CSSE132
Introduction to Computer Systems

10 : Sequential Logic

March 19, 2013

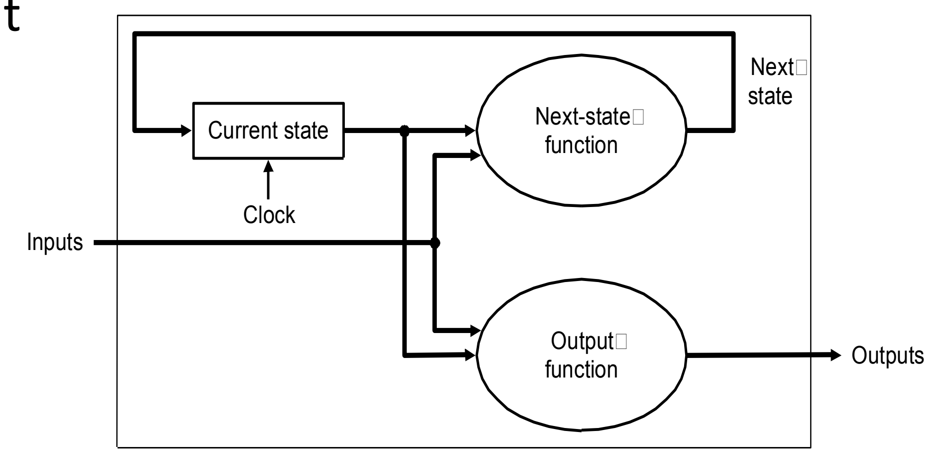
Today: Sequential Logic

- Sequential logic
- Clocks
- Latches
- Flip-flops
- Build a register file
- Memory

Sequential Logic

- **Combinational logic**
  - Defined by Boolean expression
  - Output based only on input

- **Sequential logic**
  - Maintains stored values or state
  - Retains data for later use
  - Output based on previous input
  - Can build state machines

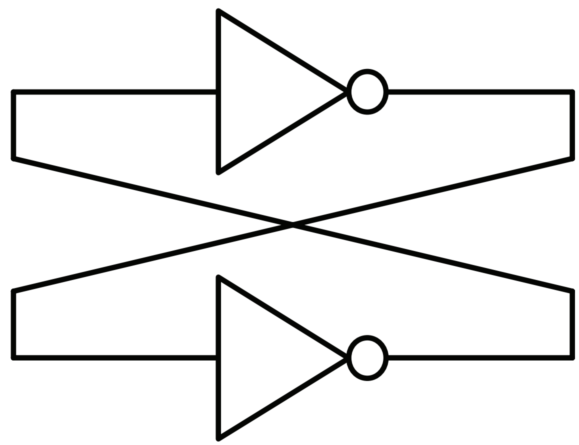


Clock

- **Produce regular changing signal**
  - Special hardware that produces oscillating signal
  - Several waveform outputs
- **Square waveform**
  - Has period (frequency)
  - Duty cycle when power is on
    - Rising edge (power up)
    - Falling edge (power down)
  - Duty cycle often 50% of period
- **Will allow us to transition between states**

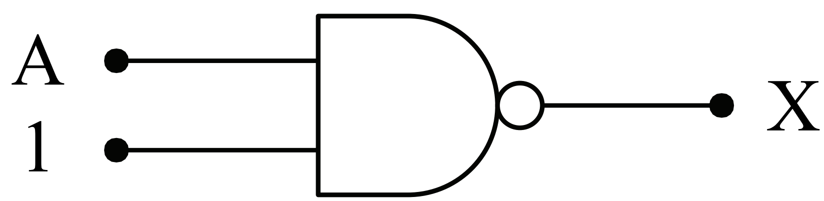
Memory circuit

- **Two invertor loop**
  - Preserve signal
- **Circuit is hard to use**
  - Can read stored value
  - Can’t update stored value
- **Idea is useful**



