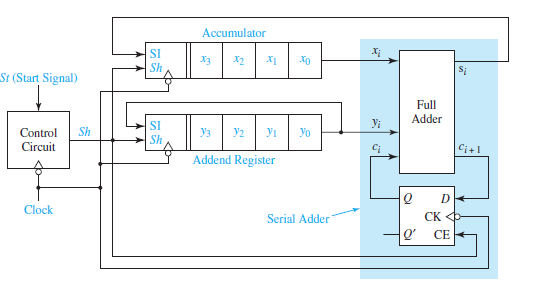
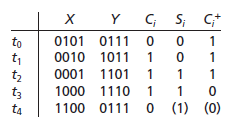
Charles H Roth Jr., Larry L. Kinney ―Fundamentals of Logic Design, Cengage Learning, 7th Edition.

**18.1 Serial Adder with Accumulator**



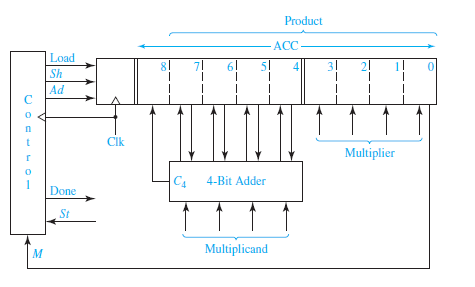
* Two shift registers are used to hold the 4-bit numbers to be added, *X* and *Y*.
* The *X* register serves as an accumulator and the *Y* register serves as an addend register.
* When the addition is completed, the contents of the *X* register are replaced with the sum of *X* and *Y*.
* When *Sh* = 1 and an active clock edge occurs, SI is entered into *x*3 (or *y*3) and the contents of the register are shifted one place to the right.
* At each clock time, one pair of bits is added.
* When Sh = 1, the falling clock edge shifts the sum bit into the accumulator, stores the carry bit in the carry flip-flop, and rotates the addend register one place to the right.

Example of the serial adder operation:



* Initially, the accumulator contains 0101 and the addend register contains 0111.
* At t0, the full adder computes 1+1+0=10, so si=0 and ci=1.
* After the first falling clock edge (time t1) the first sum bit has been entered into the accumulator, the carry has been stored in the carry flip-flop, and the addend has been cycled right.
* After four falling clock edges (time t4), the sum of X and Y is in the accumulator, and the addend register is back to its original state.

