Input : Multiple Run**, output is displayed in the **Result** column.

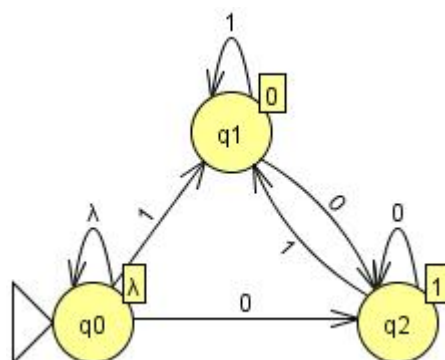


Multiple run output display

No Nondeterminism

As Moore machines map an input to a unique output, nondeterminism cannot exist in a Moore machine. Although we still will be able to build a Moore machine that has nondeterminism, we will not be able to run input on it.

For instance, let's modify our machine slightly to make:



Moore machine with nondeterminism

We we try to run it on an input, with **Input : Step...**, **Input : Fast Run...**, or **Input : Multiple Run...**, we will get an error message asking us to remove the nondeterministic states:



Select **Test : Highlight Nondeterminism** to view the nondeterministic states. Remove the nondeterminism in order to be able to run input on the Moore machine.

Output Convention

In JFLAP, we adopt the output convention that the machine produces the output associated with the start state when it is turned on. (The alternate view is that the machine does not produce output until the first symbol of input is read.) If we wish to use the alternate view for a machine, we may just add a start state with the empty string as its output, like the machine described above.

This concludes our brief tutorial on Moore machines.

Moore Machine Examples

Contents

[Back to Moore Machines](#)

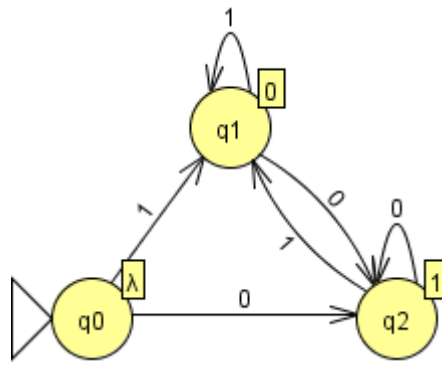
[Example 1: NOT](#)

[Example 2: Halving a Binary Number](#)

Example 1: NOT

Let's start with a simple Moore machine that takes an input bit string b and produces the output $\text{NOT}(b)$.

The machine should look like this, and is can be downloaded through [mooreNOT.jff](#):



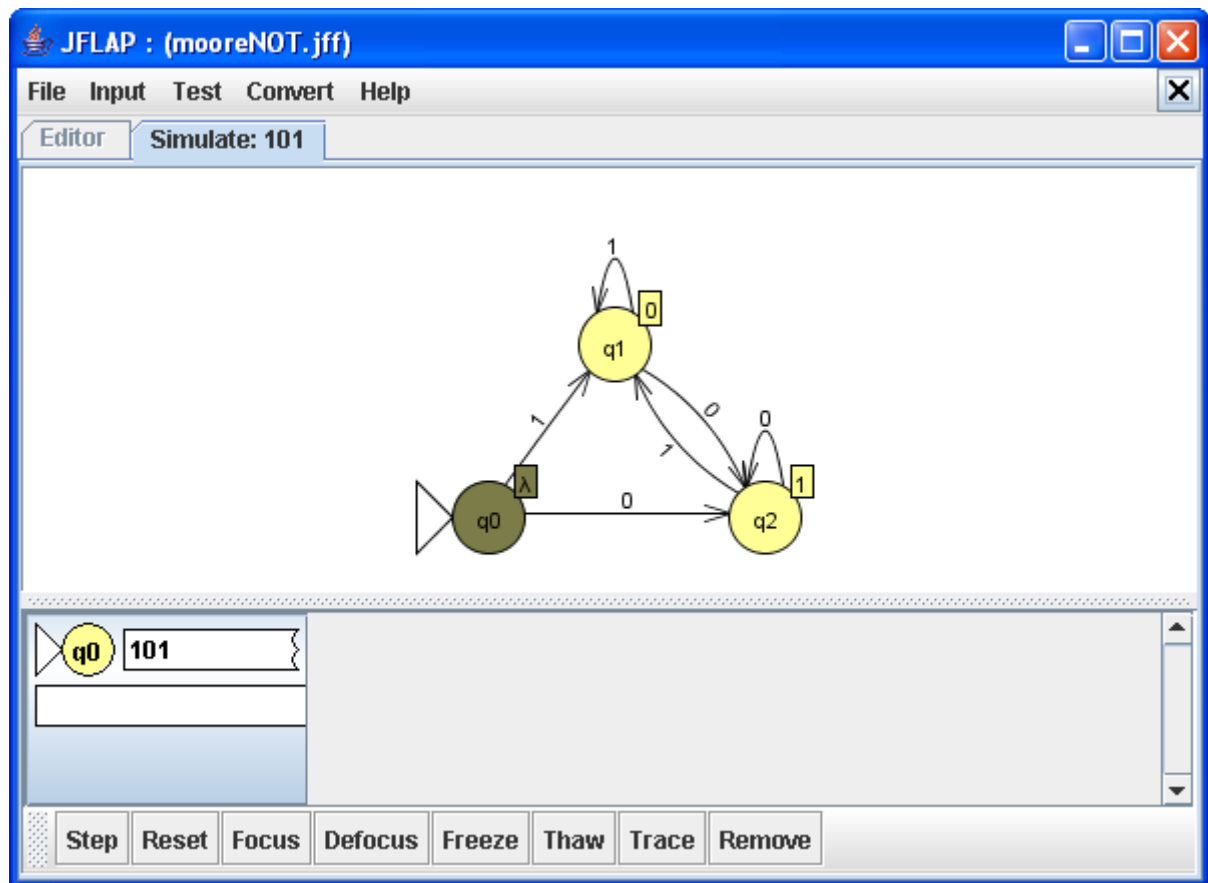
A Moore machine that produces $\text{NOT}(b)$

As you can see, this machine has three states, with q_0 producing no output, q_1 producing the output "1", and q_2 producing the output "0".

