

Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.
Seoul National University

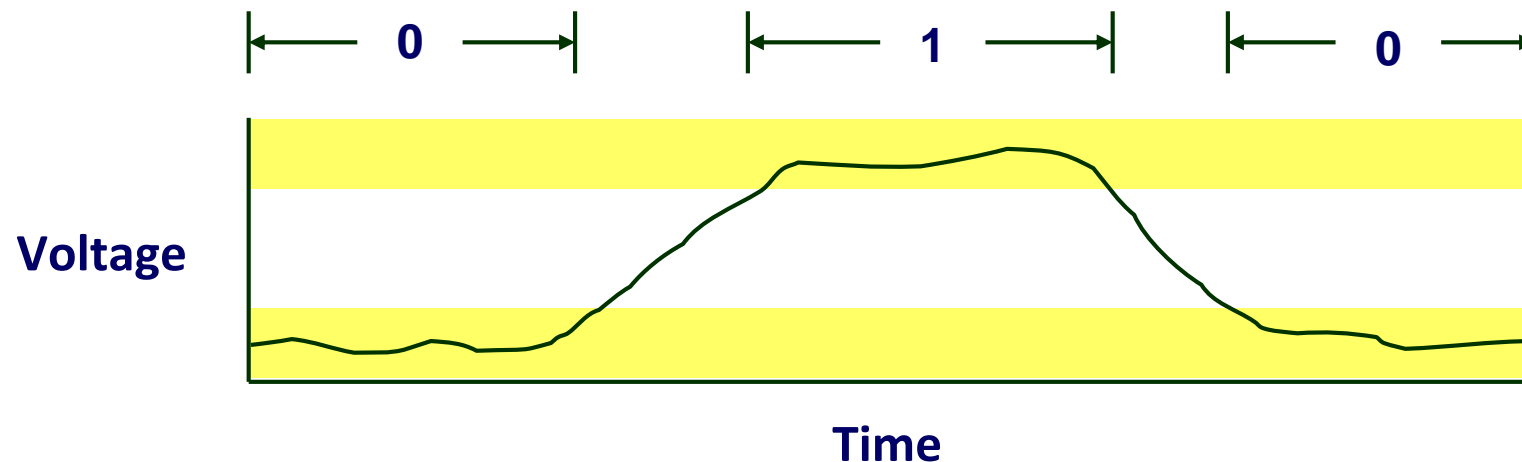
Spring 2019

Logic Design



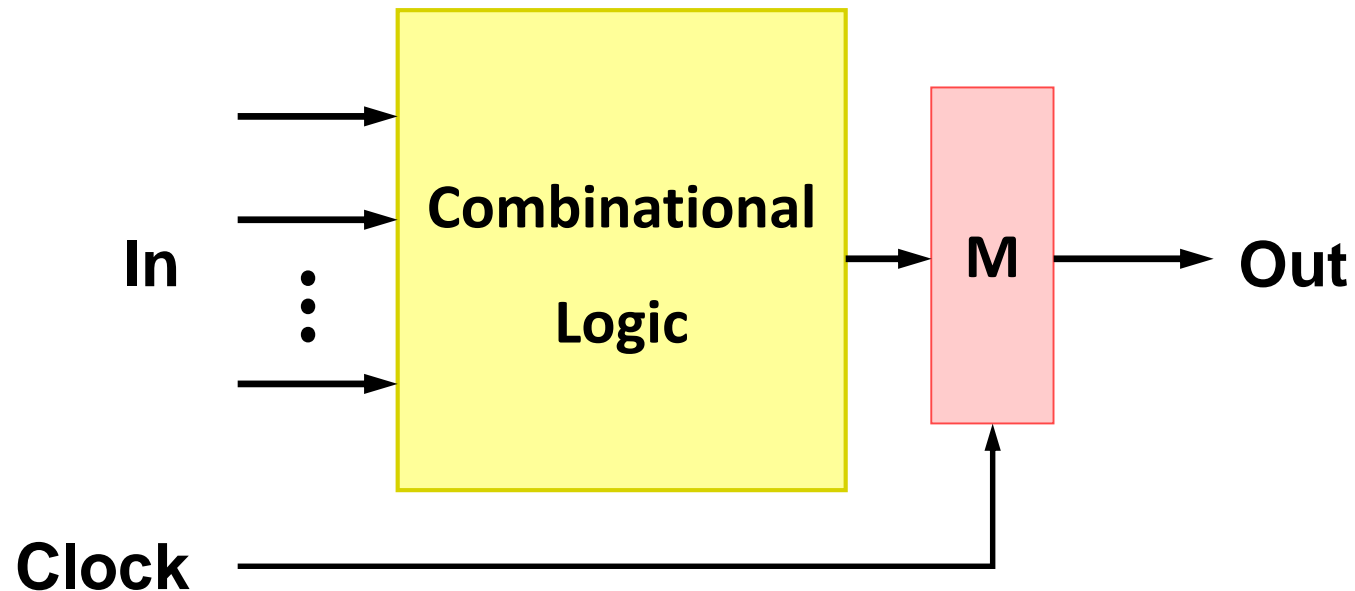
Digital Signals

- 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, fast, and robust



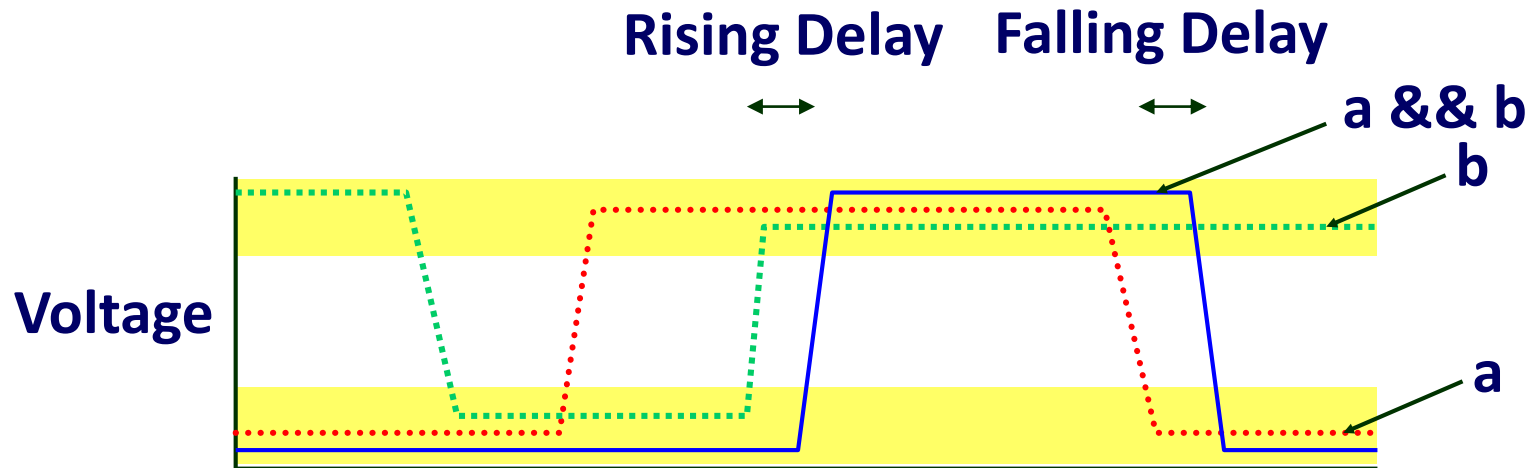
Digital Systems

- Three components required to implement a digital system
 - **Combinational logic** to compute Boolean functions
 - **Memory elements** to store bits
 - **Clock signals** to regulate the updating of the memory elements

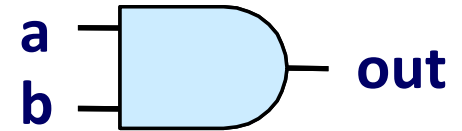


Computing with Logic Gates

- Outputs are Boolean functions of inputs
- Respond continuously to changes in inputs (with some, small delay)

