CSC 252: Computer Organization Spring 2019: Lecture 12

Instructor: Yuhao Zhu

Department of Computer Science
University of Rochester

Action Items:

- **Assignment 3 is due March 1, midnight**

Announcement

- Programming Assignment 3 is due on March 1, midnight
- Mid-term exam: March 7; in class
- Past exam & Problem set: http://www.cs.rochester.edu/courses/252/spring2019/handouts.html

MON	TUE	WED	THU	FRI
25	26	27	28	Mar 1
	Lecture		**Lecture**	**A3 due**
4	5	6	7	8
	Lecture		**Midterm**	

The Need for Storing Bits

- Assembly programs set architecture (processor) states.
  - Register File
  - Status Flags
  - Memory
  - Program Counter

The Need for Storing Bits

- Assembly programs set architecture (processor) states.
  - Register File
  - Status Flags
  - Memory
  - Program Counter
- Every state is essentially some bits that are stored/loaded.

The Need for Storing Bits

- Assembly programs set architecture (processor) states.
  - Register File
  - Status Flags
  - Memory
  - Program Counter
- Every state is essentially some bits that are stored/loaded.
- Think of the program execution as an FSM.

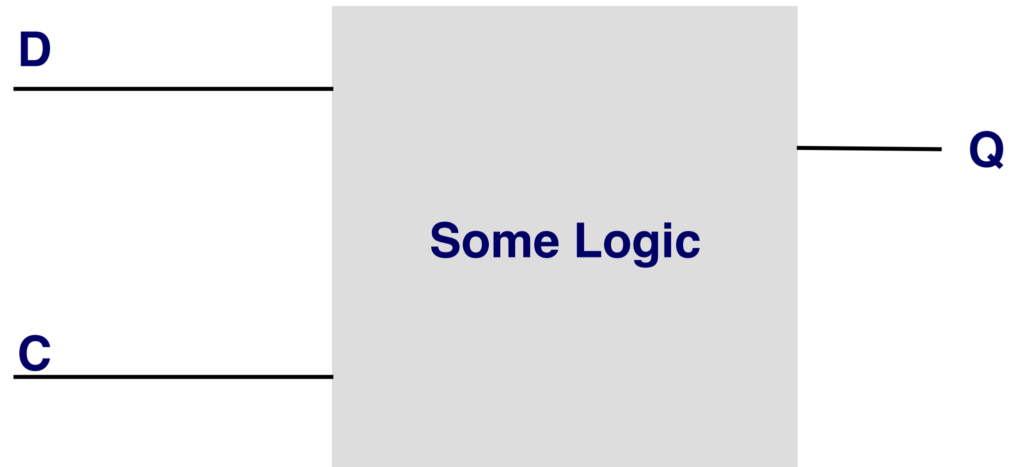
The Need for Storing Bits

- Assembly programs set architecture (processor) states.
  - Register File
  - Status Flags
  - Memory
  - Program Counter
- Every state is essentially some bits that are stored/loaded.
- Think of the program execution as an FSM.
- The hardware must provide mechanisms to load and store bits.

The Need for Storing Bits

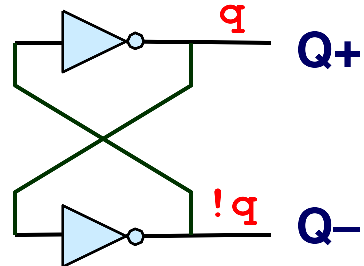
- Assembly programs set architecture (processor) states.
  - Register File
  - Status Flags
  - Memory
  - Program Counter
- Every state is essentially some bits that are stored/loaded.
- Think of the program execution as an FSM.
- The hardware must provide mechanisms to load and store bits.
- There are many different ways to store bits. They have trade-offs.

Build a 1-Bit Storage



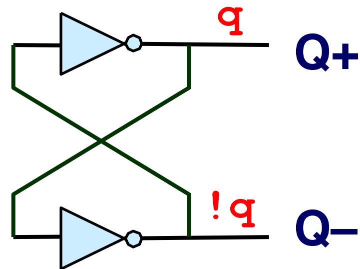
- What I would like:
  - D is the data I want to store (0 or 1)
  - C is the control signal
    - When C is 1, Q becomes D (i.e., storing the data)
    - When C is 0, Q doesn't change with D (data stored)

