RISC vs. CISC

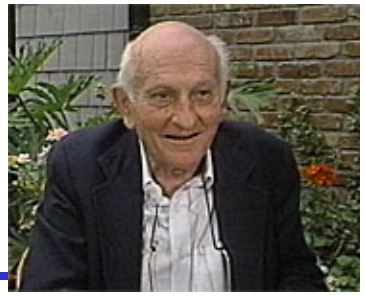
RISC vs. CISC

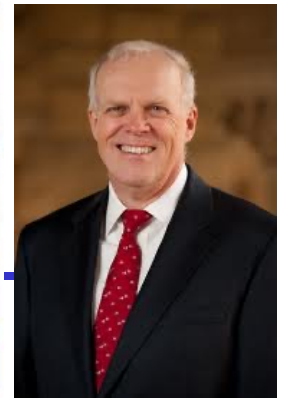
- ISA
  - Instruction set architecture
    - Instructions supported by a particular processor
    - Their byte-level encodings
- CISC
  - Complex instruction set computer
- RISC
  - Reduced instruction set computer

CISC

- Involved from the earliest computers
- Mainframe and Minicomputers
  - By the early 1980s
  - their instruction sets had grown quite large
    - Manipulating circular buffers
    - performing decimal arithmetic
    - evaluating polynomials
- Microcomputer
  - Appeared in 1970s, had limited instruction sets
    - constrained by number of transistors on a single chip
  - By the early 1980s, followed the path to increase their instruction sets

RISC





- Developed in the early 1980s
  - philosophy
    - One are able to generate efficient code for a simpler form of instruction set
    - Complex instructions are hard to generated with a compiler and seldom used
- John Cocke (1925-2002)
  - 1987 ACM Turing Award
- David Patterson(UC Berkeley), John Hennessy (Stanford U)
  - 2017 ACM Turing Award

RISC

IBM	Power
IBM and Motorola	PowerPC
Sun Microsystems	SPARC
Digital Equipment Corporation	Alpha
Hewlett Pack	Pa-risc
MIPS Technologies	MIPS
Acorn Computers Ltd	ARM(Acorn RISC Machine)

RISC vs. CISC

CICS	Early RISC
A large number of instructions	Many fewer instructions (<100)
Some instructions with long execution times	No instruction with a long execution time
Variable-length encodings. (x86-64 1~15 bytes)	Fixed-length encodings
Multiple formats for specifying operands	Simple addressing formats
Arithmetic and logical operations can be applied to both memory and register operands	Arithmetic and logical operations only use register operands. *load/store Architecture*

RISC vs. CISC

CICS	Early RISC
Implementation artifacts hidden from machine level programs	Implementation artifacts exposed to machine level programs
Condition codes	explicit test instructions store the test results in normal registers for use in conditional evaluation
Stack-intensive procedure linkage	Registers are used for procedure arguments and return addresses

RISC

- Cons for early RISC
  - More instructions
  - Not multiple cycles
  - Exposing implementation artifacts to machine level programs
- Pros for RISC
  - Well suited for pipeline (high efficient implementation)

Y86-64

- The Y86-64 instruction set
  - Includes attributes of both CISC and RISC
  - Can be viewed as taking a CISC instruction set (x86-64) and simplifying it by applying some of the principles of RISC
- On the CISC side
  - condition codes, variable-length instructions
  - uses the stack to store return addresses.
- On the RISC side,
  - a load/store architecture
  - a regular instruction encoding
  - passes procedure arguments through registers

RISC vs. CISC

- In the early 1990s, the debate diminished
  - neither RISC nor CISC in their purest forms are better than designs that incorporated the best ideas of both

RISC

- Evolved and introduced more instructions
  - many of which take multiple cycles to execute
  - Have hundreds of instructions
- Exposing implementation artifacts to machine-level programs
  - proved to be shortsighted
  - As new processor models were developed
    - using more advanced hardware structures,
    - many of these artifacts became irrelevant,
    - but they still remained part of the instruction set

CISC

- The core of RISC design
  - is well suited to execution on a pipelined machine
- X86 incorporates RISC features
  - Dynamic translating CISC instructions to RISC-like ops and taking pipeline architectures
  - X86-64 introduces more RISC features

CISC

- Marketing issues determine the success of the ISAs
  - X86 wins in high-end server, desktop, laptop machines
  - RISC win in market of embedded processors and the smart phones

CISC

- Intel made it easy to keep moving from one generation of processor to the next
  - By maintaining compatibility with its existing processors
- As integrated-circuit technology improved
  - x86 processor manufacturers could overcome the inefficiencies created by the original 8086 instruction set design
  - using RISC techniques to produce performance comparable to the best RISC machines.

RISC

- RISC processors have done very well in the market for *embedded processors*
- In these applications, saving on cost and power is more important than maintaining backward compatibility
  - In terms of the number of processors sold
- this is a very large and growing market

Logical Design & HCL

Topics

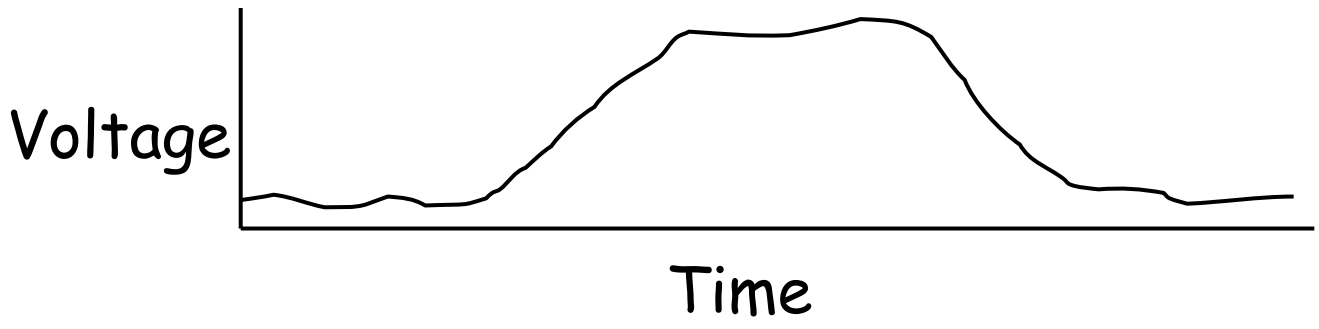
- Logic design
- Hardware Control Language HCL
- Suggested Reading: 4.2

Logic Design

- Digital circuit
  - What is digital circuit?
  - Know what a CPU will base on?
- Hardware Control Language (HCL)
  - A simple and functional language to describe our CPU implementation
  - Syntax like C

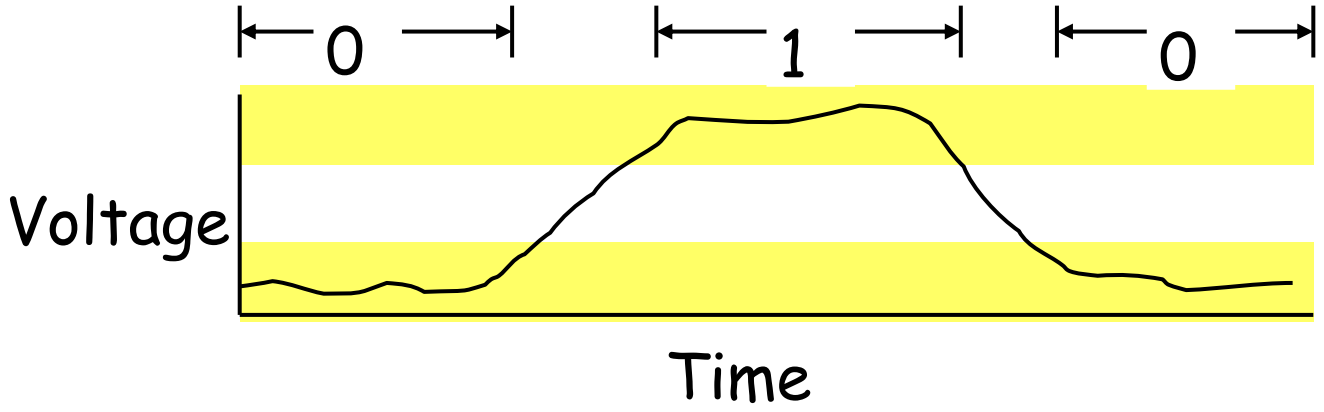
Category of Circuit

- Analog Circuit
  - Use all the range of Signal
  - Most part is amplifier
  - Hard to model and automatic design
  - Use transistor and capacitance as basis
  - We will not discuss it here



Category of Circuit

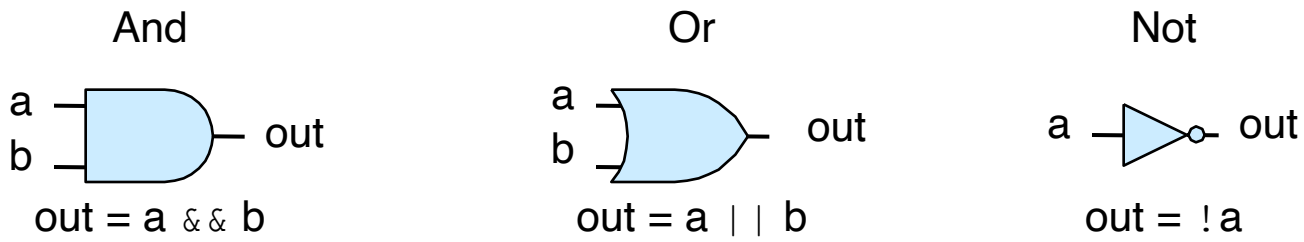
- Digital Circuit
  - Has only two values, 0 and 1
  - The voltage of 1 is differ in different kind circuit.
    - E.g. TTL circuit using 5 voltage as 1
  - Use voltage thresholds to extract discrete values from continuous signal