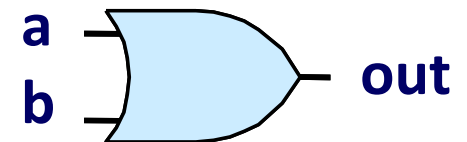


And



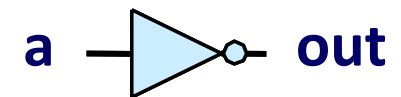
$$\text{out} = a \ \&\& \ b$$

Or



$$\text{out} = a \ || \ b$$

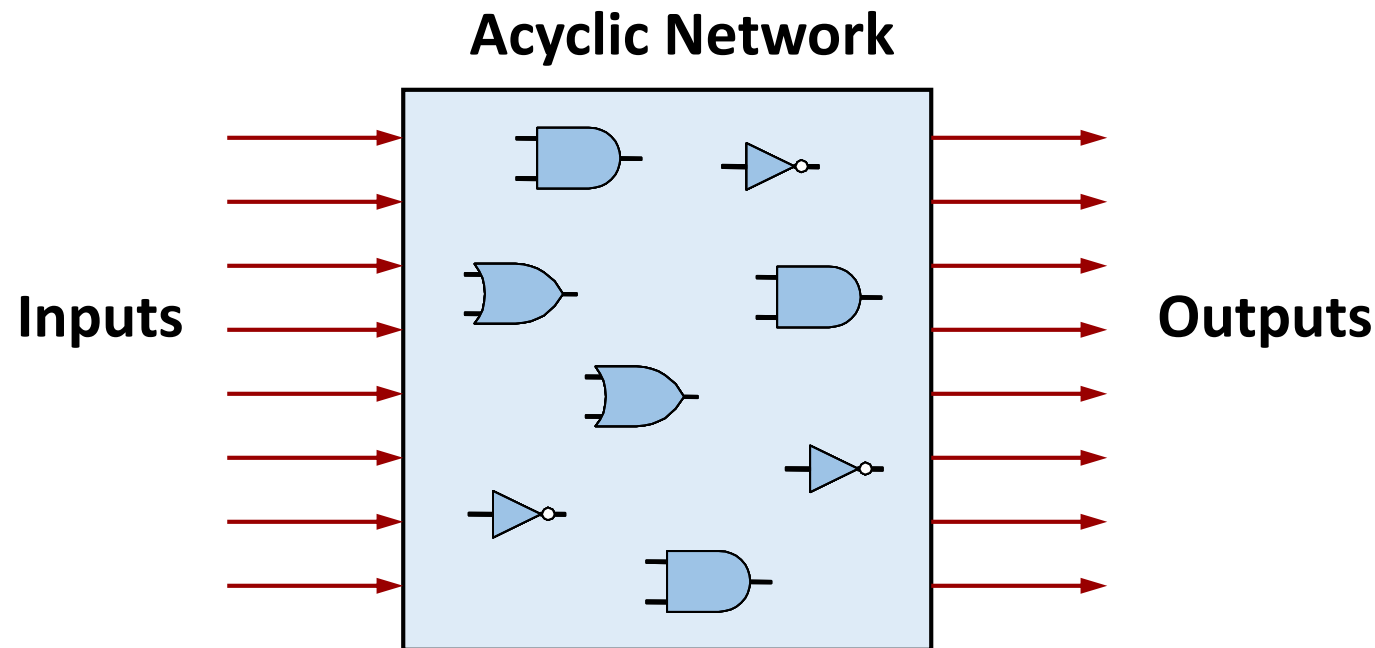
Not



$$\text{out} = !a$$

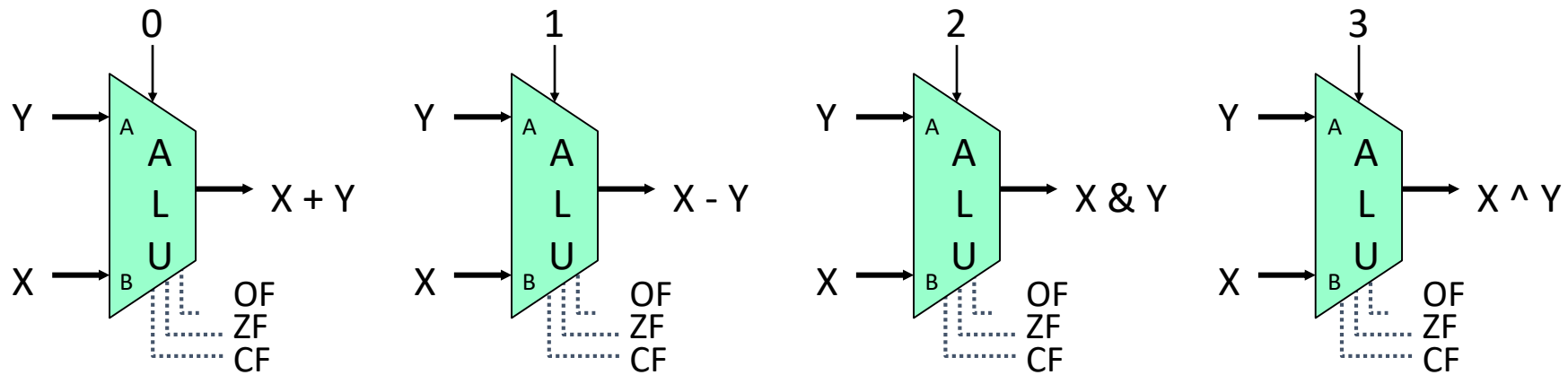
Combinational Circuits

- Acyclic network of logic gates
 - Continuously responds to changes on primary inputs
 - Primary outputs become (after some delay) Boolean functions of primary inputs



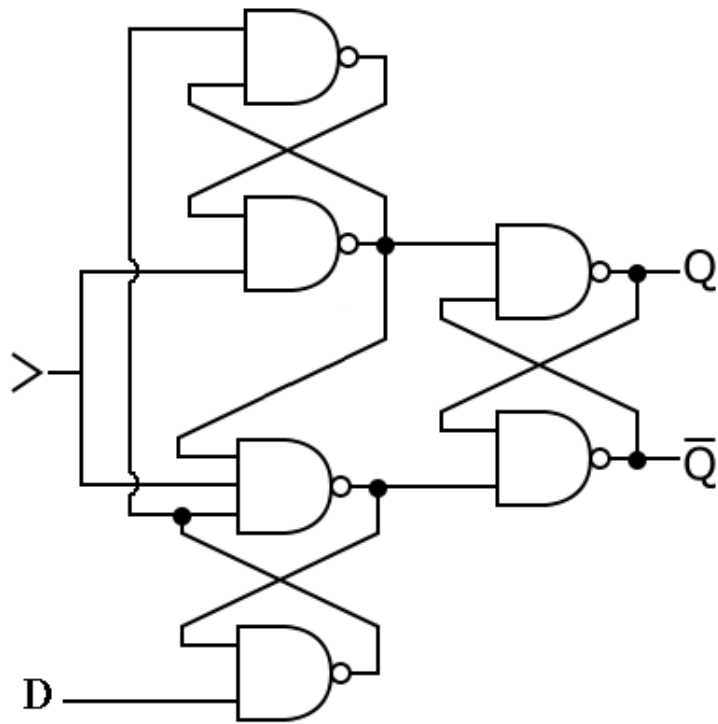
Arithmetic Logic Unit

- **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**

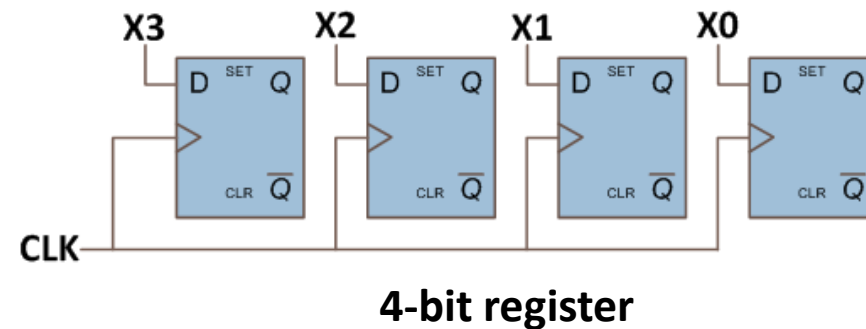
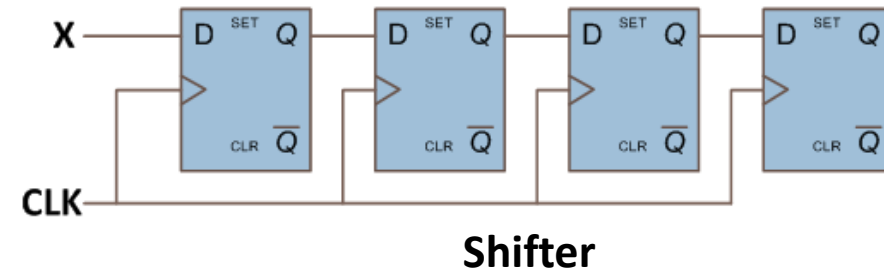
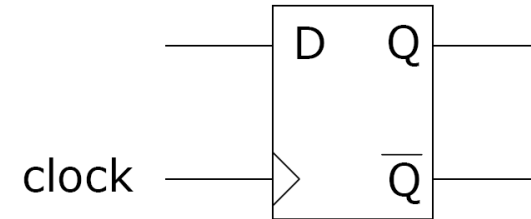


