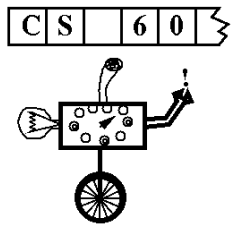


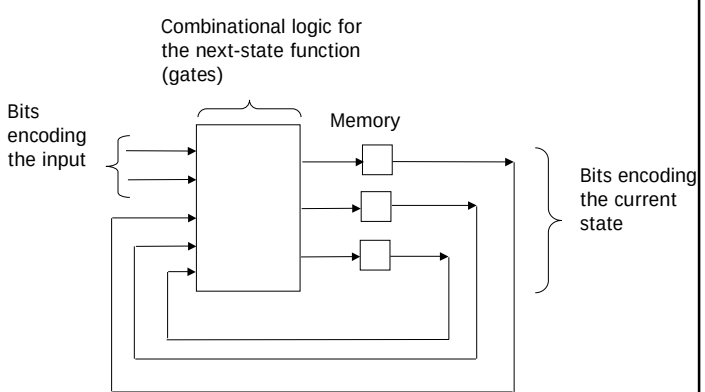
State Machines In Hardware

December 3, 2001

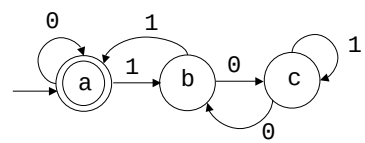
State Machines in Hardware

- A hardware implementation of a state machine needs two major parts
  - Finite memory (to remember which state we're currently in)
  - Logic to determine what the next state should be, given the current state and the next input.
- We can encode an arbitrary set of states just as we encode any set, in terms of some number of bits.
- We implement the next-state function using ***combinational*** logic gates: given an encoded version of the current state and the input, produce the encoding of the next state.

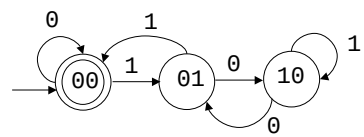
Next-State Sequential Logic for a Finite-State Machine



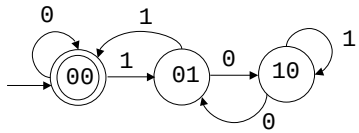
Example: A Multiples-of-3 Machine



Suppose we encode the state set using 2 bits, thus:



Transition Table



The next state is summarized by the following table:

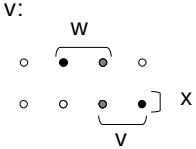
next state	input	
current state	0	1
0 0	0 0	0 1
0 1	1 0	0 0
1 0	0 1	1 0

But we already know how to implement such a table in logic!

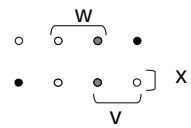
Karnaugh Maps

next vw	input (x)	
current state (vw)	0	1
0 0	0 0	0 1
0 1	1 0	0 0
1 0	0 1	1 0

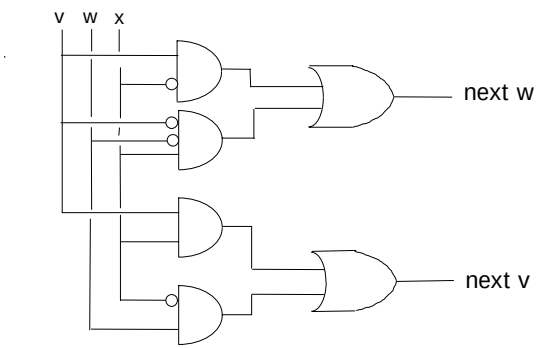
next v:



next w:



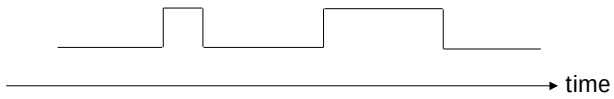
Logic Diagram



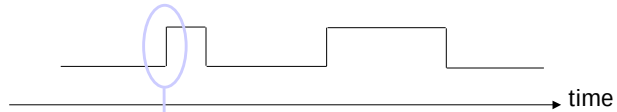
Representing a Discrete Sequence in Continuous Time

- From our viewpoint, time appears to be a continuous variable.
- For a digital sequence, we want discrete values $[x_0, x_1, x_2, x_3, ...]$, not a continous function x(t).
- The typical way to handle this is to use a *clock*.
- The continuous sequence is “sampled” at regularly-spaced times, when the clock “ticks”.

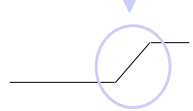
Sampling a Signal



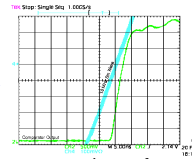
Sampling a Signal



(In reality, rises and falls aren't so vertical.)