The initial state, q_0 produces no output as, in this case, we do not want the machine to produce an output if it is not given any input. (We adopt the [output convention](#) that the initial state produces output even if there is no input.) As you can see, if we do not want the initial state to produce output, we can use a "dummy state," like q_0 .

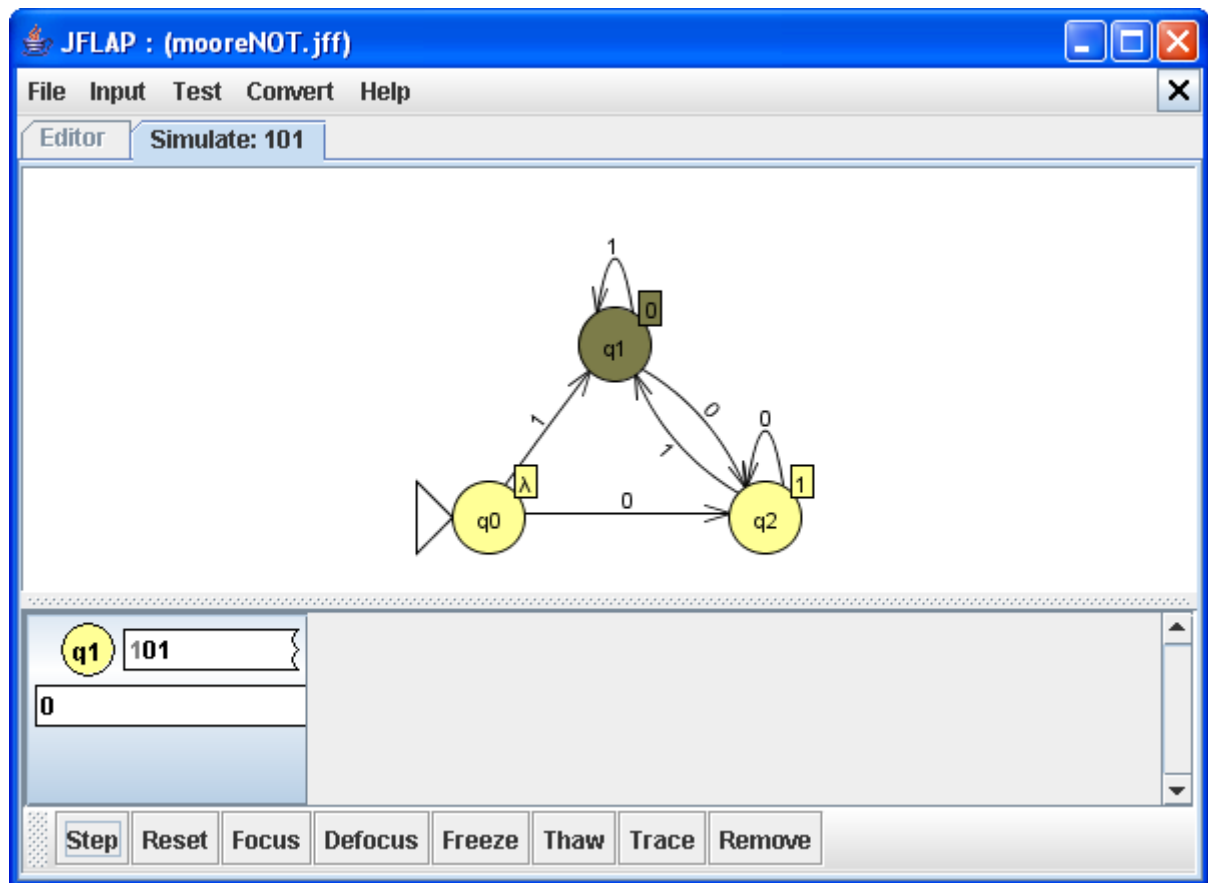
There are two other states, q_1 that produces the output "0", and q_2 that produces the output "1". Any outgoing transition on the input "0" goes to q_2 , so the input bit "0" always produces the output bit "1". Similarly, any outgoing transition on the input "1" goes to q_1 , so the input bit "1" always produces the output bit "0".

Let's step the machine through the input "101".



