Overview of Logic Design

CS2011: Introduction to Computer Systems
Lecture 16 (4.2)

Overview of Logic Design

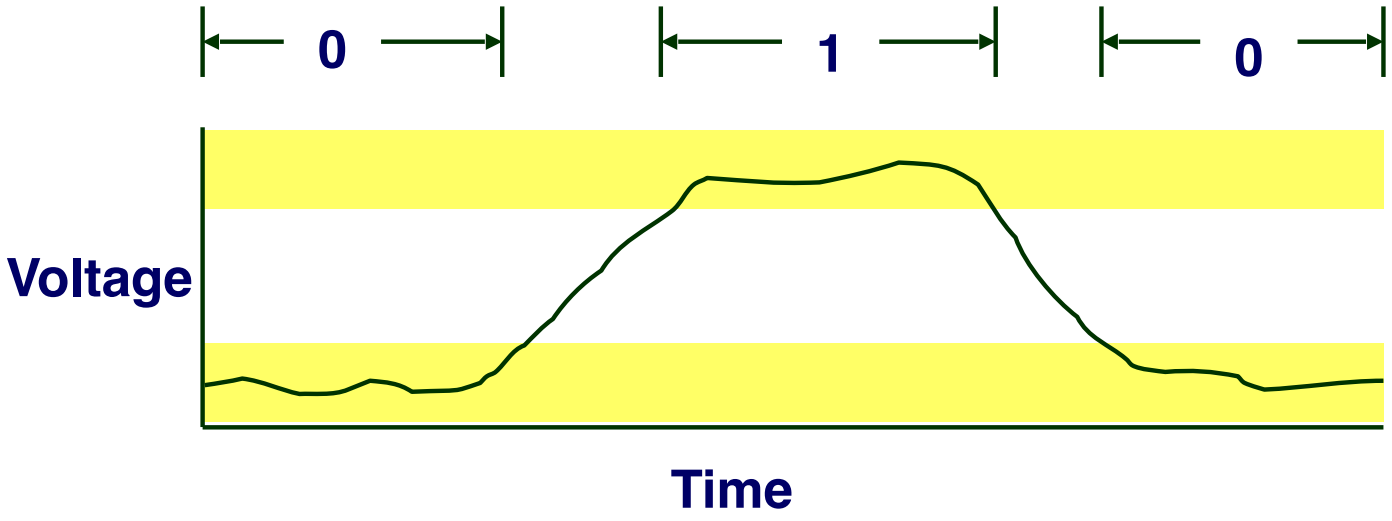
Fundamental Hardware Requirements

- Communication
  - How to get values from one place to another
- Computation
- Storage

Bits are Our Friends

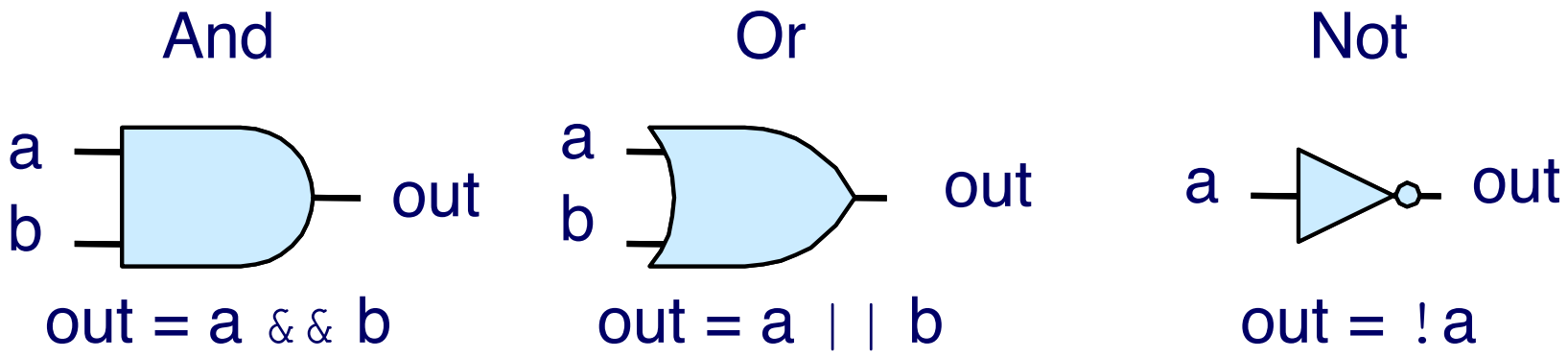
- Everything expressed in terms of values 0 and 1
- Communication
  - Low or high voltage on wire
- Computation
  - Compute Boolean functions
- Storage
  - Store bits of information

Digital Signals

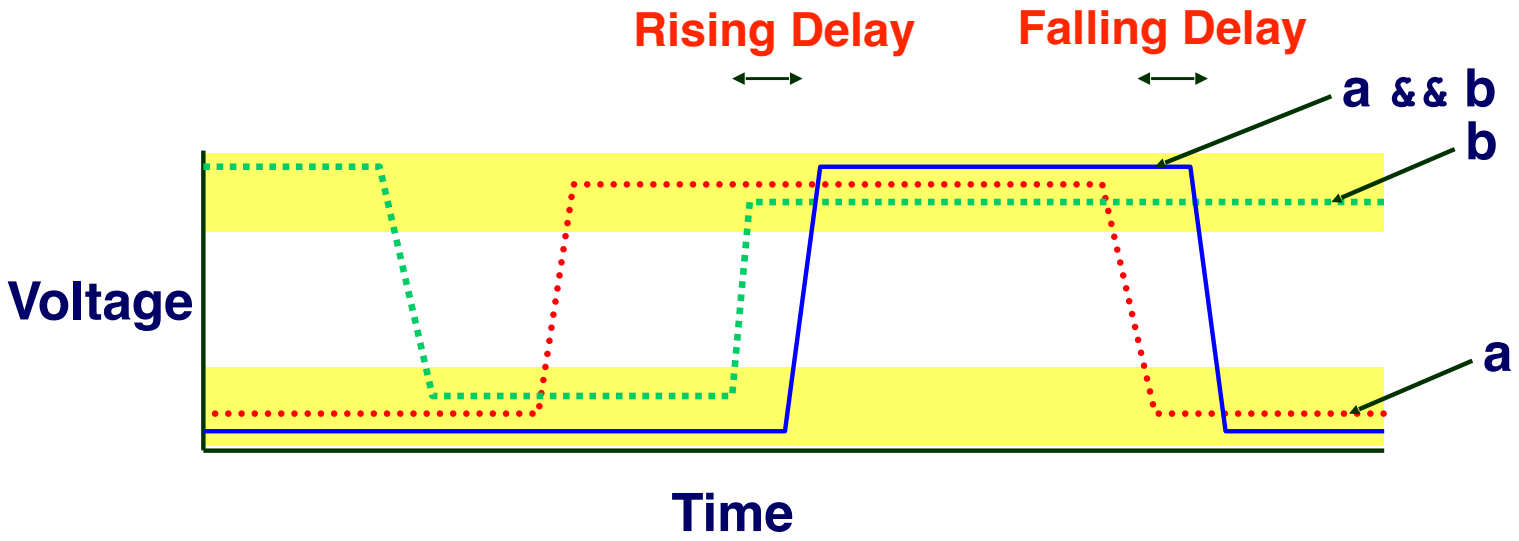


- Use voltage thresholds to extract discrete values from continuous signal
- Simplest version: 1-bit signal
  - Either high range (1) or low range (0)
  - With guard range between them
- Not strongly affected by noise or low quality circuit elements
  - Can make circuits simple, small, and fast

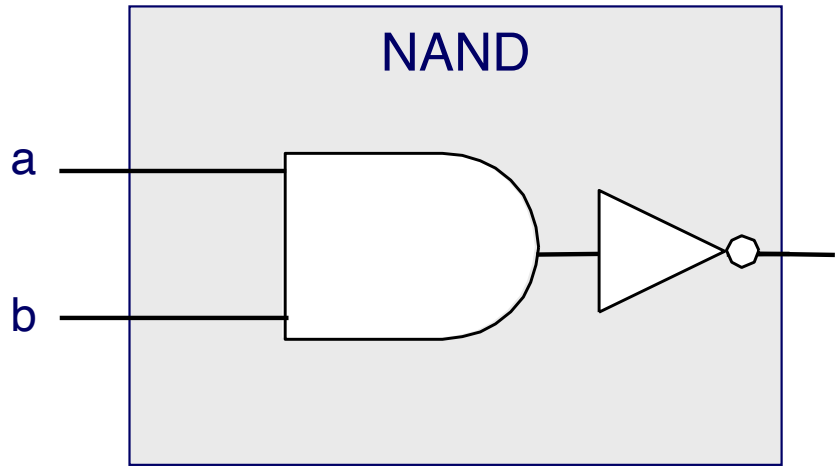
Computing with Logic Gates

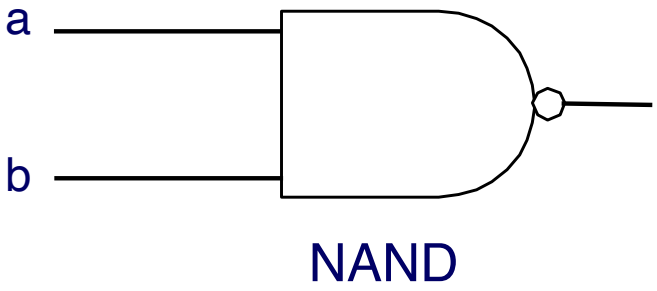


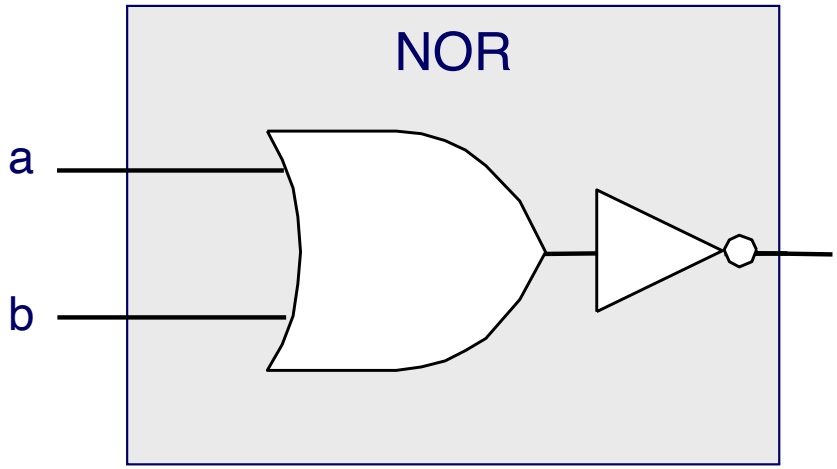
- **Outputs are Boolean functions of inputs**
- **Respond continuously to changes in inputs**
  - **With some, small delay**

