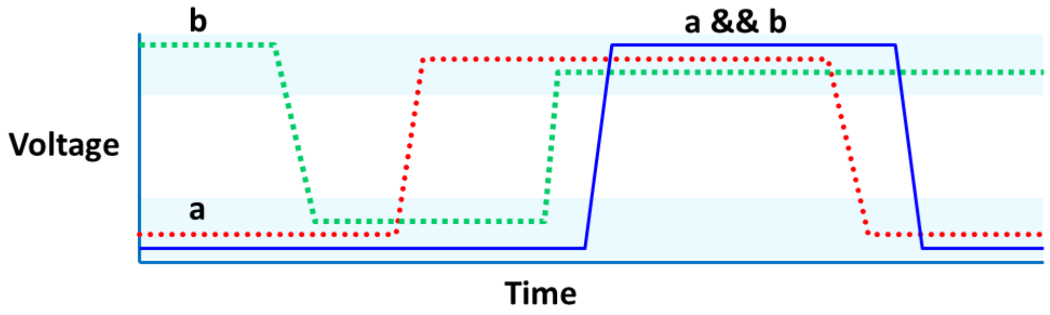
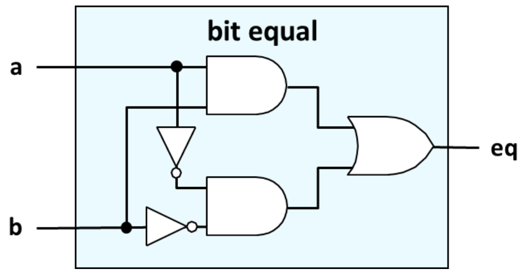
Processor Architecture

15 min Logic Design Recap



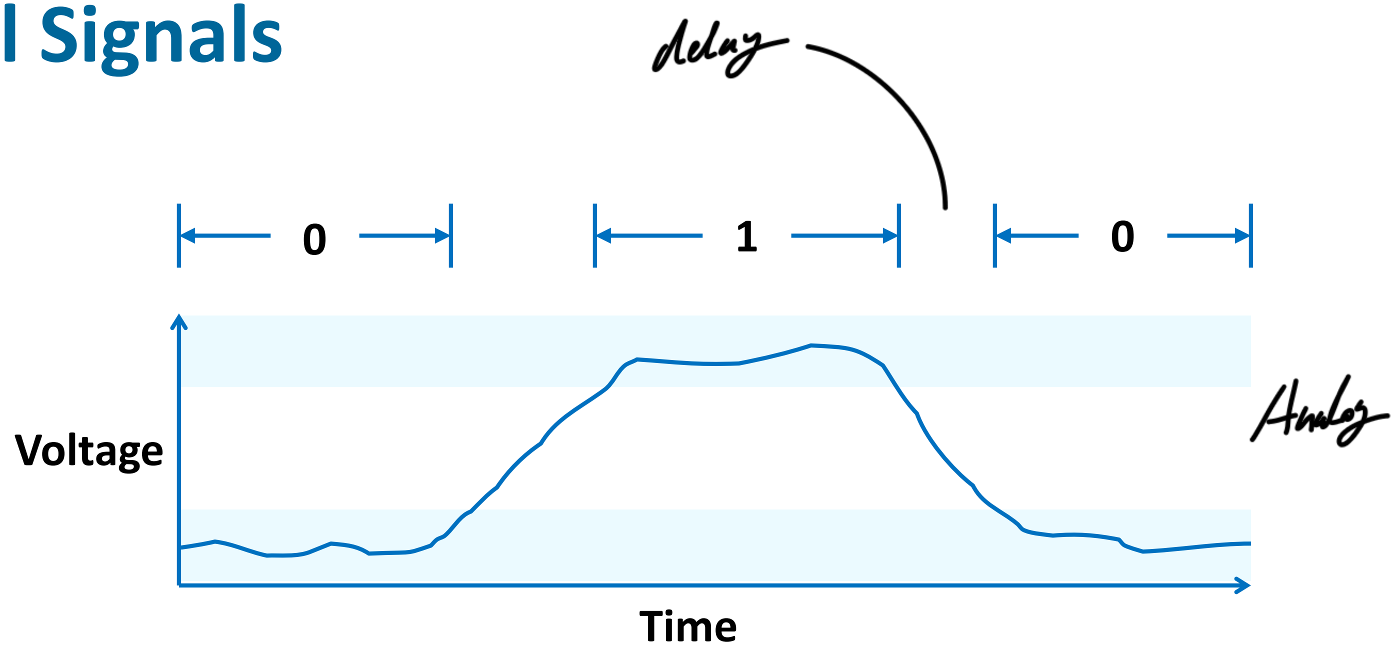
Module Outline

- Combinational Circuits
- Sequential Circuits
- Sequential Operation
- Module Summary