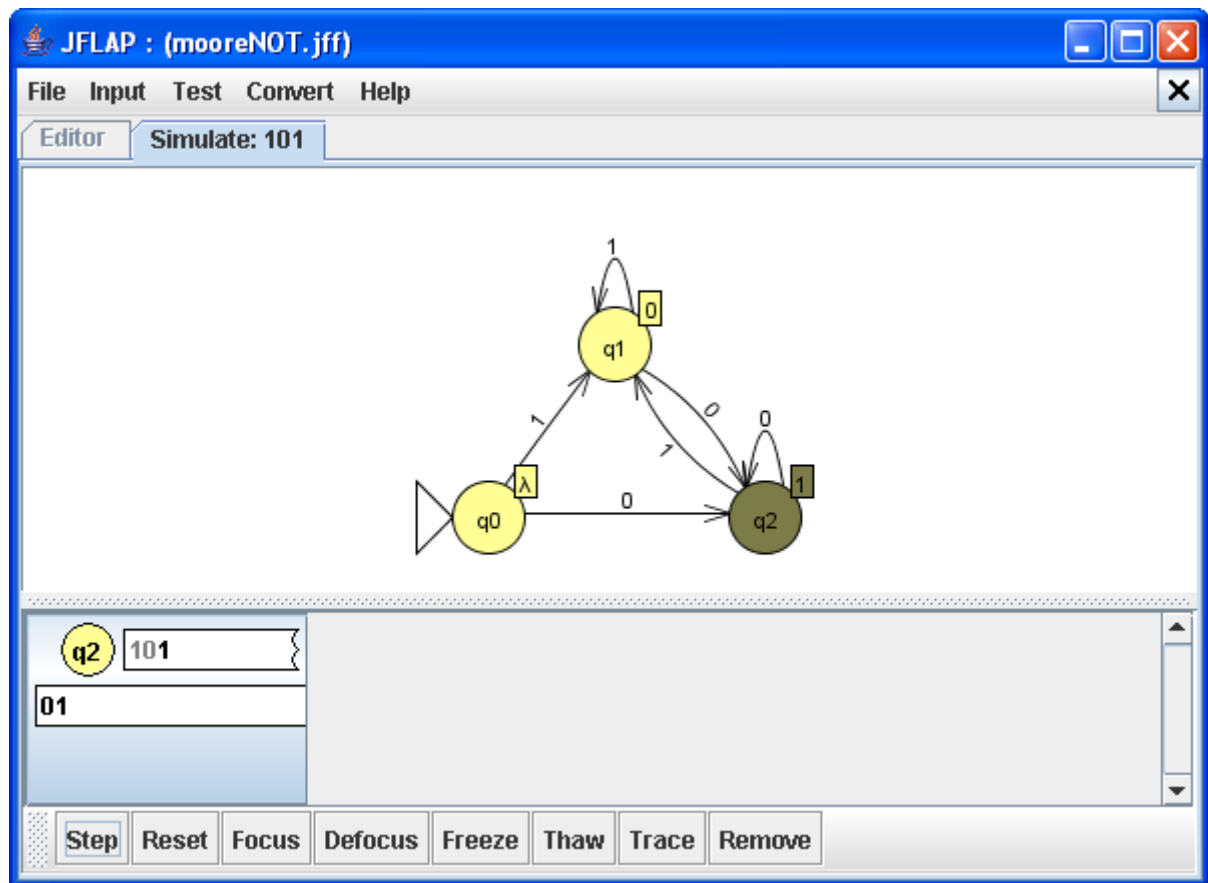
Initial set up

The machine is in its initial state, q_0 . It does not produce any output because the output assigned to that state is λ , or the empty string. Click **Step** to process the next bit of the input.