Memory circuit

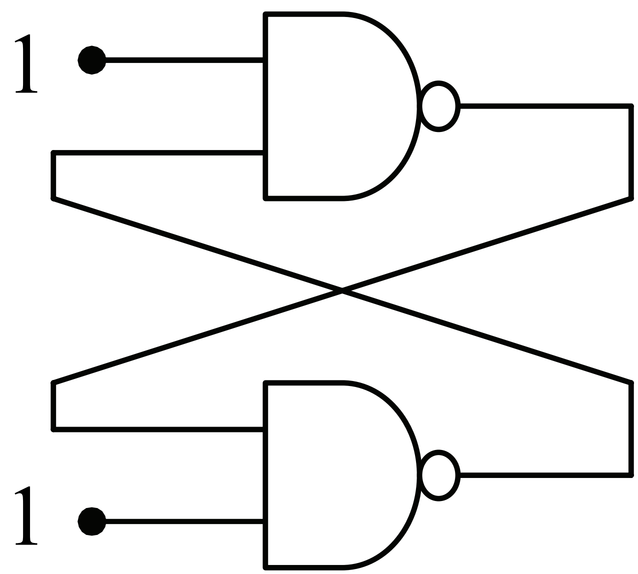
- **Build invertor with NAND**
  - Set inputs to 1
  - Same as invertor



A	1	X
0	1	1
1	1	0

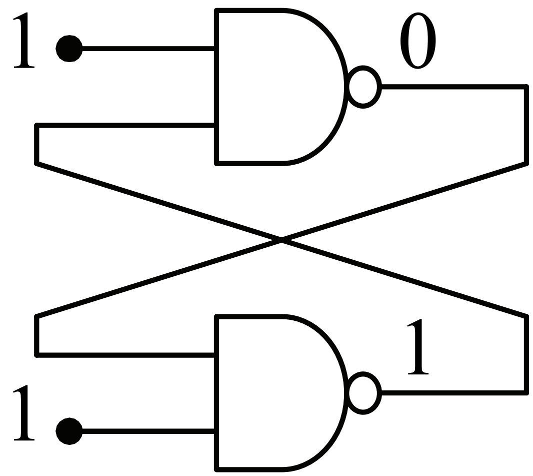
Memory circuit

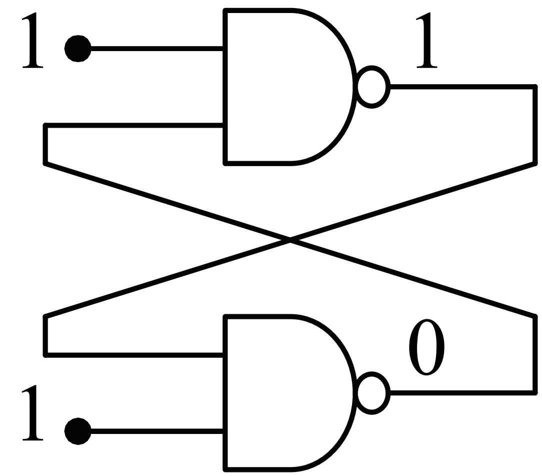
- **Build loop with NAND**
  - Same idea



Memory circuit

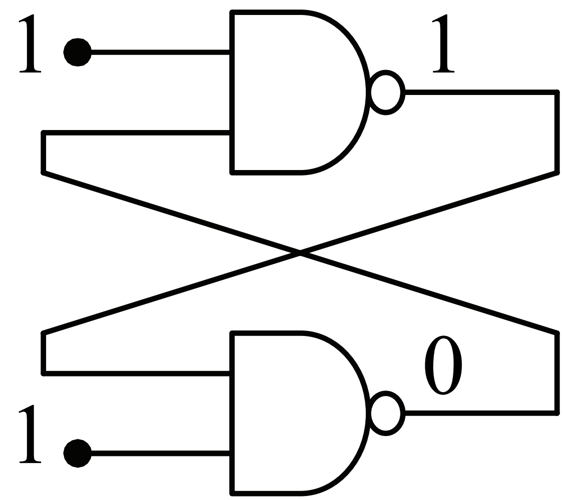
- **Build loop with NAND**
  - Same idea
  - Can store 0 or 1



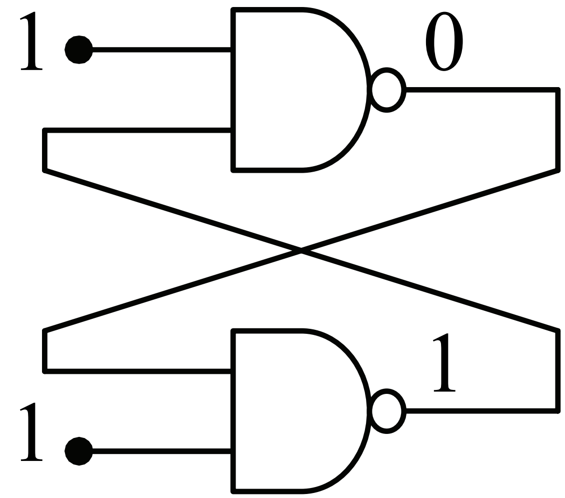


Changing value

- **Toggle <u>top</u> input**
  - Set to 0
  - Wait a bit
  - Set back to 1



- **What new output if originally**
  - Top NAND output is 1?
  - Top NAND output is 0?