**18.2 Design of a Parallel Multiplier**

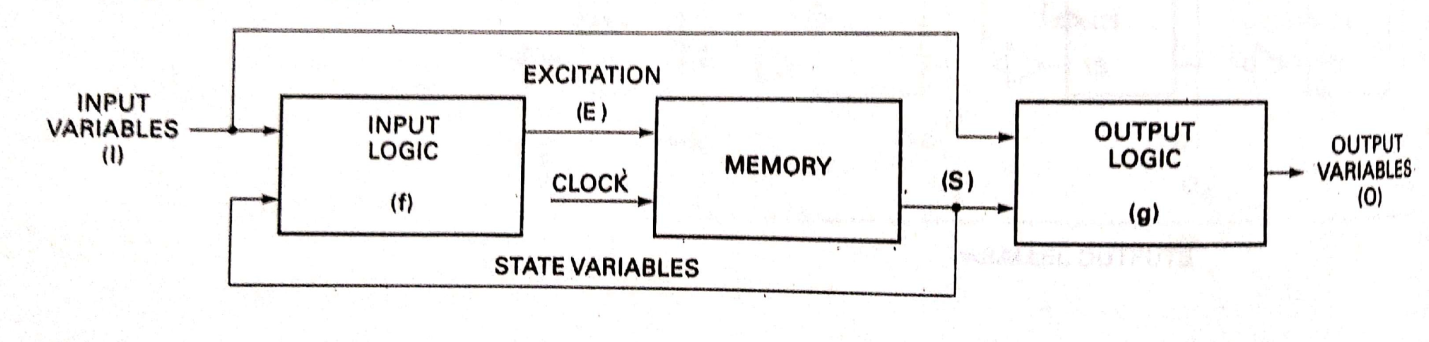
****

* The multiplication of two 4-bit numbers requires a 4-bit multiplicand register, a 4-bit multiplier register, and an 8-bit register for the product.
* The product register serves as an accumulator to accumulate the sum of the partial products.
* 4 bits from the accumulator and 4 bits from the multiplicand register are connected to the adder inputs.
* The adder calculates the sum of its inputs when an add signal (*Ad*) occurs, the adder outputs are stored in the accumulator by the next rising clock edge, thus causing the multiplicand to be added to the accumulator.
* An extra bit at the left end of the product register temporarily stores any carry (*C*4) which is generated when the multiplicand is added to the accumulator.
* The lower four bits of the product register store the multiplier.
* As each multiplier bit is used, it is shifted out the right end of the register to make room for additional product bits.
* The Load signal loads the multiplier into the lower four bits of ACC and clears the upper 5 bits.
* The shift signal (*Sh*) causes the contents of the product register to be shifted one place to the right when the next rising clock edge occurs.
* The control circuit puts out the proper sequence of add and shift signals after a start signal (*St =* 1) has been received.
* If the current multiplier bit (*M*) is 1, the multiplicand is added to the accumulator followed by a right shift.
* If the multiplier bit is 0, the addition is skipped and only the right shift occurs.

Example of the Parallel Multiplier operation:



**Mealy Machine:** In a Mealy machine, the outputs are a function of the present state and the value of the input. The outputs may change asynchronously in response to any change in the inputs.

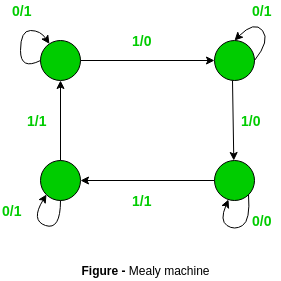


The Mealy Model is characterized by

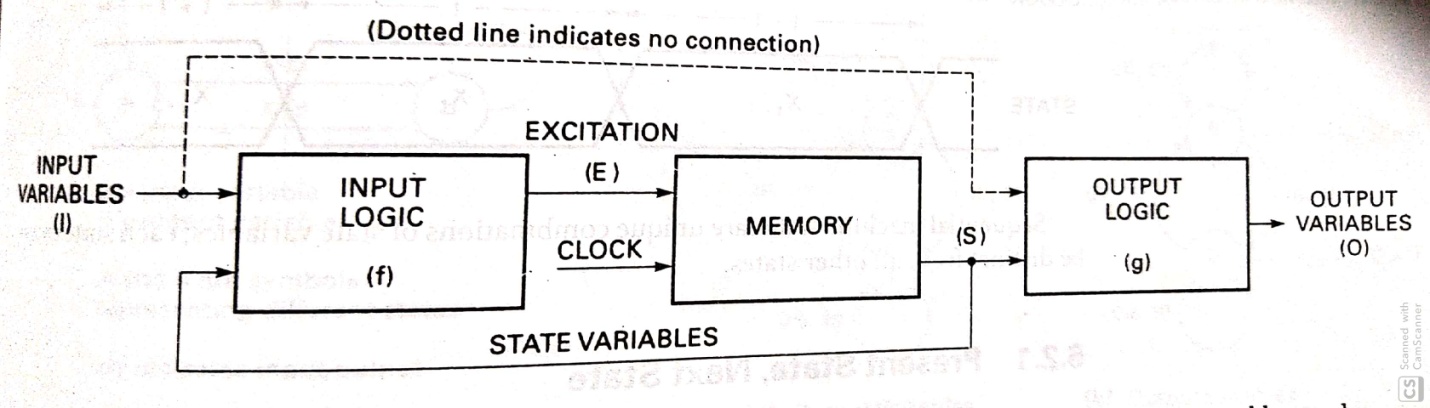
E+ = f1(I, S)

O = f2(I, S)

State diagram example:



**Moore machine:** In a Moore machine the outputs depend only on the present. A combinational logic block maps the inputs and the current state into the necessary flip-flop inputs to store the appropriate next state. The outputs are computed by a combinational logic block whose inputs are only the flip-flops state outputs. The outputs change synchronously with the state transition triggered by the active clock edge.

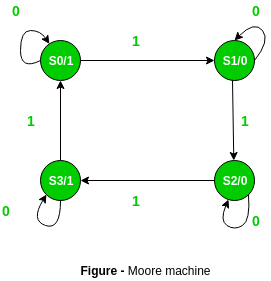


The Moore Model is characterized by

E+ = f1(I, S)

O = f2(S)

State diagram example:



**Differences between Moore Machine and Mealy Machine**

|  |  |
| --- | --- |
| **Moore Machine** | **Mealy Machine** |
| 1. Output depends only upon present state. 2. If input changes, output does not change. 3. More number of states are required. 4. There is more hardware requirement. 5. They react slower to inputs. 6. Synchronous output and state generation. 7. Output is placed on states. 8. Easy to design. | 1. Output depends on present state as well as present input. 2. If input changes, output also changes. 3. Less number of states are required. 4. There is less hardware requirement. 5. They react faster to inputs. 6. Asynchronous output generation. 7. Output is placed on transitions. 8. It is difficult to design. |