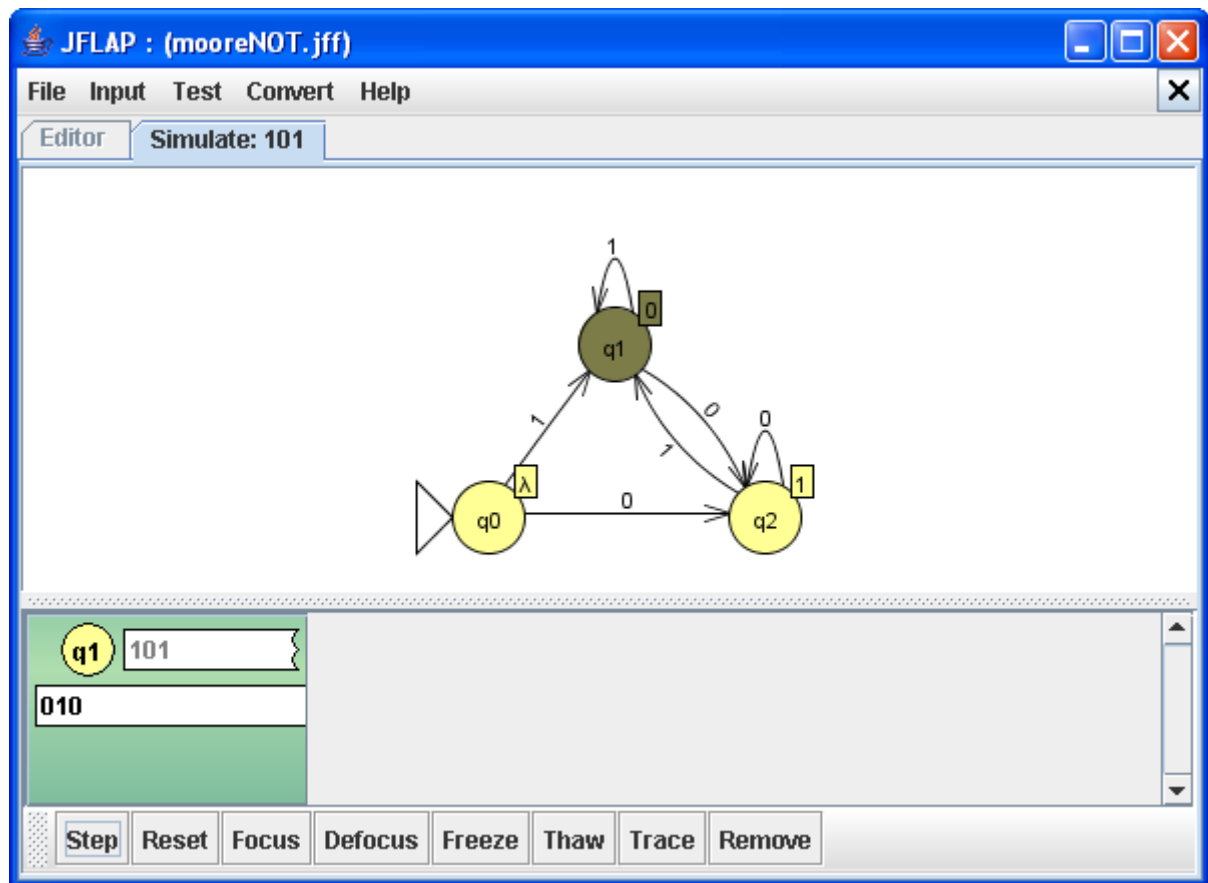
Step 1

Upon processing the first bit of the input, "1", the machine takes the transition to q_1 . At q_1 , it produces the output bit "0", which is the negation of the input bit. If we examine q_0 , we can see that it will always correctly handle the first bit of input. The state itself produces no output. If the first bit of input is "1", it will take the transition to q_1 , that will produce the output of "0". On the other hand, if the first bit of input is "0", it will take the transition to q_2 , producing the output of "1". Click **Step** to process the next bit of the input.



Step 2

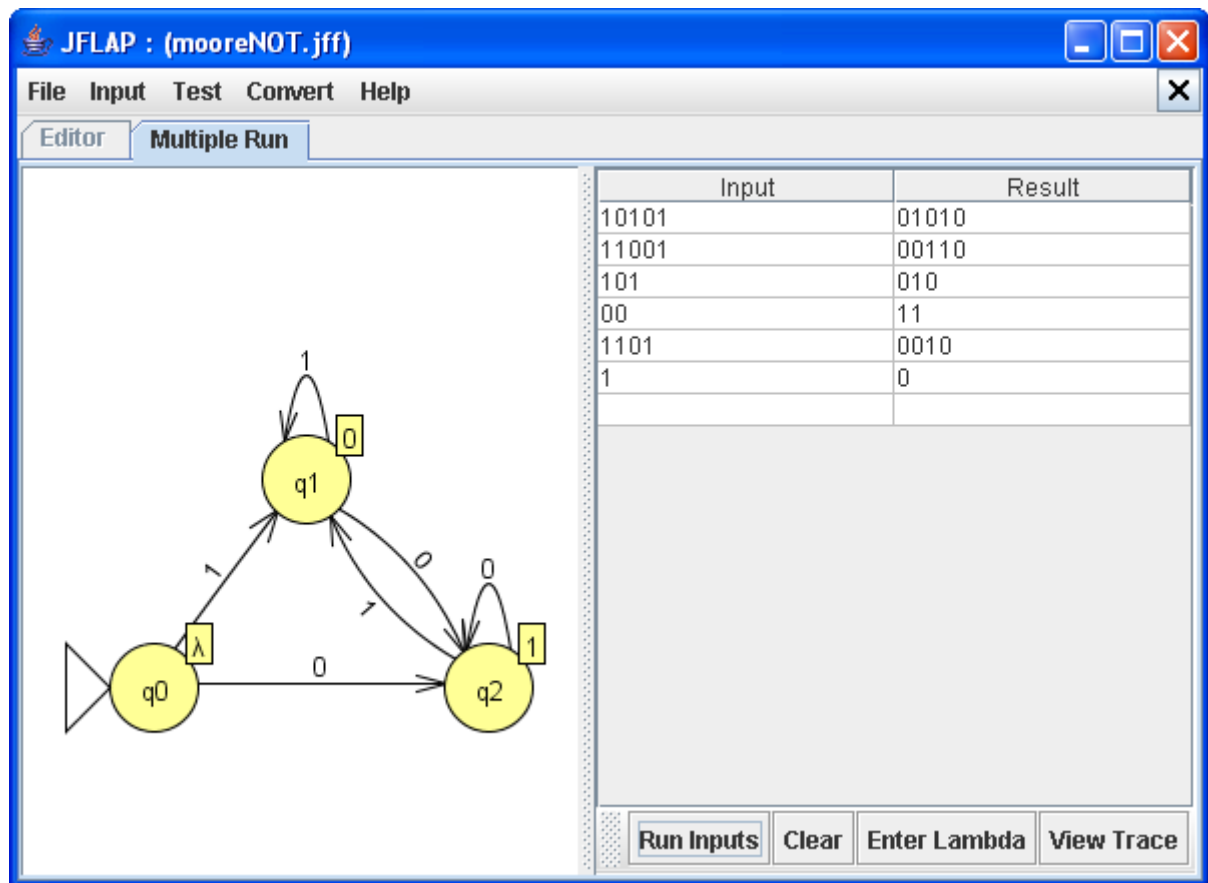
Upon processing the next bit of the input, "0", the machine takes the transition to q_2 . At q_2 , it produces the output bit "1", which is the negation of the input bit. If we examine all the incoming transitions to q_2 we will see that, from any state, the input of "0" will cause the machine to enter q_2 . Thus, from any state, the input bit "0" will create the output of "1". Click **Step** to process the next bit of the input.