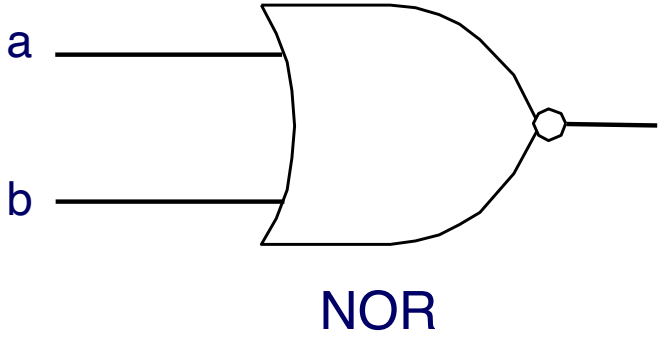


Computing with Logic Gates

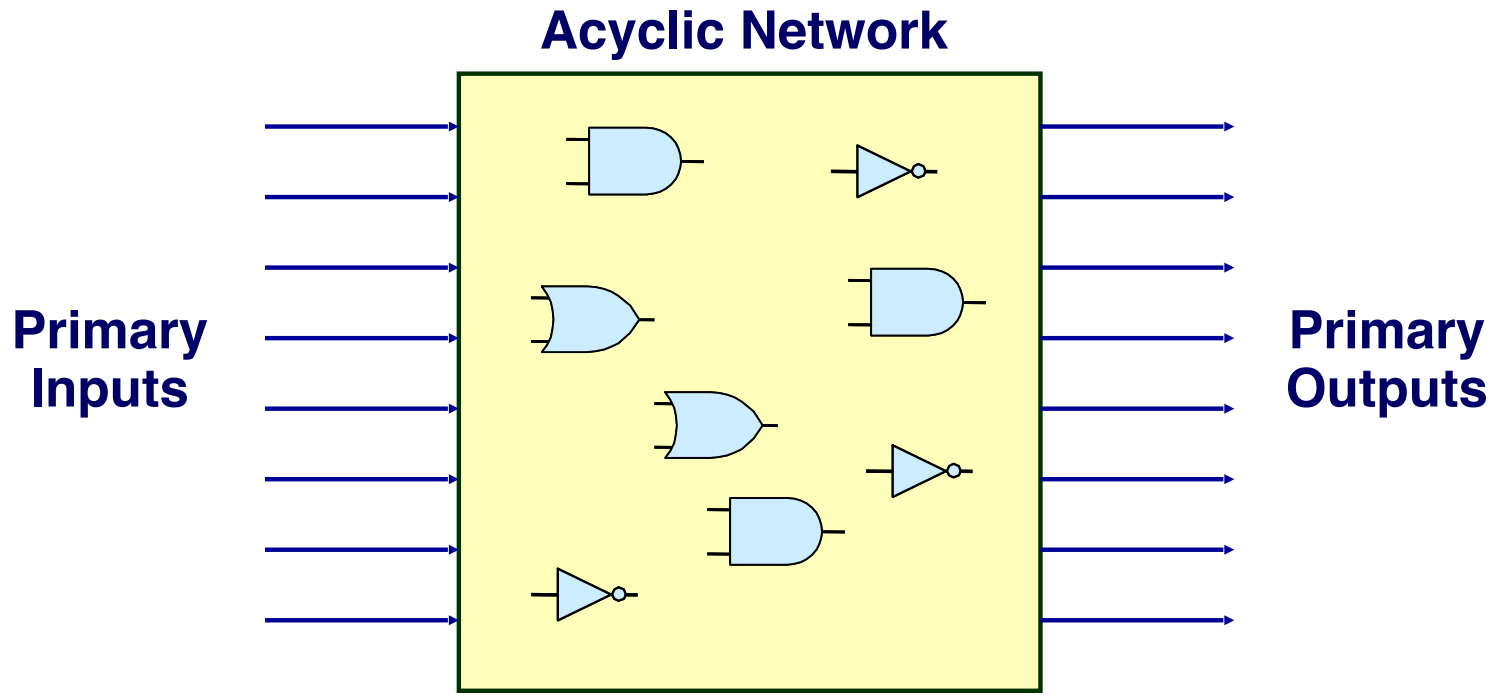








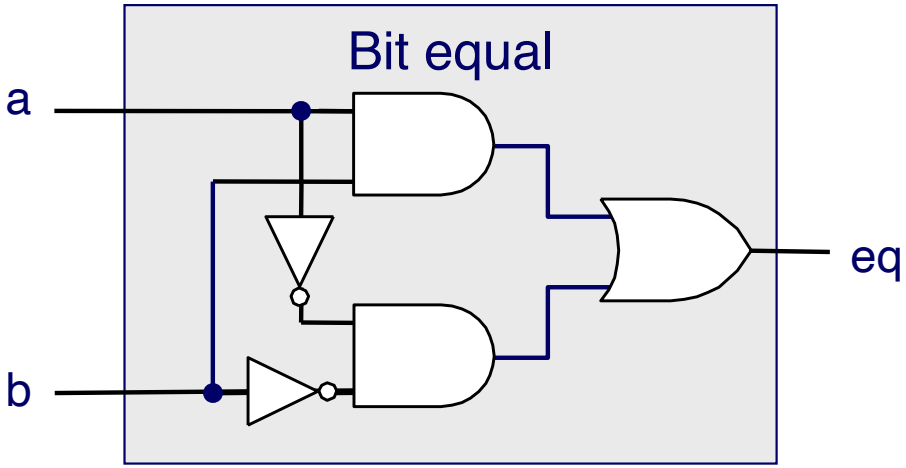
Combinational Circuits



Acyclic Network of Logic Gates

- Continously responds to changes on primary inputs
- Primary outputs become (after some delay) Boolean functions of primary inputs

Bit Equality



HCL Expression

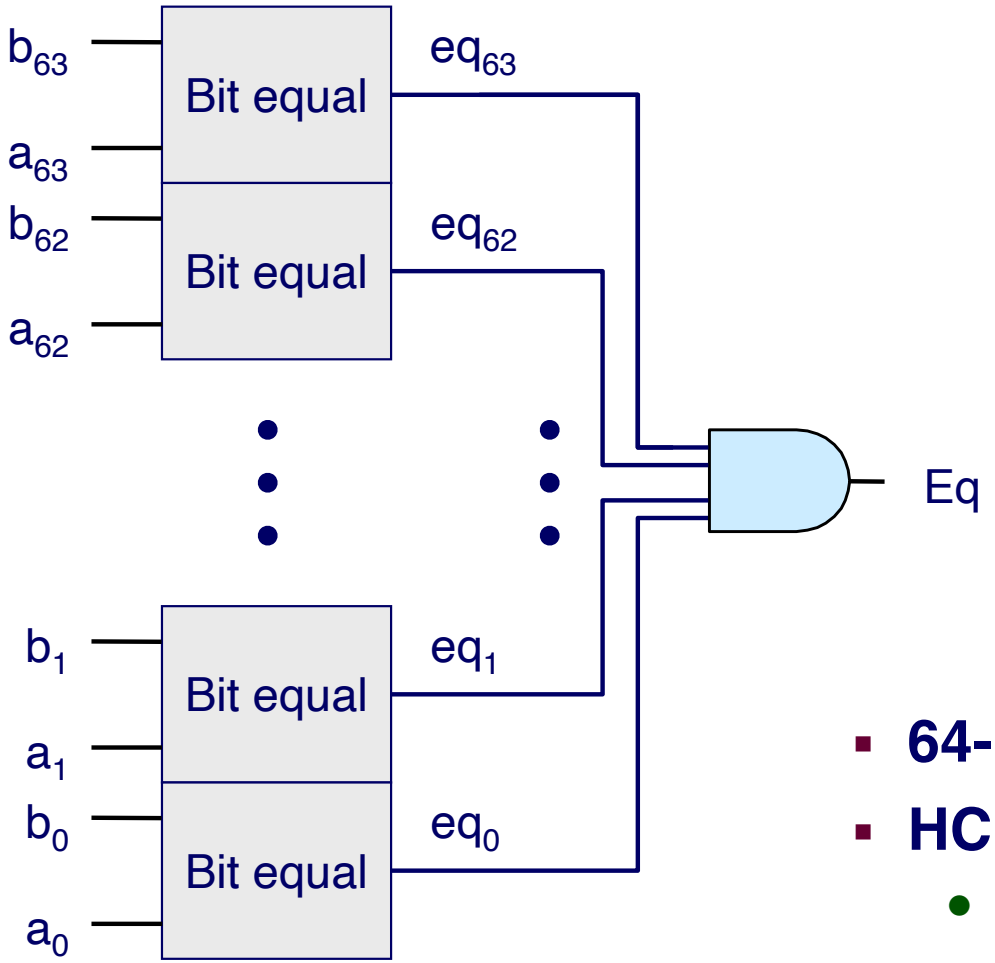
```
bool eq = (a&&b)||(!a&&!b)
```

- Generate 1 if a and b are equal

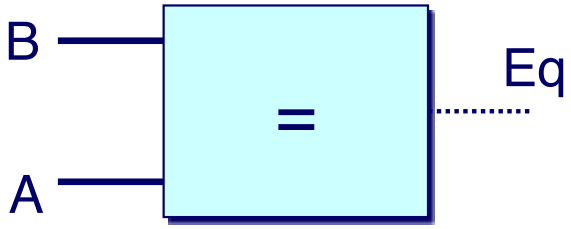
Hardware Control Language (HCL)