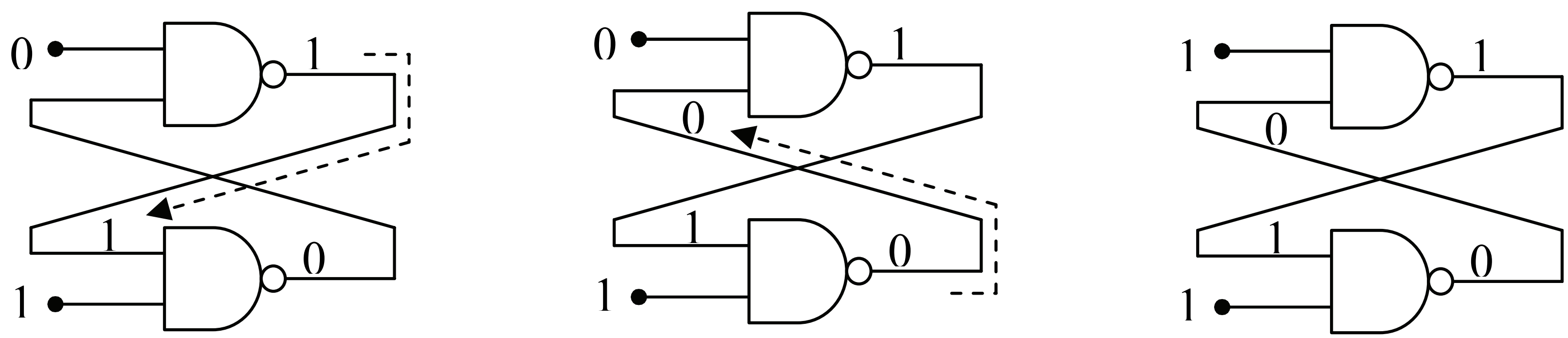


A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Changing value

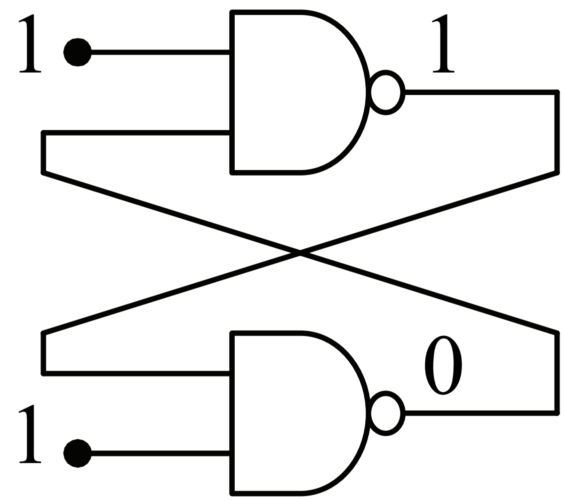
- **Change top input**
  - Top input set to 0
  - Stored value becomes 1
  - 1 value is retained even if input goes to 1