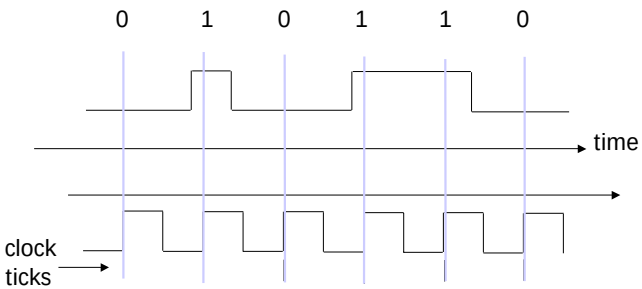


as sometimes shown
in engineering drawings



an actual waveform

Sampling a Signal



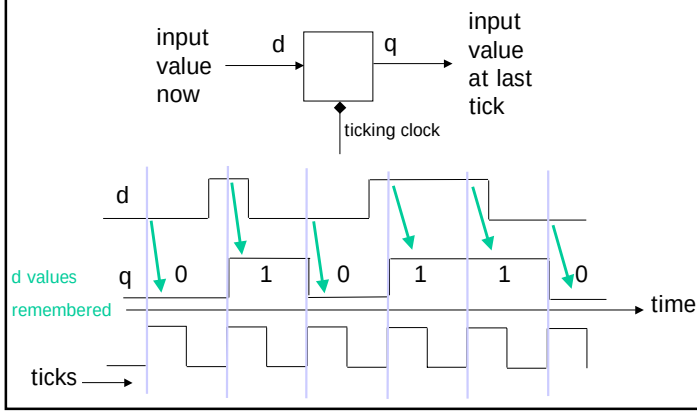
Clock Rate

- The clock is analogous to the conductor of a symphony orchestra: it keeps all of the players in synch.
- The rate at which the clock ticks is the quoted rate of the processor, e.g. 500 MHz (500,000,000 ticks per second).
- It is possible to design systems that don't have clocks ("asynchronous systems") but these are rare.

The Basic Unit of Memory is the Flip-Flop

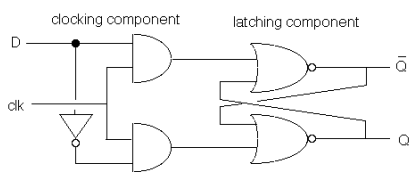
- A flip-flop remembers **one bit**, either a 0 or 1.
- The presence of a synchronizing clock is assumed.
- The bit is held from one clock-tick to the next.
- Each time the clock ticks, whatever value (0 or 1) exists at the flip-flop's input is **remembered**; the old value is lost.

Flip-Flop Behavior

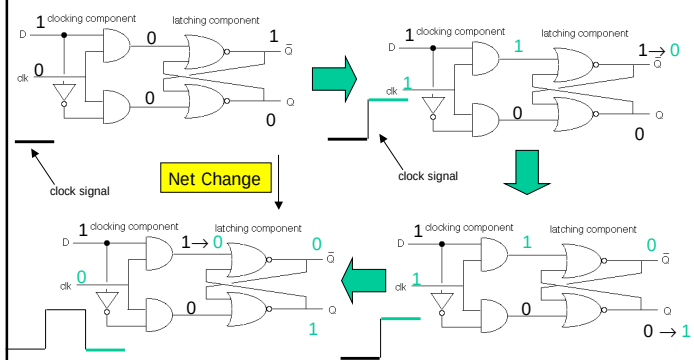


Inside a Flip-Flop

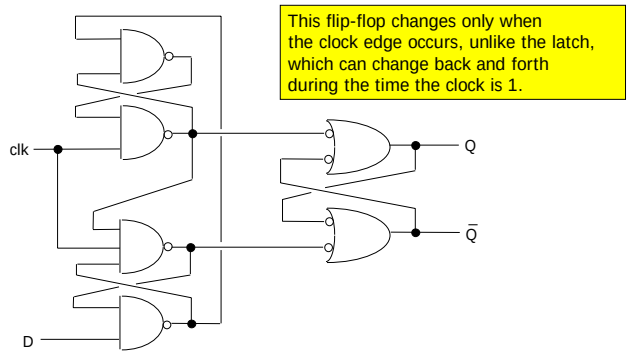
- A flip-flop can be constructed from ordinary gates (which have some associated switching ***delay***) and feed-back connections.
- A first approximation, called a ***clocked latch***, is:



Clocked Latch Behavior

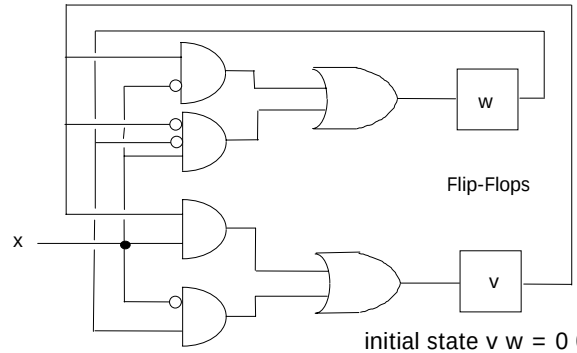


Edge-Triggered Flip-Flop



Analysis is left to the reader.

Logic Diagram in Context



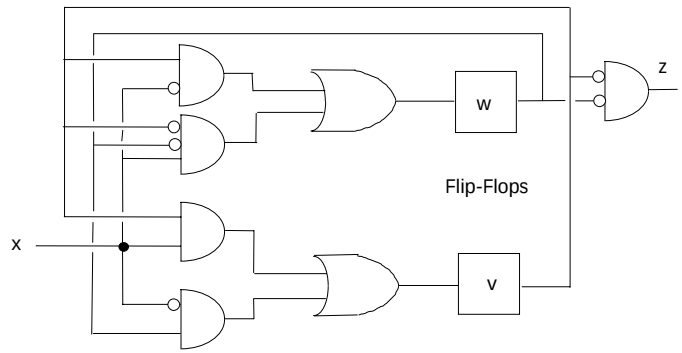
initial state v w = 0 0

Output Considerations

- We need to drive the output from the encoded state.
- The output is coded, just like the state and input.
- Let's say that the output is z which has value 1 for accepting, 0 for rejecting.
- Since the only accepting state is 00, the output function is

$$z = v'w'$$

Final Circuit, with Output



initial state v w = 0 0

Physical Computers

(as distinguished from
"virtual" computers, such as
Turing machines)

Computer Components

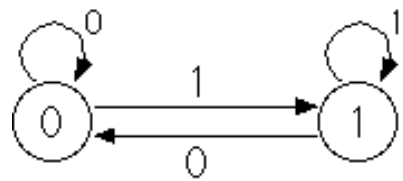
- Finite-state machines
- Combinational logic
- Buses

Register

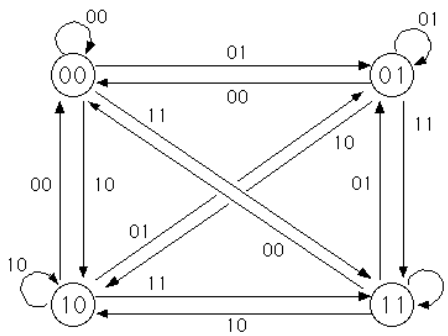
- A register is a finite-state machine that remembers values as bit vectors.
- A register may perform other functions as well:
  - Clearing
  - Incrementing, decrementing
  - Shifting

Simplest register

Remembers one bit = Flip-Flop

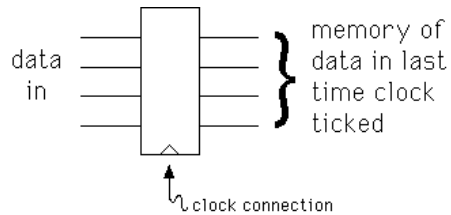


Two-Bit Register

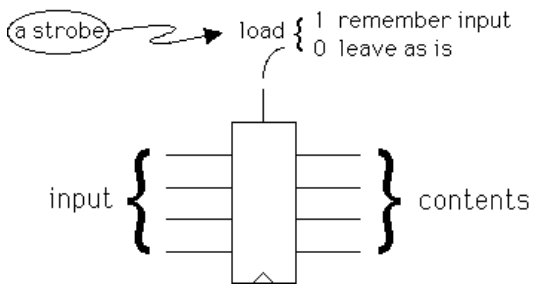


Inputs to a Register

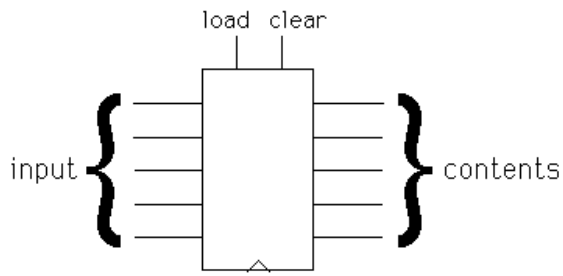
- Data (e.g. value to be remembered)
- "Strobe": function to be performed
- Simplest register has no strobe inputs