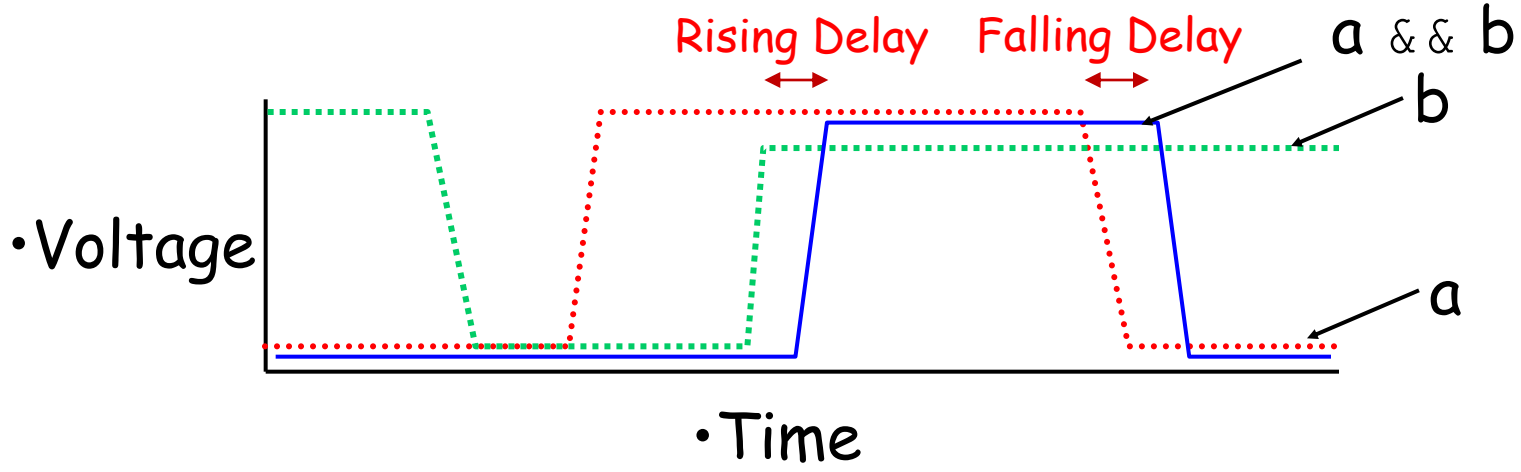
Digital Signals

- Digital Circuit (cont.)
  - 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
  - Easy to model and design
  - Use true table and other tools to analyze
  - Use gate as the basis

Computing with Logic Gates



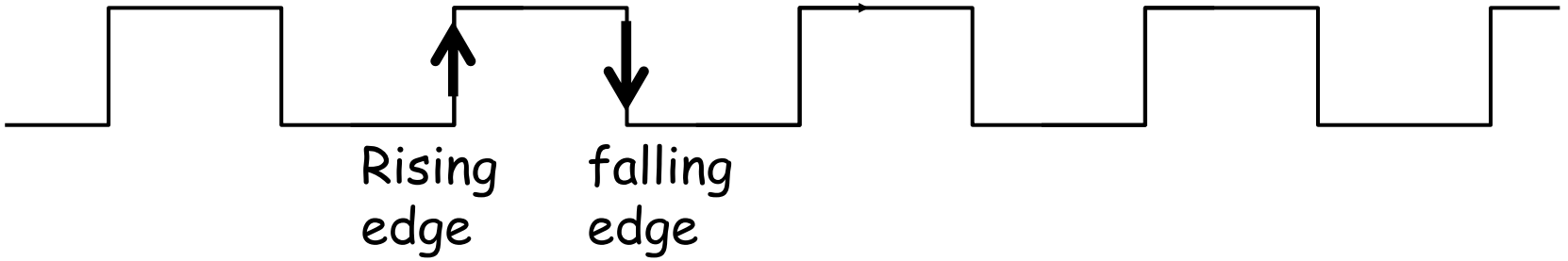
- Outputs are Boolean functions of inputs
- Not an assignment operation, just give the circuit a name
- Respond continuously to changes in inputs
  - With some, small delay



Overview of Logic Design

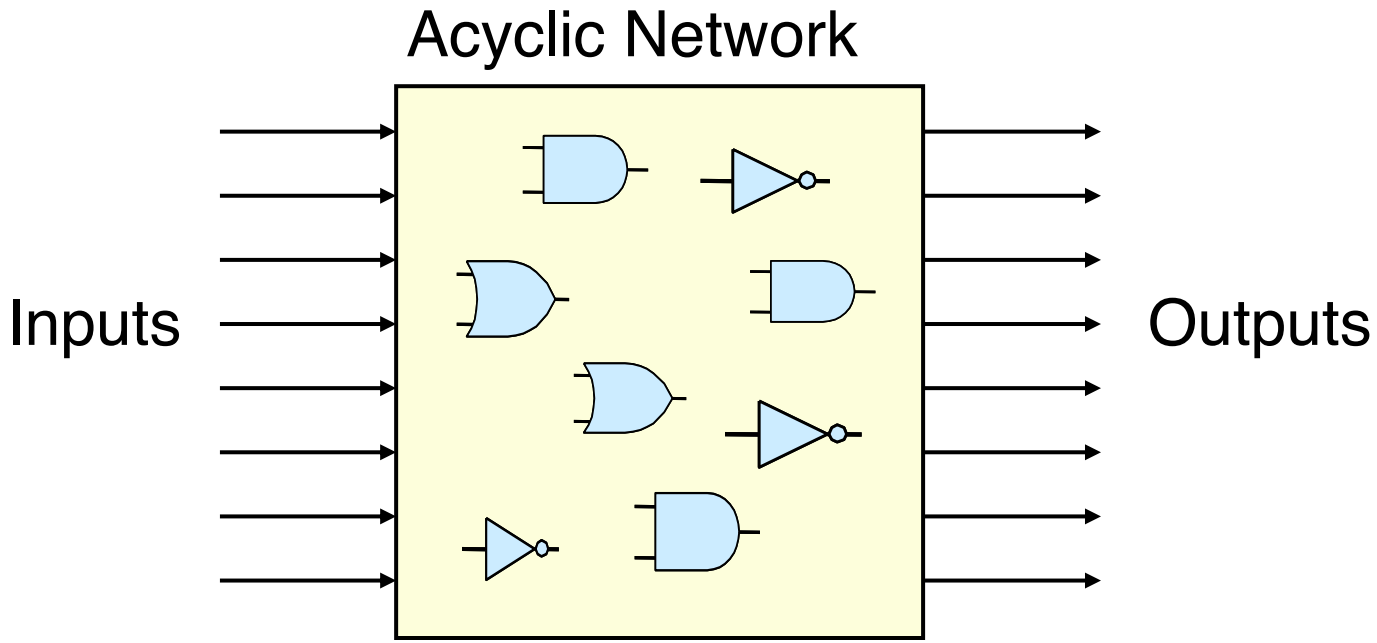
- Electronic circuits are used to
  - Compute functions on bits (computational logic)
  - Store bits in different kind of memory elements (memory elements)
- Clock Signals
  - Are used to regulate updating of the memory elements

Clock : a periodic signal



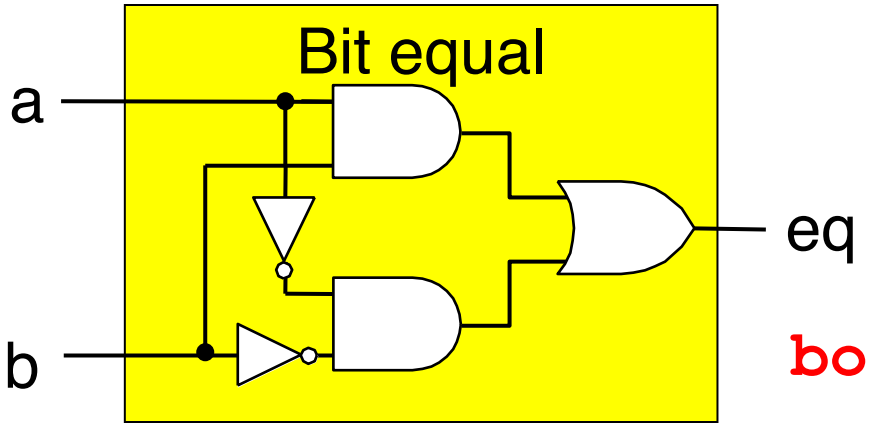
Combinational Circuits

Combinational Circuits



- Acyclic Network of Logic Gates
  - Continuously responds to changes on inputs
  - Outputs become (after some delay) Boolean functions of 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

Hardware Description Language

- Once designers created circuit designs by
  - drawing schematic diagrams of logic circuits
    - first with paper and pencil
    - later with computer graphics terminals
- Now most designs are expressed in a HDL
  - a textual notation that looks similar to a programming language
    - that is used to describe hardware structures rather than program behaviors

Hardware Description Language

- The most commonly used languages are
  - Verilog, having a syntax similar to C
  - VHDL, having a syntax similar to the Ada
- These languages were originally designed
  - For creating simulation models of digital circuits
- *Logic synthesis* programs were created
  - that could generate efficient circuit designs from HDL descriptions
  - in the mid-1980s

Hardware Description Language

- There are now a number of commercial synthesis programs
  - Logic synthesis has become the dominant technique for generating digital circuits
- Similar shifts
  - from hand-designed circuits to synthesized
  - from assembly programming to high-level language programming