Combinational Circuits

Digital Signals

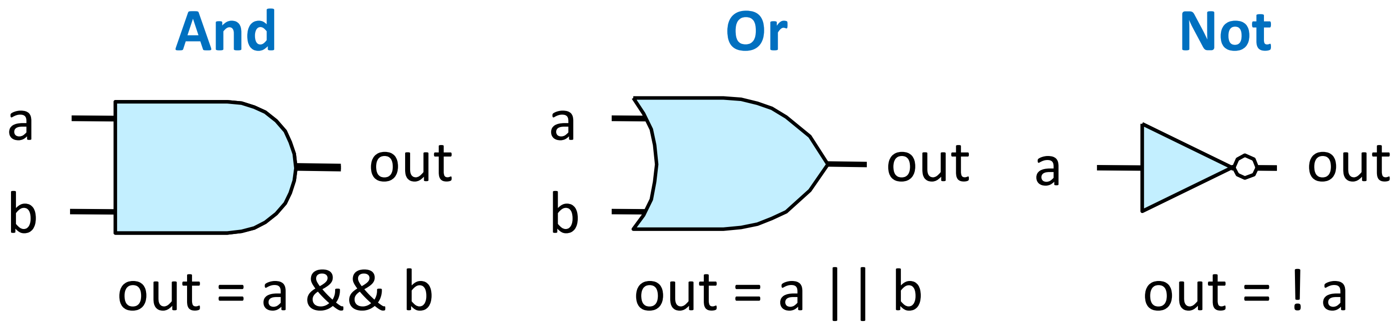


- Use voltage thresholds to extract discrete values from continuous signal

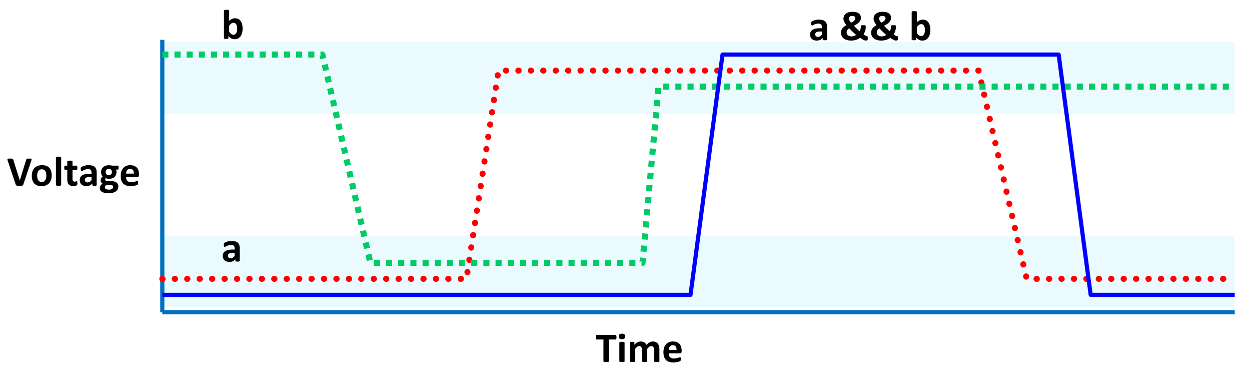
- Simplest version: 1-bit signal
  - Either high range (1) or low range (0)
  - With guard range between them

- Not strongly affected by noise or low quality circuit elements
  - Can make circuits simple, small, and fast

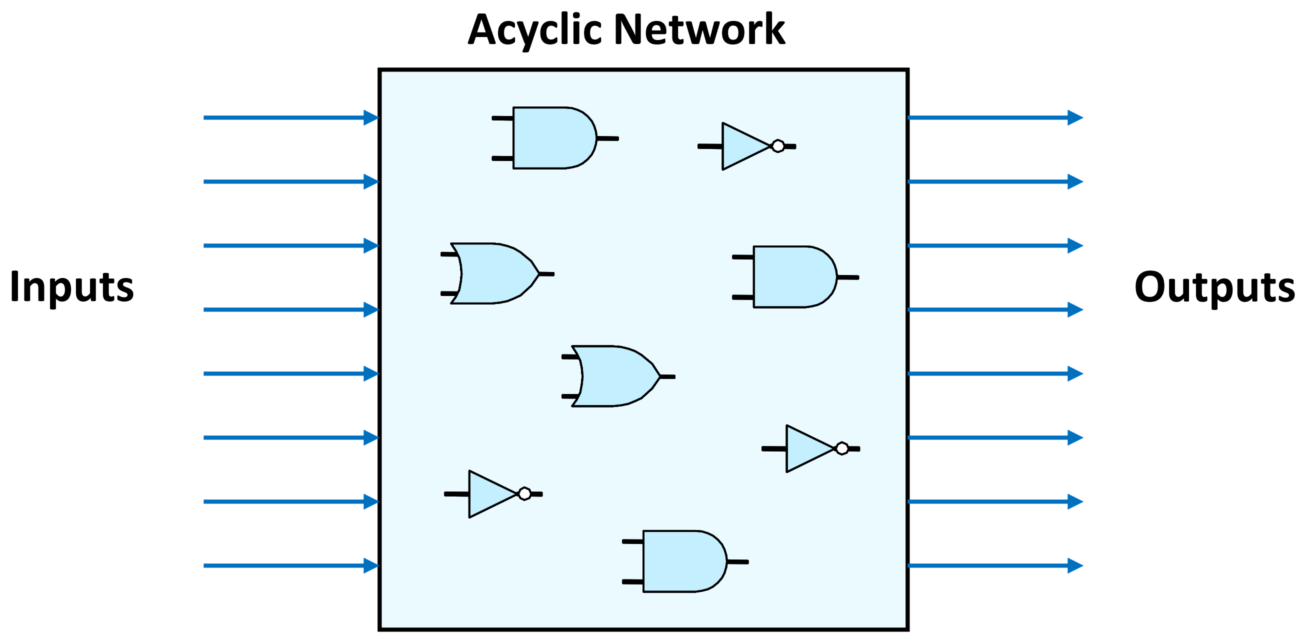
Computing with Logic Gates



- Outputs are Boolean functions of inputs
- *Continuously* respond to changes in inputs
  - With some small delay

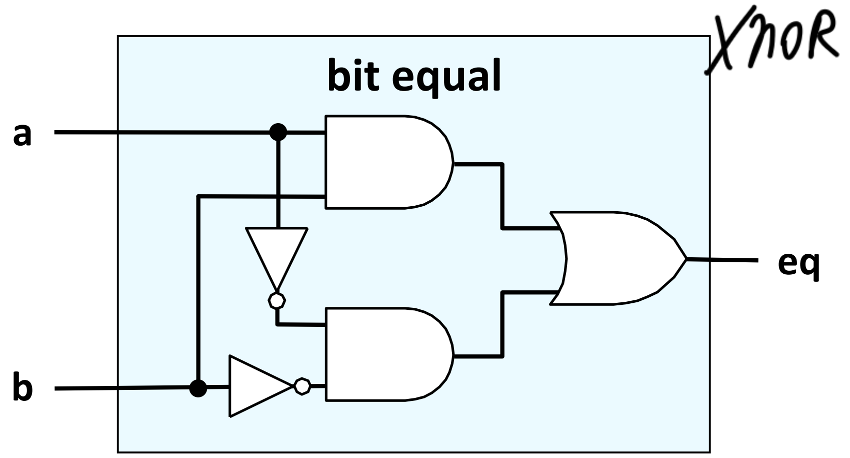


Combinational Circuits



- Acyclic network of logic gates
  - Continuously responds to changes on inputs
  - Outputs become (after some delay) boolean functions of inputs