Register with Strobe Input

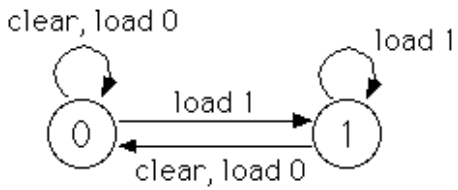


Register with Two Strobe Inputs



1-bit register with load & clear

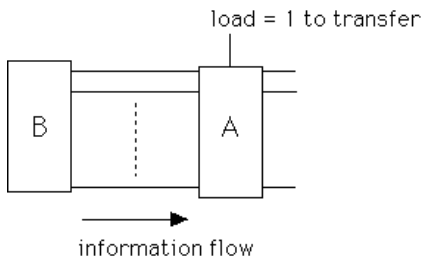
- {clear, load} is 1-hot (never both 1 simultaneously)



Strobe Possibilities

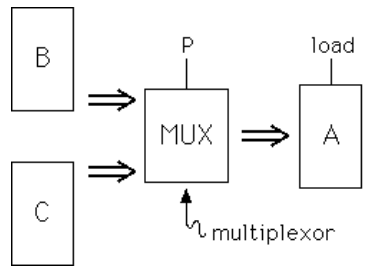
- load
- clear
- increment
- decrement
- complement
- left-shift
- right-shift

Register Transfer



Equivalent Java: `A = B;`

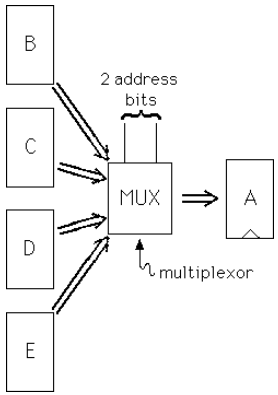
Selective Register Transfer



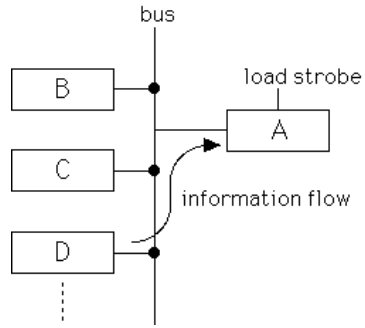
See also: Boole/Shannon Expansion

Equivalent Java: `A = P ? B : C;`

4-way selection



Selection Using a Bus

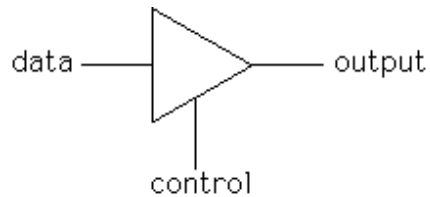


For this to make sense, we need another register output value separate from 0,1.

Implementing Bus Connection

- We can't simply use AND-gates; the output of an AND will *always* be 0 or 1.
- Connecting together wires with 0 and 1 simultaneously would be fatal.
- For the bus, use a third possible output value:
  - "high impedance", "high Z", or
  - NC (no connection)

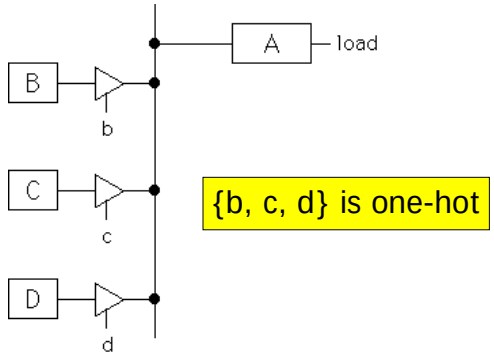
3-State Buffer



output = control ? data : NC;

No Connection

Selection using Bus

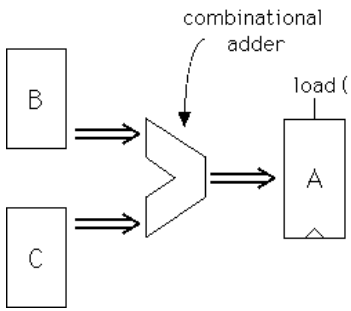


{b, c, d} is one-hot

Bus vs. Multiplexor

- The bus-type connection allows selection from a large number of inputs without requiring a multiplexor tree or other complex logic.

Computing using Combinational Functions



Java: A = B + C;

The actual computation takes place "between" clock ticks.

The clock simply strobes the result into the register.

Next class:

ISC Processor Structure