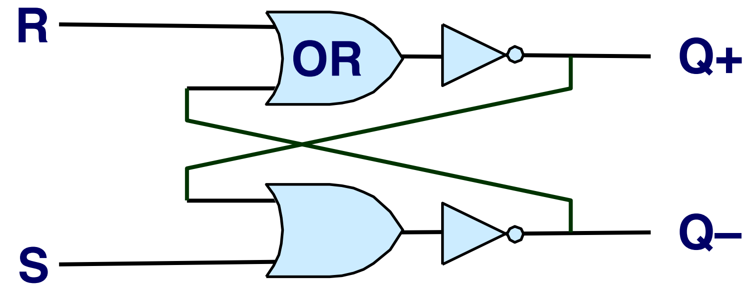
Building Block: RS Latch



Building Block: RS Latch

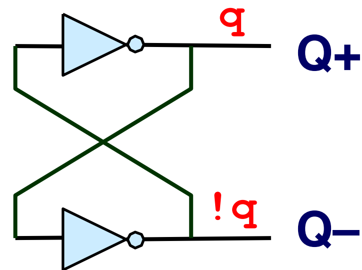
Bistable Element



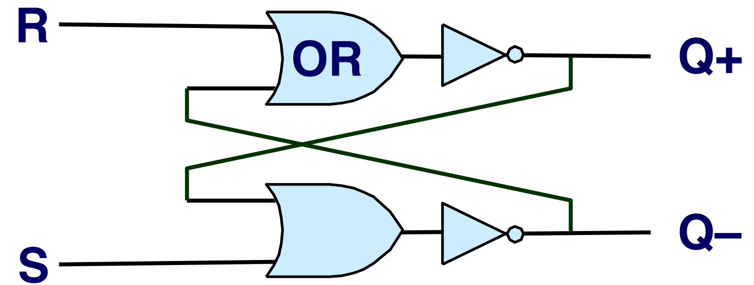


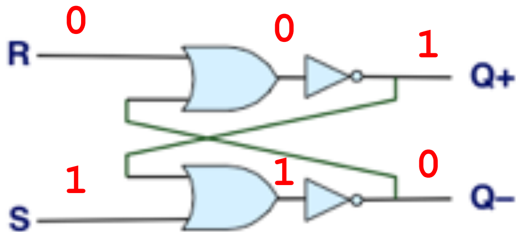
Building Block: RS Latch

Bistable Element



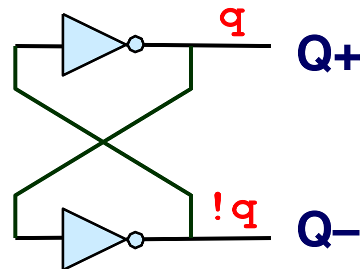
q = 0 or 1

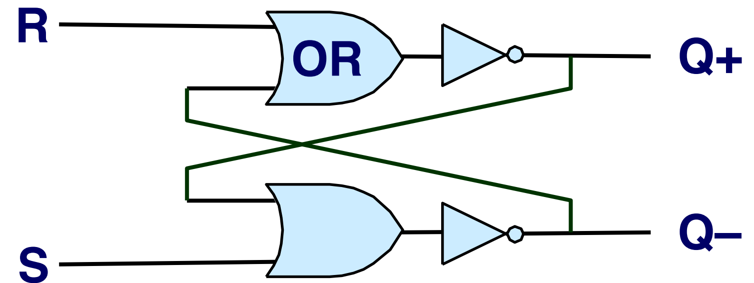


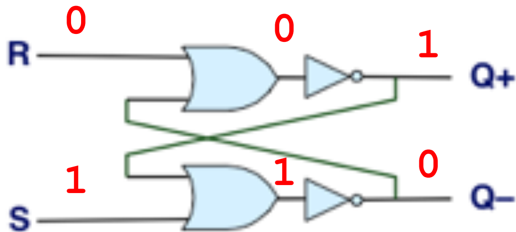


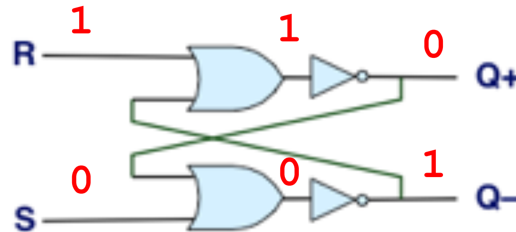
Building Block: RS Latch

Bistable Element



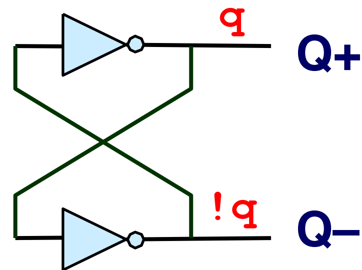