Sequential Logic

- Flip-flops

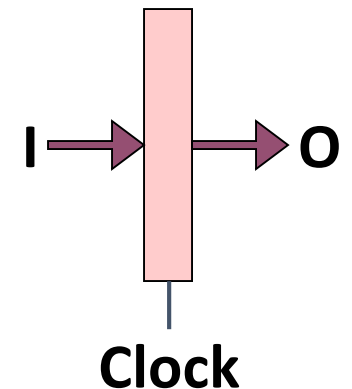
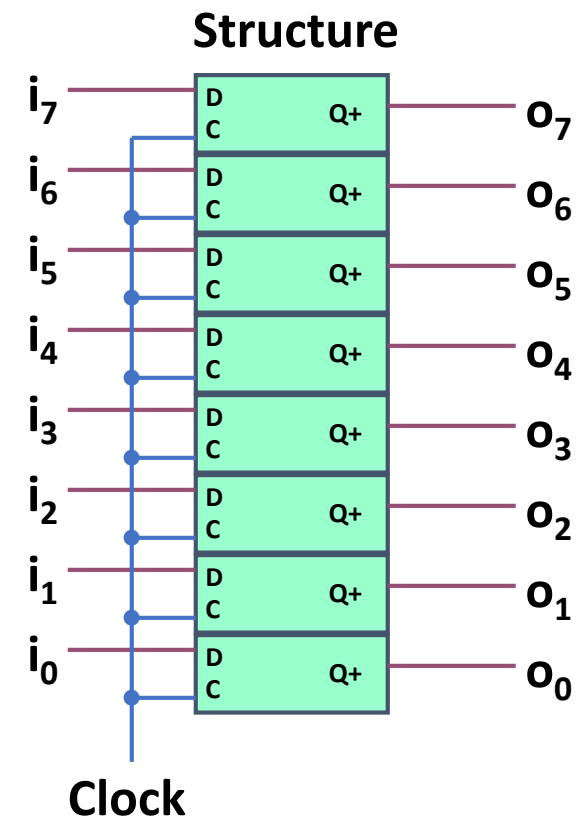


Edge triggered D flip-flop



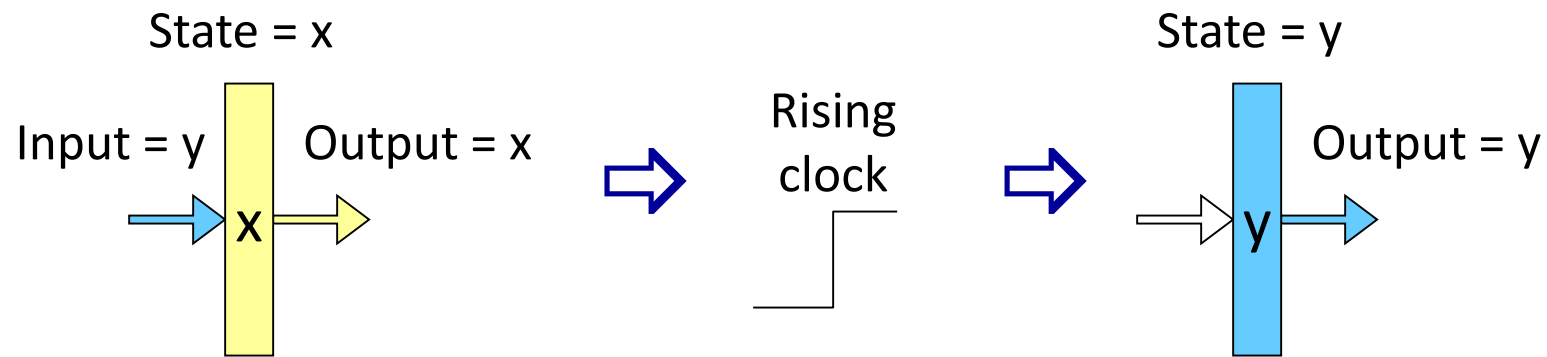
(Hardware) Registers

- Stores word of data
 - Different from **program registers** seen in assembly code
- Collection of edge-triggered latches
- Loads input on rising edge of clock



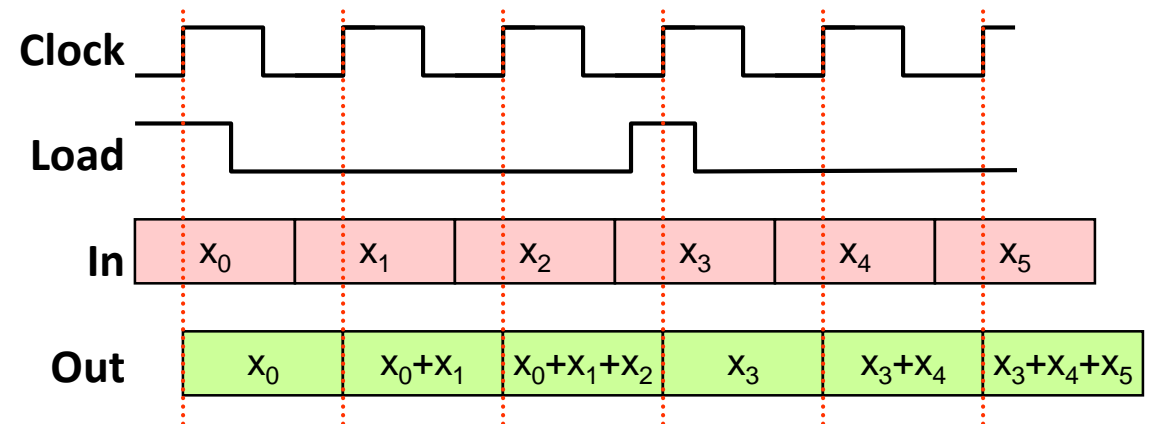
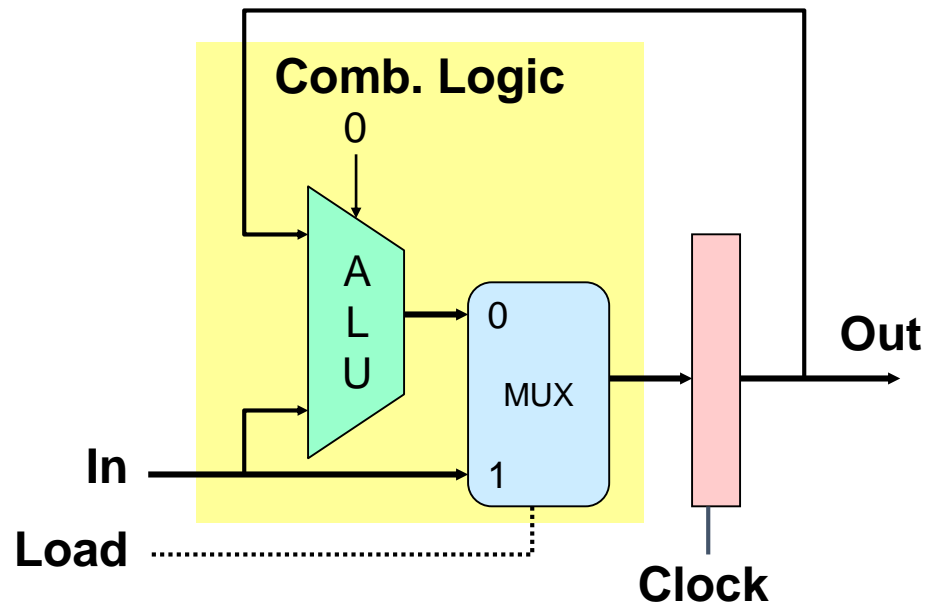
Register Operation