Hardware Control Language

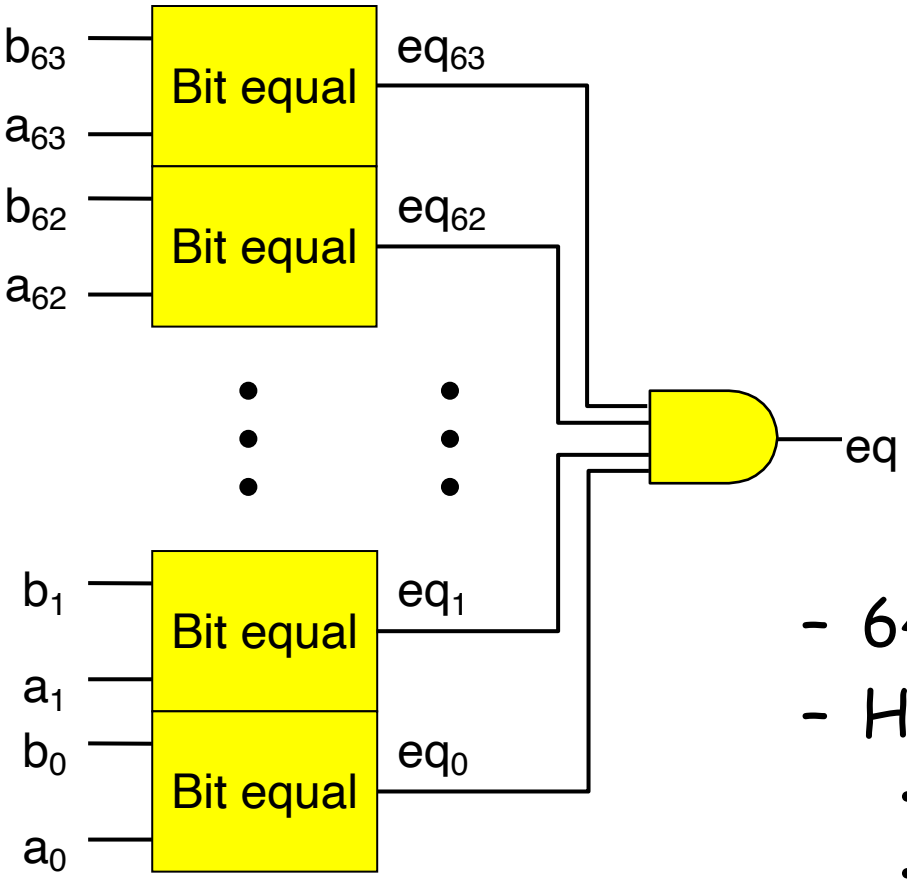
- A language to describe
  - the control logic of the different processor designs
- The control logic
  - is the most difficult part of designing a microprocessor
- Carefully separating out, designing, and testing the control logic
  - we can create a working microprocessor with reasonable effort
- There are tools that can directly translate HCL into Verilog

Practice Problems

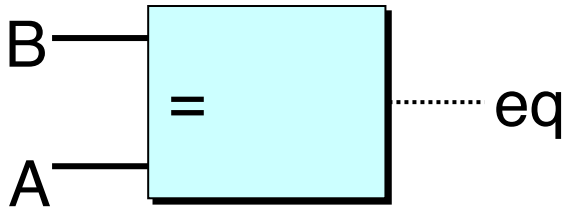
- Write an HCL expression for a signal xor
  - equal to the exclusive-or of inputs a and b.

- What is the relation
  - between the signals xor and eq ?

Word Equality



Word-Level Representation

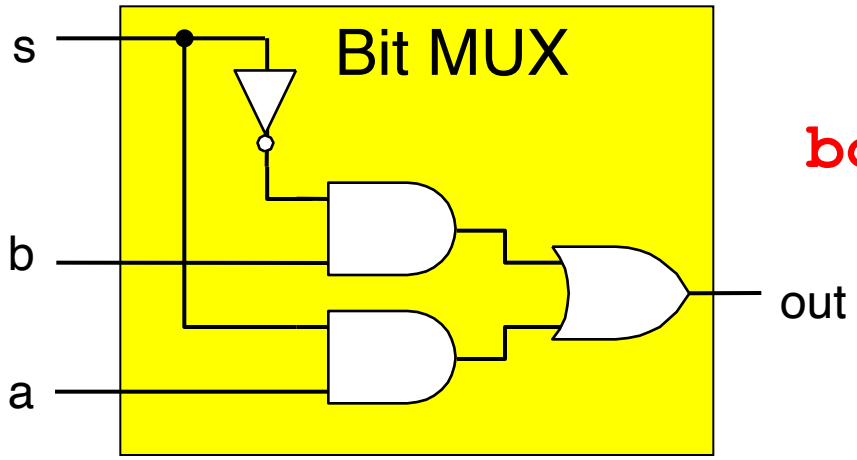


HCL Representation

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

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

Bit-Level Multiplexor

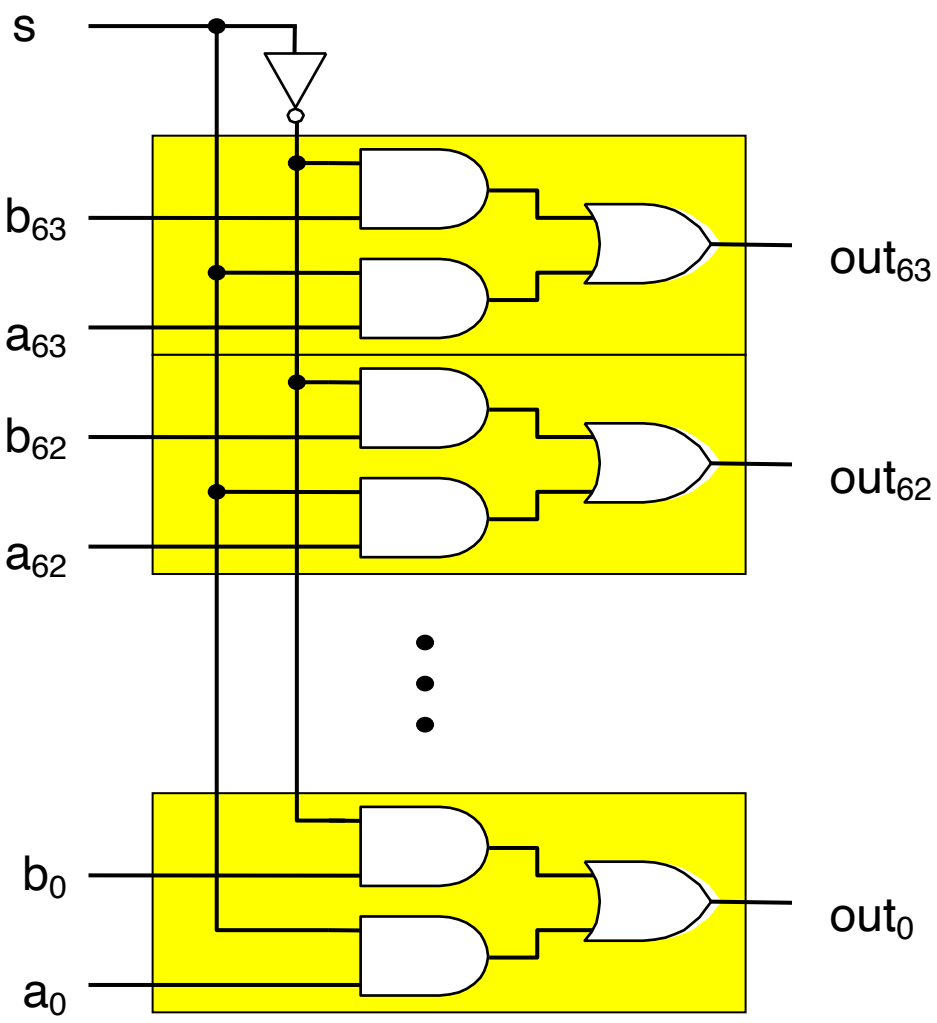


HCL Expression

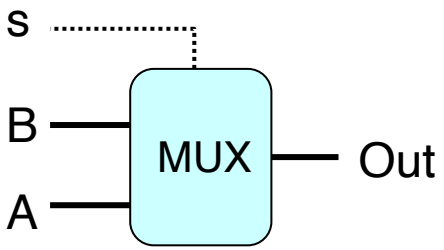
```
bool out = (s&&a)||(!s&&b)
```

- Control signal s
- Data signals a and b
- Output a when s=1, b when s=0
- Its name: MUX
- Usage: Select one signal from a couple of signals

Word Multiplexor



Word-Level Representation



HCL Representation ?

Word Multiplexor

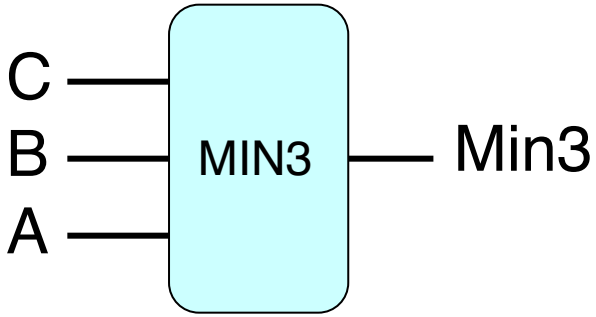
- HCL Representation
  - Select input word A or B depending on control signal s
  - HCL representation
    - Case expression
    - Series of test : value pairs (Don't require mutually)
    - Output value for **first** successful test

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

default case

HCL Word-Level Examples

Minimum of 3 Words



HCL Representation

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

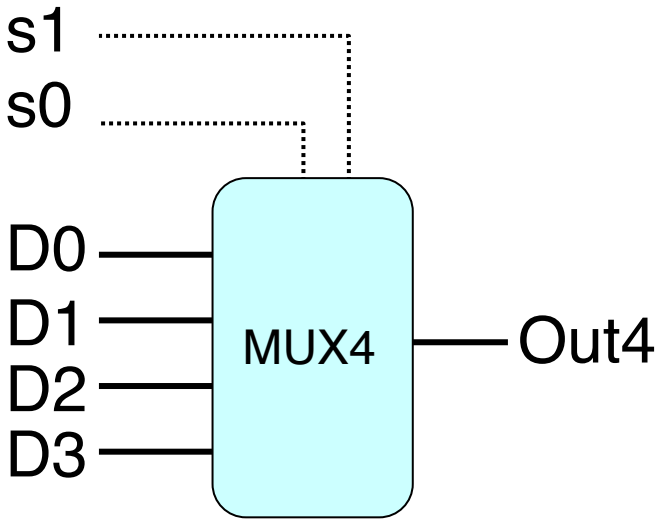
- Find minimum of three input words
- HCL case expression
- Final case guarantees match