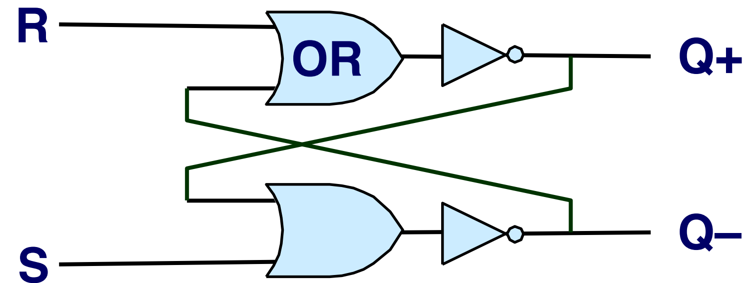


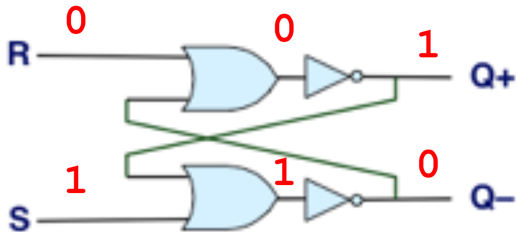
Building Block: RS Latch

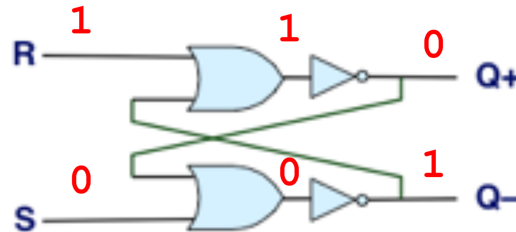
Bistable Element

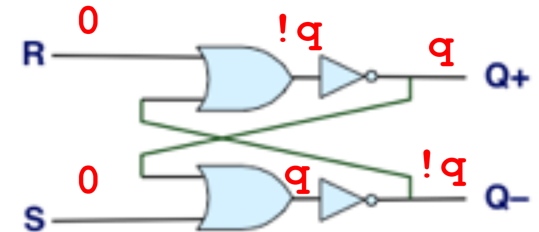


q = 0 or 1



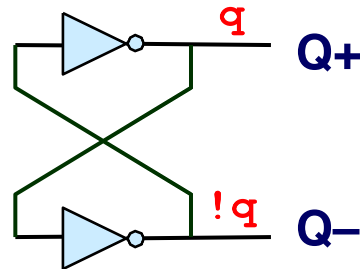






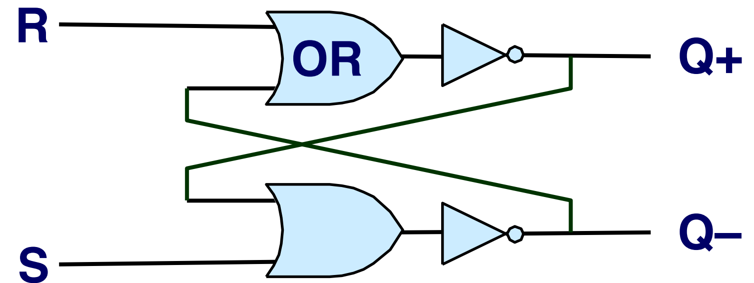
Building Block: RS Latch

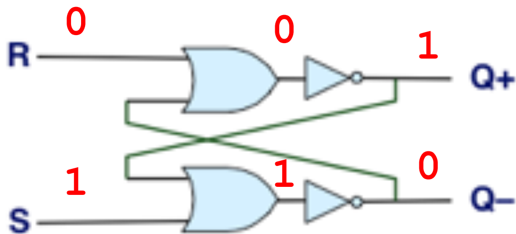
Bistable Element

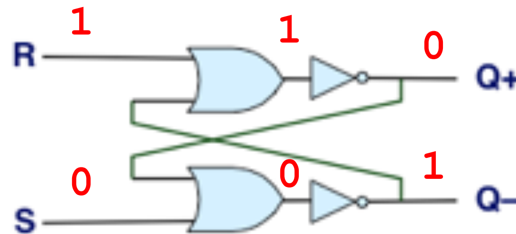


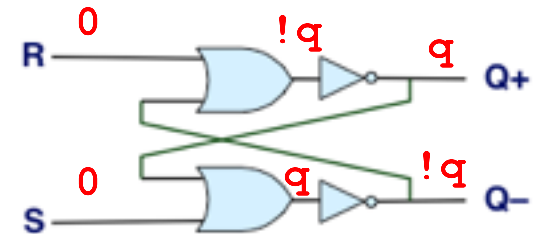
q = 0 or 1

R-S Latch

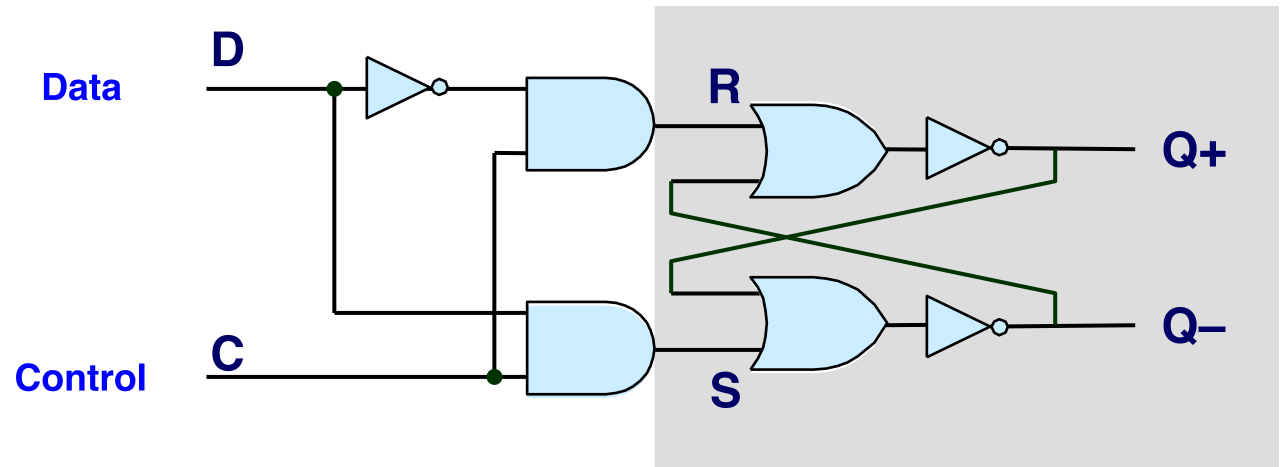




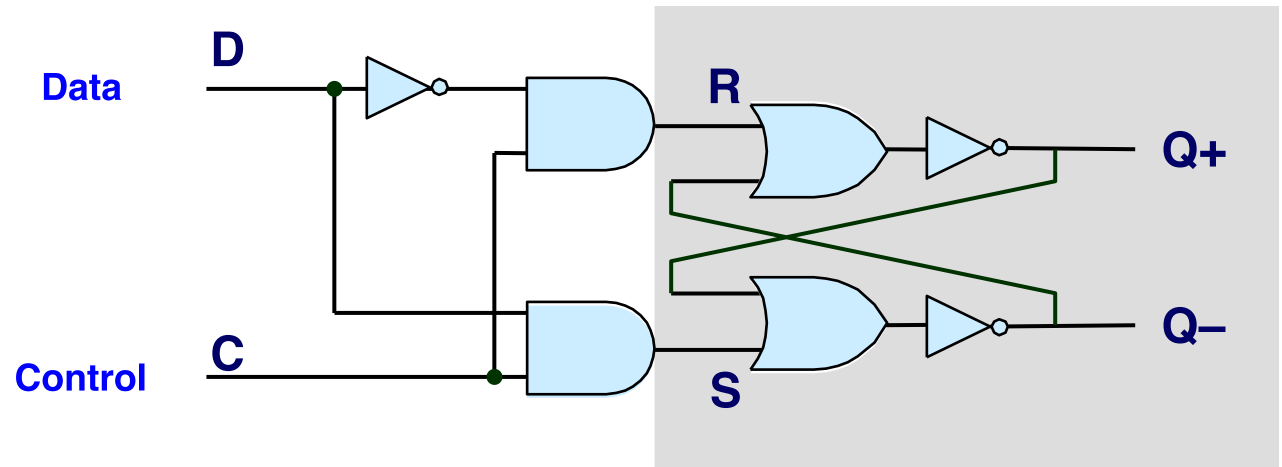




