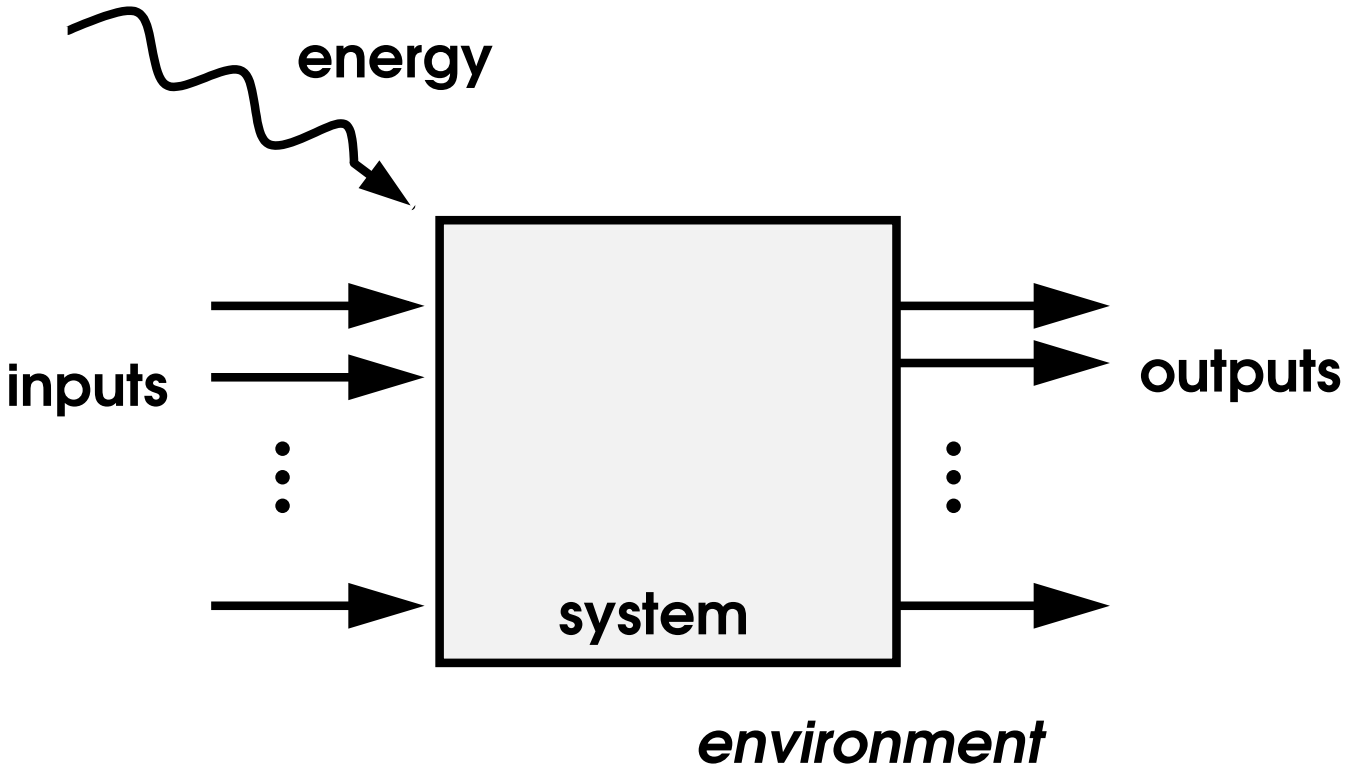
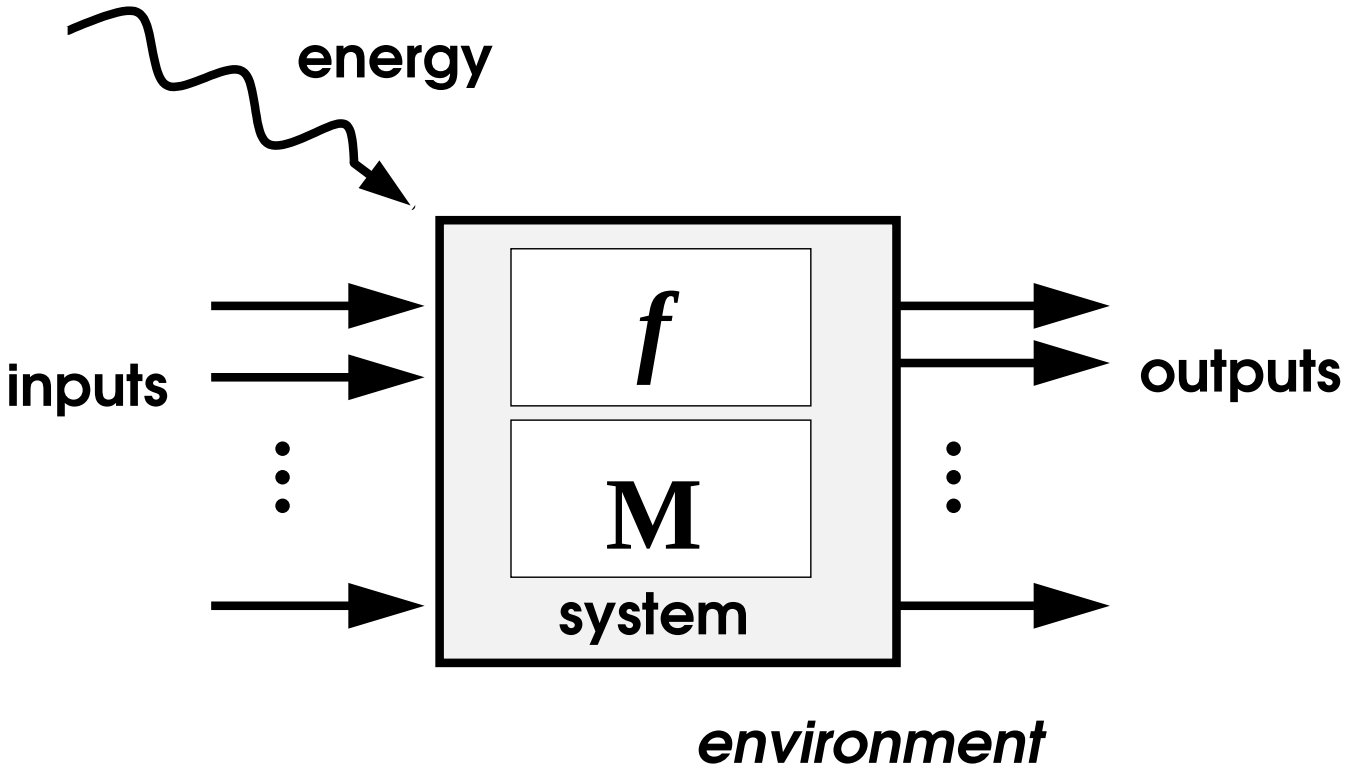
Digital Computing Systems