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



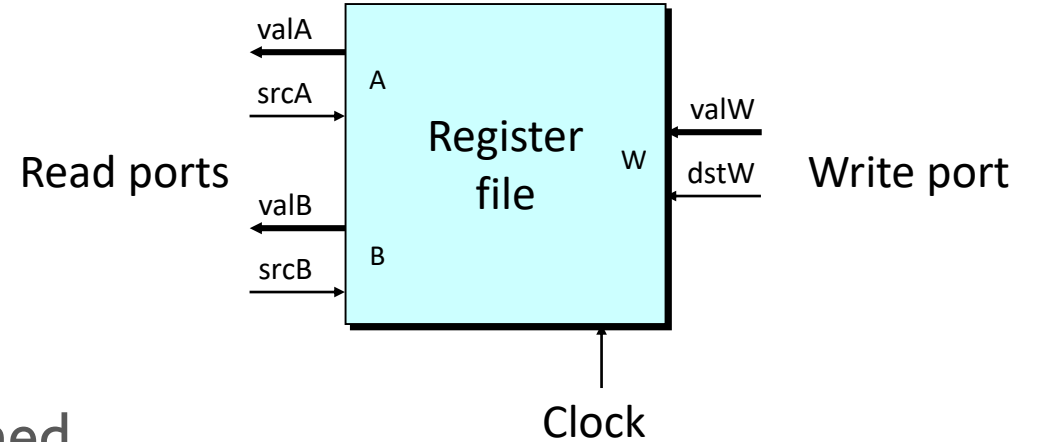
State Machine Example

- Accumulator circuit
- Load or accumulate on each cycle



Random-Access Memory

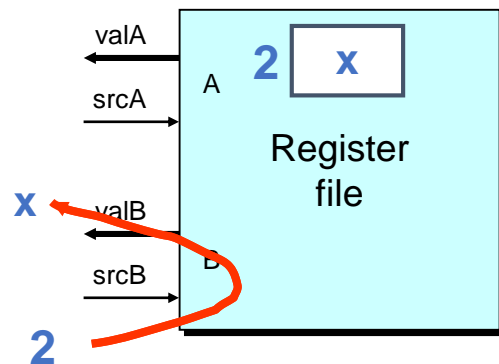
- Stores multiple words of memory
 - Address input specifies which word to read or write
- Register file
 - Holds values of program registers
 - %rax, %rsp, etc.
 - Register identifier serves as address
 - ID 15 (0xF) implies no read or write performed
- Multiple ports
 - Can read and/or write multiple words in one cycle
 - Each has separate address and data input/output



Register File Timing

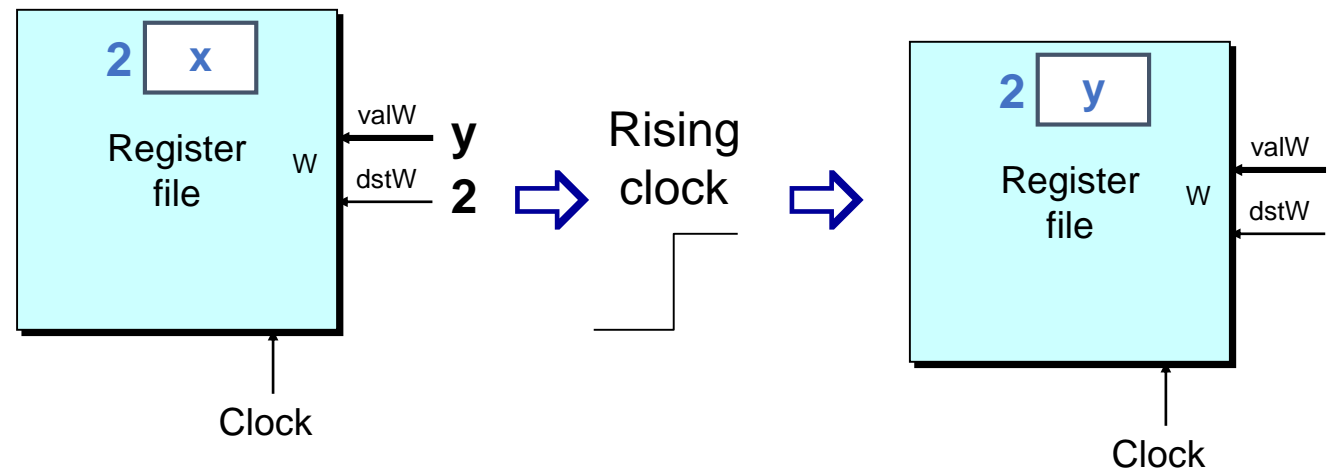
■ Reading

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



■ Writing

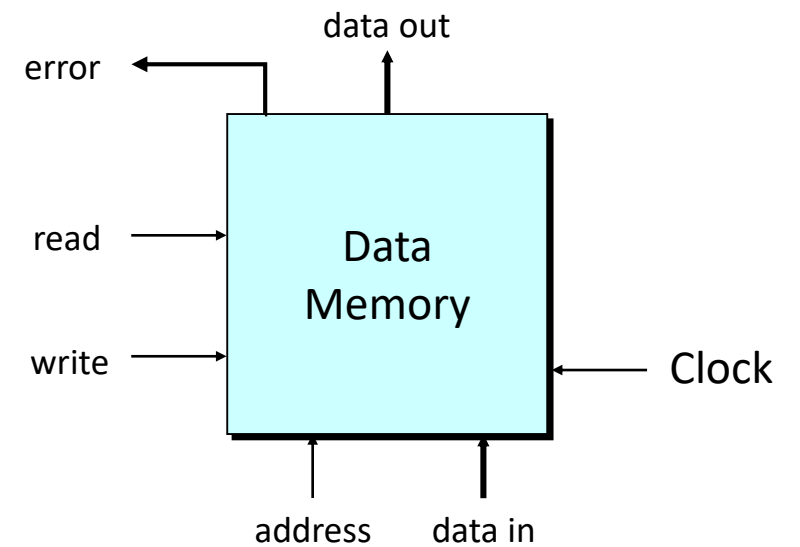
- Like register
- Update only as clock rises