Bit Equality Circuit



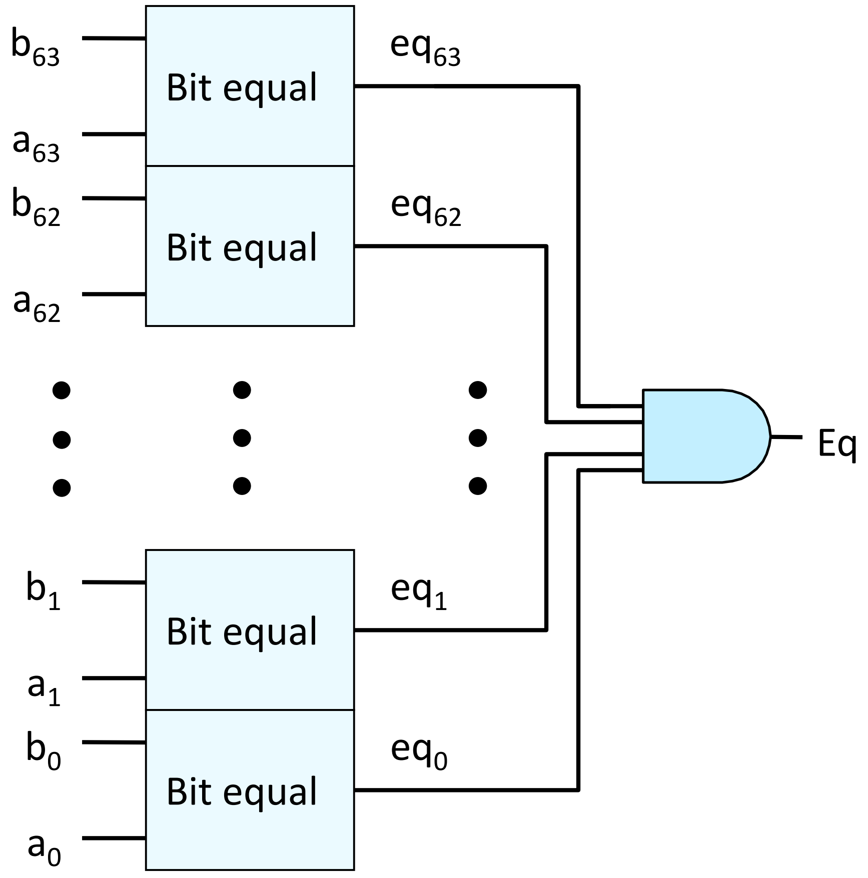
Generate 1 iff a and b are equal

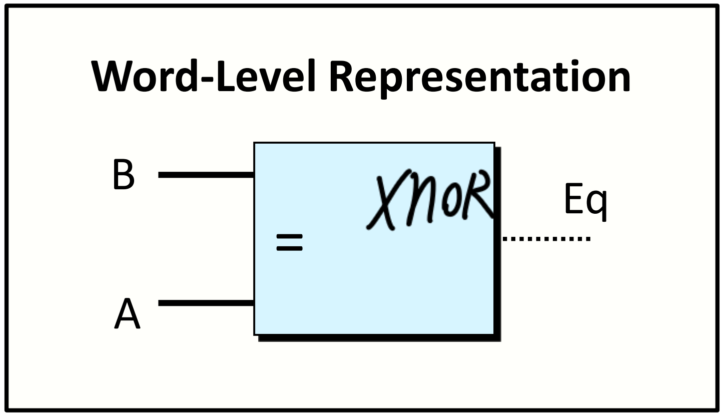
HDL Expression

```
bool eq = (a&&b)||(!a&&!b)
```

- Hardware Description Language (HDL)
  - Hardware description language
    - Boolean operations have syntax similar to C logical operations
  - Verilog, VHDL, Bluespec, …

Word Equality Circuit



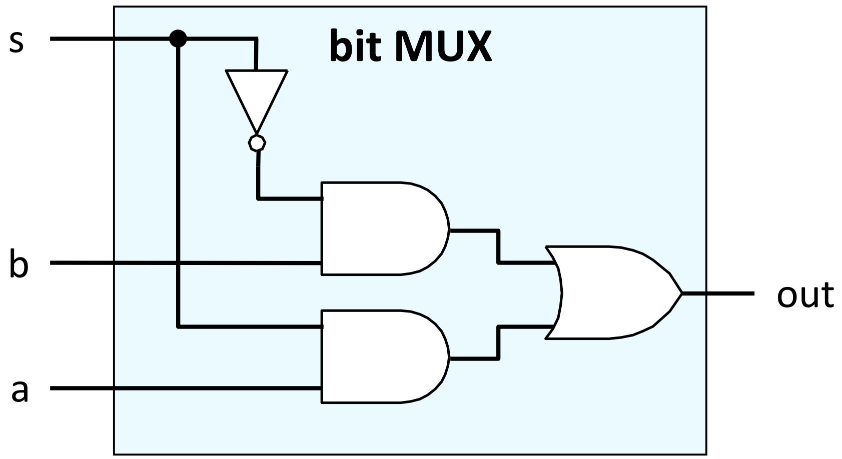


HDL Representation

```
bool Eq = (A == B)
```

- 64-bit word size
- HDL representation
  - Equality operation
  - Generates Boolean value