- Overview of Computing Systems
- Combinational Circuits
- Sequential Circuits

Abstract View of Computer System

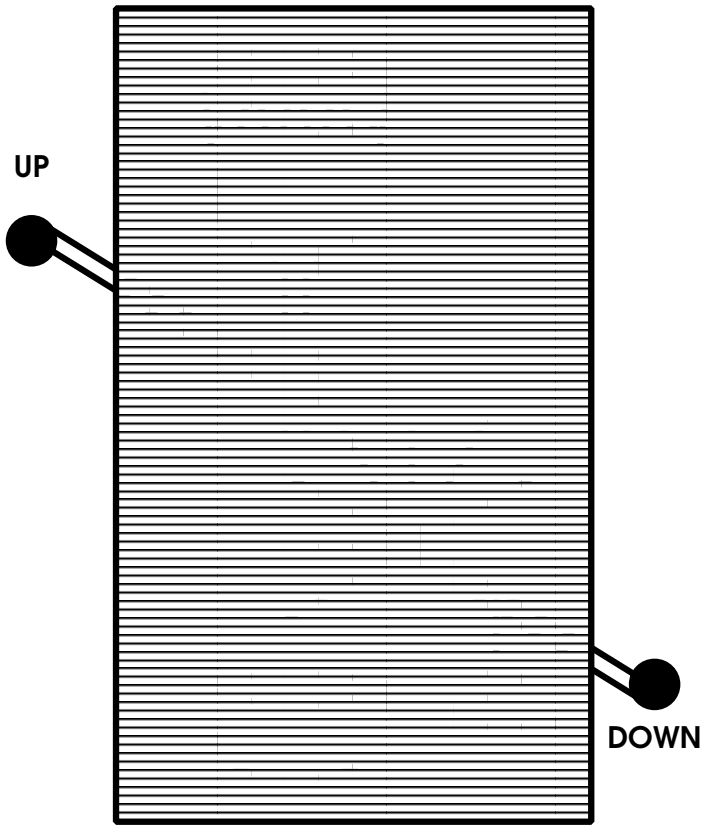


Abstract View of Computer System

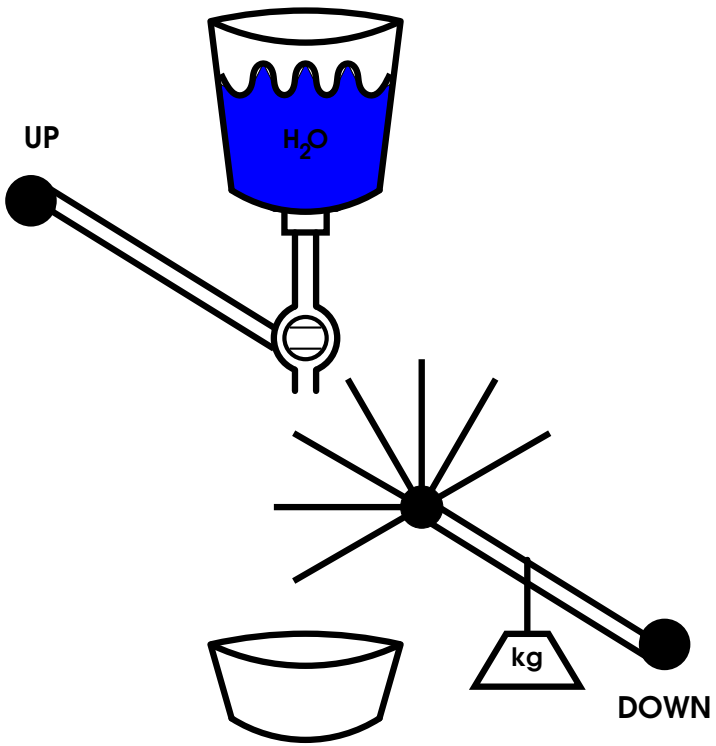


Mechanical Computing Machine

- input on the left, output on the right

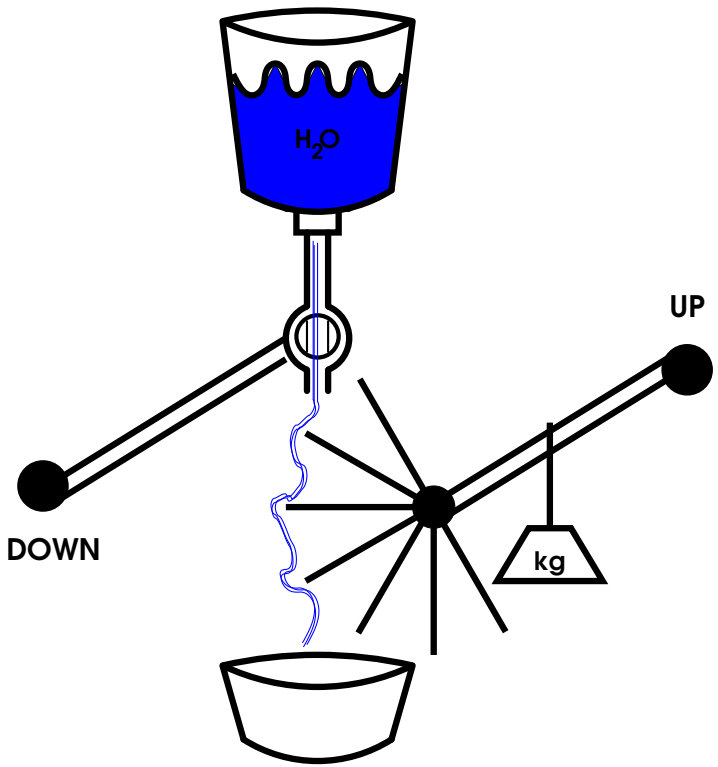


Mechanical Computing Machine



- input on the left, output on the right