Step 3

On processing the last bit of the input, "1", the machine takes the transition to q_1 . At q_1 , it produces the output bit "0", which is the negation of the input bit. Examining all incoming transitions to q_1 , we will see that, very much like q_2 , from any state, the input of "1" will cause the machine to enter q_1 . Therefore, from any state, the input bit of "1" will create the output of "0". Thus, we have seen that the machine will produce the negation of any bit string.

We can also test this machine on multiple input strings.



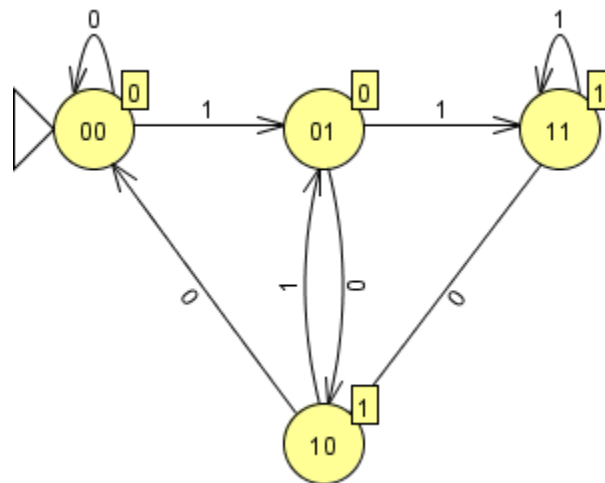
Testing the machine on multiple strings

As you can see, it produces $\text{NOT}(b)$ in every instance.

Example 2: Halving a Binary Number

Next, let's discuss a slightly more complex machine that halves a binary number, truncating any decimal places. It should be noted that halving a binary number merely involves dropping the least significant bit, or shifting the number right by one bit. However, we will make this a little more complicated by dictating that the machine will receive input starting from the most significant bit to the least significant bit, or from left to right. Thus, this would require the machine to remember the most recent two bits of input.

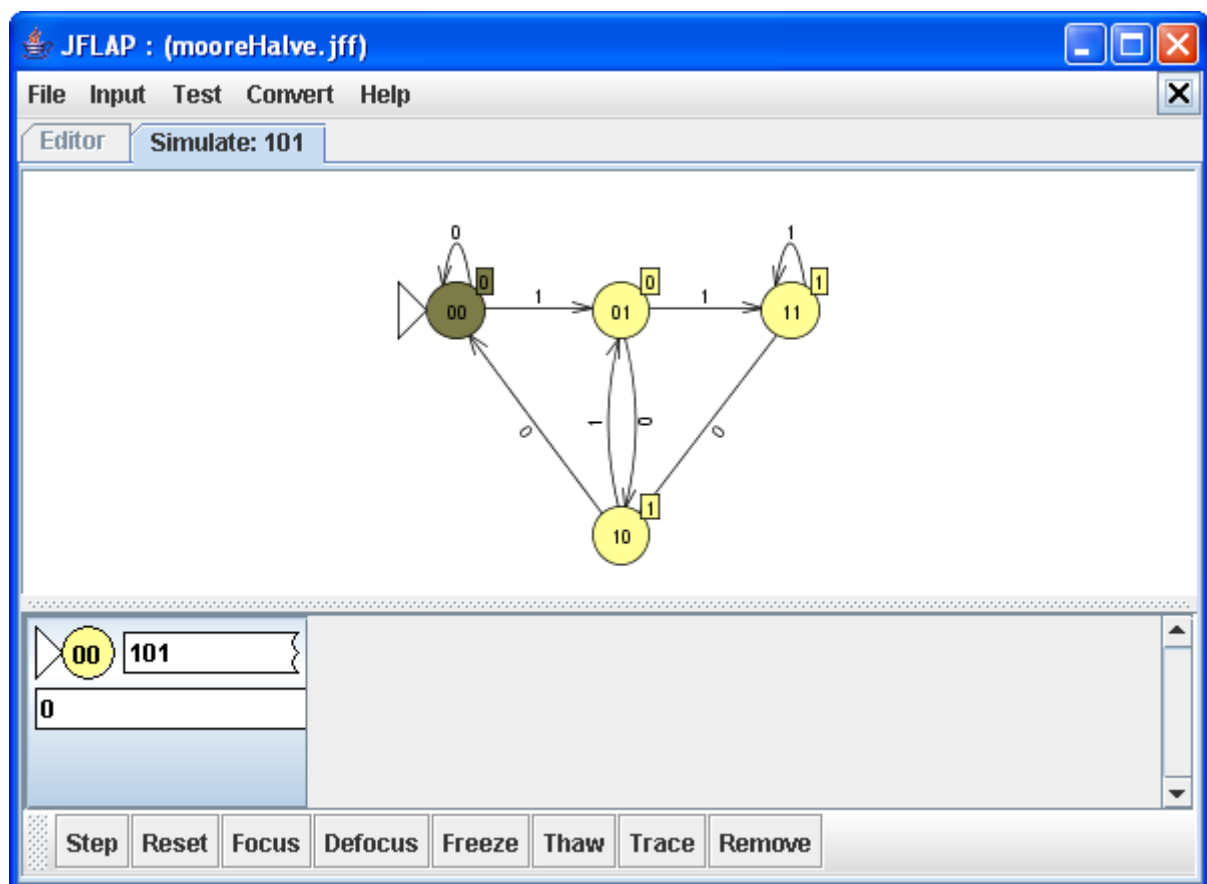
The machine should look something like this, and can be downloaded through [mooreHalve.jff](#):



A Moore machine that halves binary numbers

Let's take a look at the machine. There are four possible permutations of two consecutive bits, 00, 01, 10, and 11, and each is represented by a state in the machine. For instance, the state 01 means that the most recent input bit was 1, and the input bit before that was 0. Each state produces the output equivalent to the second-most recent input bit, and stores the most recent bit to relay that information to the next state (where it will become the second-most recent input bit).