Bit-Level Multiplexor

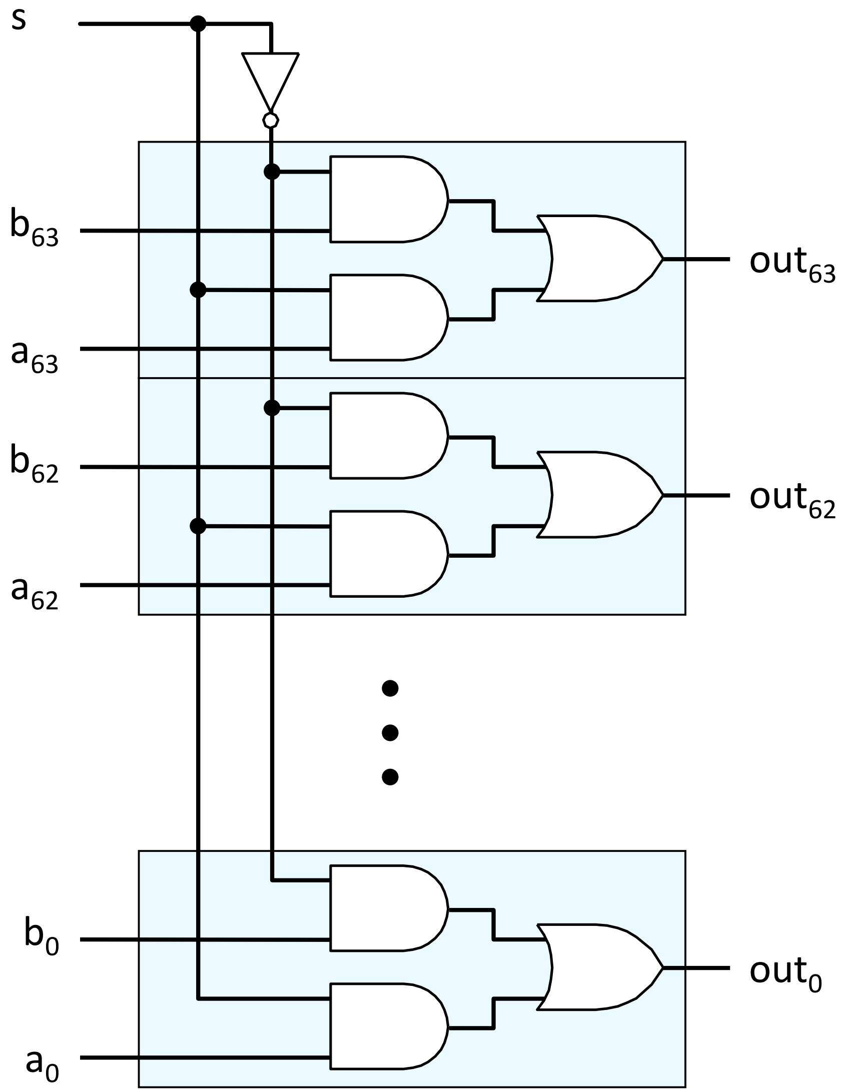


HDL Expression

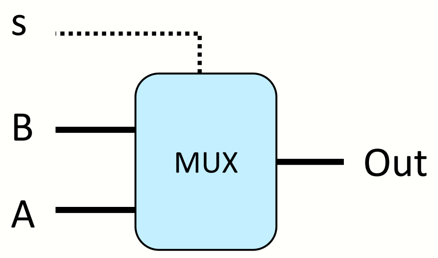
```
bool out = (s&&a)||(!s&&b)
```

- Input
  - Control signal s
  - Data signals a and b
- Output
  - s=1: a
  - s=0: b

Word Multiplexor



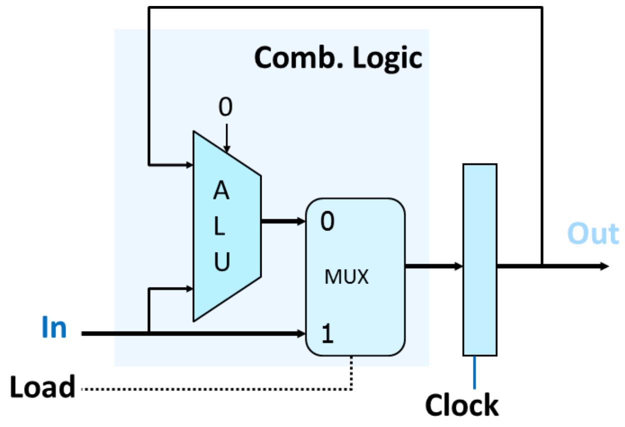
Word-Level Representation



HDL Representation

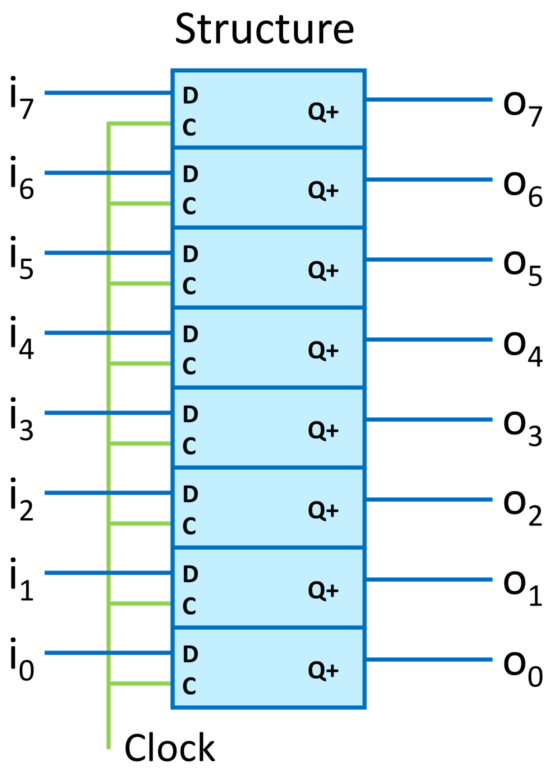
```
int Out = [
  s : A;
  1 : B;
];
```

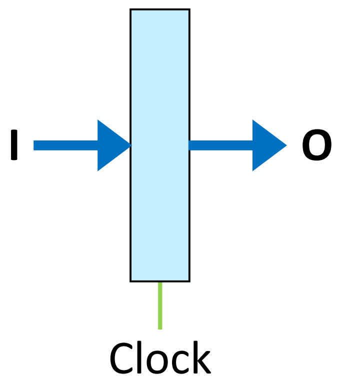
- Select input word A or B depending on control signal s
- HDL representation
  - Case expression
  - Series of test : value pairs
  - Output value of ***first successful test***