- valve, water wheel, weight

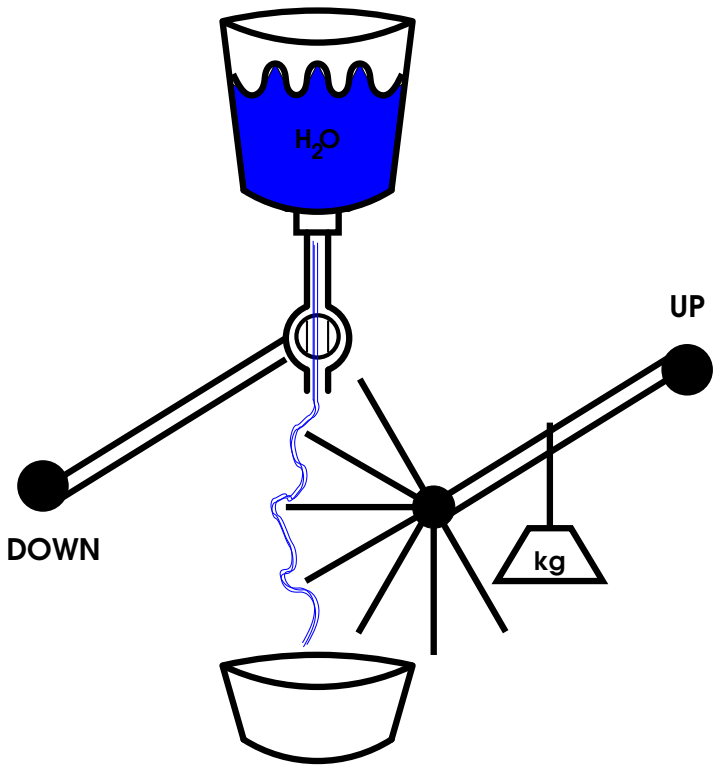
Mechanical Computing Machine



- input on the left, output on the right
- valve, water wheel, weight
- change input and output changes

What have we built?

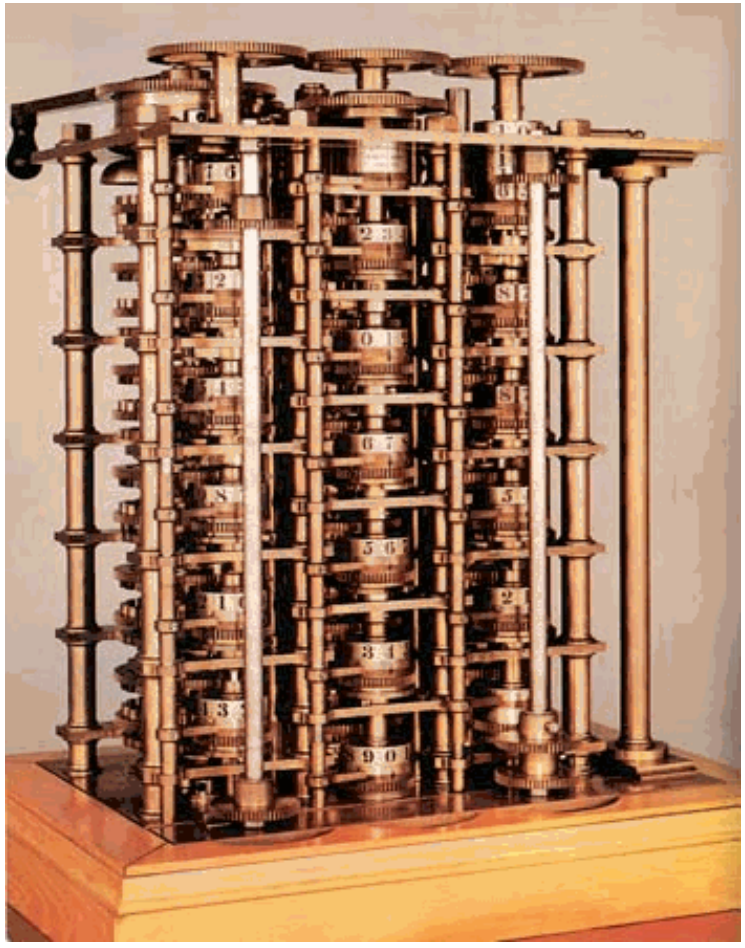
Mechanical Computing Machine



- input on the left, output on the right
- valve, water wheel, weight
- change input and output changes