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

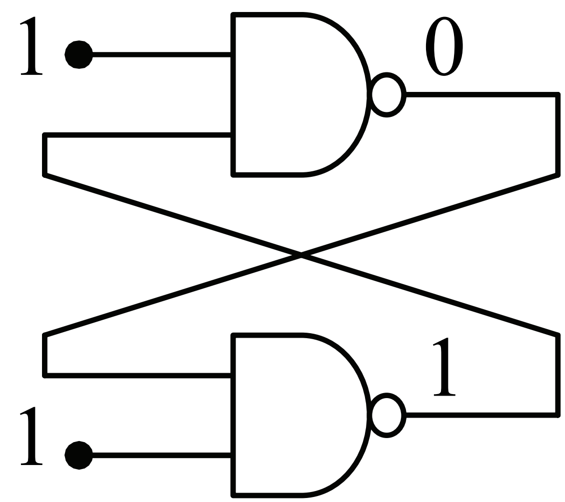


Changing value

- **Toggle <u>bottom</u> input**
  - Set to 0
  - Wait a bit
  - Set back to 1



- **Initial value does not matter!**

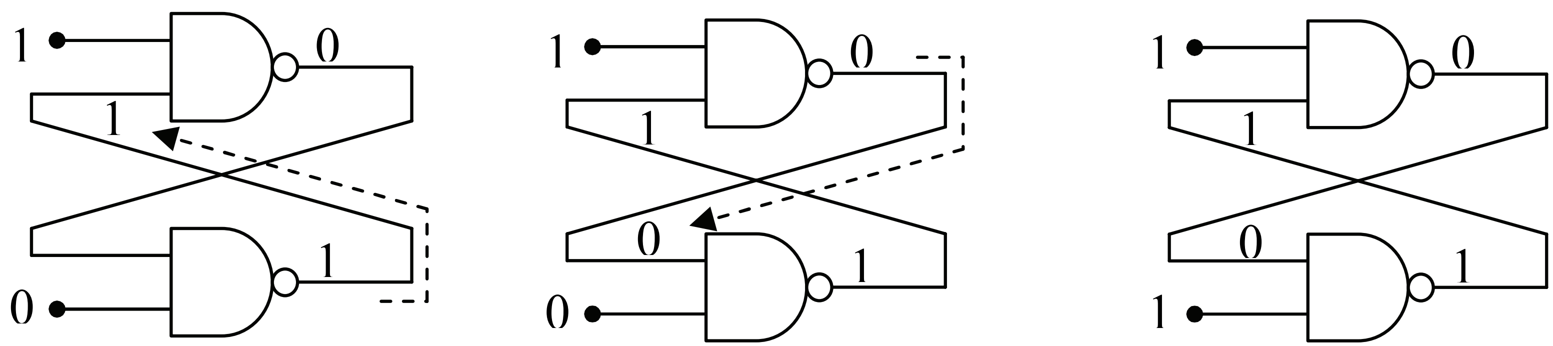


A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Changing value

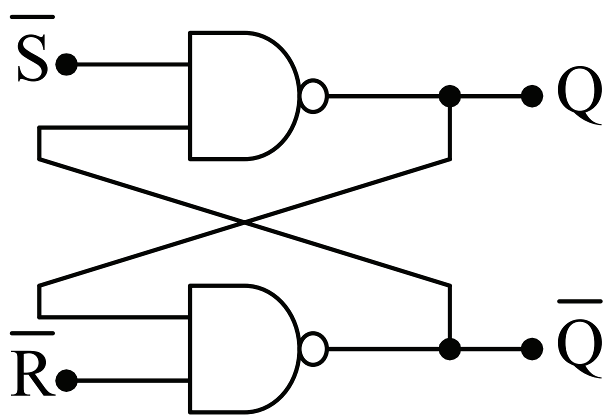
- **Change bottom input**
  - Bottom input set to 0
  - Stored value becomes 0
  - 0 value is retained even if input goes to 1

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0



SR Latch

- **Two inputs, S,R (set, reset)**
  - Change stored value between 1,0
- **Two outputs Q, Q’**
  - Q is stored value
  - Q’ must always be opposite of stored value



S’	R’	Q	Q’
0	0	U	U
0	1	1	0
1	0	0	1
1	1	Q_0	Q_0’