Data Memory

- Random access memory for storing program data
- Operations similar to registers
- Reading: like combinational logic
- Writing: update only as clock rises
- error = 1 for invalid address

- Another read-only memory for instructions
- Dual-port memory
 - One read port for instructions, another read or write port for data



Hardware Control Language

- Very simple hardware description language
- Can only express limited aspects of hardware operation
- More info at <http://csapp.cs.cmu.edu/3e/waside/waside-hcl.pdf>
- Data types
 - `bool`: Boolean
 - `int`: words (Does not specify word size – bytes, 64-bit words, ...)
- Statements
 - `bool a = bool-expr;`
 - `int A = int-expr;`

HCL Operations

- Classify by type of value returned

- Boolean expressions

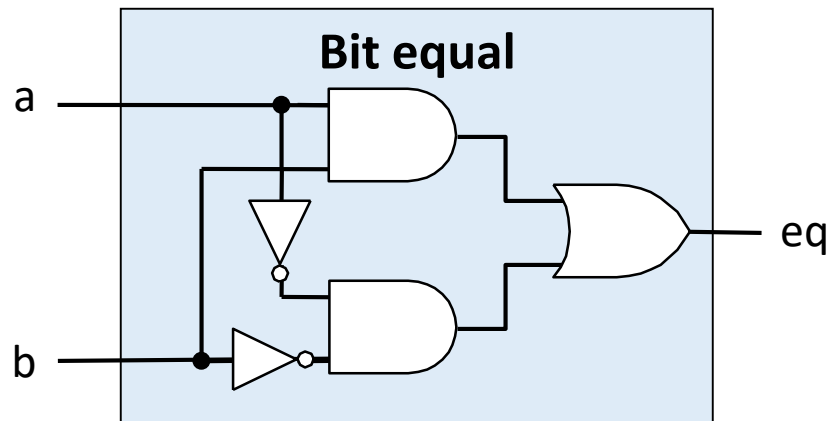
- Logic operations: `a && b, a || b, !a`
- Word comparisons: `A == B, A != B, A < B, A <= B, A >= B, A > B`
- Set membership: `A in { B, C, D }`
(Same as `A == B || A == C || A == D`)

- Word expressions

- Case expressions: `[a : A; b : B; c : C]`
- Evaluate test expressions `a, b, c, ...` in sequence
- Return word expression `A, B, C, ...` for first successful test

Bit Equality

- 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

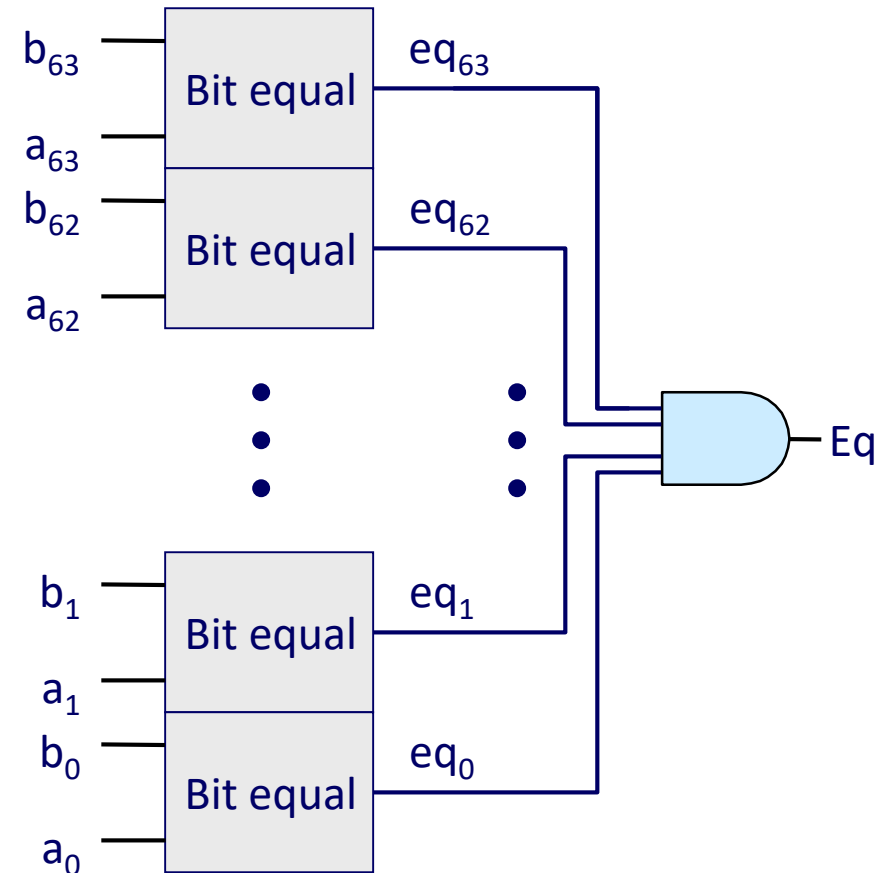
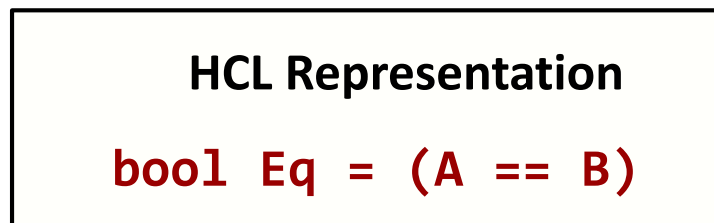
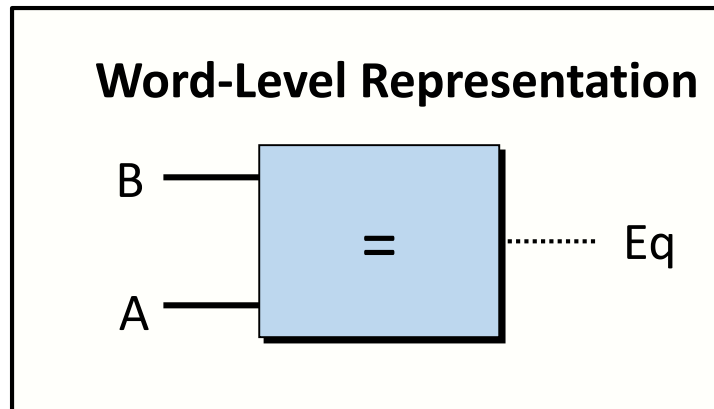