What have we built? ***an inverter***

Babbage Analytical Machine — An Early Example



- first general-purpose computer (1833)
- used punch cards to encode instructions
- first programmer: Lady Ada Byron, Countess of Lovelace (1842)
- note: Boolean Logic didn't arrive until 1854

Commonality of All Computing Systems

Think about the basic components:

- input
- output
- function or operation

Inputs And Output Mechanisms

Inputs and ouputs have some mechanism:

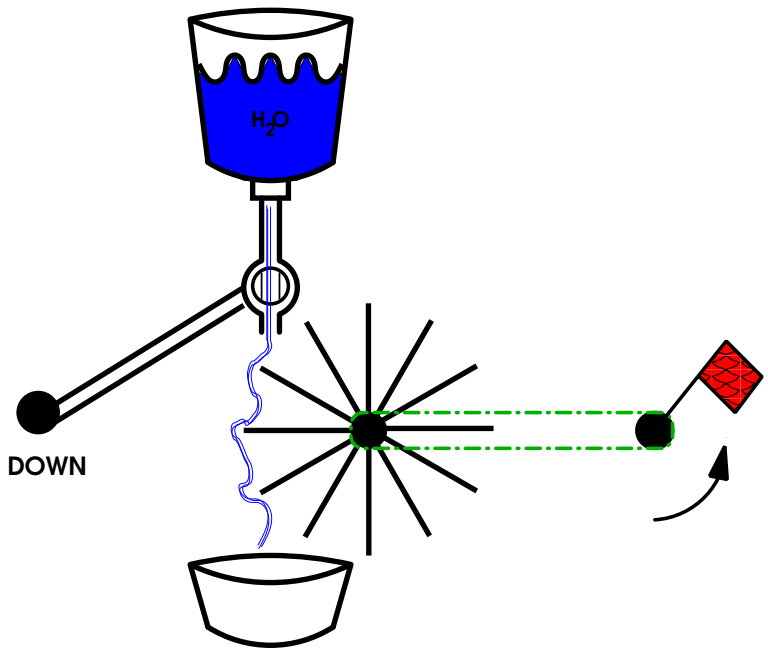
- inputs: sensing
- outputs: signaling
- encoding
- semantics

Sensing/Signalling

- sensing/signaling are physical phenomenon
  - DC voltage
  - light
  - sound waves
  - current

Sensing/Signalling

- sensing/signaling are physical phenomenon
  - DC voltage
  - light
  - sound waves
  - current
- or a waving flag: input mechanism does NOT have to be the same as output mechanism!



Encoding

Lots of standard encoding:

- DC voltage 0V and 5V (for 0 and 1)
- DC voltage -12V and +12V (for 1 and 0)
- LEDs: on or off
- pushbuttons: depressed or not

Encoding (2)

once the input/output has been encoded to 0 or 1, we might do another layer of encoding; groups of bits:

- an integer (binary; i.e. $x_7 \times 2^7 + x_6 \times 2^6 + \cdots$)
- an integer (BCD)
- a character (ASCII)
- *a non-standard encoding*

Combinational Circuits

to make designing these systems reasonable, we represent

- inputs and outputs with Boolean variables

$$x \quad y \quad z$$

- semantics of complex ideas with Boolean expressions

$$x'y + z$$

- connection between inputs and outputs with Boolean functions

$$f = x'y + z$$

Combinational Circuits in VHDL

- the function

$$f = x'y + z$$

- in VHDL would be

```
entity fun is
  port ( x, y, z: in bit;
         f: out bit);
end entity fun ;
architecture dataflow of fun is
begin
        f <= not(x) and y or z ;
end dataflow ;
```

Characteristics of Combinational Circuits

- outputs depend only on the current set of inputs
- expressed strictly as a set of Boolean functions

$$A = x'y + z$$

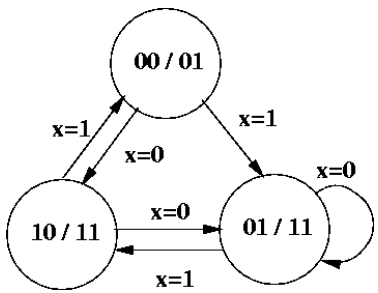
$$B = x'yz + xy$$

(Synchronous) Sequential Circuits

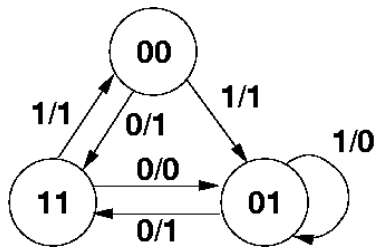
- superset of combinational circuits
- outputs depend not only on the current set of inputs but all previous inputs since reset
- includes memory elements (flip-flops or latches) these hold the "current state" of the machine
- includes a clock and reset
- derived from a Finite State Machine (FSM)

State Chart

- FSM can be draw with a state chart
- also known as state diagram
- Moore-type: outputs are based on state
- Mealy-type: outptus are based on transition



Moore



Mealy

Sequential Circuit (Assemble)

from the state chart we can build a sequential circuit

1. translate chart into state table (present state, next state, output)
2. start with template (next state function, memory, output function)
3. derive functions from state table; minimize
4. assemble circuit

Sequential Circuit (VHDL)

1. decide whether to code in 1, 2, or 3 VHDL processes
   - 2 is most common, 3 is a reasonable choice
   - all synthesize; decision is based on what's easier to understand and code
2. create a type for each state
3. write process 1: a `CASE` statement with a `WHEN` for each state; infer memory
4. write process 2: a `CASE` statement that drives output signals