- Very simple hardware description language
  - Boolean operations have syntax similar to C logical operations
- We'll use it to describe control logic for processors

Word Equality



Word-Level Representation

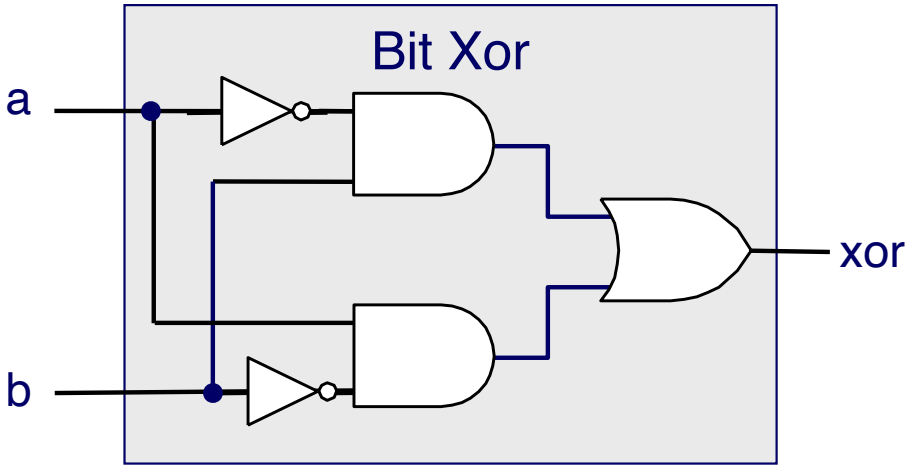


HCL Representation

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

- 64-bit word size
- HCL representation
  - Equality operation
  - Generates Boolean value (dotted line is 1 bit)

Bit Xor



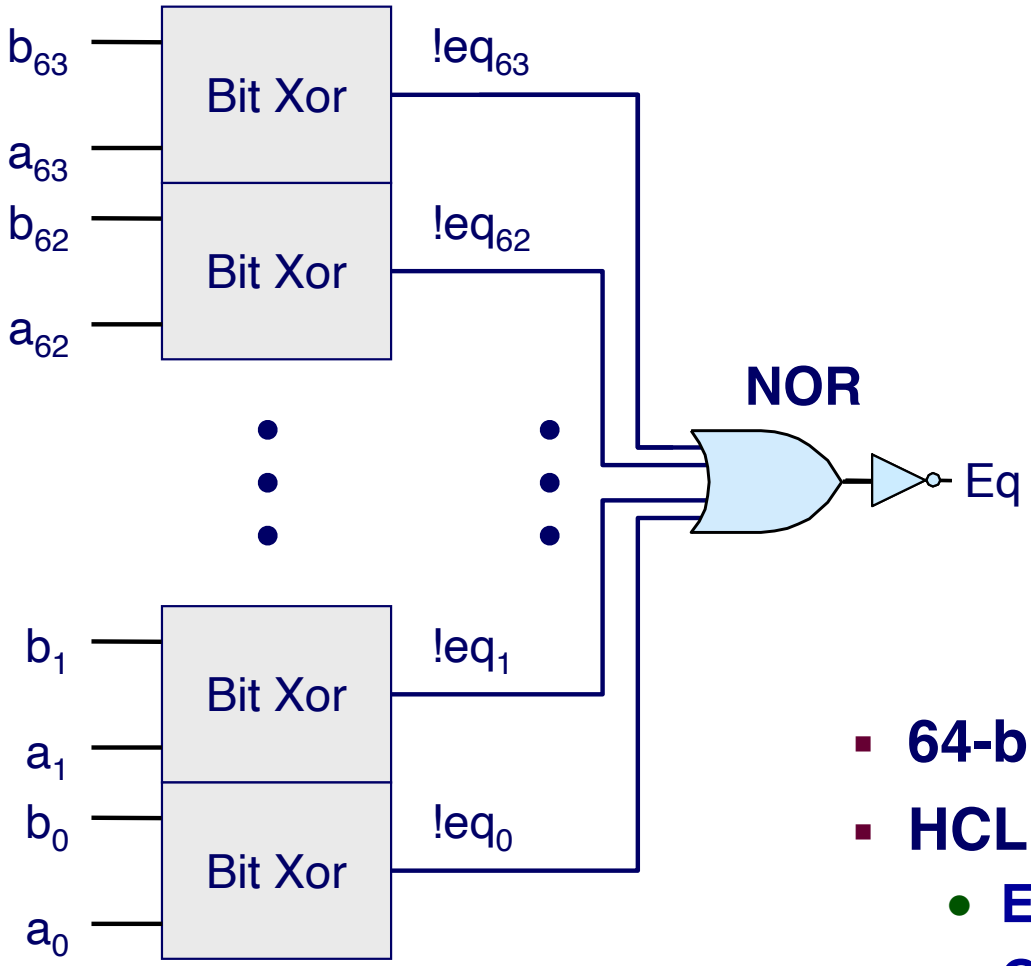
HCL Expression

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

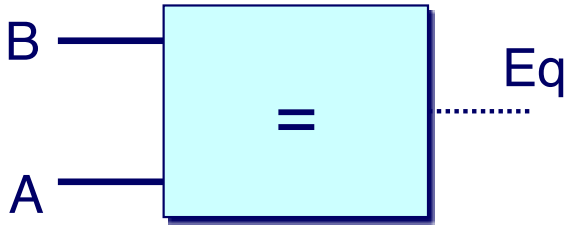
- **Generate 1 if a and b are not equal**
- **Output is complement of the bit equality value**

```
xor = !eq
```

Word Equality using Bit Xor



Word-Level Representation

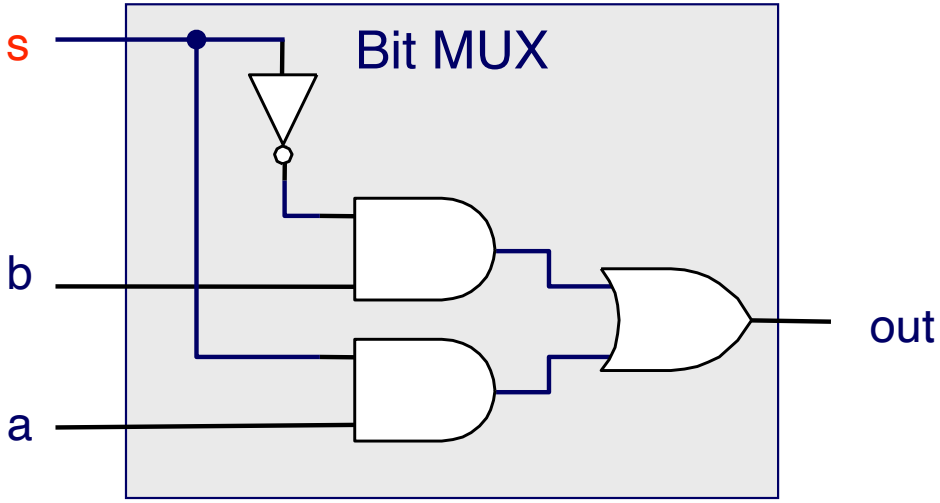


HCL Representation

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

- **64-bit word size**
- **HCL representation**
  - **Equality operation**
  - **Generates Boolean value (dotted line is 1 bit)**