HCL Expression

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

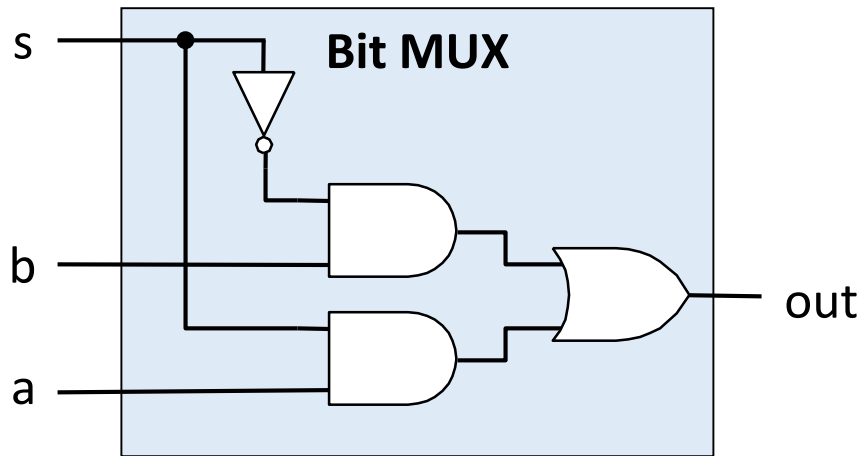

Word Equality

- HCL representation of 64-bit word size
 - Equality operation
 - Generates Boolean value



Bit-Level Multiplexor

- HCL representation
 - Control signal s
 - Data signals a and b
 - Output a when $s=1$, b when $s=0$