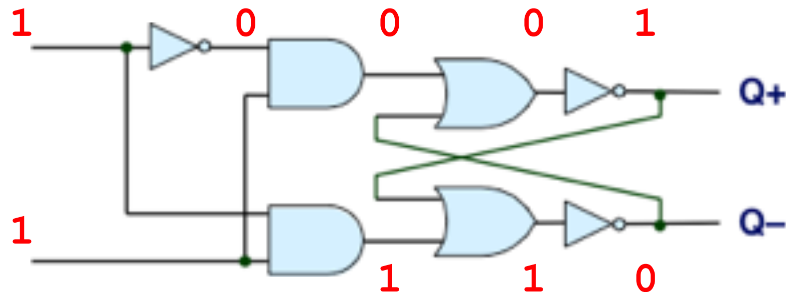
A Simple Way of Storing/Accessing 1 Bit



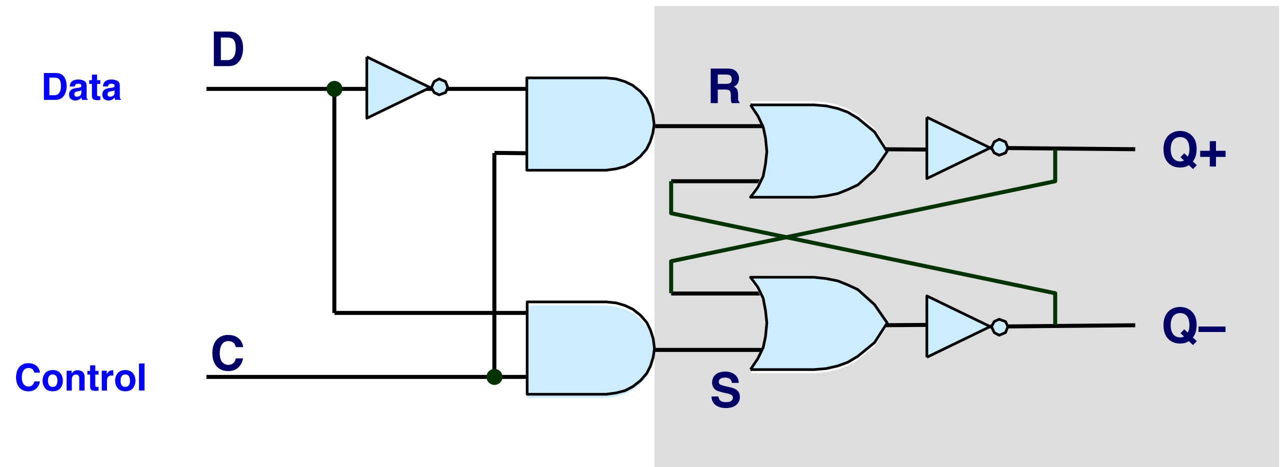
A Simple Way of Storing/Accessing 1 Bit

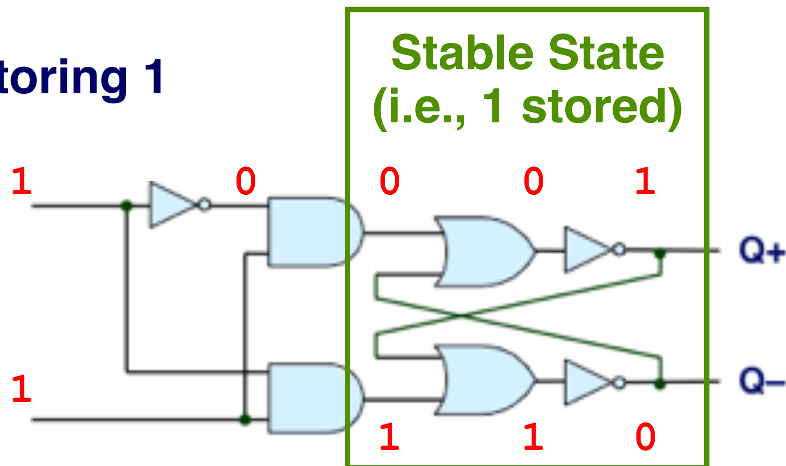


Storing 1

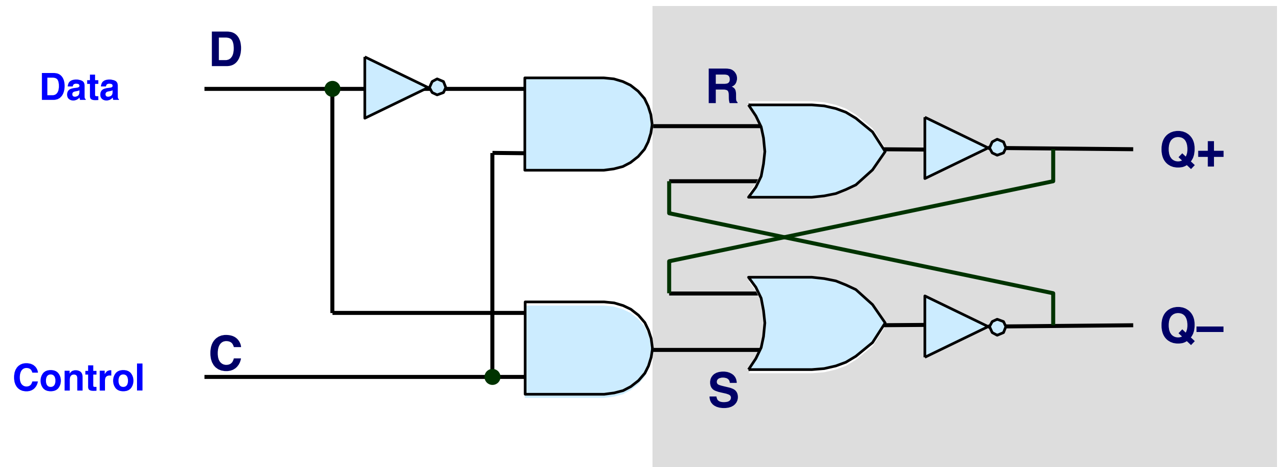


A Simple Way of Storing/Accessing 1 Bit

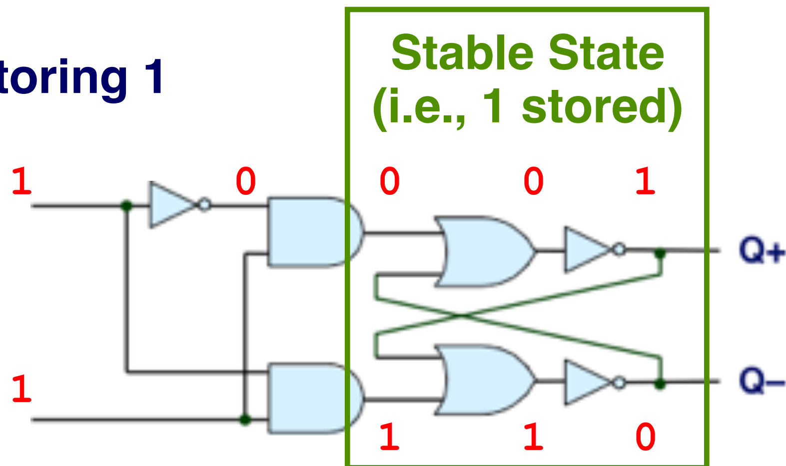




A Simple Way of Storing/Accessing 1 Bit

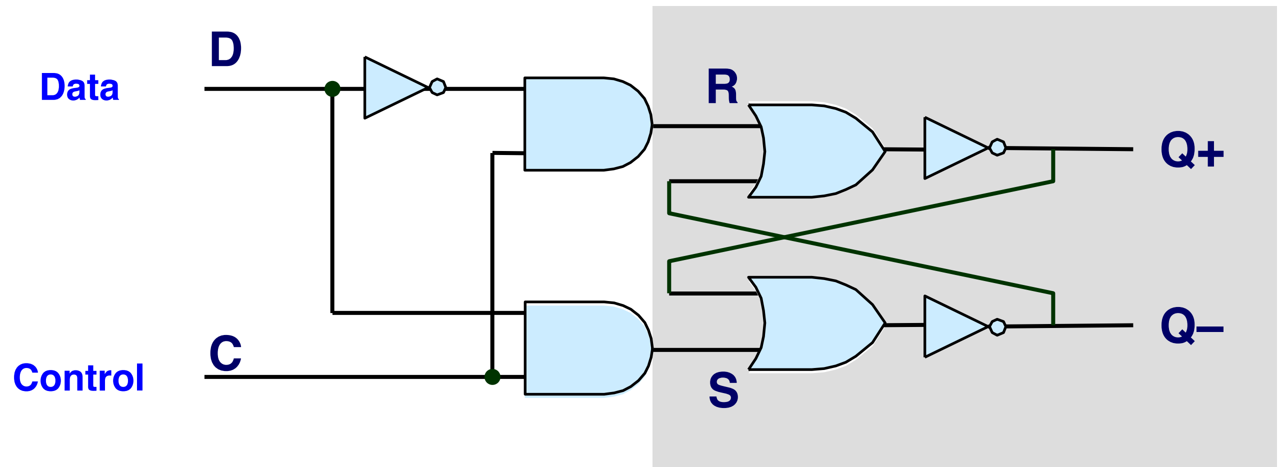


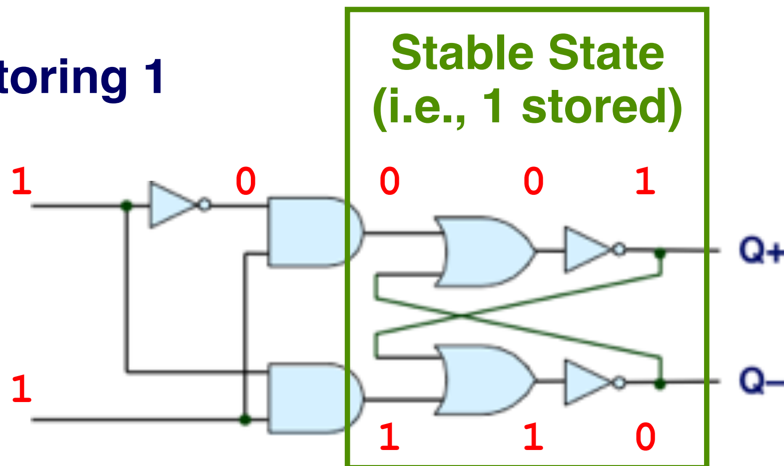
Storing 1



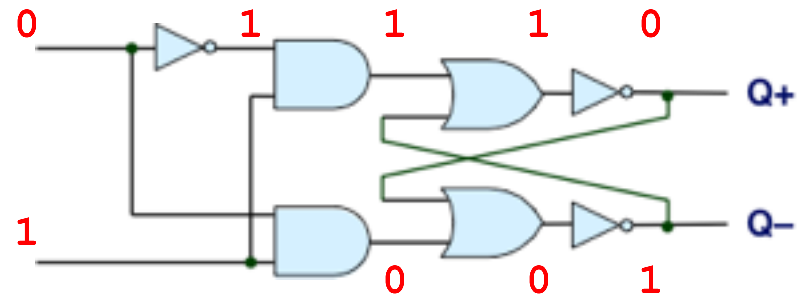
When C is 1, D is 1, Q+ will eventually become 1.