Example: Sequential Circuit (VHDL) — 1

As with all VHDL, we start with an entity:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity fsm_2 is
    port ( clk, reset, x1 : IN std_logic;
           outp           : OUT std_logic);
end entity;
```

Example: Sequential Circuit (VHDL) — 2

```
architecture beh1 of fsm_2 is
    type state_type is (s1,s2,s3,s4);
    signal state: state_type ;
begin
    process1: process (clk,reset)
    begin
        if (reset ='1') then state <=s1;
        elsif (clk='1' and clk'Event) then
            case state is
                when s1 =>  if x1='1' then
                                state <= s2;
                            else
                                state <= s3;
                            end if;
                when s2 => state <= s4;
                when s3 => state <= s4;
                when s4 => state <= s1;
            end case;
        end if;
    end process process1;
```

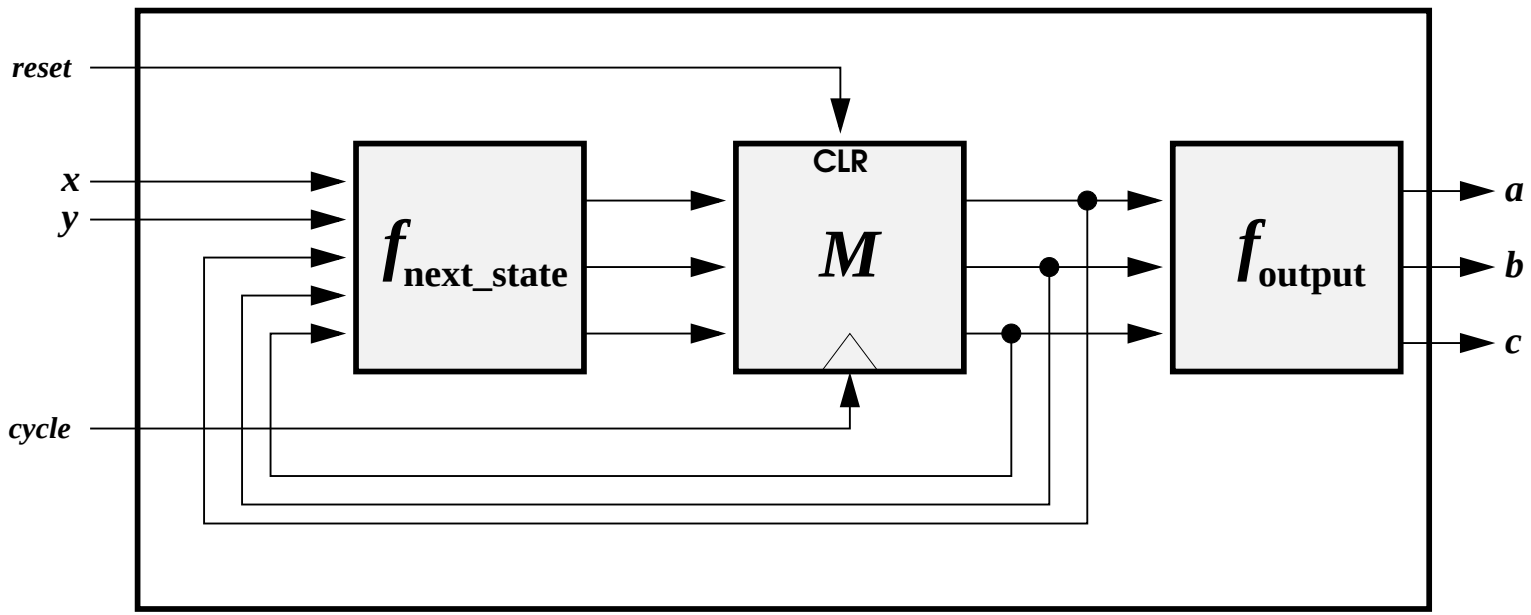
Example: Sequential Circuit (VHDL) — 3

... then the second process.

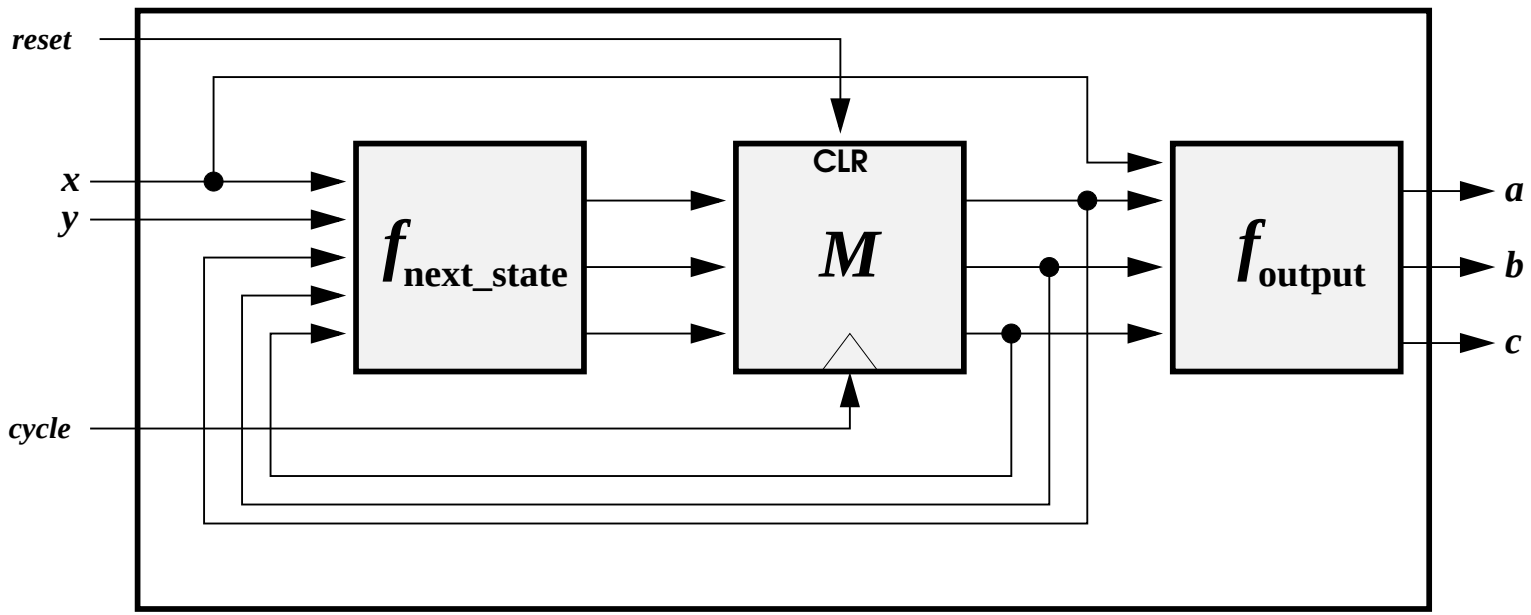
```
    process2 : process (state)
    begin
        case state is
            when s1 => outp <= '1';
            when s2 => outp <= '1';
            when s3 => outp <= '0';
            when s4 => outp <= '0';
        end case;
    end process process2;
end beh1;
```

(last line ends the architecture blocked started on previous slide)

Moore-type Machine



Mealy-type Machine



Example: Compute 'next day'

- given the current day of the week, compute the next day