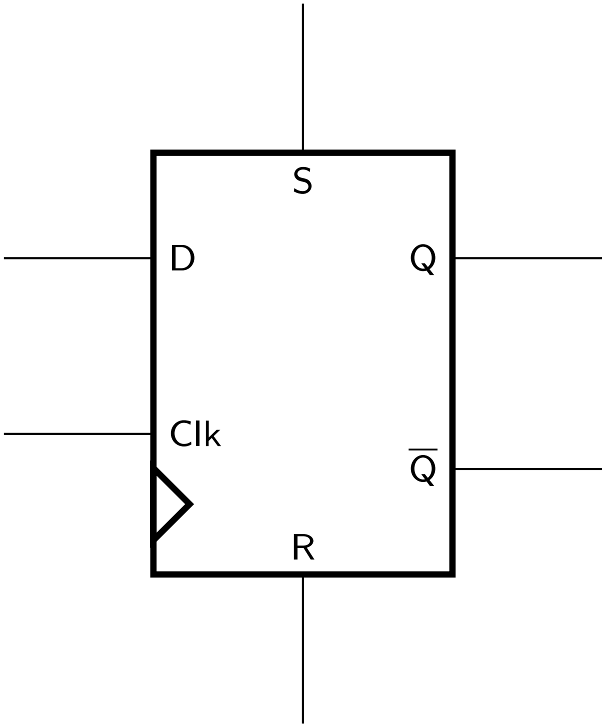
Storage cells

- **Many different kinds**
  - Simple ones called ‘latches’
  - Bigger, clocked ones called ‘flip-flops’
- **Maintain state/stored value**
  - Represented by Q
  - Can transition between states
    - Many conventions
    - Previous/initial state: Q_0, Q_{prev}, Q_{t-}
    - Next state : Q, Q_{next}, Q_{t+}
- **Can have undefined state**
  - Represented by U

Clocked storage

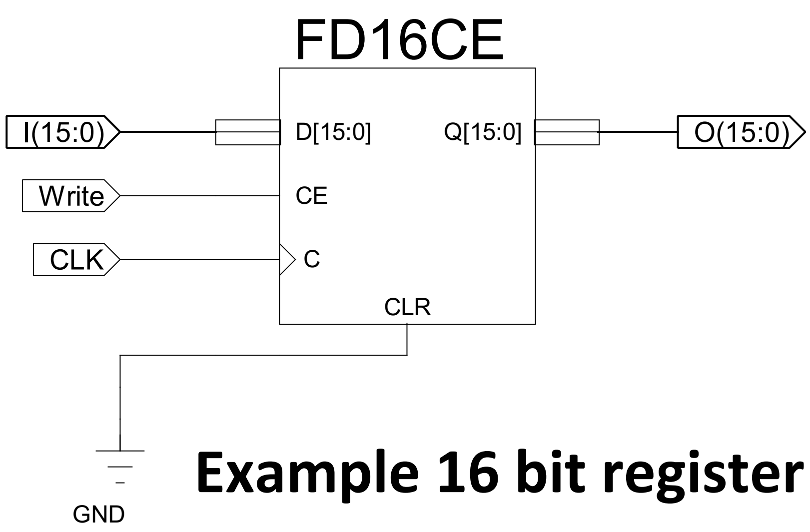
- **D flip-flop**
  - Has 4 inputs (Data, Set, Reset, Clock)
  - Has 2 outputs (Q, Q’)
  - Changes value on clock edge
    - We will use rising edge