Practice Problems

- The HCL code given for computing the minimum of three words contains
  - four comparison expressions of the form *X* <= *Y*
- Rewrite the code to compute the same result
  - but using only three comparisons

HCL Word-Level Examples

4-Way Multiplexor

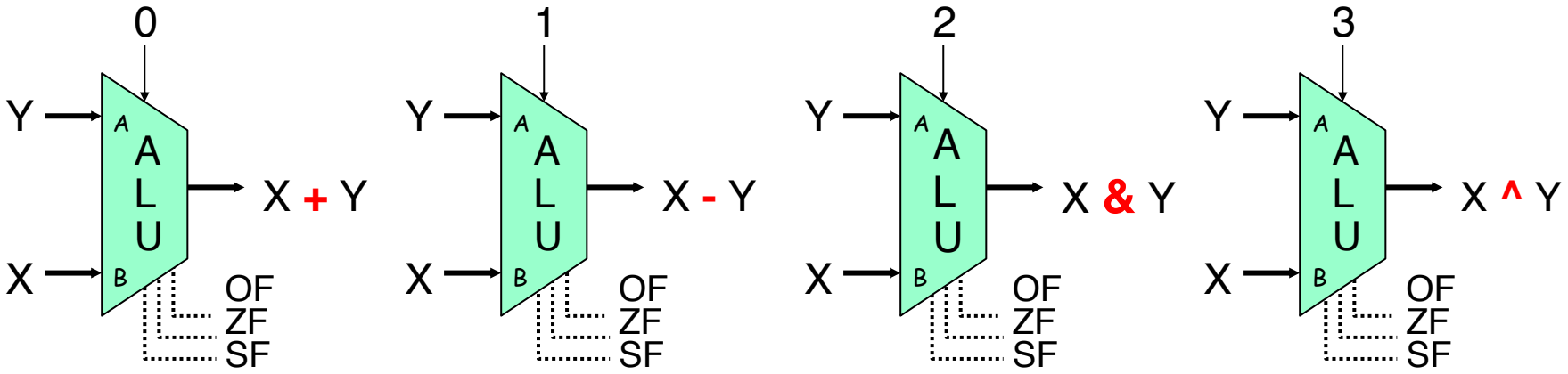


HCL Representation

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

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

Arithmetic Logic Unit

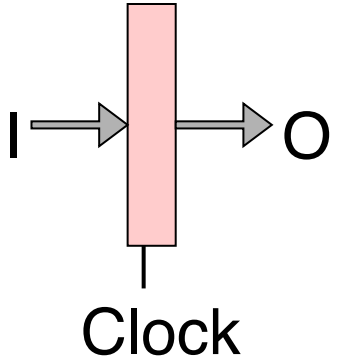


- Combinational logic
  - Continuously responding to inputs
- Control signal selects function computed
  - Corresponding to 4 arithmetic/logical operations in Y86
- Also computes values for condition codes
- We will use it as a basic component for our CPU

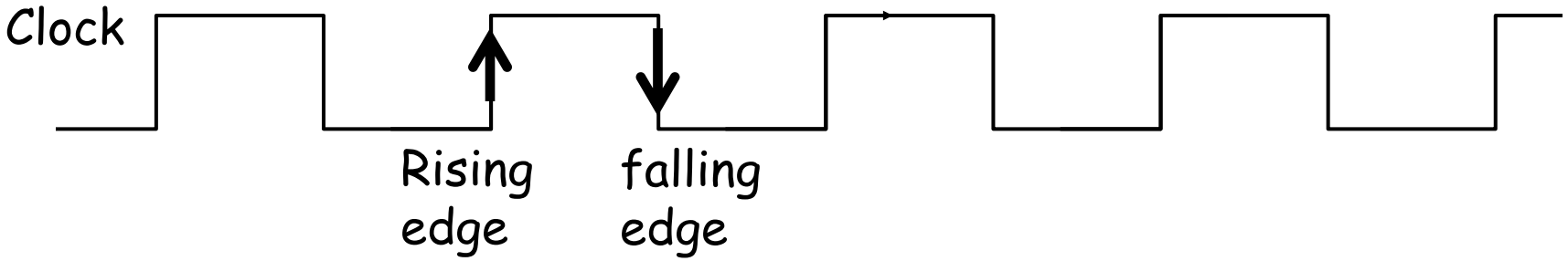
Storage
(Sequential Circuits)

Storage (Clocked Registers)

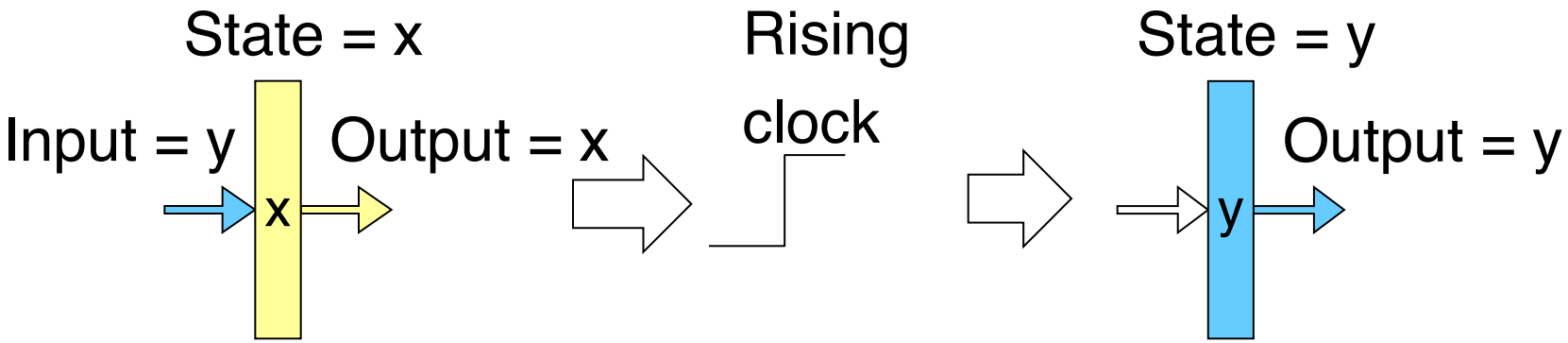
- Clocked Registers
  - e.g. Program Counter(PC), Condition Codes(CC)
  - Hold single words or bits
  - Loaded as clock rises
  - Not "program registers"