- start by defining semantics and encoding the inputs and outputs

Day Encoding	
MONDAY	000
TUESDAY	001
WEDNESDAY	010
THURSDAY	011
FRIDAY	100
SATURDAY	101
SUNDAY	110

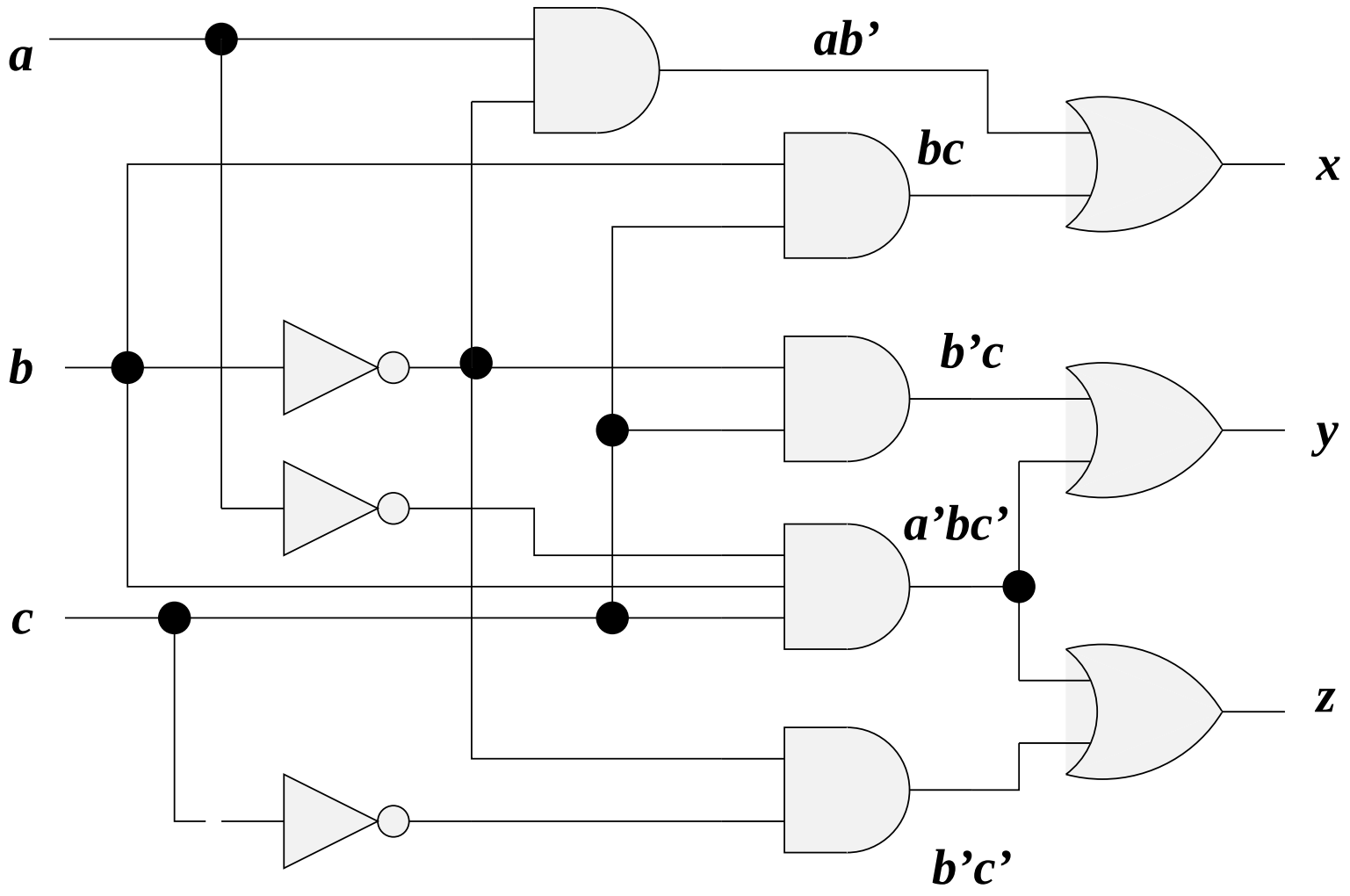
Determine Equations

- next, define the operation by relating the inputs and outputs with Boolean functions
- (minimize)

Governing Equations

$$\begin{aligned} x &= ab' + bc \\ y &= b'c + a'bc' \\ z &= b'c' + a'bc' \end{aligned}$$

Next-Day Circuit



What If We Changed the Encoding?

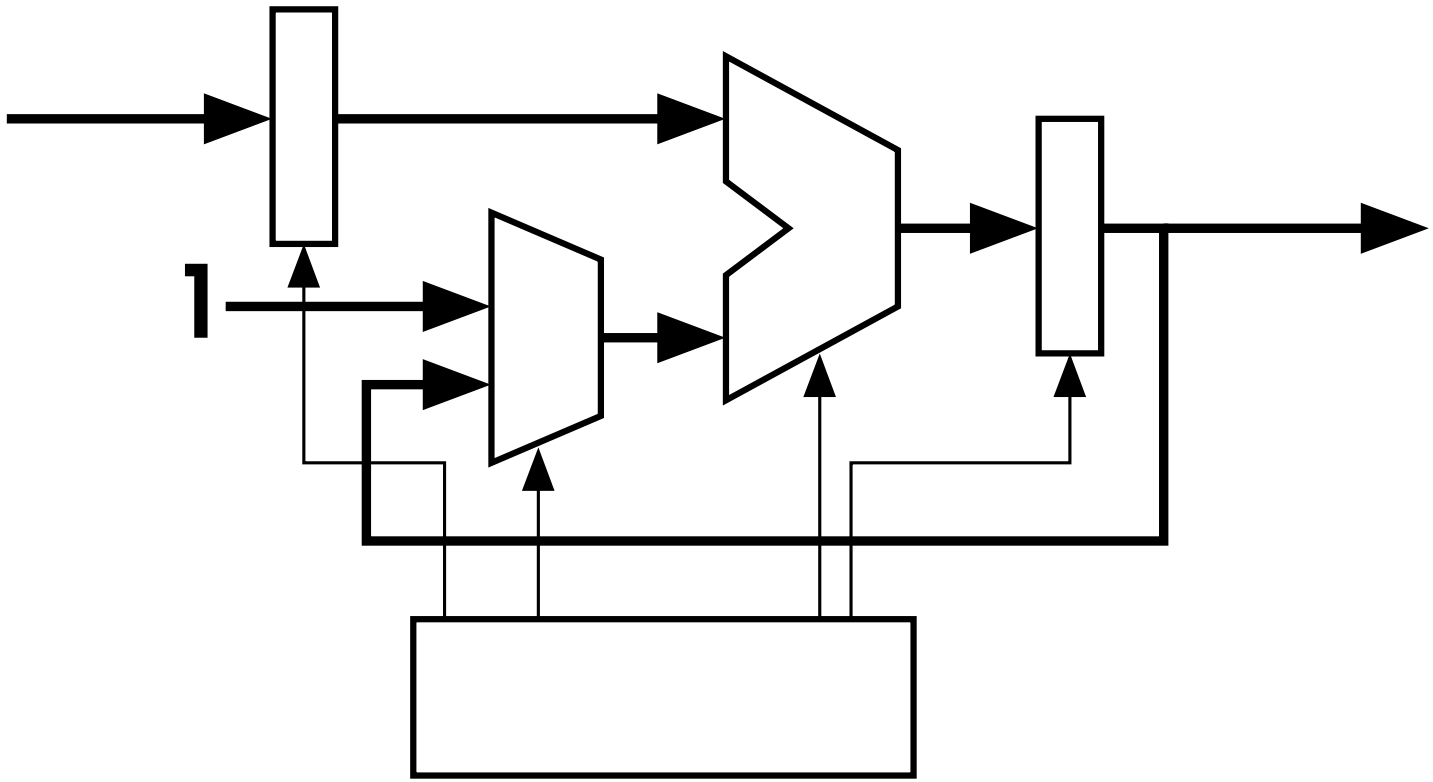
Day	Output Encoding
Monday	110
Tuesday	000
Wednesday	001
Thursday	010
Friday	011
Saturday	100
Sunday	101

Block Diagrams

- as the number of states grows from 0 to $5 - 10$, this analysis tends to overwhelm the human mind
- the solution is abstract and build hierarchically