Bit-Level Multiplexor

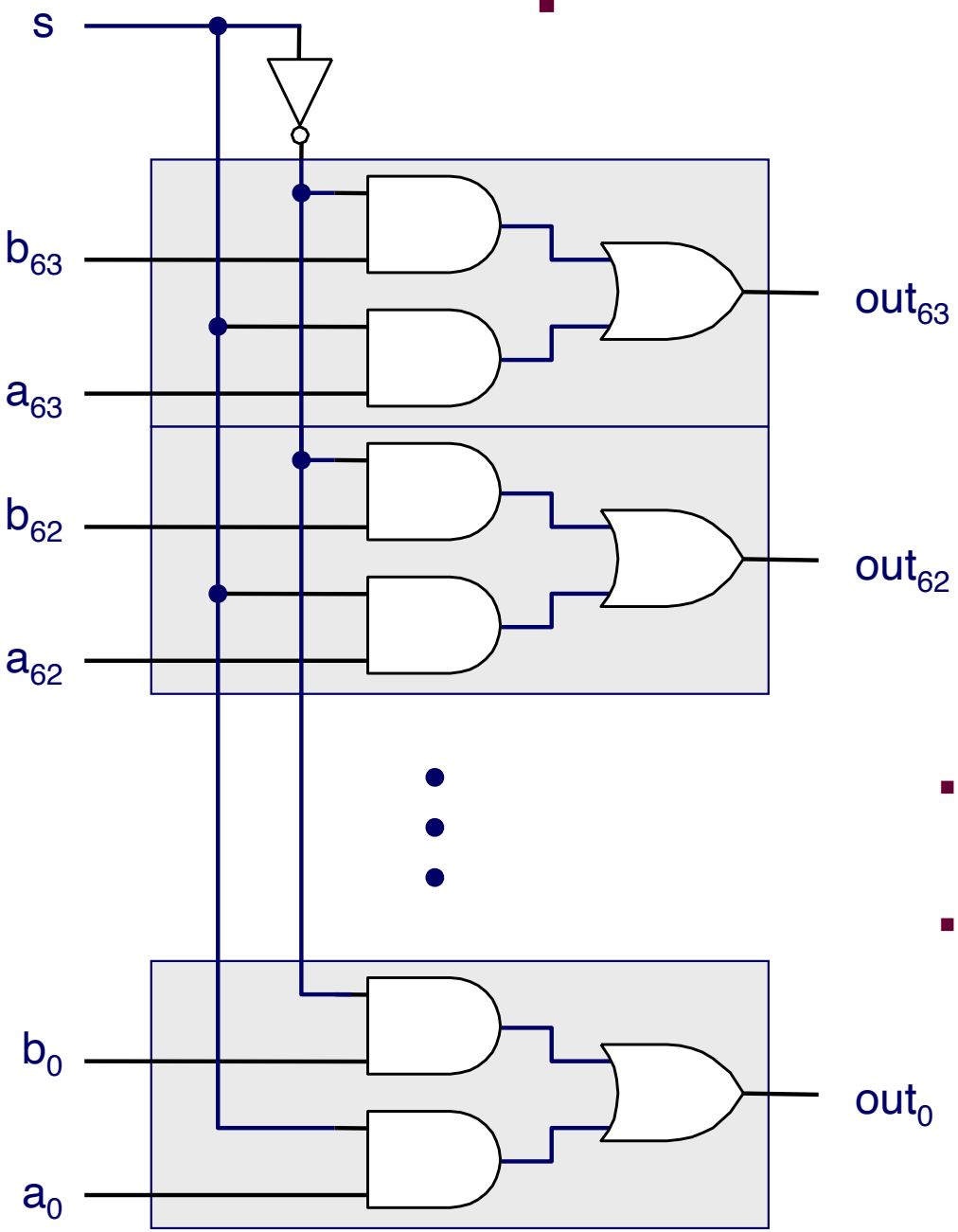


HCL Expression

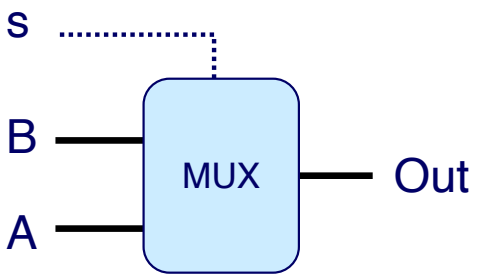
```
bool out = (s&&a)||(!s&&b)
```

- **Control signal s**
- **Data signals a and b**
- **Output a when s=1, b when s=0**

Word Multiplexor



Word-Level Representation



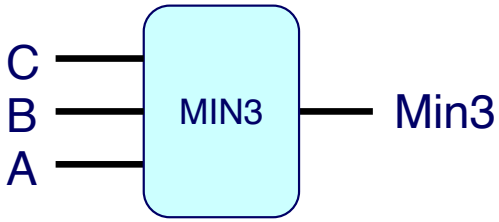
HCL Representation

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

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

HCL Word-Level Examples

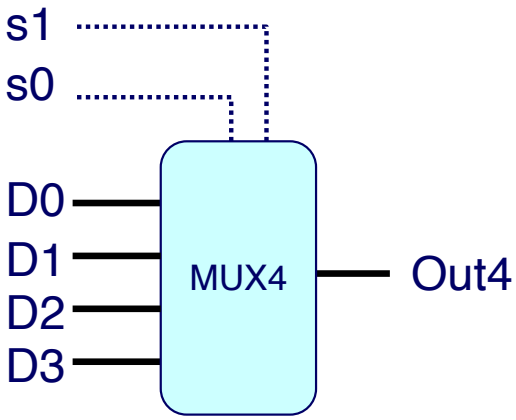
Minimum of 3 Words




```
int Min3 = [
  A < B && A < C : A;
  B < A && B < C : B;
  1              : C;
];
```

- Find minimum of three input words
- HCL case expression
- Final case guarantees match

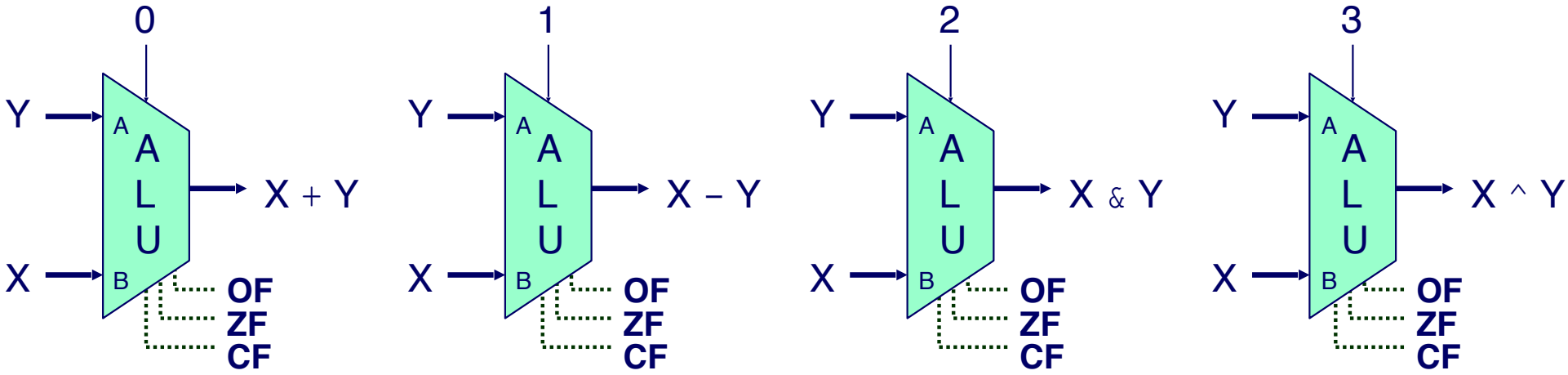
4-Way Multiplexor



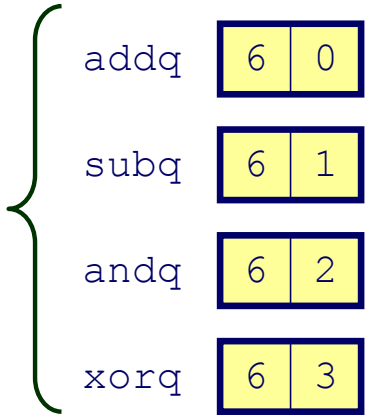

```
int Out4 = [
  !s1&&!s0: D0; #00
  !s1     : D1; #01 !s1&&s0
  !s0     : D2; #10
  1       : D3; #11
];
```

- Select one of 4 inputs based on two control bits
- HCL case expression
- Simplify tests by assuming sequential matching