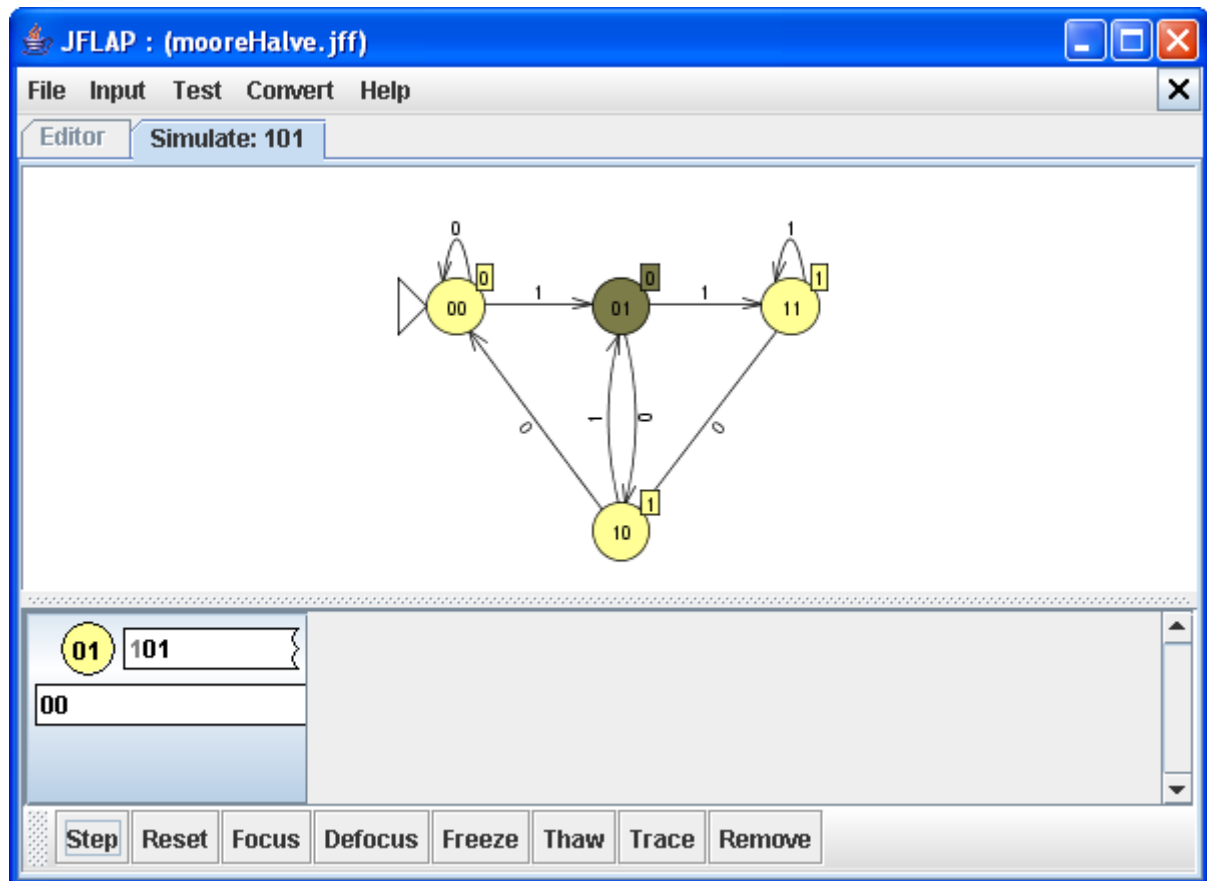
Let's step through the input "101".



Initial set up

We chose state 00 as the initial state as it is the only state that accurately represents the binary number before any bits have been processed. In other words, adding two 0's as the most significant bits in a binary number will not change the value of the number. Similarly, the output of state 00, "0", added as the most significant bit of the entire output, will not change its value.