Sequential Circuits

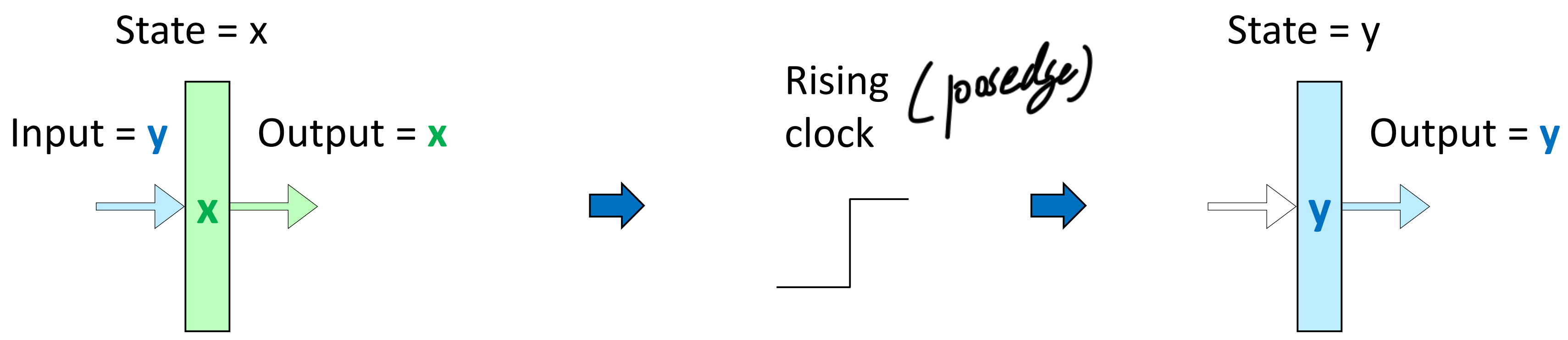
Registers

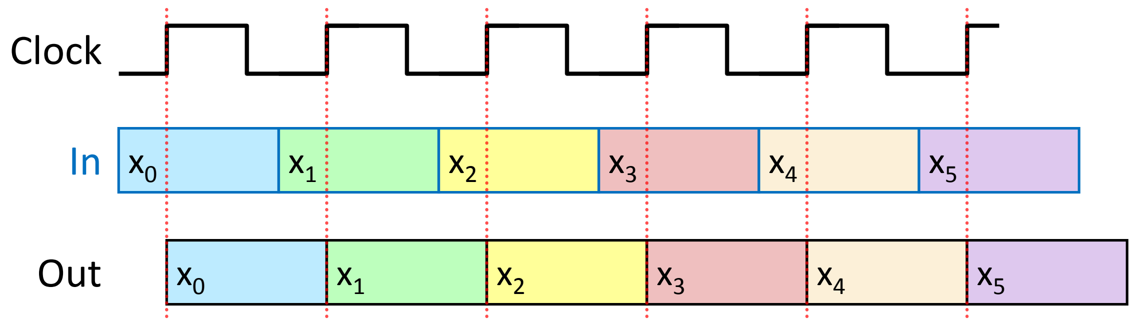




- Stores word of data *for one cycle*
  - Different from *program registers* seen in assembly code
- Loads input on rising edge of clock
- Acts as barrier between input and output