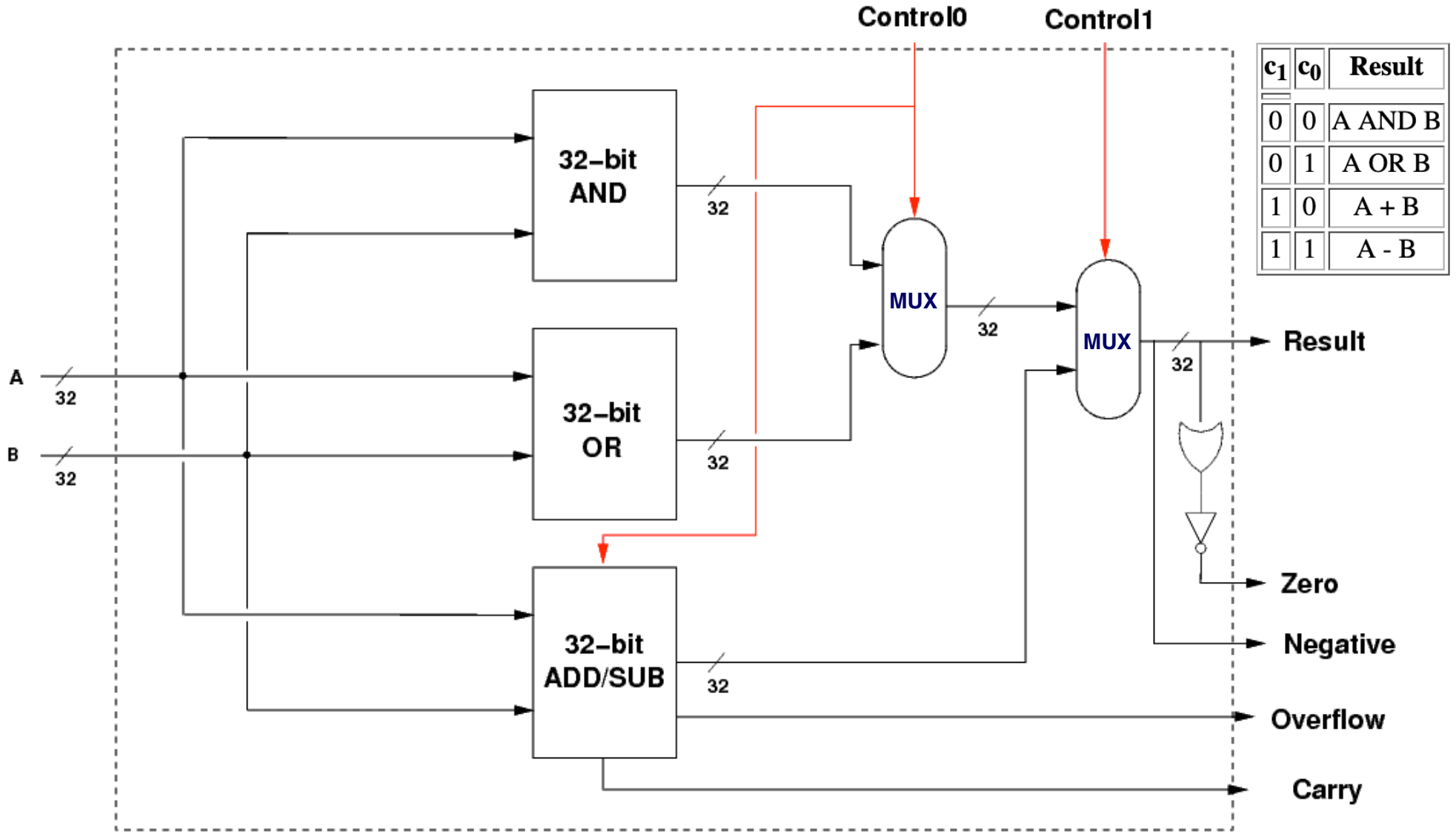
Arithmetic Logic Unit



- **Combinational logic**
  - **Continuously responding to inputs**
- **Control signal selects function computed**
  - **Corresponding to 4 arithmetic/logical operations in Y86-64**
- **Also computes values for condition codes**



Simple Arithmetic Logic Unit (Example Implementation) - AND/OR/ADD/SUB

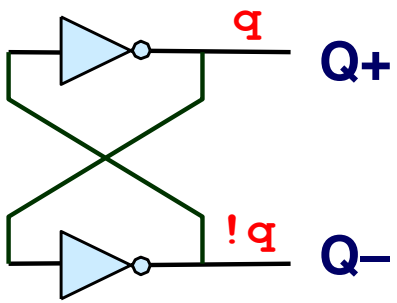


c_1	c_0	Result
0	0	A AND B
0	1	A OR B
1	0	A + B
1	1	A - B

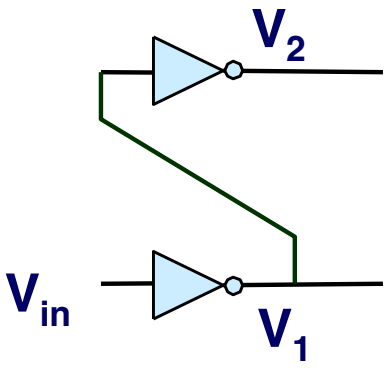
Source: http://www.csc.villanova.edu/~mdamian/Past/csc2400fa13/assign/ALU.html