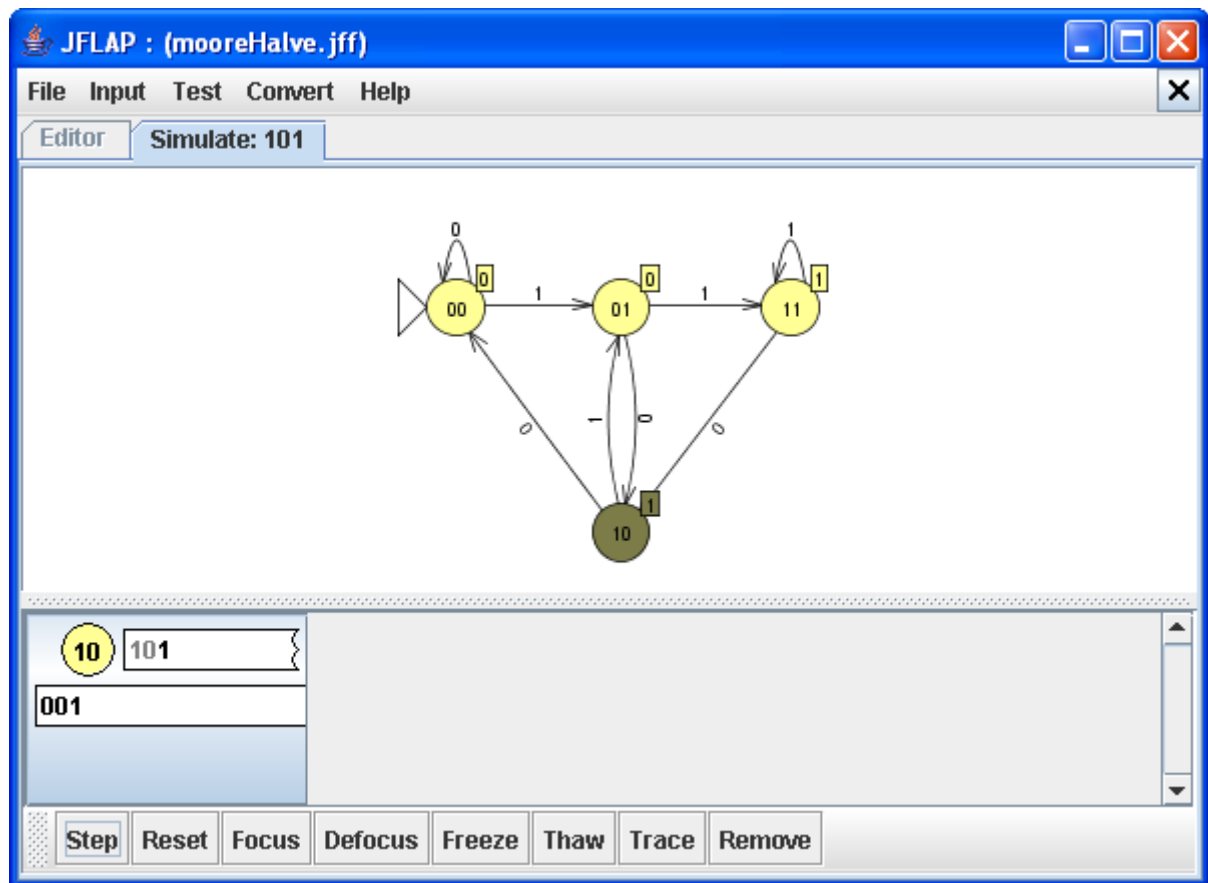
The state produces the output of "0". Click **Step** to process the next bit.



First bit, "1", processed

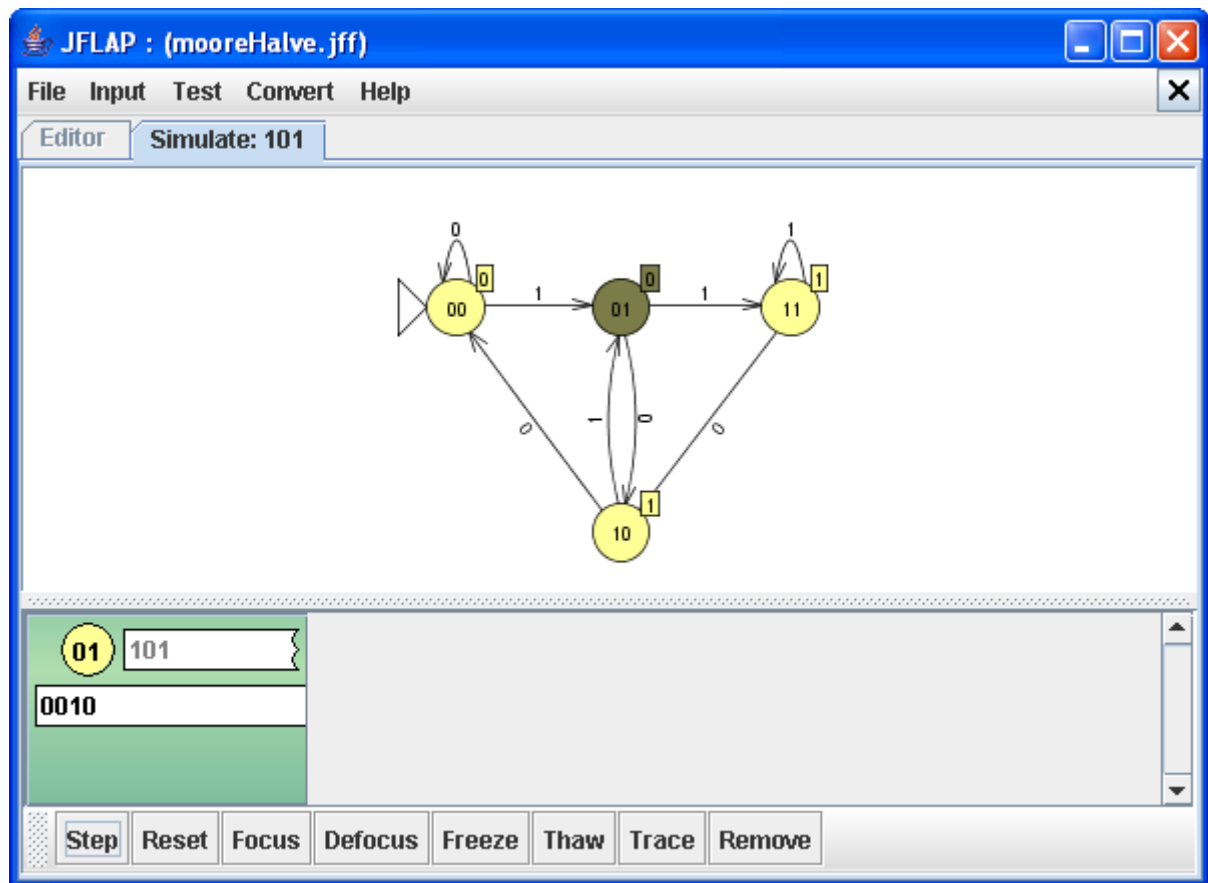
The machine has processed the first bit of input, "1", taking transition 1 from state 00 to state 01. State 01 then produces the output "0". Although this state, state 01, produces the same output as the initial state, state 00, it is different because it remembers that the most recent input bit is "1". Thus, it takes different transitions than the initial state for the same input.