  1. design a simple component and give it a name
  2. assemble systems from components

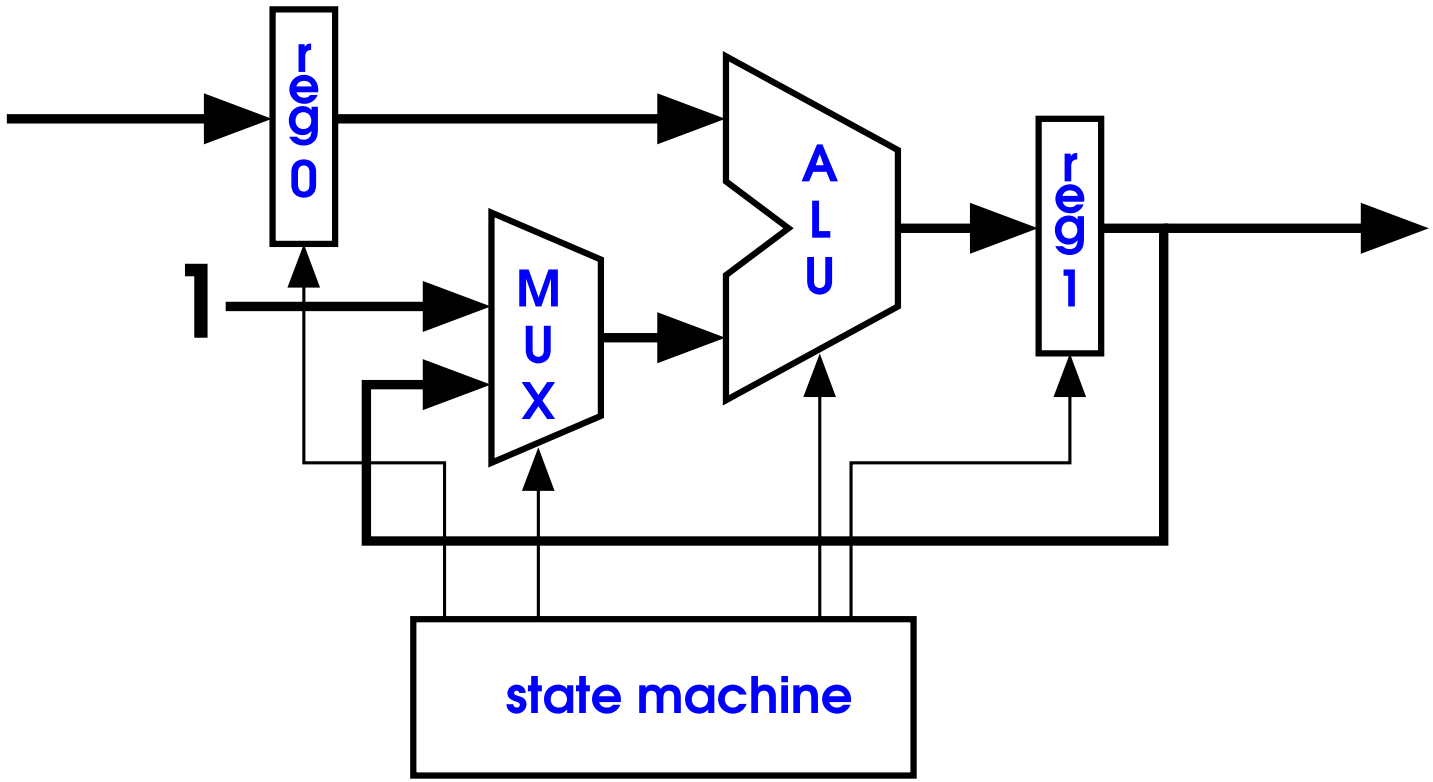
Example: Block Diagram

- a common example... recognize it?



Example: Block Diagram

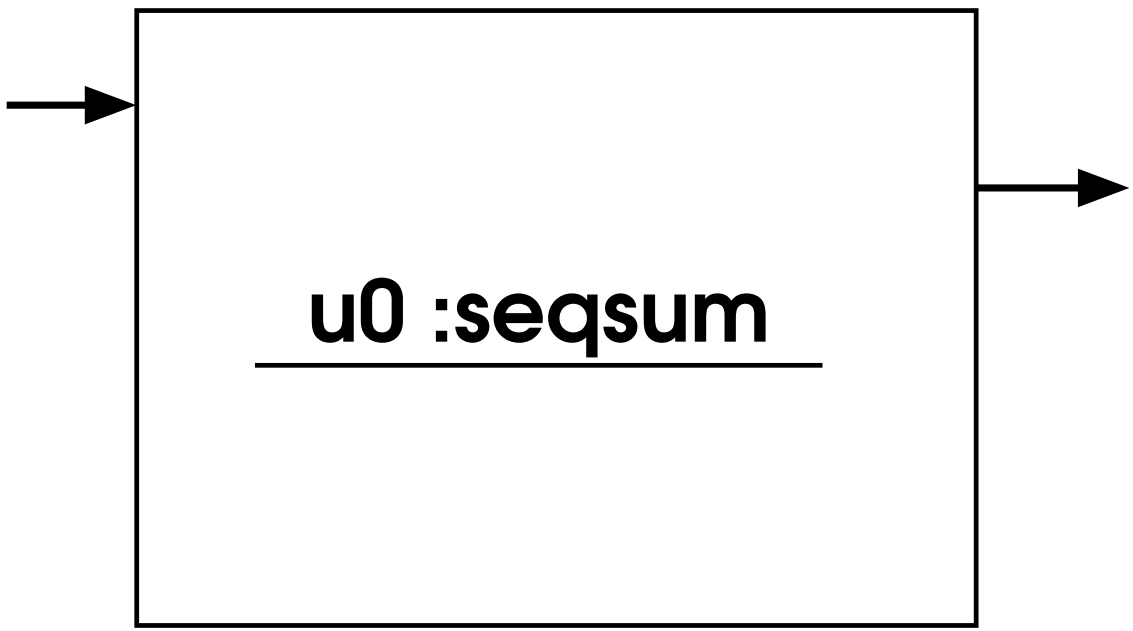
- a common example... recognize it?



- (a somewhat customized) a datapath

Hierarchy

- Lastly, we can take systems designed from components and give them names to make new compoents



- and so on...

Structural VHDL

insert VHDL example here!

```
library IEEE;
use IEEE.std_logic_1164.all;
entity sumseq is
    port ( clk, reset, x : IN std_logic_vector(31 downto 0);
           sum           : OUT std_logic_vector(31 downto 0));
    component ...
    component ...
    component ...
end entity;
```