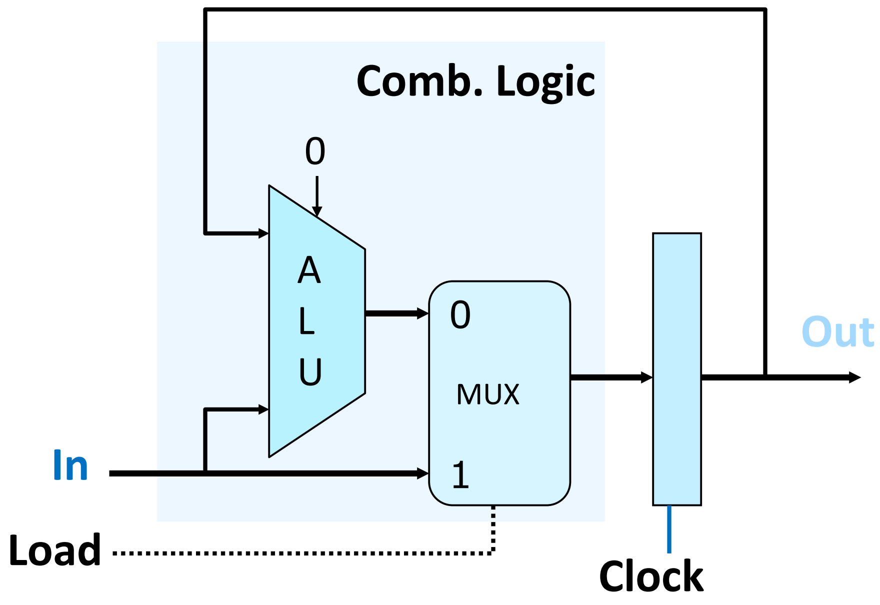
Register Operation

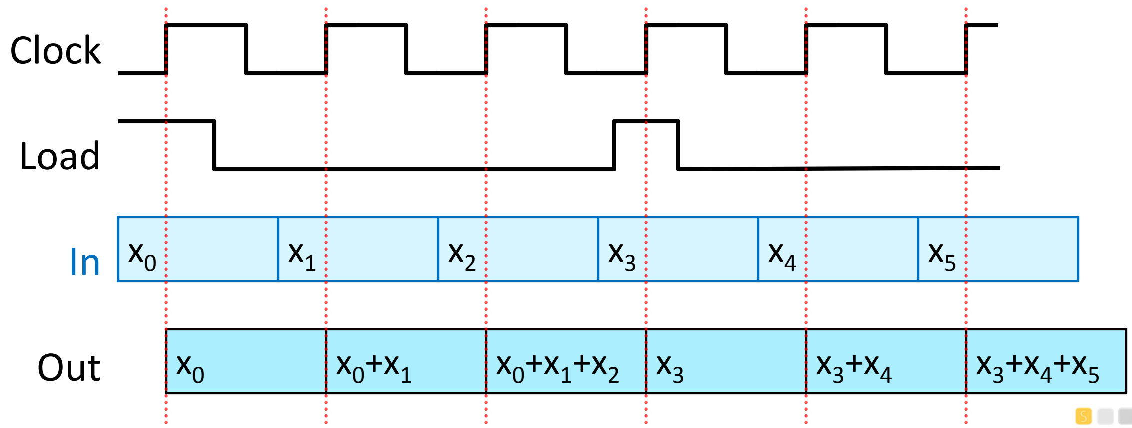




State Machine Example

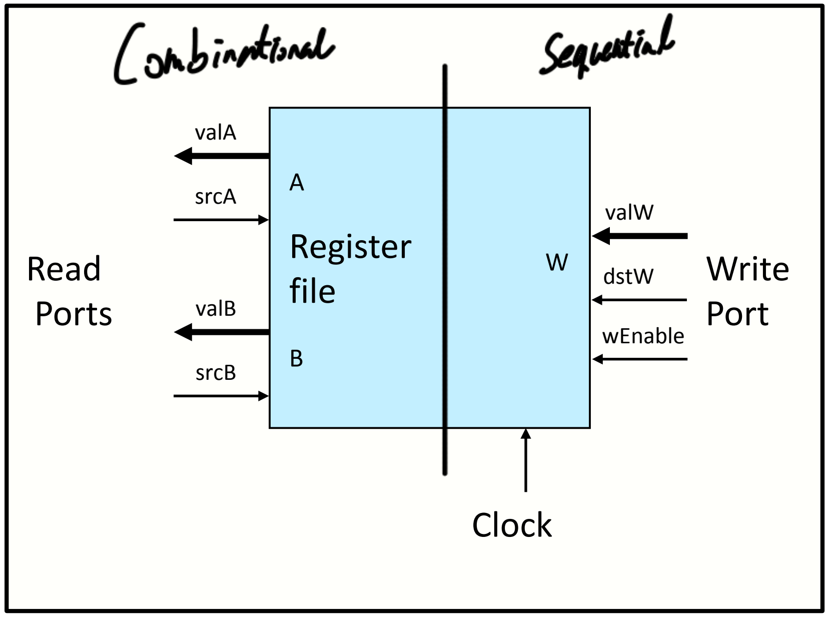
- **Accumulator circuit**
  - Load or accumulate on each cycle





Random-Access Memory

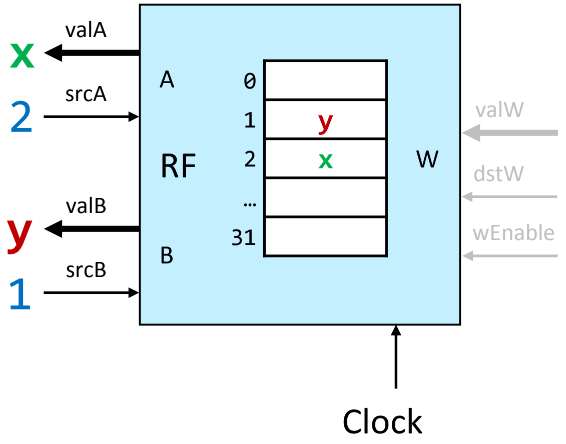
- Stores multiple words of memory
  - Address input specifies which word to read or write
- Register file
  - Holds values of program registers
  - x1, x2, etc.
  - Register index serves as address



- Multiple Ports
  - Can read and/or write multiple words in one cycle
    - Each has separate address and data input/output

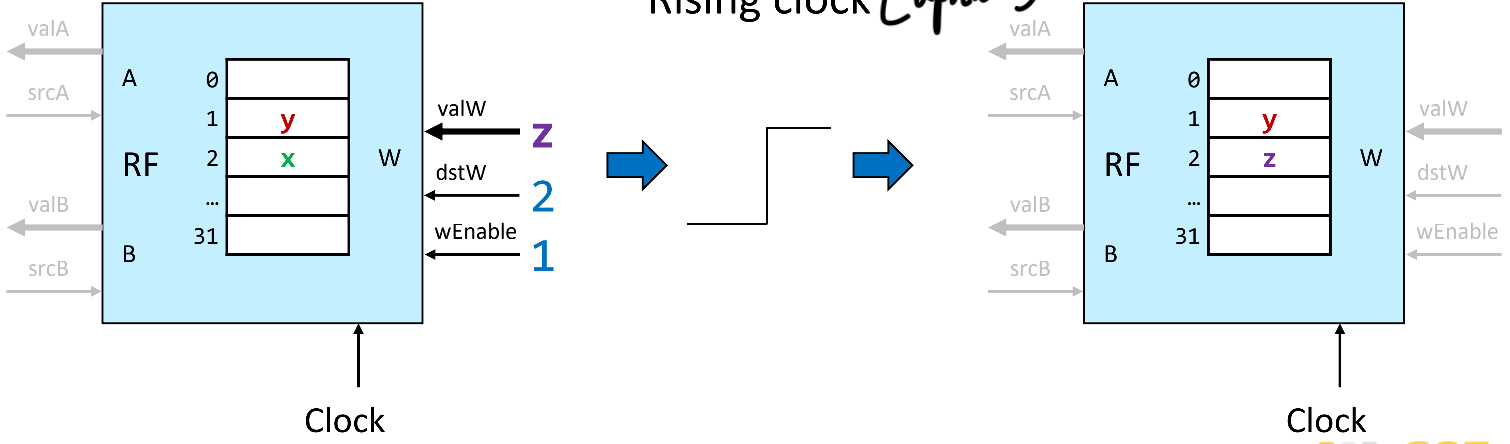
Register File Timing

- **Reading**
  - Like combinational logic
  - Output data generated based on input address (after some delay)

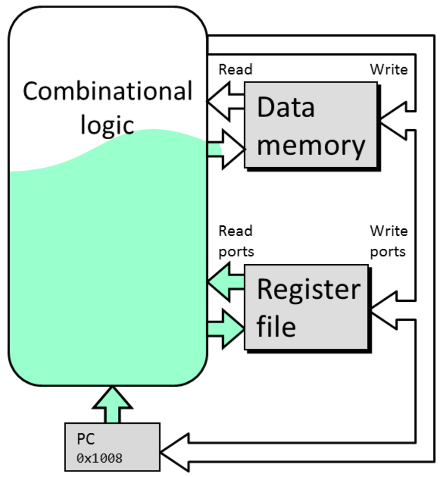


- **Writing**
  - Like register (Sequential)
  - Update only as clock rises

Rising clock (update)