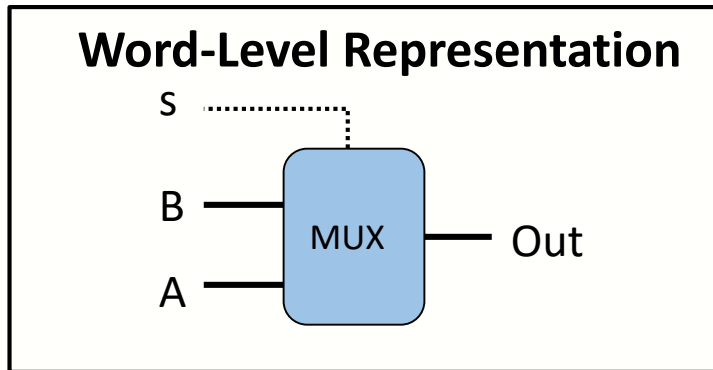


HCL Expression

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

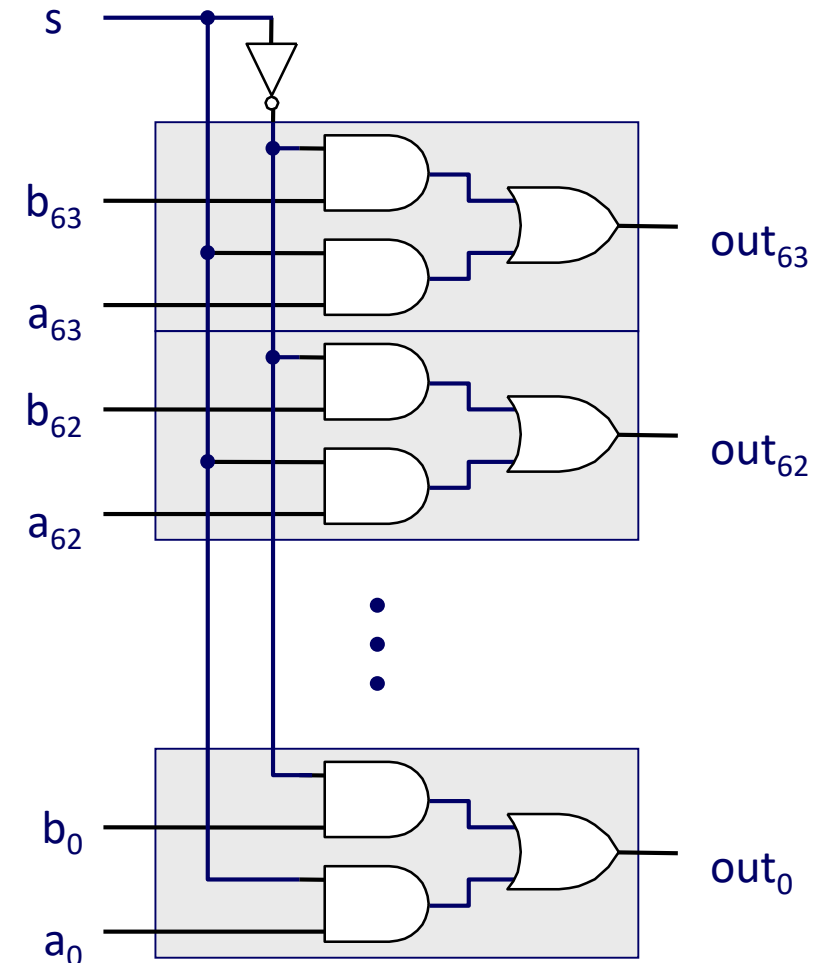
Word Multiplexor

- 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



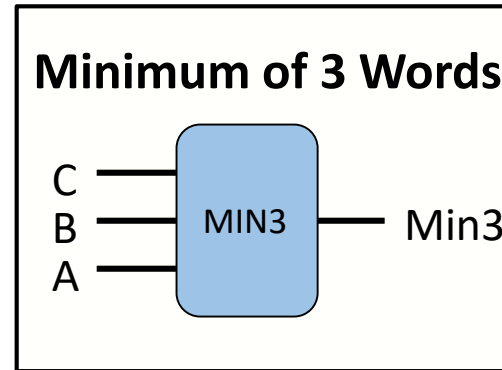
HCL Representation

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



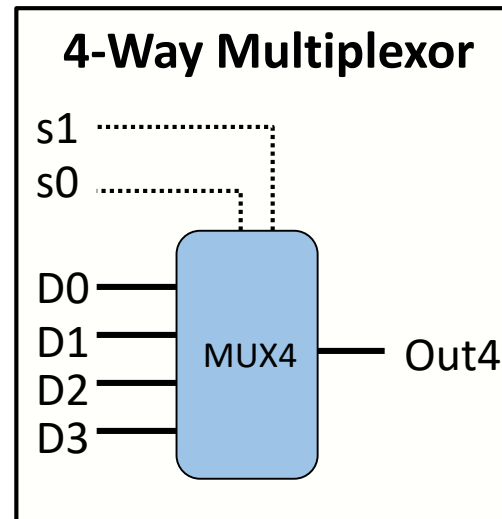
HCL Word-Level Examples

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



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

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

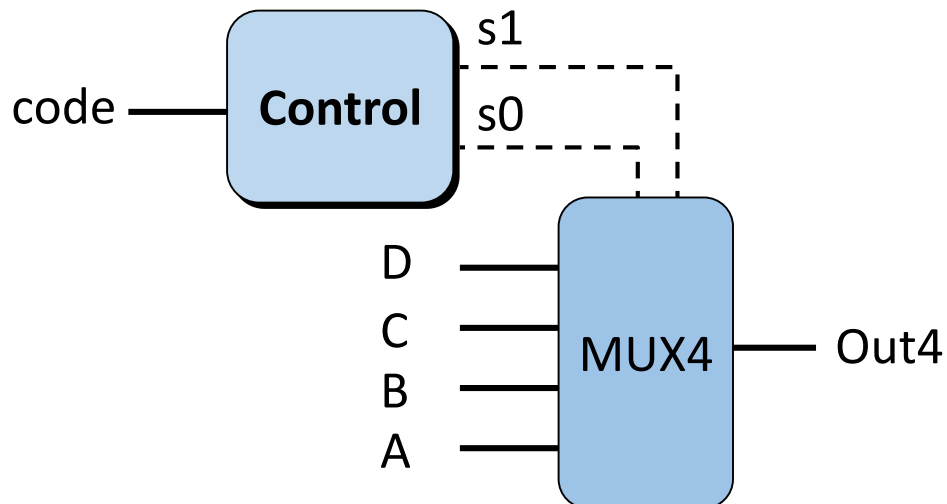


```
int Out4 = [  
  !s1 && !s0 : D0;  
  !s1        : D1;  
  !s0        : D2;  
  1          : D3;  
];
```