A Simple Way of Storing/Accessing 1 Bit

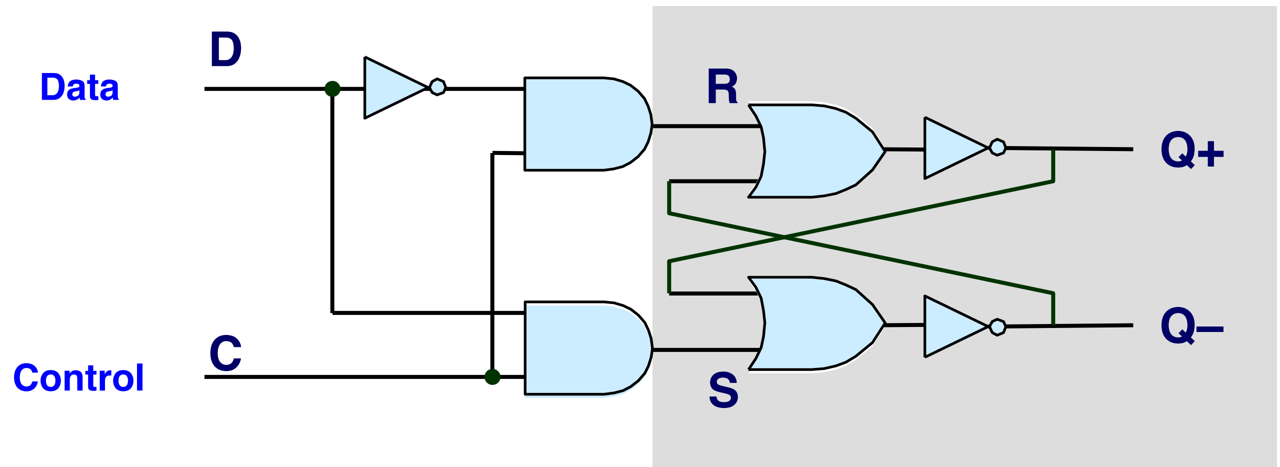


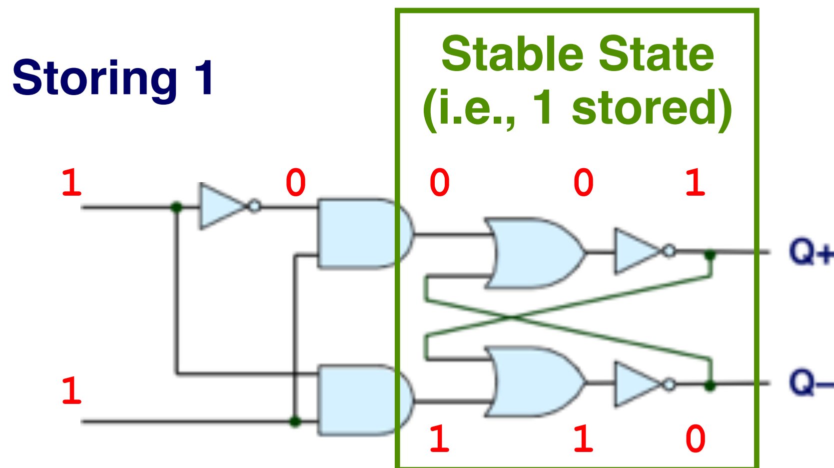


When C is 1, D is 1, Q+ will eventually become 1.