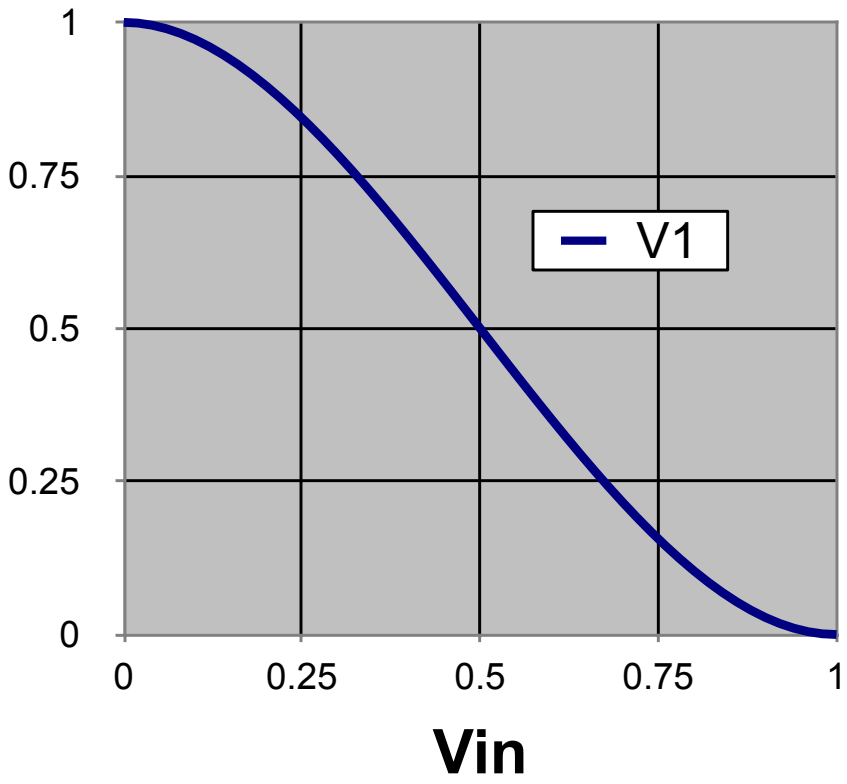
Storing 1 Bit

Bistable Element



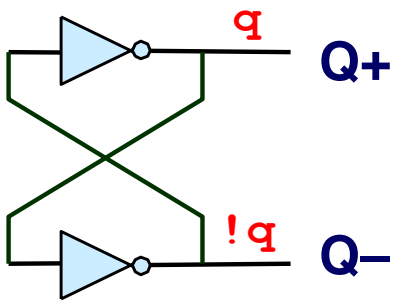
q = 0 or 1



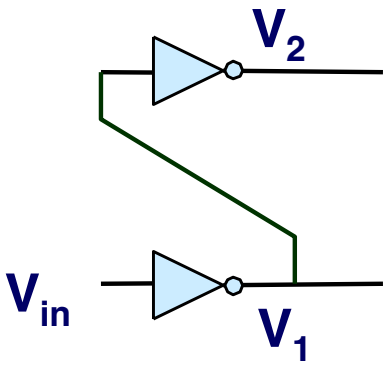


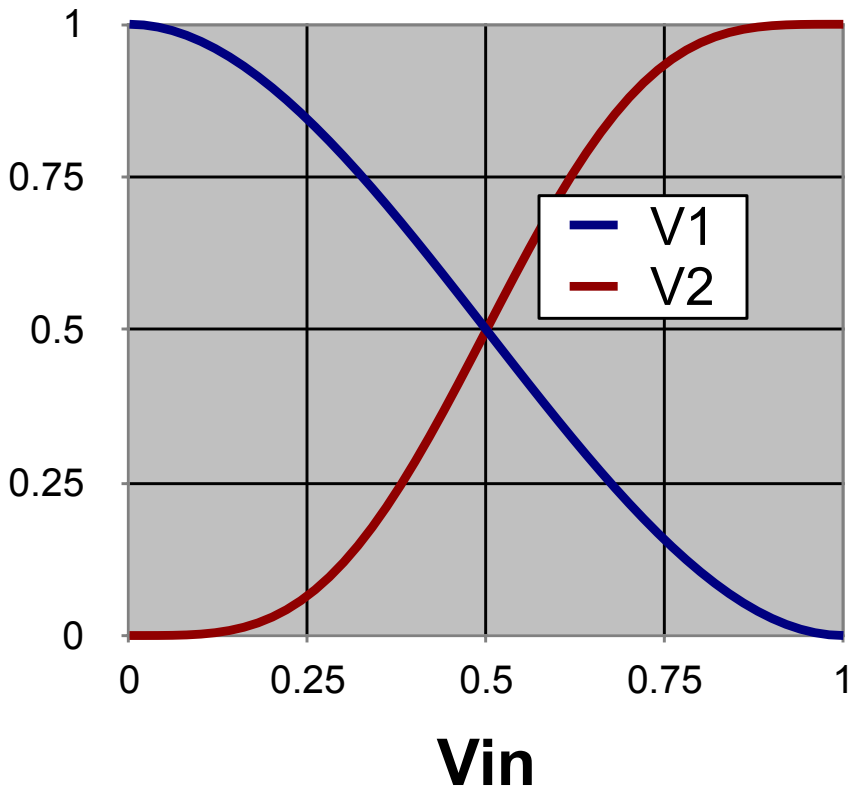
Storing 1 Bit

Bistable Element

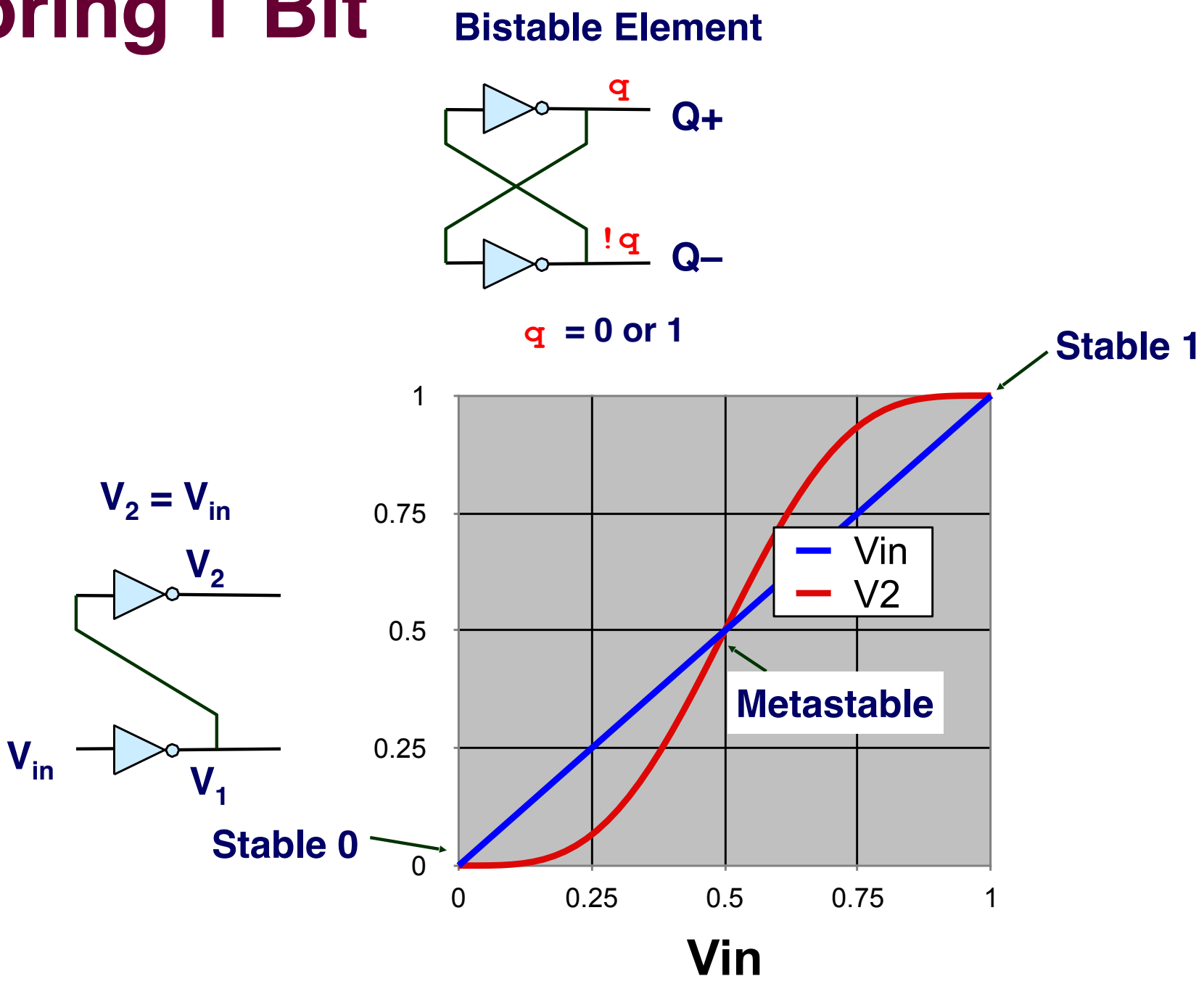


q = 0 or 1

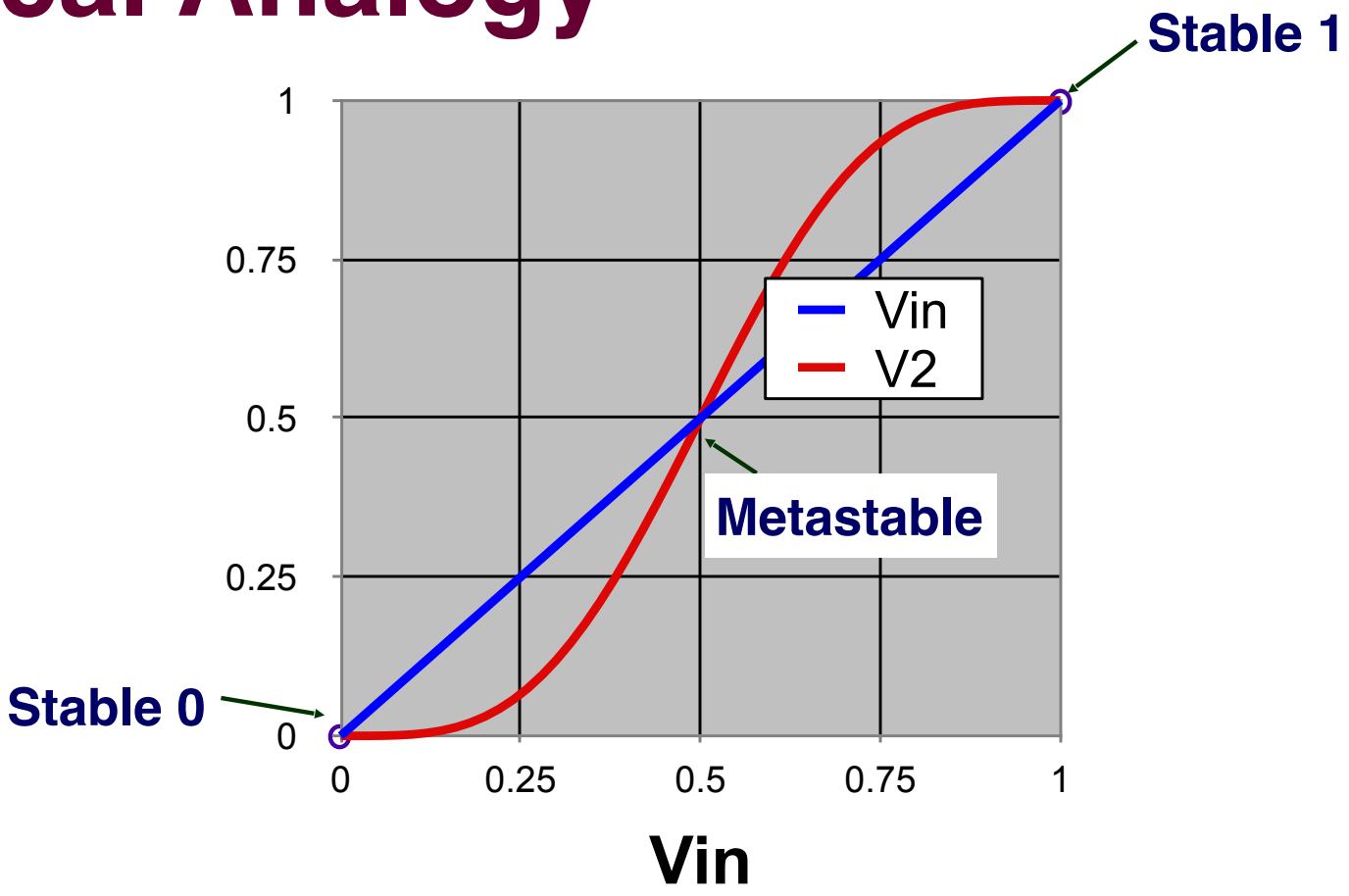


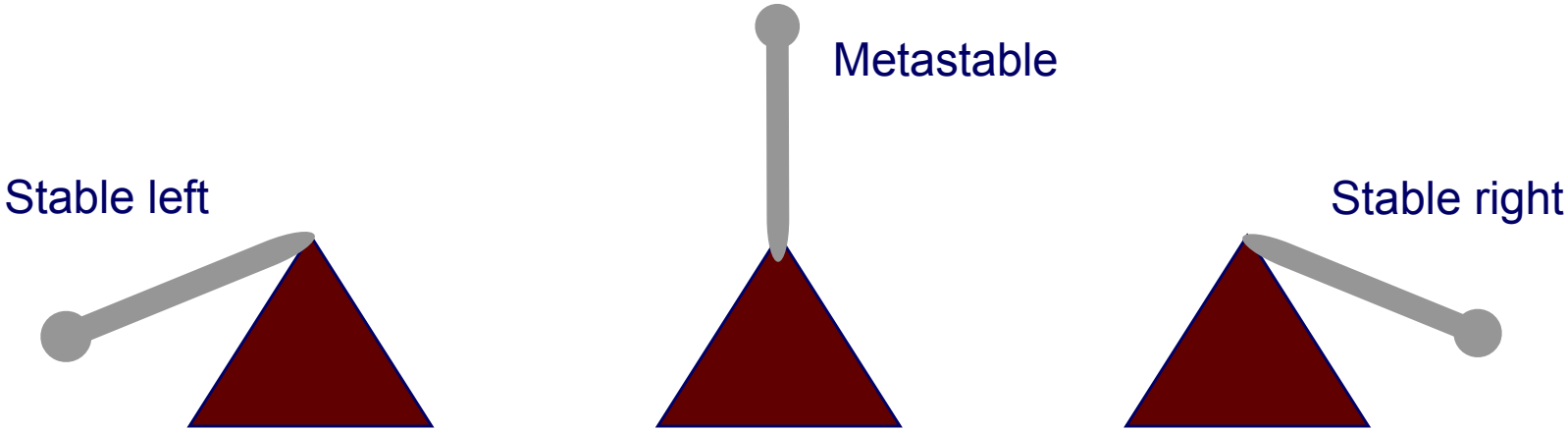


Storing 1 Bit



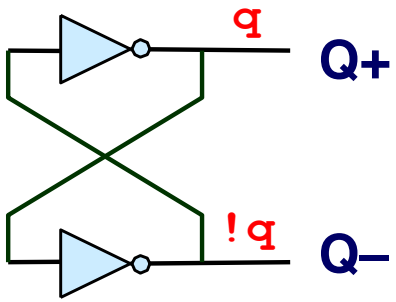
Physical Analogy





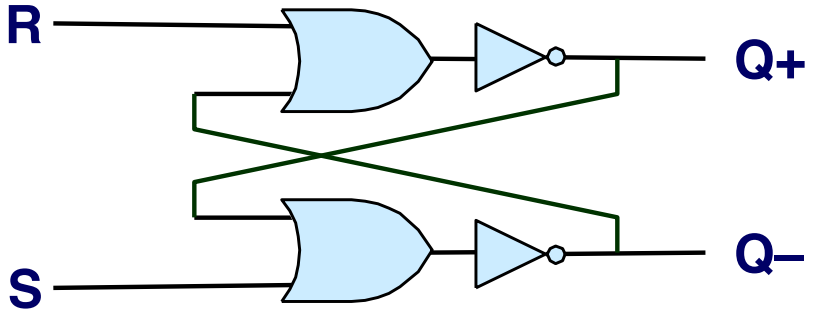
S-R Latch

Bistable Element

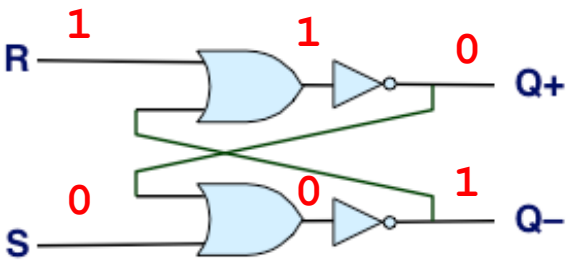


q = 0 or 1

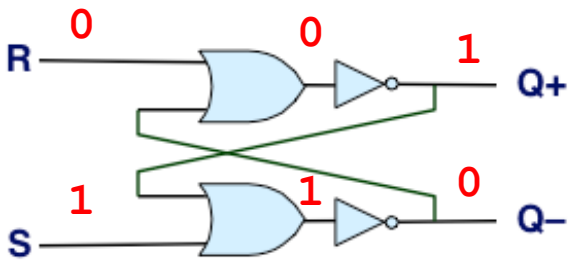
S-R Latch



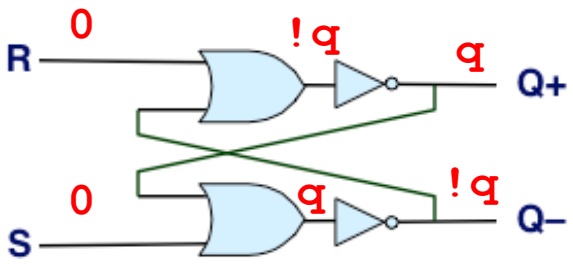
Resetting