Set Membership

■ HCL representation

- Input code
- Output signal s_0, s_1
- s_1 is 1 when code is in the set $\{2, 3\}$
- s_0 is 1 when code is in the set $\{1, 3\}$

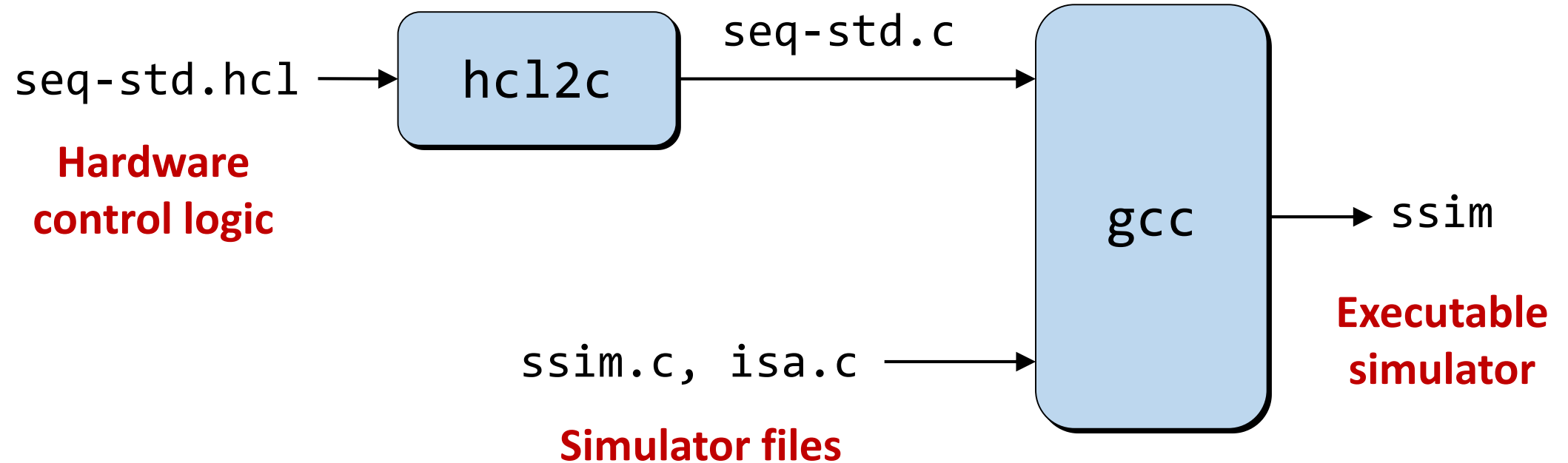


HCL Expression

```
bool s1 = code == 2 || code == 3;  
bool s0 = code == 1 || code == 3;
```

```
bool s1 = code in {2, 3};  
bool s0 = code in {1, 3};
```

Using HCL



Summary

■ Computation

- Performed by combinational logic
- Computes Boolean functions
- Continuously reacts to input changes

■ Storage

- Registers
 - Hold single words, Loaded as clock rises
- Random-access memories
 - Hold multiple words
 - Possibly multiple read or write ports
 - Read word when address input changes
 - Write word as clock rises