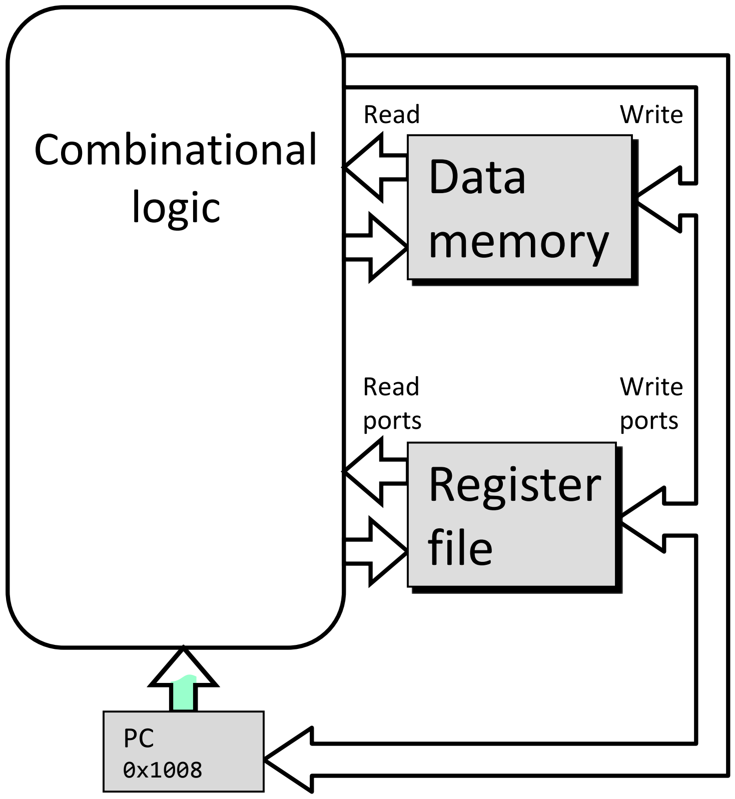


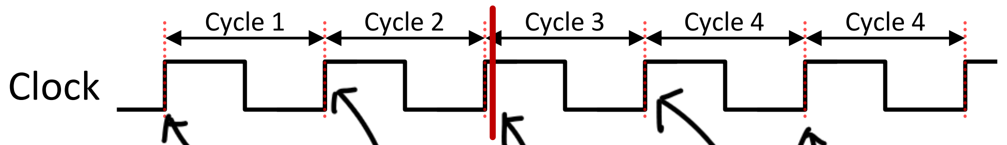
CSE 컴퓨터공학부 Department of Computer Science & Engineering



Sequential Operation

Sequential Operation: Executing an Instruction

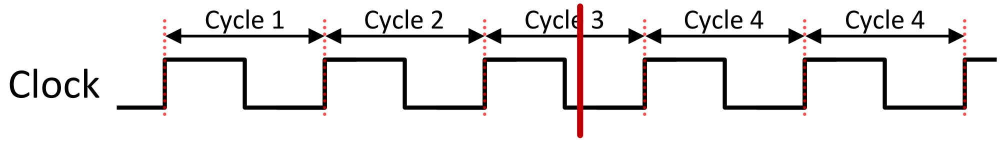




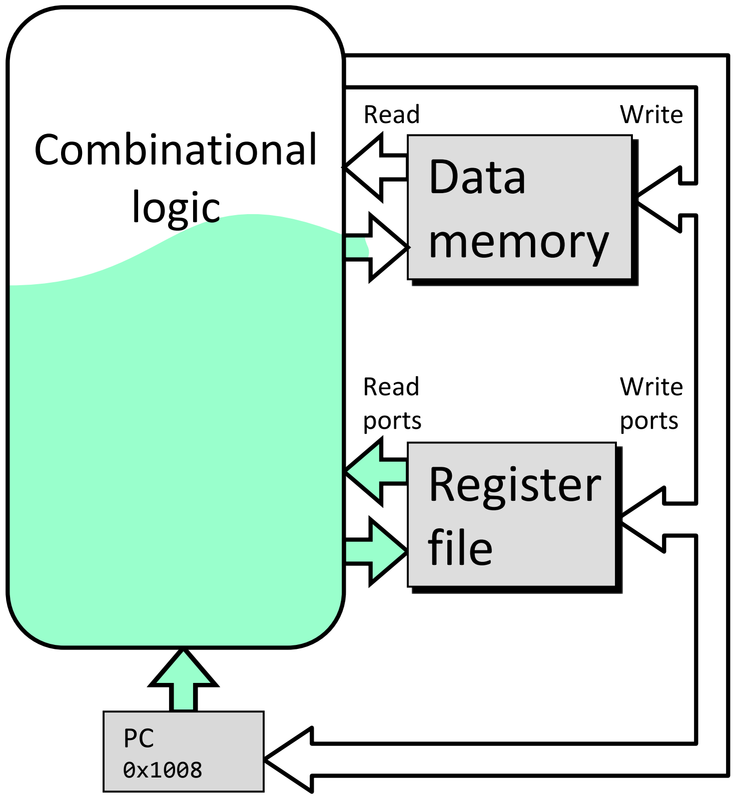
Cycle 1	`0x1000: add`	`a1,a5,a1`
Cycle 2	`0x1004: slli`	`a1,a1,0x4`
Cycle 3	`0x1008: add`	`a0,a0,a1`
Cycle 4	`0x100c: sub`	`a0,a2,a0`
Cycle 5	`0x1010: ret`	

- Timing: immediately after start of cycle 3 (rising clock edge)
- Combinational logic starting to react to state changes

Sequential Operation: Executing an Instruction

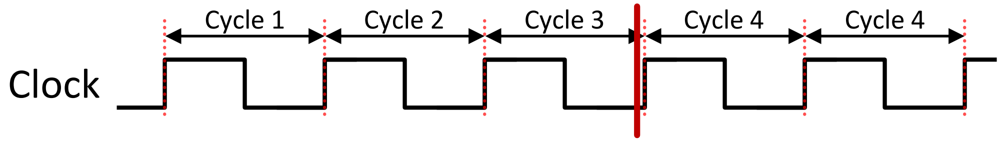


Cycle 1	`0x1000: add a1,a5,a1`
Cycle 2	`0x1004: slli a1,a1,0x4`
Cycle 3	`0x1008: add a0,a0,a1`
Cycle 4	`0x100c: sub a0,a2,a0`
Cycle 5	`0x1010: ret`