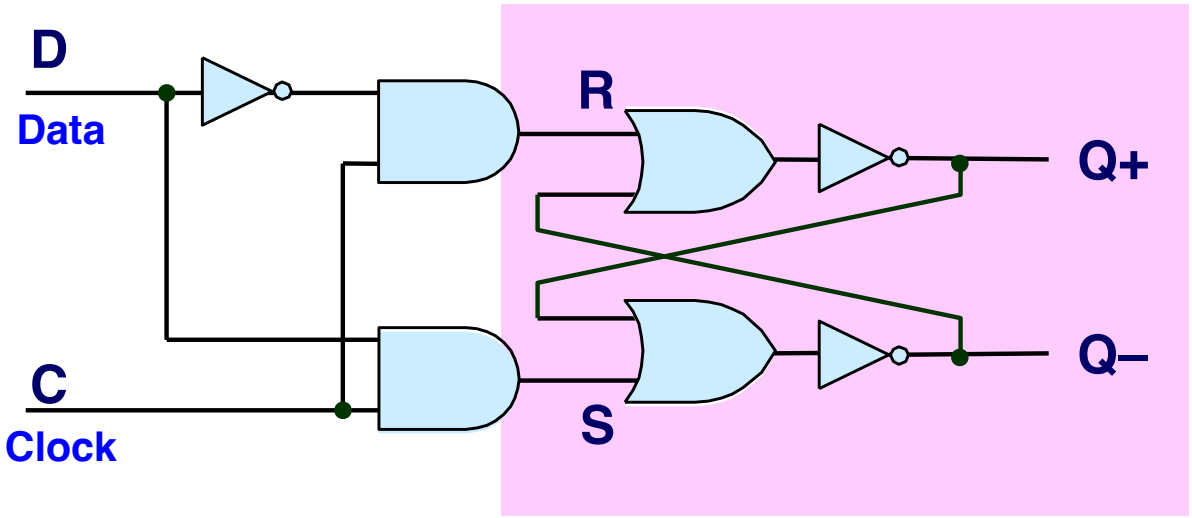
Setting



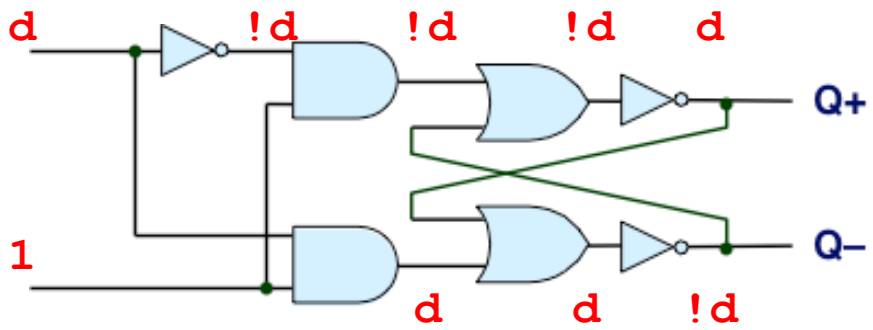
Storing



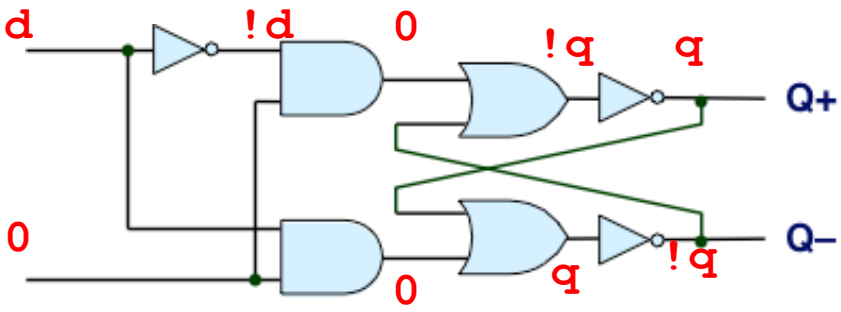
Transparent 1-Bit D-Latch



Latching

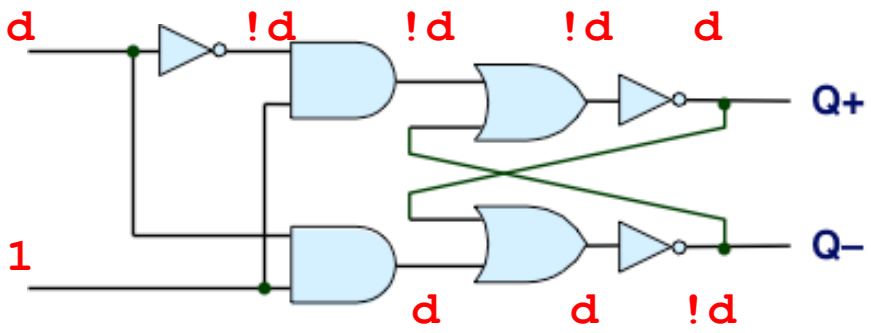


Storing

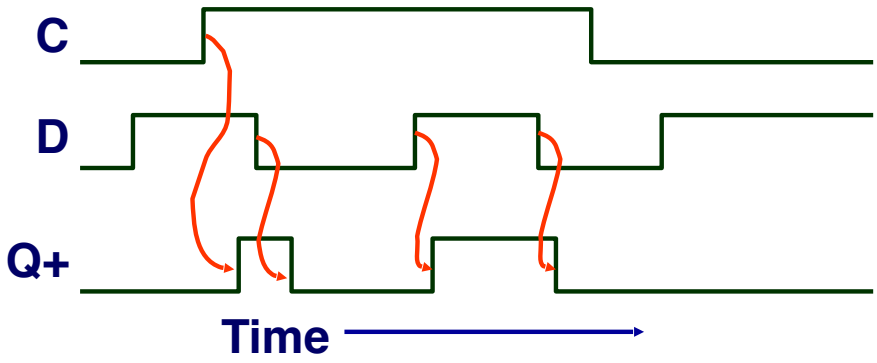


Transparent 1-Bit D-Latch

Latching

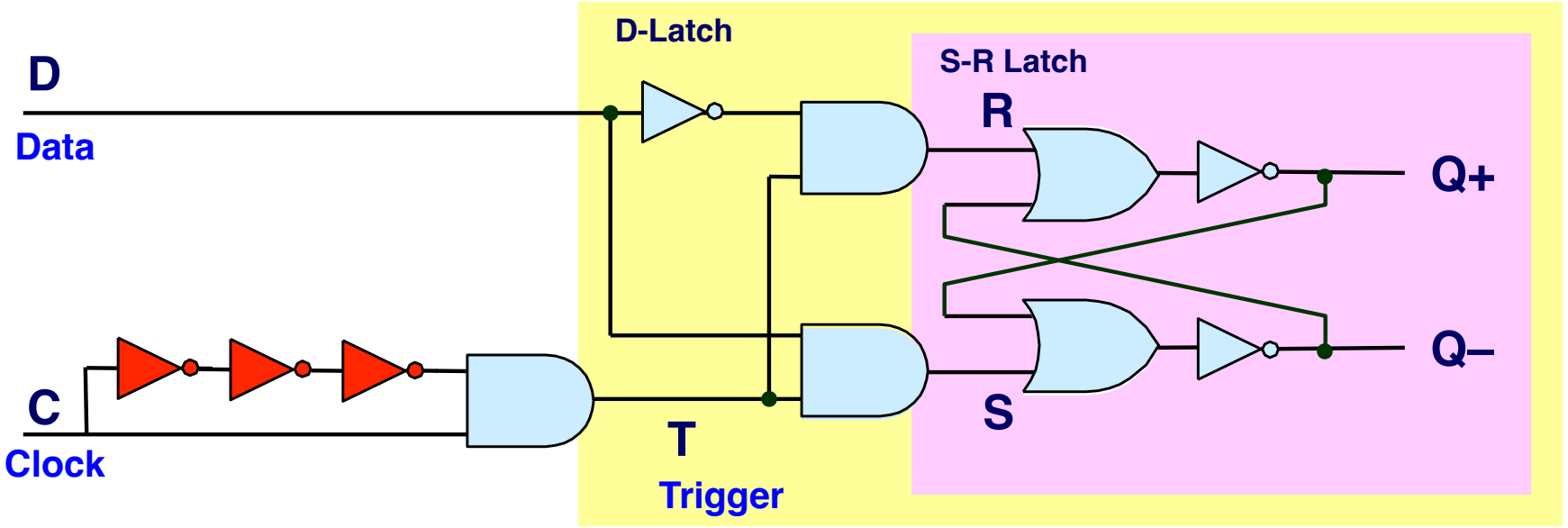


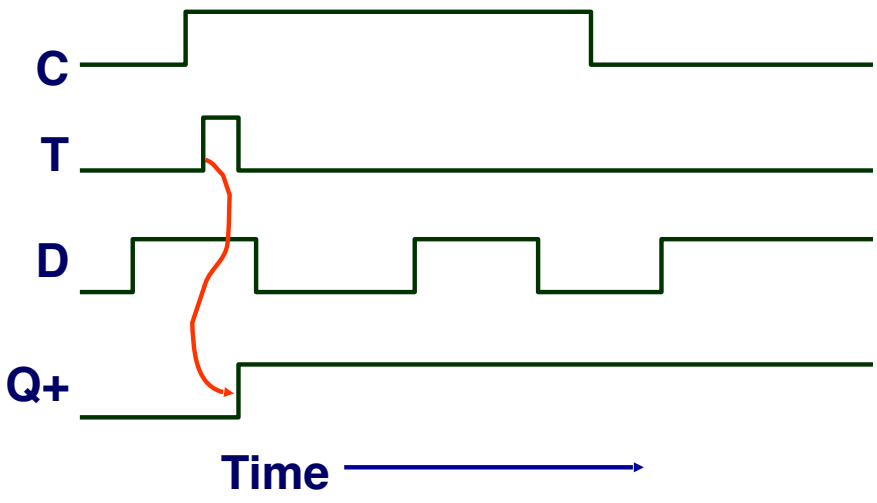
Changing D



- When in latching mode, combinational propogation from D to Q+ and Q–
- Value latched depends on value of D as C falls

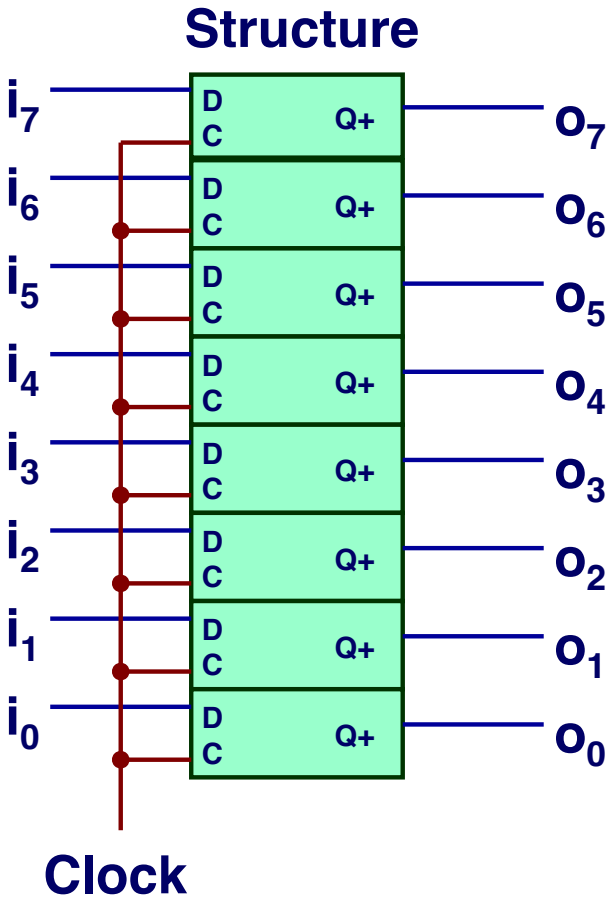
Edge-Triggered Latch (D Flip-Flop)

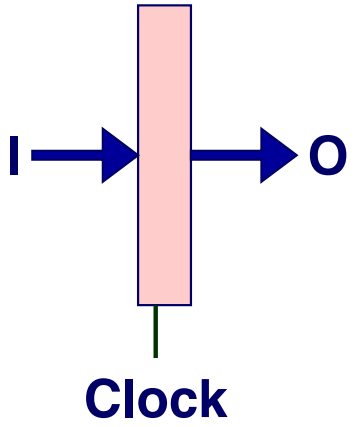




- Adding three inverters help us delay the Clock change and get a PULSE/ Trigger only on the rising edge of the clock