Notice that all outgoing transitions from this state, state 01 go to states that produce an output of "1". This is equivalent to the function of shifting the bits right by one, as the machine's output is the second-most recent bit. Click **Step** to process the next bit.



Second bit, "0", processed

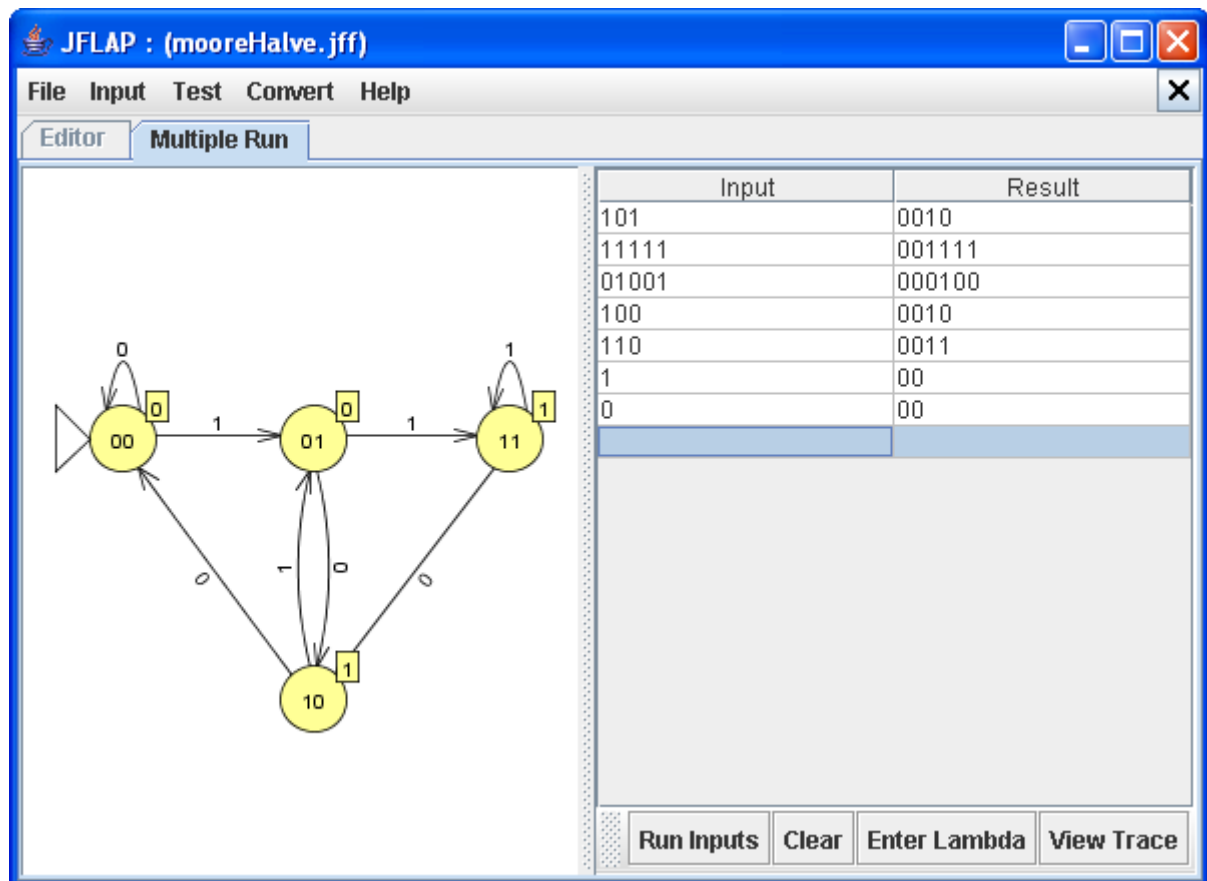
The machine processes the next bit of input, "0", and takes transition 0 to state 10. State 10 then produces the output "1", which corresponds to the previous input bit, "1". This state now remembers that the most recent input bit was "0", and all outgoing transitions are to states which have the output "0". Click **Step** to process the next bit.



Last bit, "1", processed

The machine processes the last bit of input, "1", and takes transition 1 to state 01. It produces the output "0", completing the output. Although the most recent, and last, input bit was "1", this is ignored, effectively shifting all the bits to the right by one. This concludes the machine's process of dividing a binary number by two.

Let's try our machine out on different numbers.



Running the machine on different numbers

As you can see, the machine produces the correct results in each instance. You might notice that in every instance, the output starts with "00", which is unnecessary, even if it does not change the number. This is because the machine starts in state 00, which produces the output "0" even if there is no input. We could have avoided this unnecessary "0" by adding a dummy initial state as we did in the [previous example](#).

The second "0" is due to the fact that all the outgoing transitions from state 00, lead to states that have the output "0". Regardless of the first bit of the input, the first bit of output is "0" because the number is divided by two, that would shift all bits to the right by one. This "0" is also necessary for single-bit numbers such as 0 and 1.