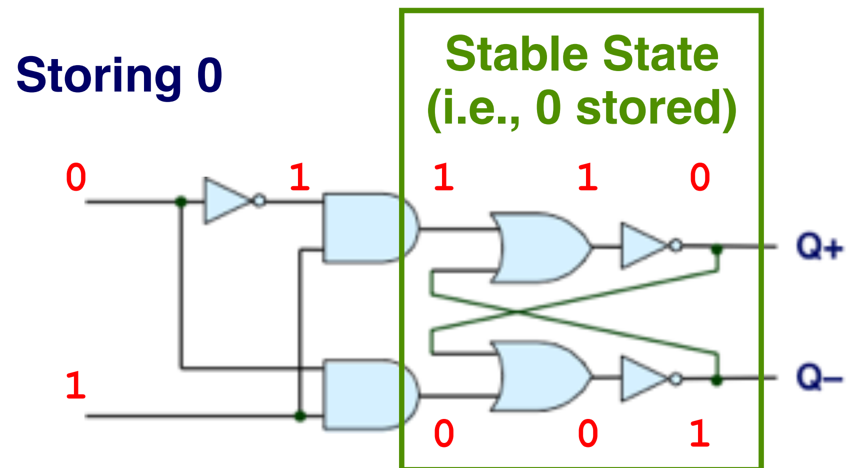


A Simple Way of Storing/Accessing 1 Bit

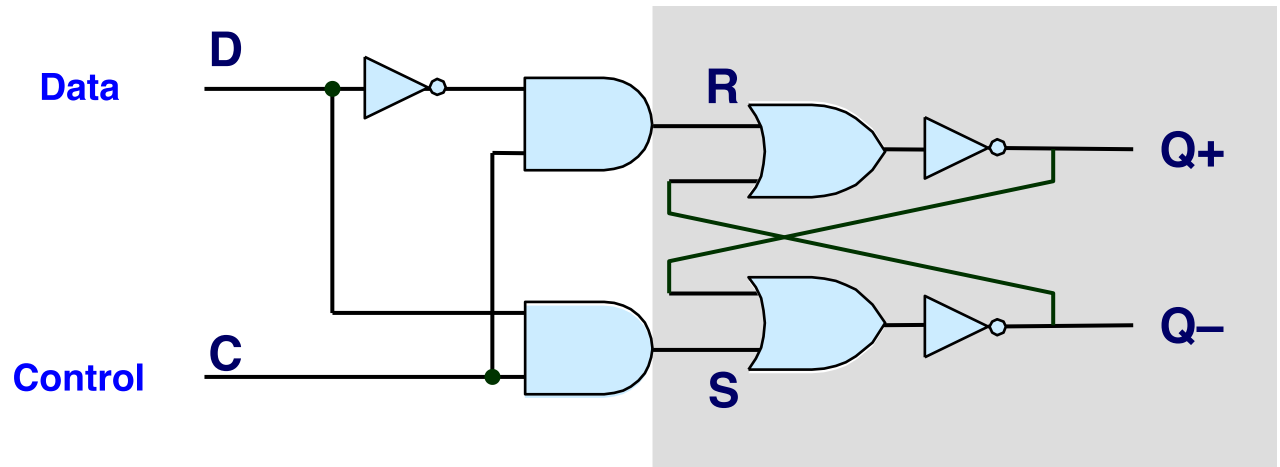


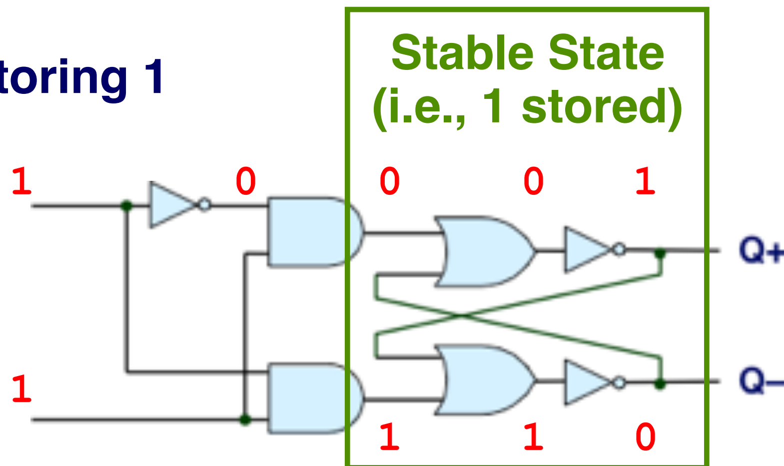


When C is 1, D is 1, Q+
will eventually become 1.

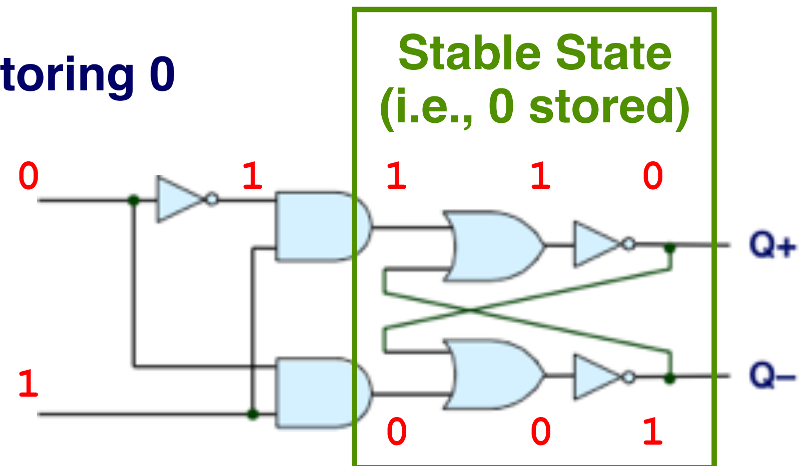


A Simple Way of Storing/Accessing 1 Bit





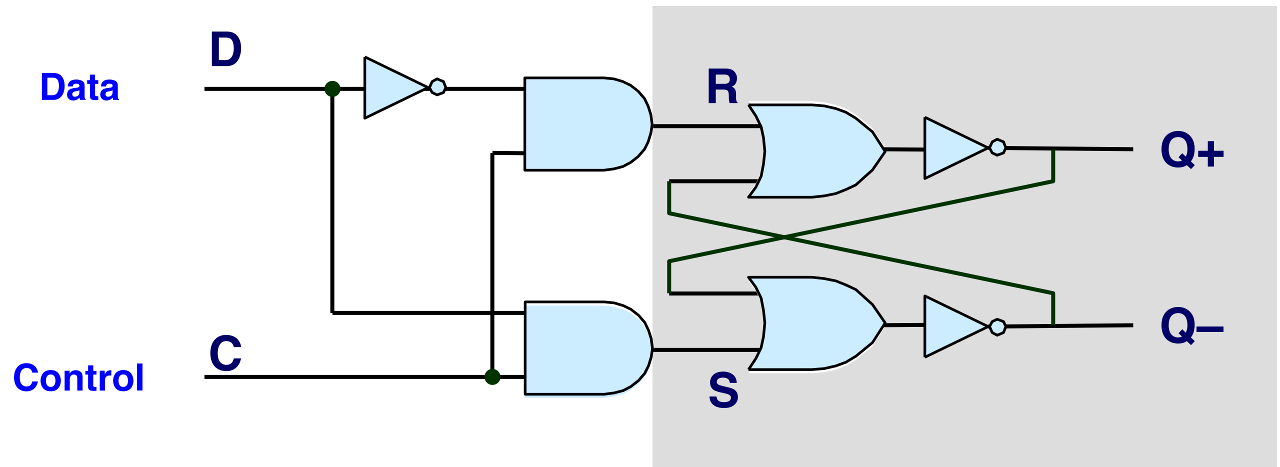
When C is 1, D is 1, Q+ will eventually become 1.



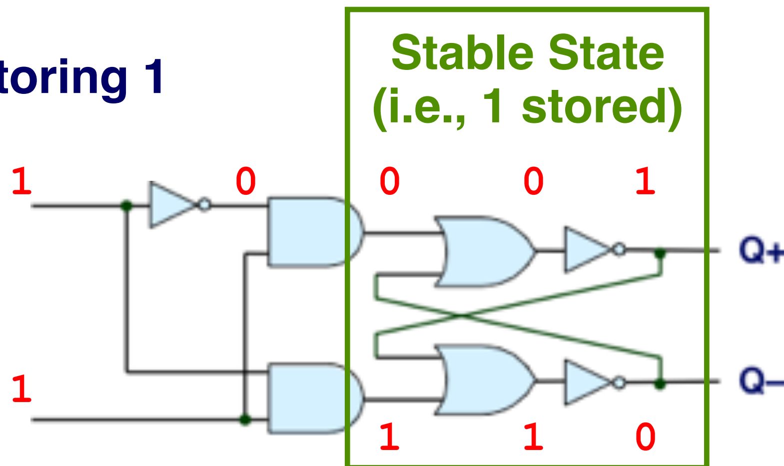
When C is 1, D is 0, Q+ will eventually become 0.

A Simple Way of Storing/Accessing 1 Bit

D Latch

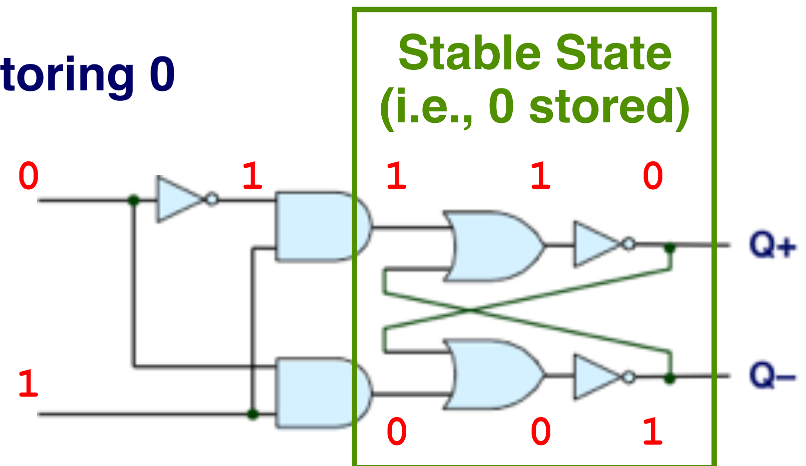


Storing 1



When C is 1, D is 1, Q+ will eventually become 1.