- Clock
  - determines when new values are to be loaded into the devices

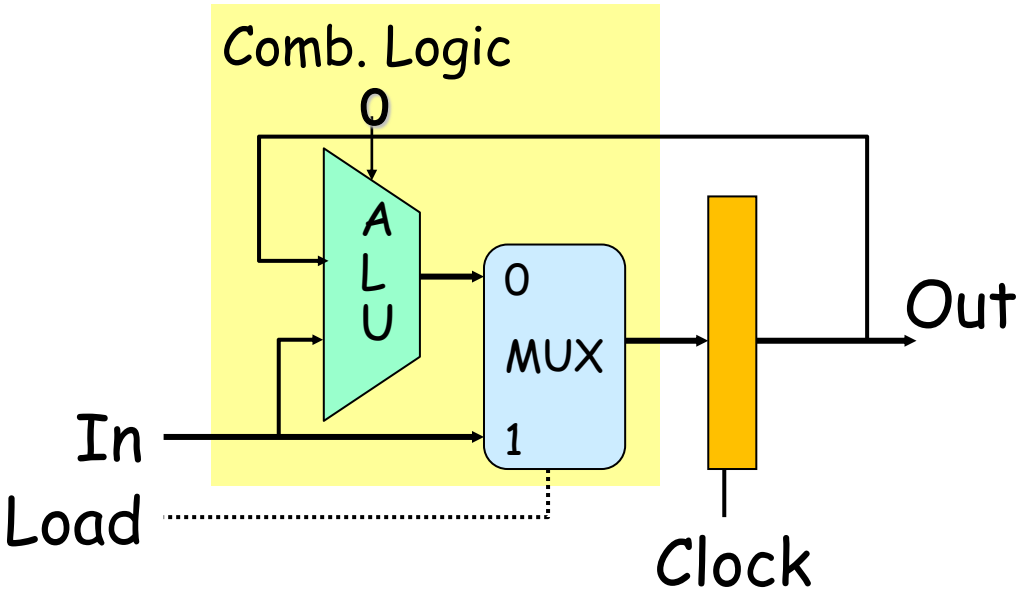


Register Operation

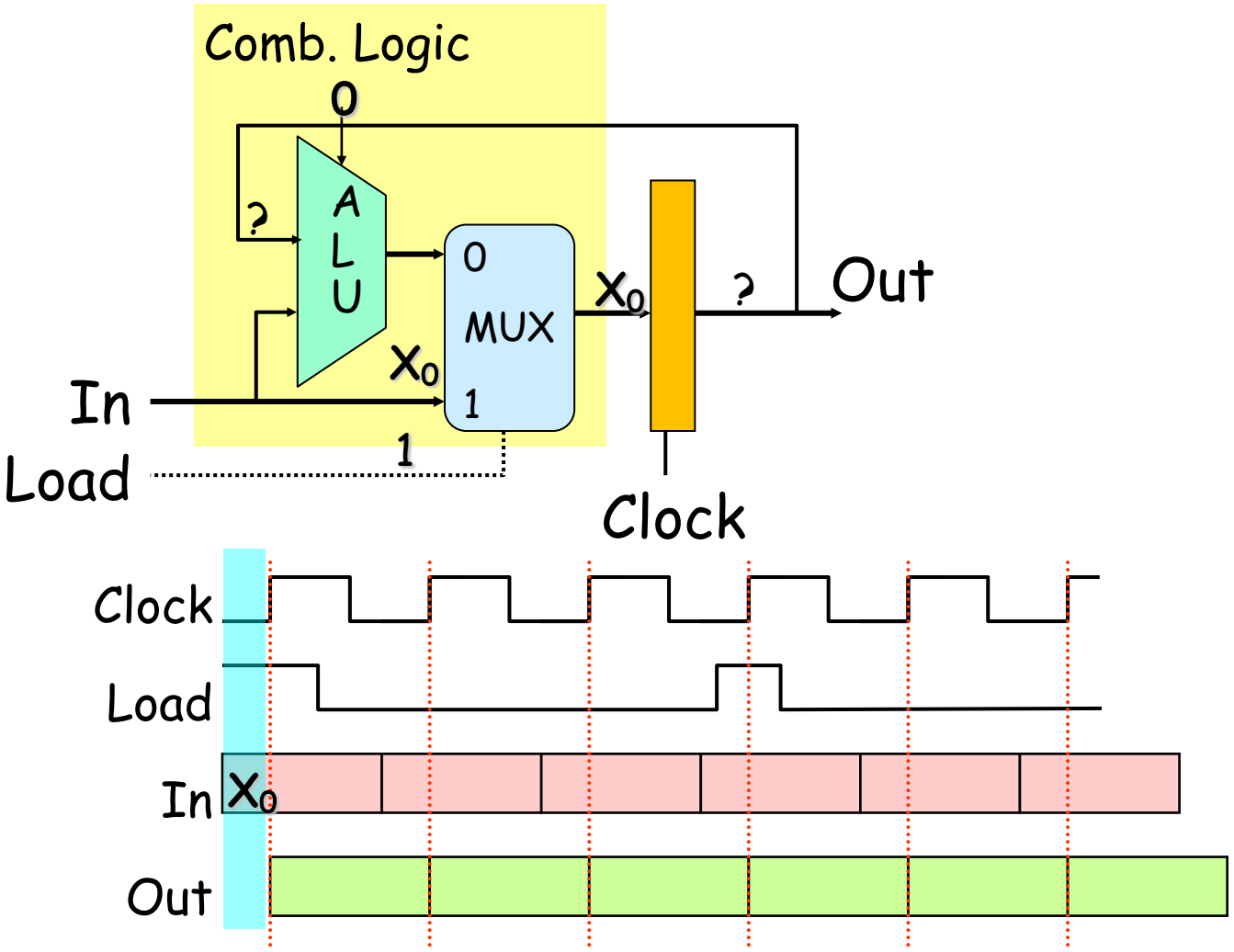


- Stores data bits
- For most of time acts as barrier between input and output
- As clock rises, loads input