- Only in latching mode for brief period
  - Rising clock edge
- Value latched depends on data as clock rises
- Output remains stable at all other times

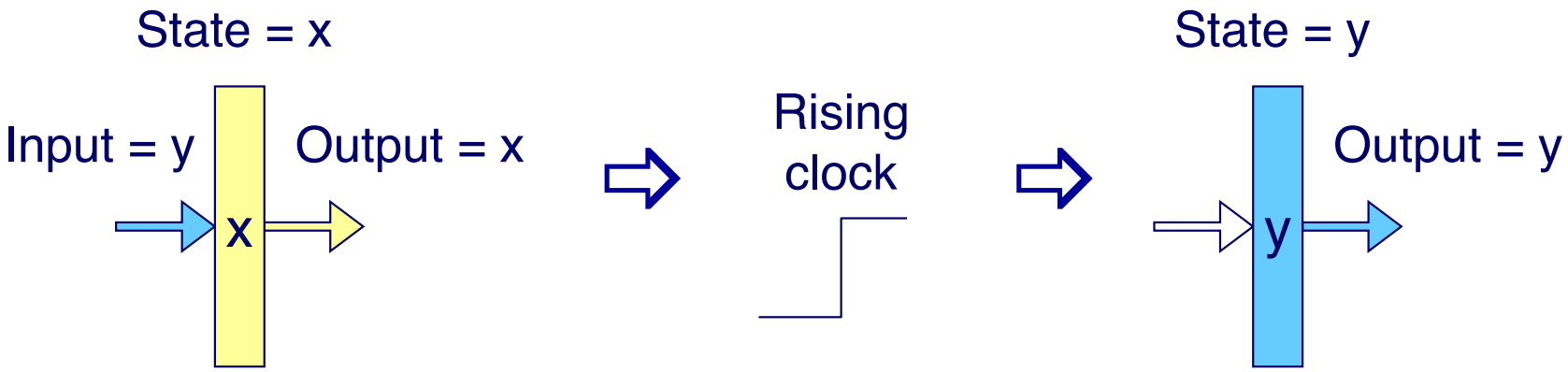
Hardware Registers





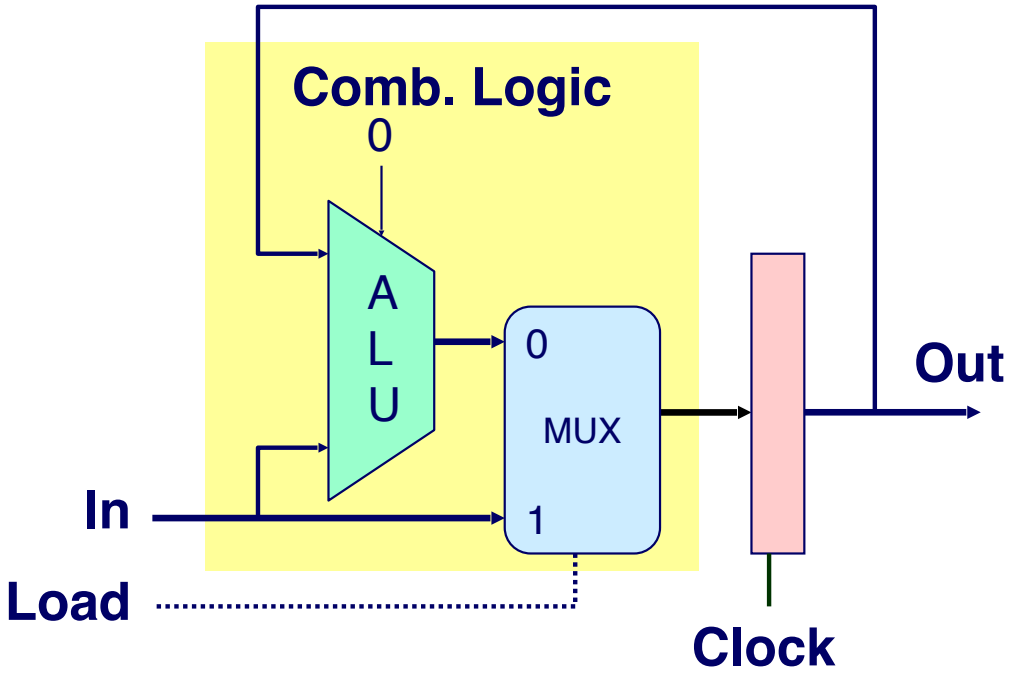
- Stores individual word of data
  - Different from *program registers* seen in assembly code
- Collection of edge-triggered latches (D Flip-Flops)
- Loads input on rising edge of clock

Register Operation



- **Stores data bits**
- **For most of time acts as barrier between input and output**
- **As clock rises, loads input**

State Machine Example



- Accumulator circuit
- Load or accumulate on each cycle