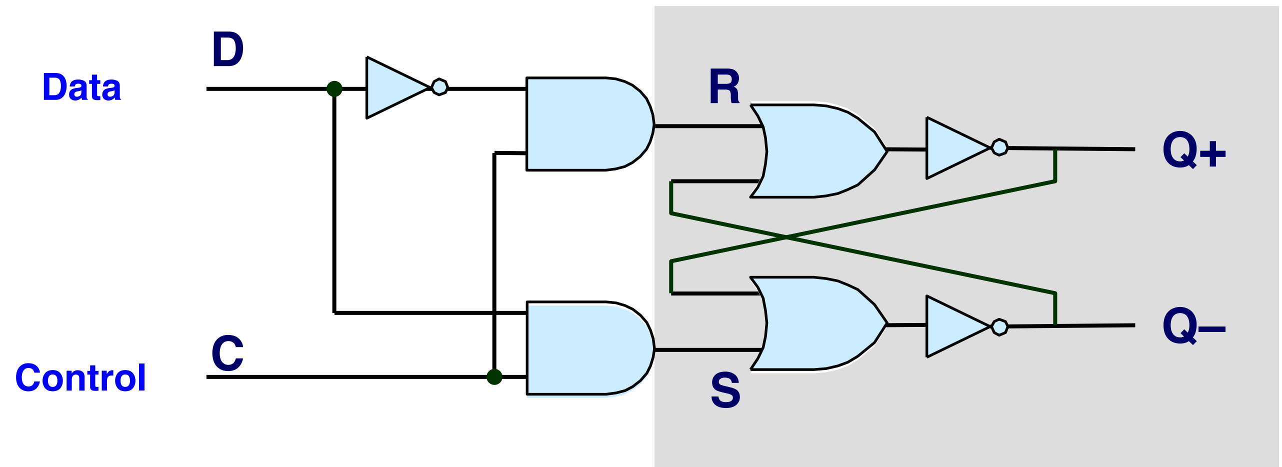
Storing 0



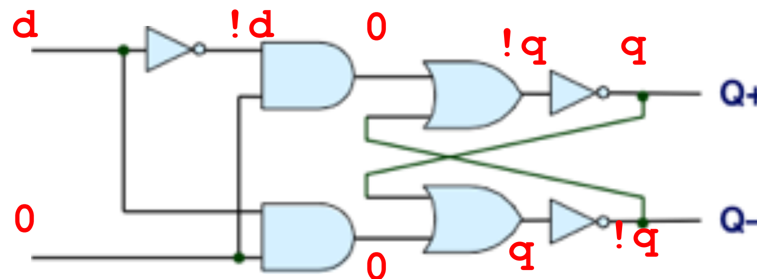
When C is 1, D is 0, Q+ will eventually become 0.