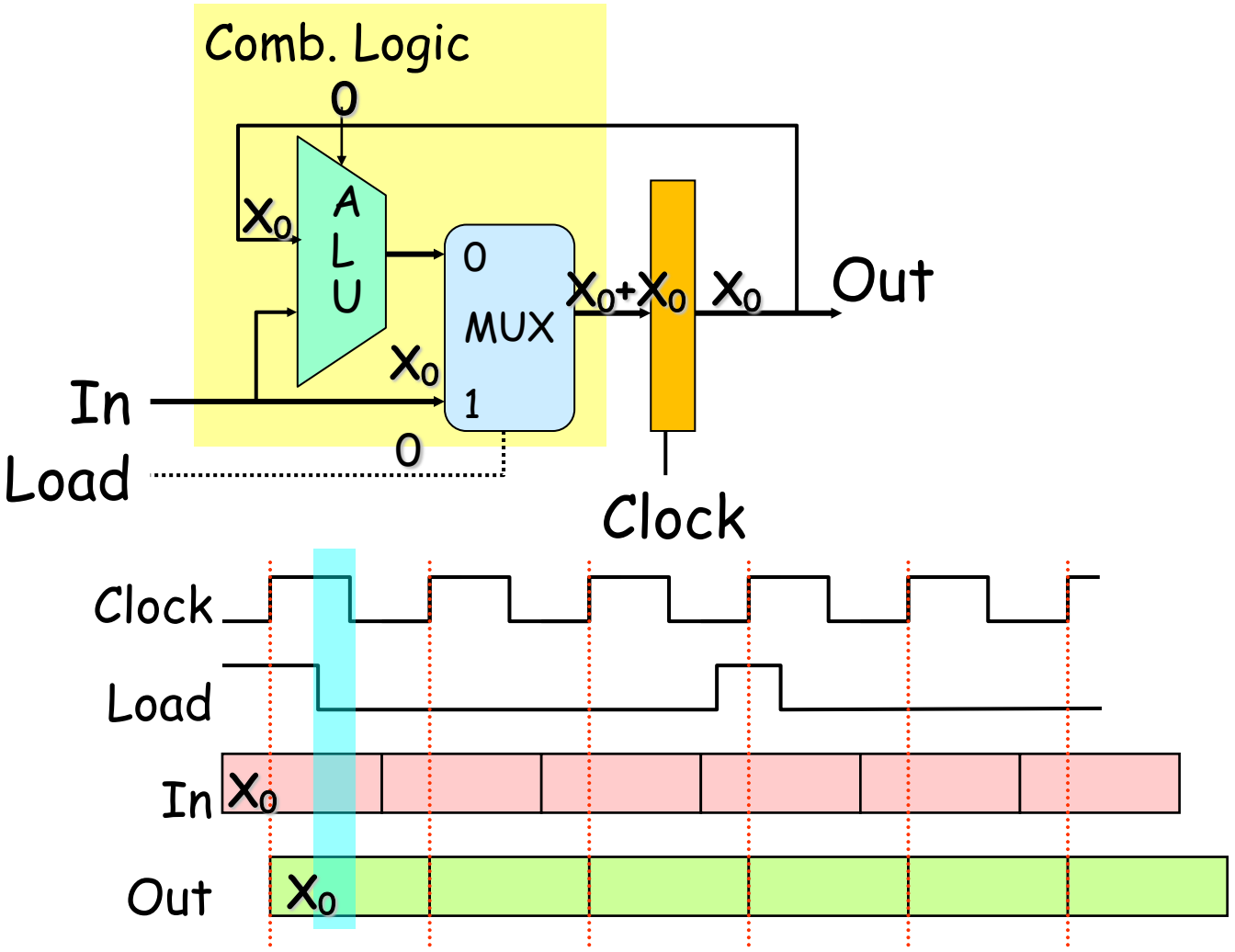
State Machine Example



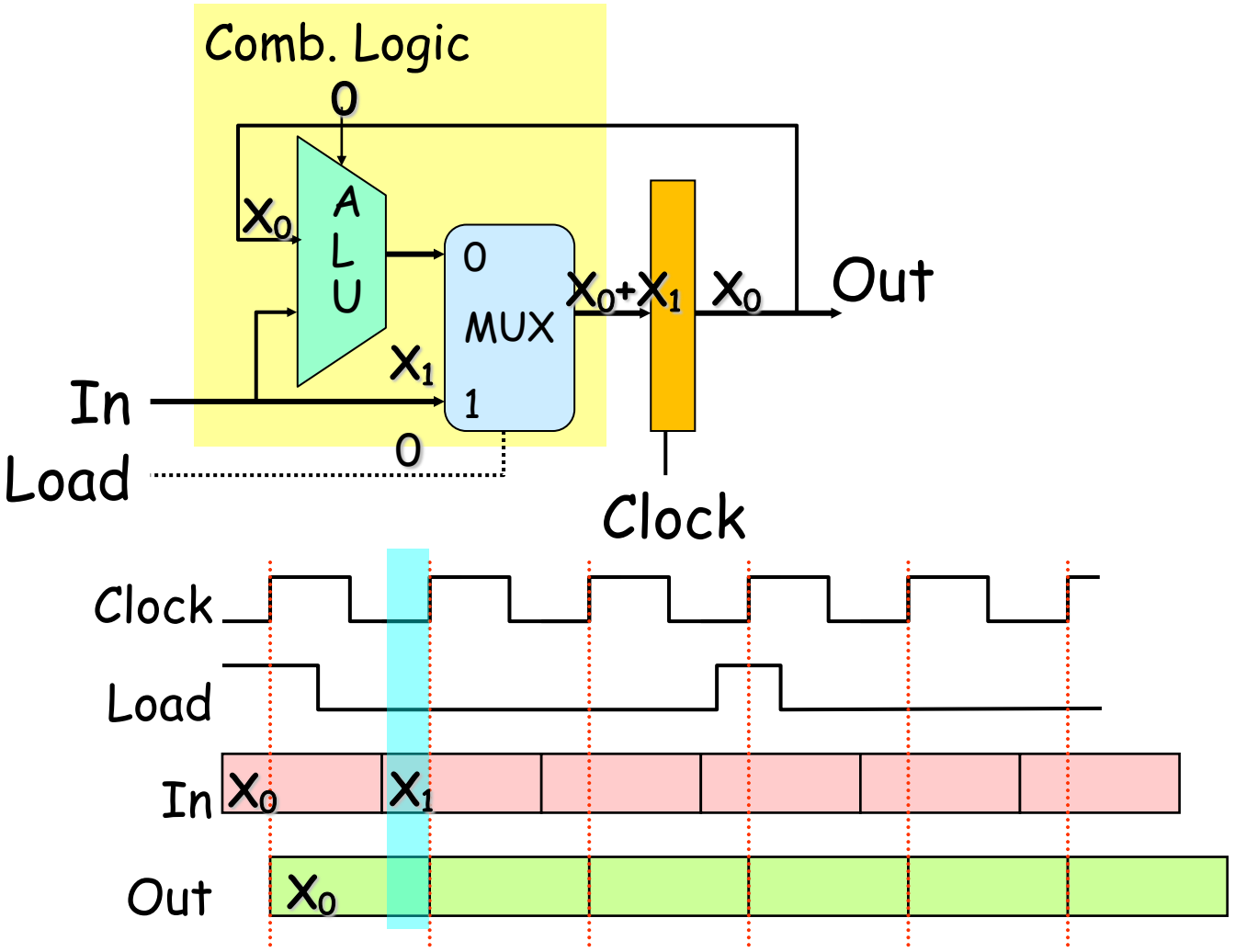
State Machine Example



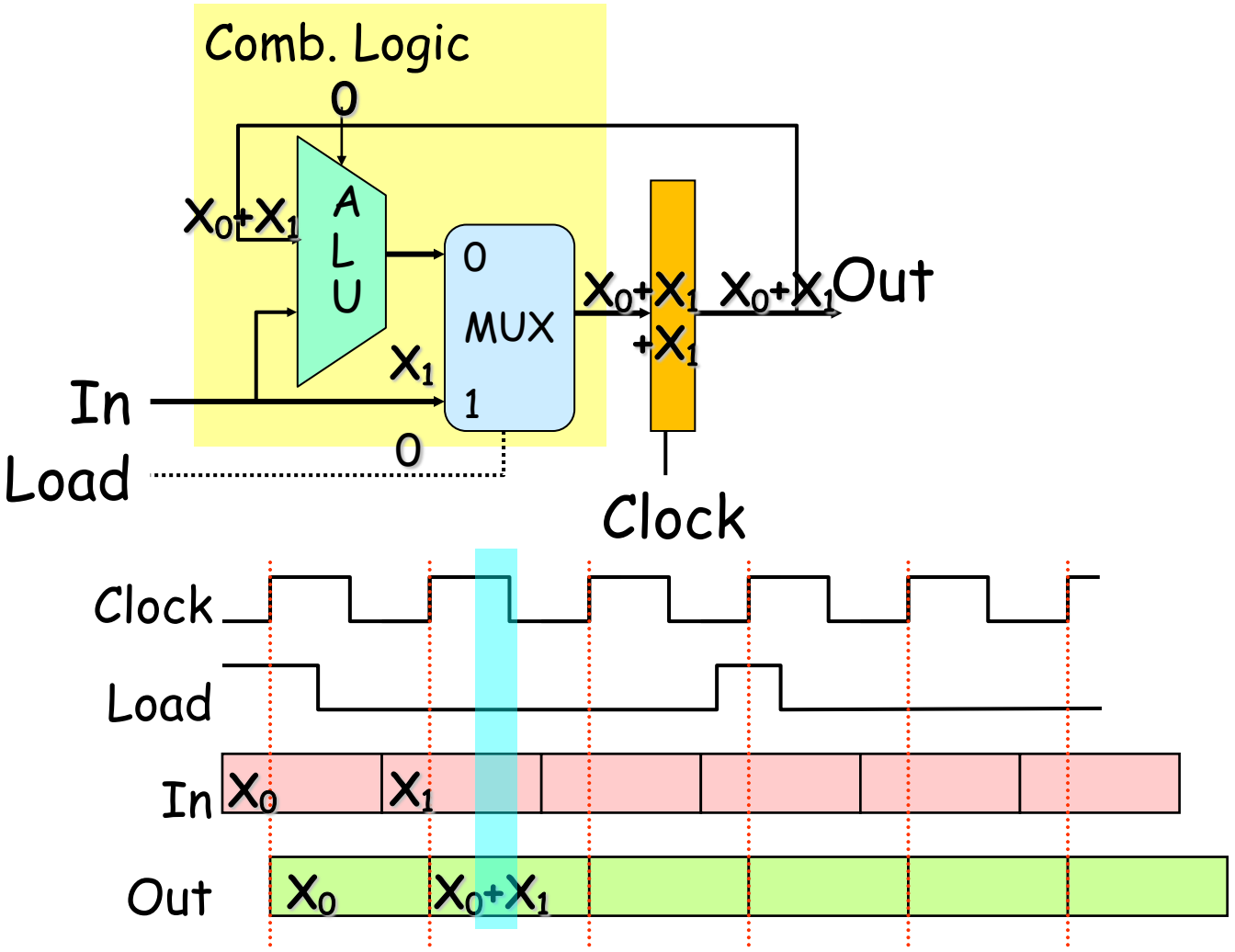
State Machine Example



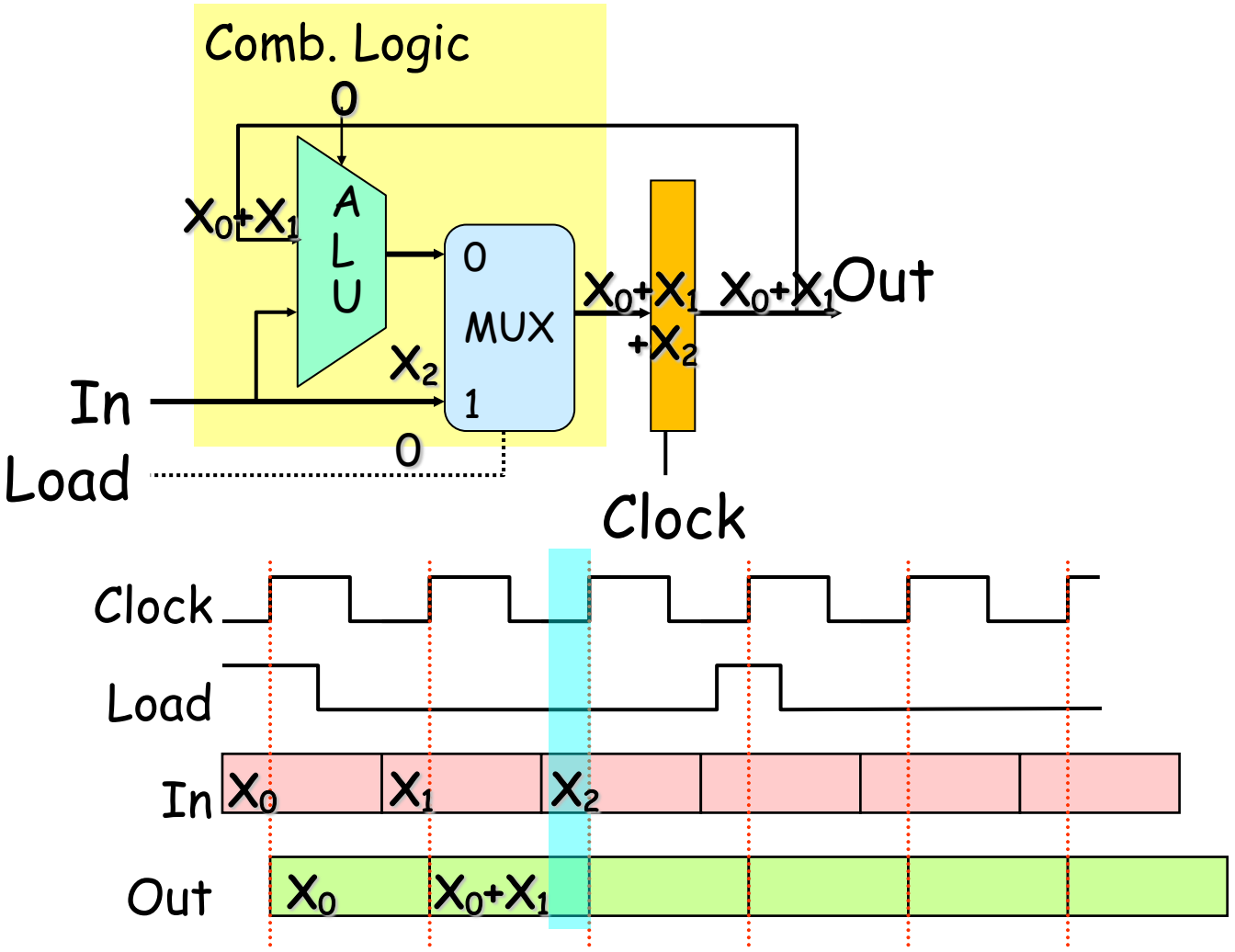
State Machine Example



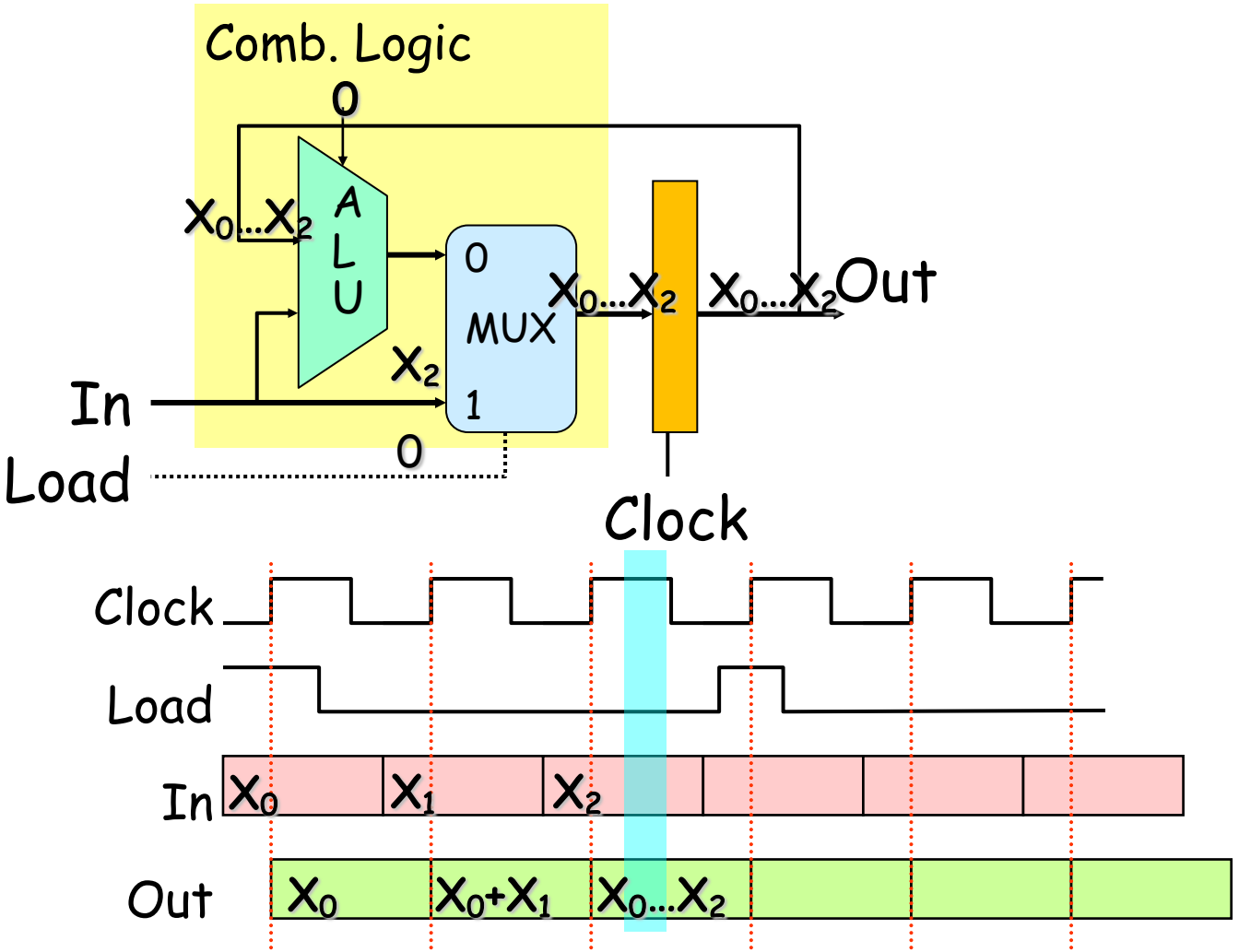
State Machine Example



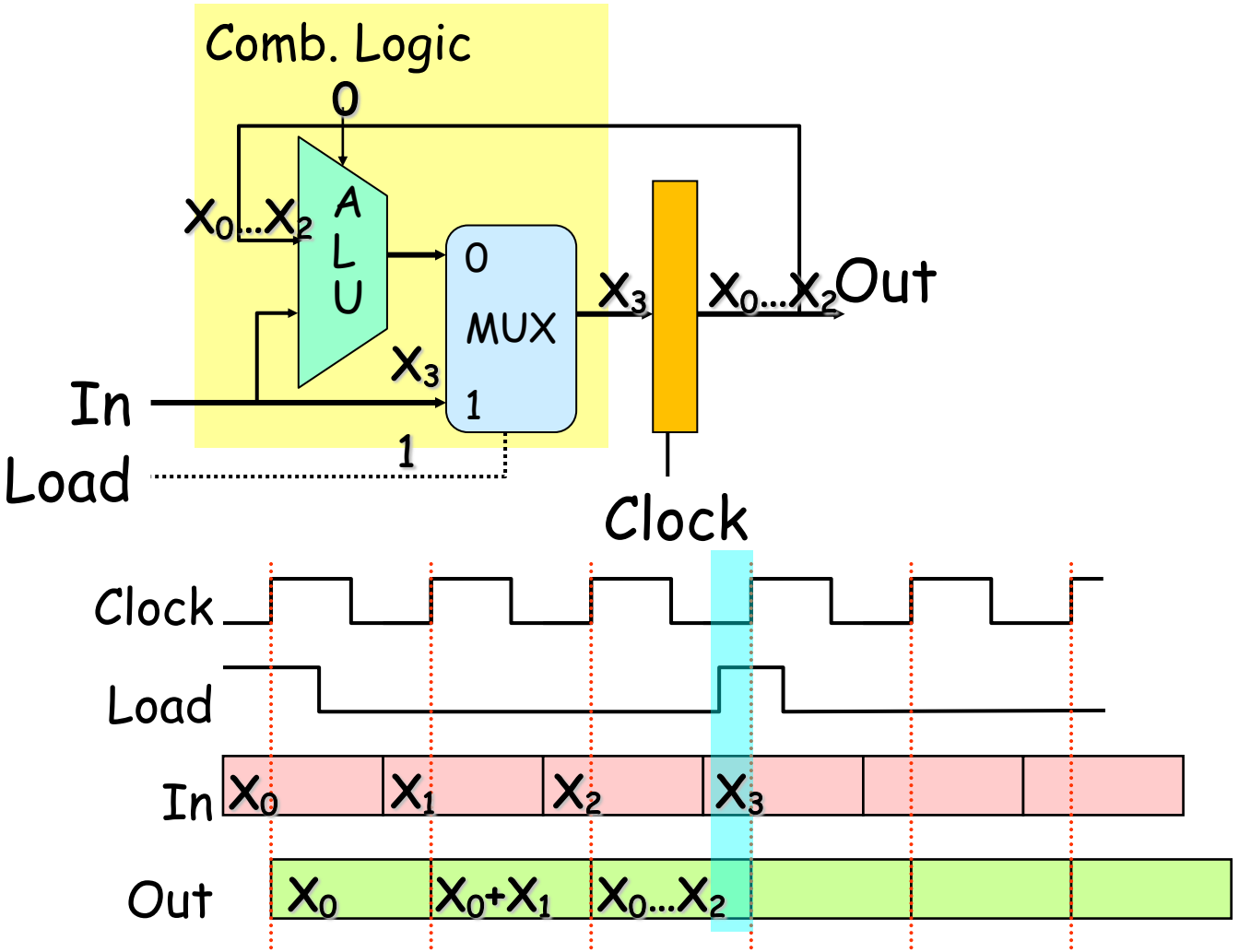
State Machine Example



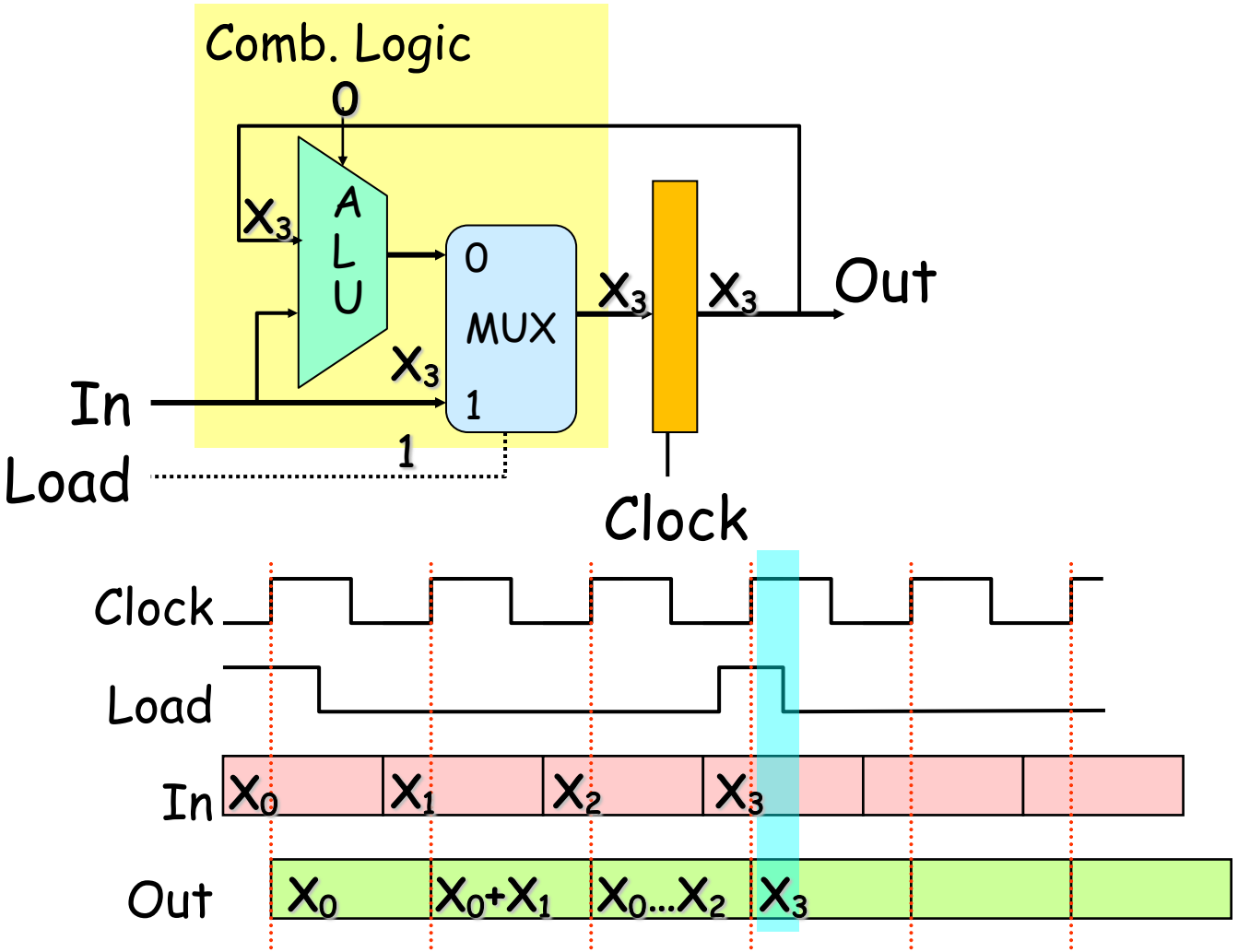
State Machine Example



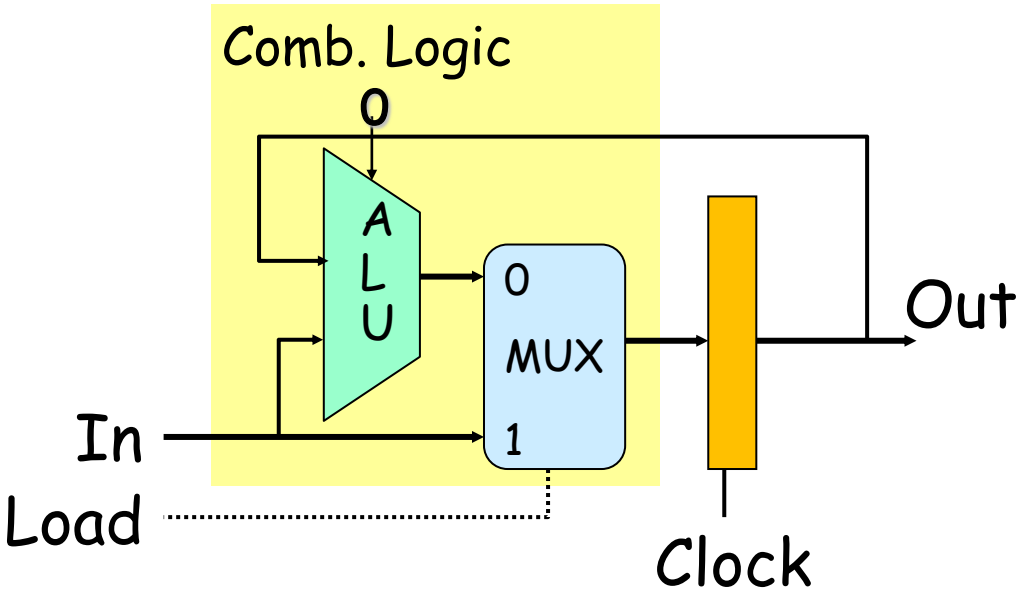
State Machine Example



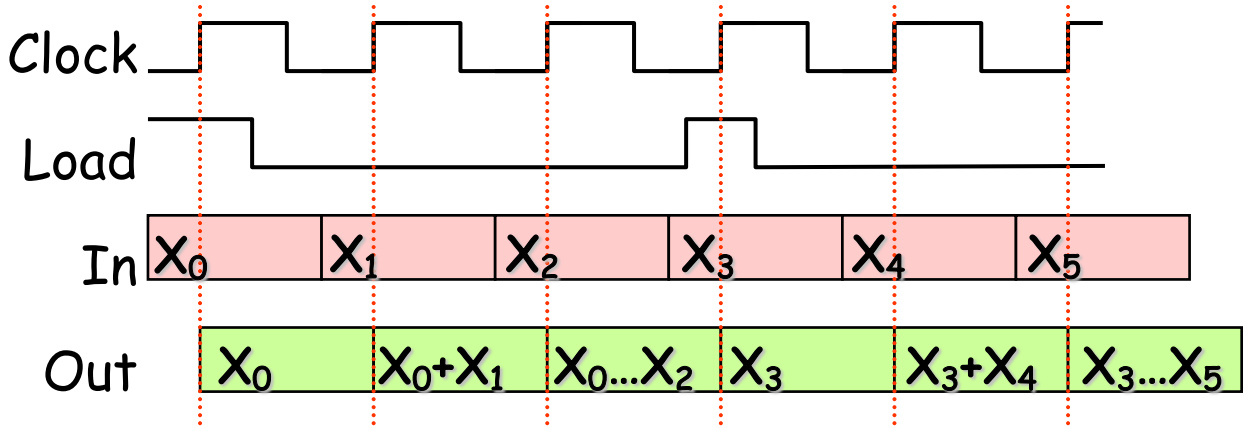
State Machine Example



State Machine Example



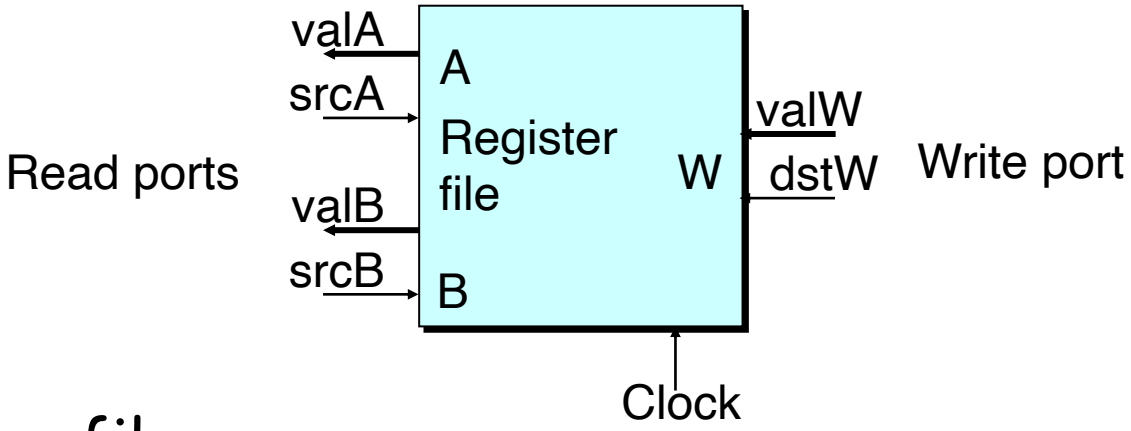
- Accumulator circuit
- Load or accumulate on each cycle