D	Clk	Q	Q’
X	0	Q_0	Q_0’
X	1	Q_0	Q_0’
X	dn	Q_0	Q_0’
0	up	0	1
1	up	1	0



Register

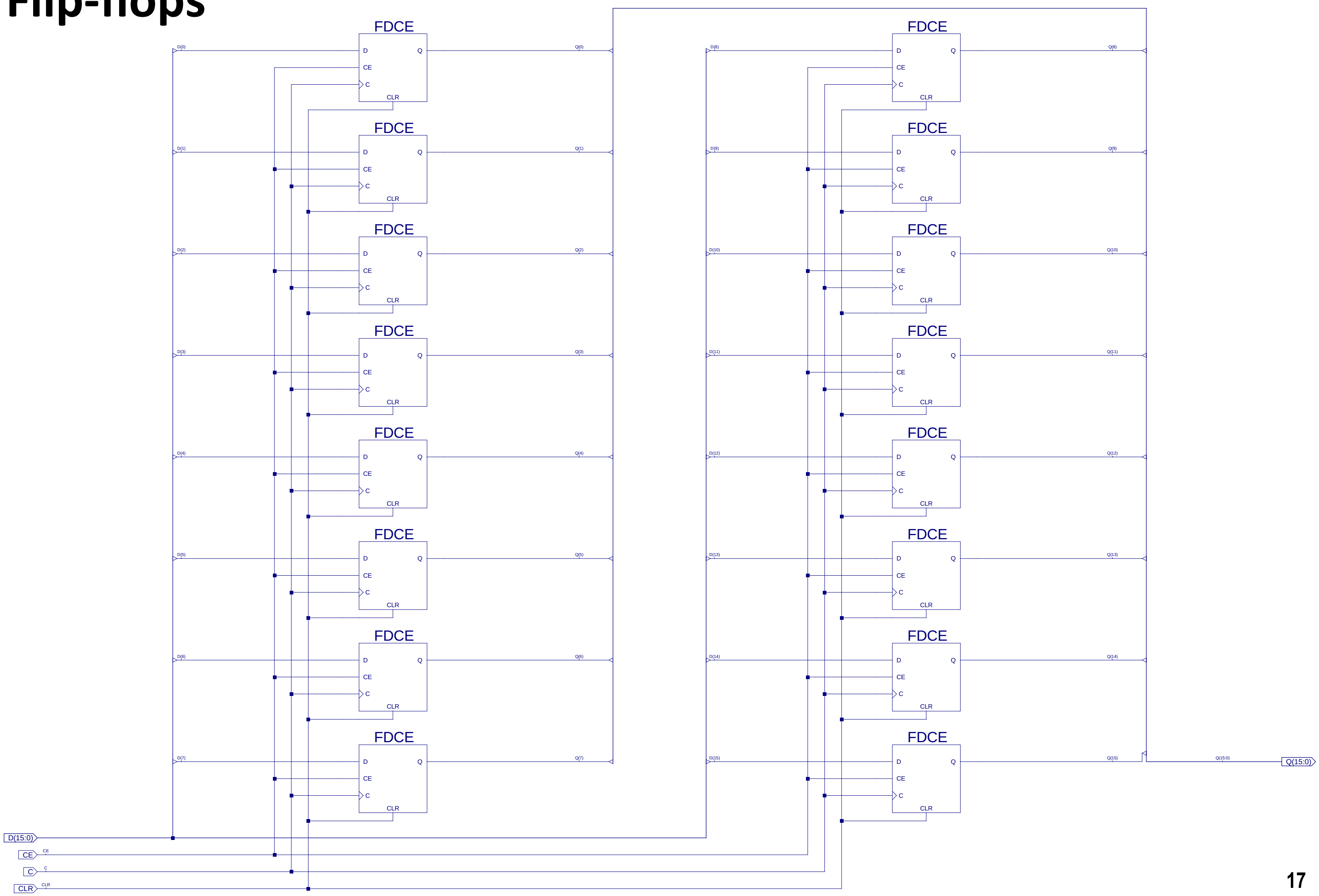
- **Stores binary values**
  - Several flip-flops grouped together
  - Can store 1 bit for each flip-flop
- **Records new value on clock edge**
  - Can be controlled with write-enable bit
- **Allows values to be saved in CPU**
  - Results of calculations
  - Query results from memory
  - Current executing instruction
  - Often word sized



Example 16 bit register

16 bit Register Internal

- 16 D Flip-flops

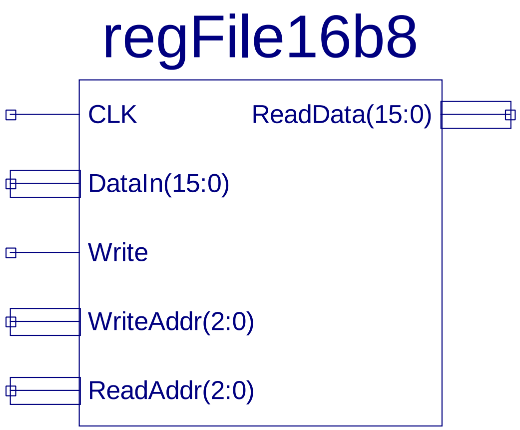


More registers

- **Useful to save several values at once**
  - Multiple register to hold values
- **Give each register/container an ID**
  - Probably a number
- **Useful to select specific register**
  - For reading or writing

Register File

- **Collection of registers**
- **Method to select a single register**
  - Input read or write address
- **Read or write values**
  - Input write data, output read data



- **Basic storage unit for CPU**
  - Stores memory fetches
  - Stores calculation results
  - Programmer elects to read or write registers

```
put 0xff, reg@2
store reg@3, mem@0xec
add 3, -5, reg@3
```

Mux to select read output

Decoder to select write target

Example 8 register file

Memory

- **Similar to a large register file**
  - Much larger
  - Often slower
- **Address selects byte to manipulate**
  - Read data at byte address
  - Write data at byte address
- **Modern memory**
  - More complex model
  - Hierarchy for read/write
  - Read/writes word size chunks