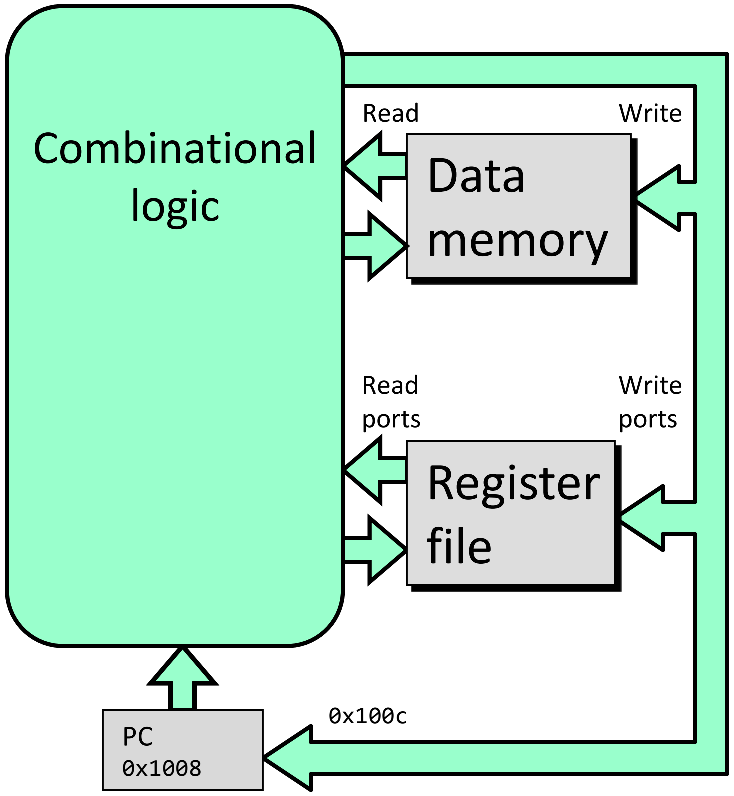


- Combinational logic generates results for current instruction (add)
- Not all results may have been generated in the middle of the cycle

Sequential Operation: Executing an Instruction



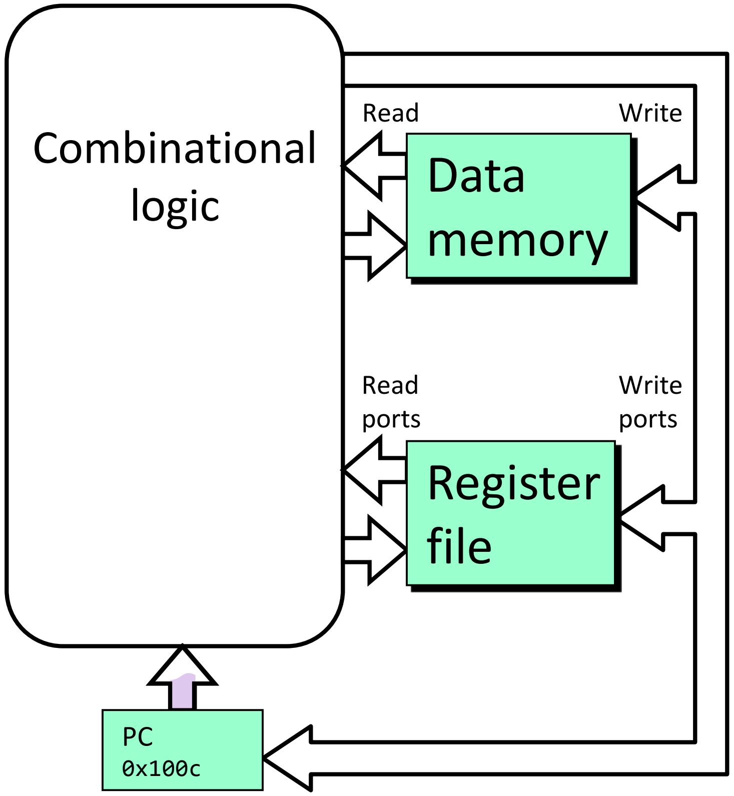
Cycle 1	`0x1000: add a1,a5,a1`
Cycle 2	`0x1004: slli a1,a1,0x4`
Cycle 3	`0x1008: add a0,a0,a1`
Cycle 4	`0x100c: sub a0,a2,a0`
Cycle 5	`0x1010: ret`

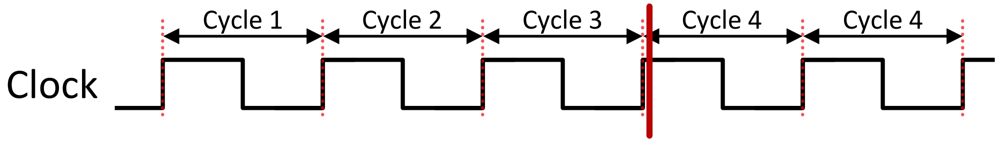


- Combinational logic generates results for current instruction (add)
- All results must be generated and have arrived at the clocked elements before the end of the cycle

(waiting for update)

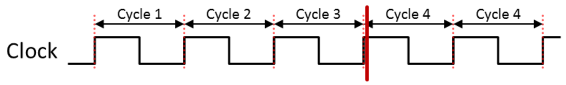
Sequential Operation: Executing an Instruction





Cycle 1	`0x1000: add`	`a1,a5,a1`
Cycle 2	`0x1004: slli`	`a1,a1,0x4`
Cycle 3	`0x1008: add`	`a0,a0,a1`
Cycle 4	`0x100c: sub`	`a0,a2,a0`
Cycle 5	`0x1010: ret`	

- Rising clock edge to start cycle 4
- State set according to add instruction
- Combinational logic starting to react to state changes (=execute next instruction)



Cycle 1	`0x1000: add a1,a5,a1`
Cycle 2	`0x1004: slli a1,a1,0x4`
Cycle 3	`0x1008: add a0,a0,a1`
Cycle 4	`0x100c: sub a0,a2,a0`
Cycle 5	`0x1010: ret`

Module Summary

Module Summary – Logic Design

- **Combinational circuits**
  - Desired operation implemented with a combination of basic logic gates
  - Acyclic circuits
  - Continuous operation – state propagates from inputs to outputs
- **Sequential circuits**
  - Store state
  - State chance only in reaction to a particular event
    - Such as the rising edge of a clock signal
- **Processors are made up from a combination of combinational and sequential circuits**