Storage (Random-access memories)

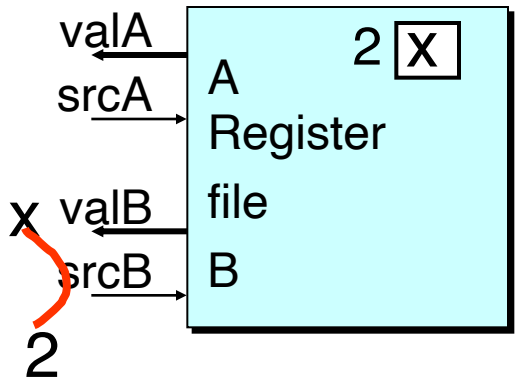
- Random-access memories
  - e.g. Register File, Memory
  - Hold multiple words
    - Address input specifies which word to read or write
  - Possible multiple read or write ports
  - Read word when address input changes
  - Write word as clock rises

Register File

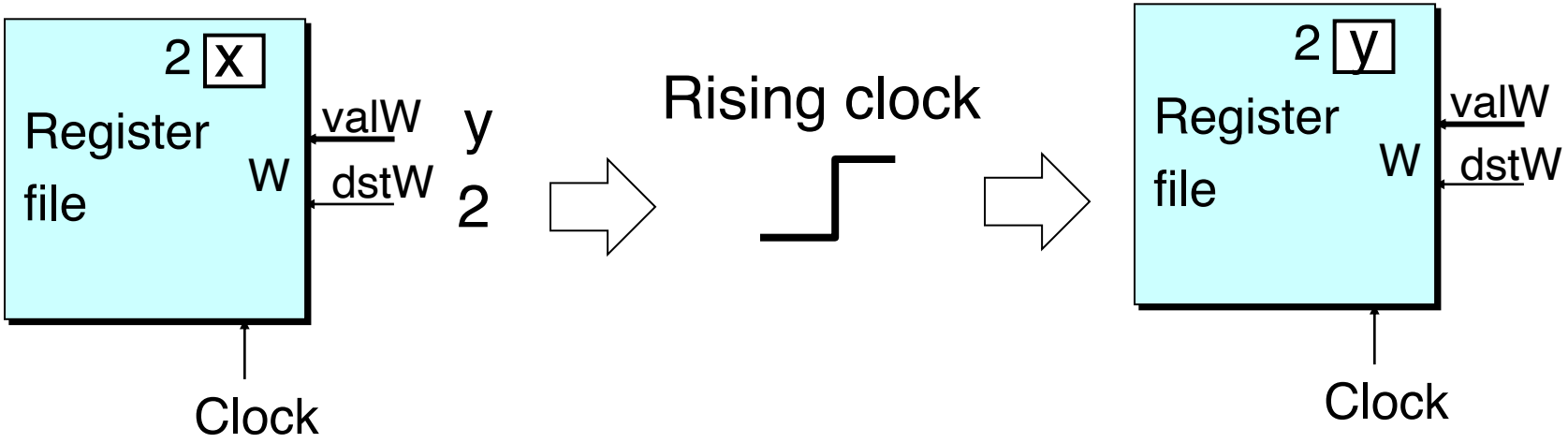


- Register file
  - Holds values of program registers
  - `%rax`, `%rsp`, etc.
  - Register identifier serves as address
    - ID "F" implies no read or write performed
- Multiple Ports
  - Can read and/or write multiple words in one cycle
    - Each has separate address and data input/output

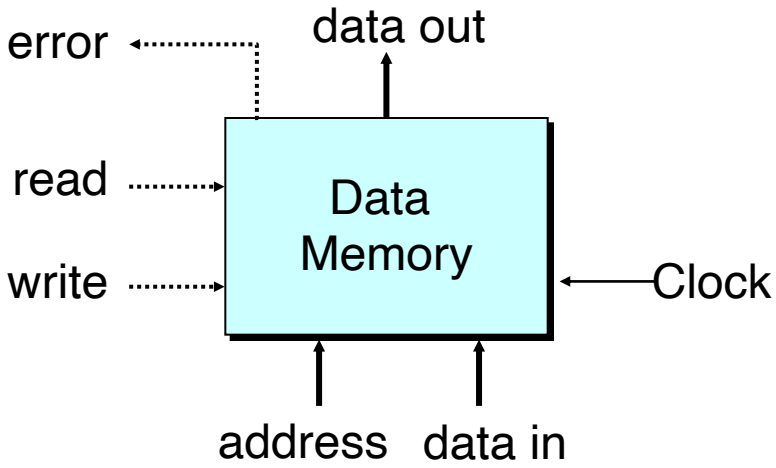
Register File Timing



- Reading
  - Like combinational logic
  - Output data generated based on input address (After some delay)
- Writing
  - Like register
  - Update only as clock rises



Memory



- Memory
  - Holds program data and instructions
- Ports
  - A single address input
  - A data input for writing, and a data output for reading
  - Error signal means invalid address

Usage of the Storage Circuits

- Clocked Registers
  - Hold single words (e.g., PC, CC)
  - Loaded as clock rises
- Random-access memories
  - Hold multiple words (e.g., Register File, Memory)
  - Possible multiple read or write ports
  - Read word when address input changes
  - Write word as clock rises