A Simple Way of Storing/Accessing 1 Bit

D Latch

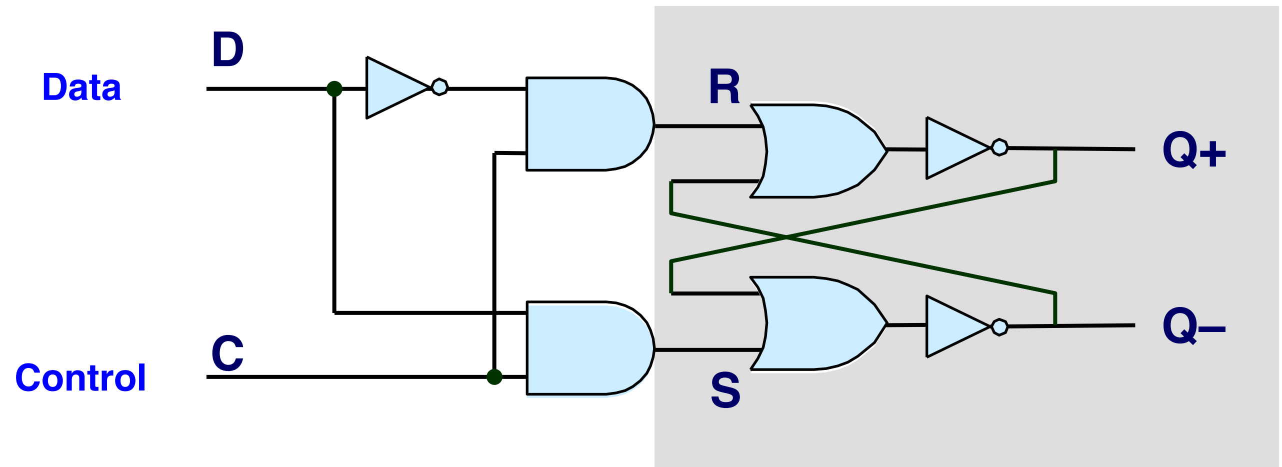


Holding Data

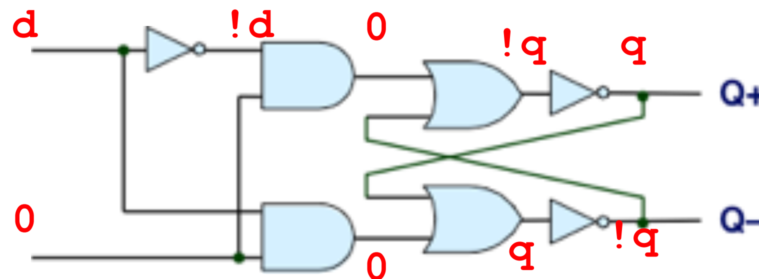


A Simple Way of Storing/Accessing 1 Bit

D Latch



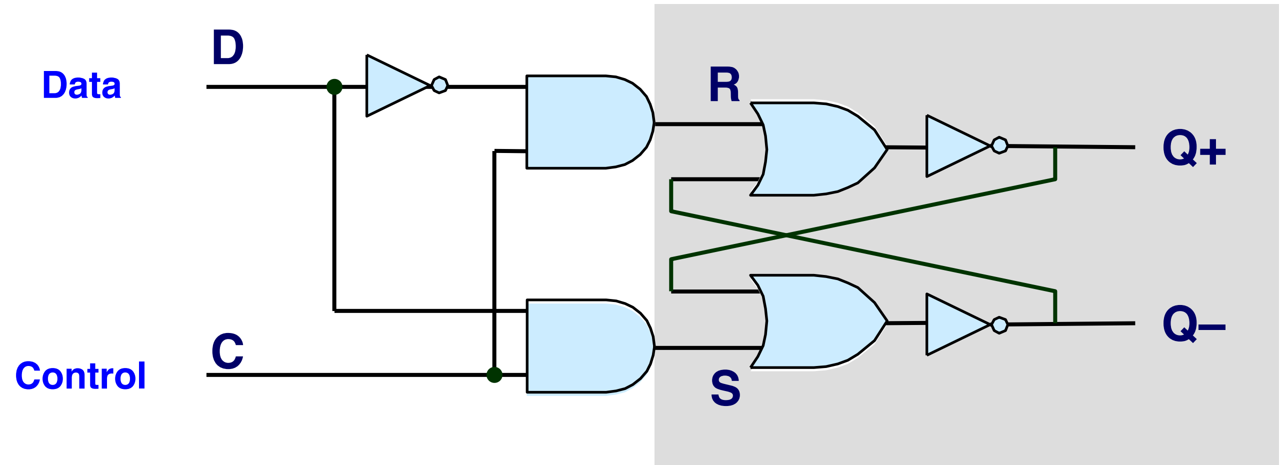
Holding Data



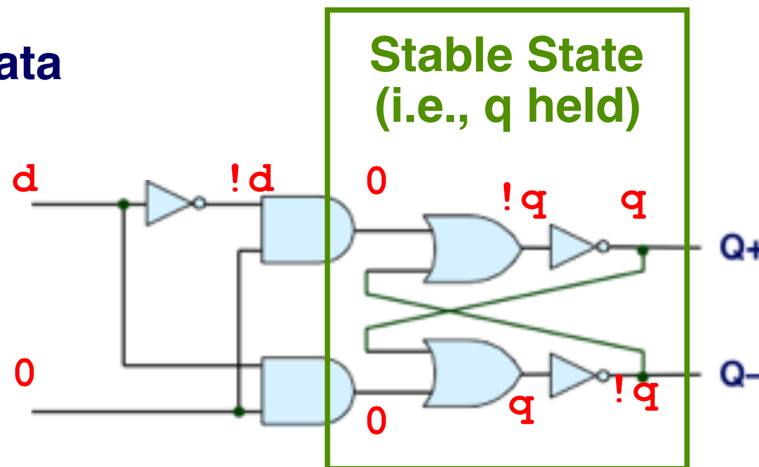
If C == 0, Q+ doesn't change with d

A Simple Way of Storing/Accessing 1 Bit

D Latch



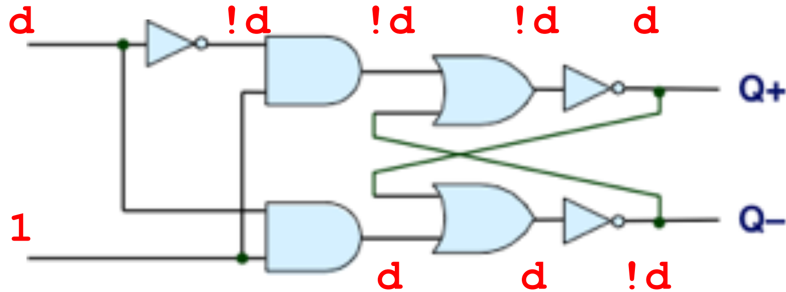
Holding Data



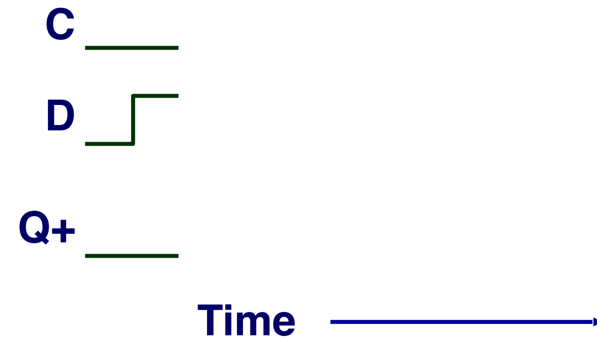
If C == 0, Q+ doesn't change with d

D-Latch is "Transparent"

Latching

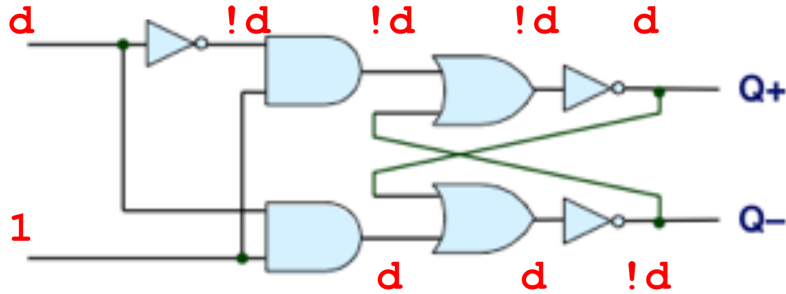


Changing D

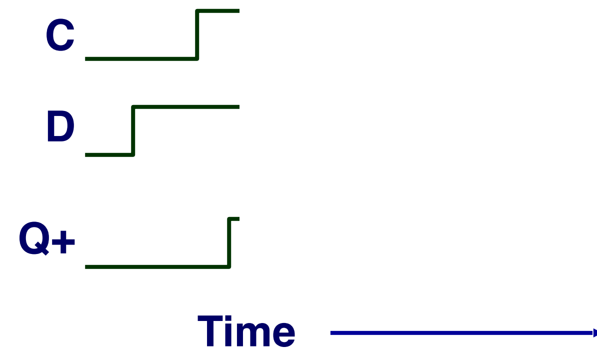


D-Latch is "Transparent"

Latching

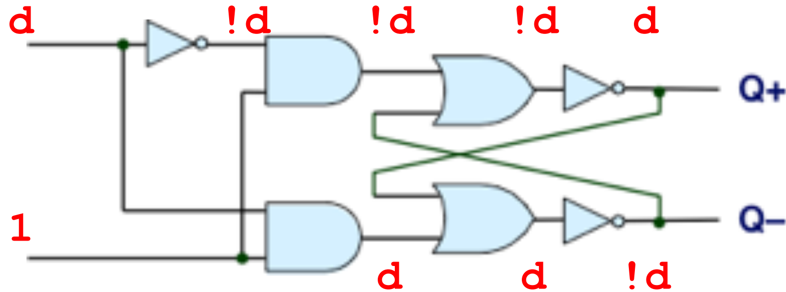


Changing D

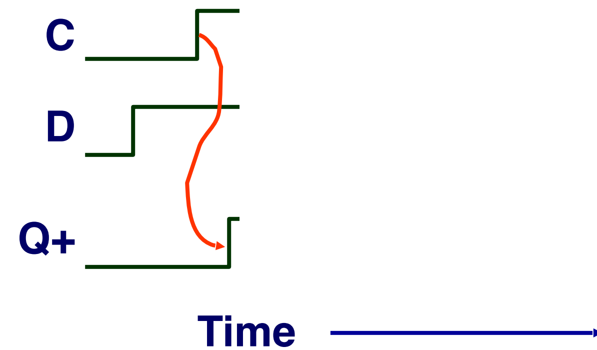


D-Latch is “Transparent”

Latching

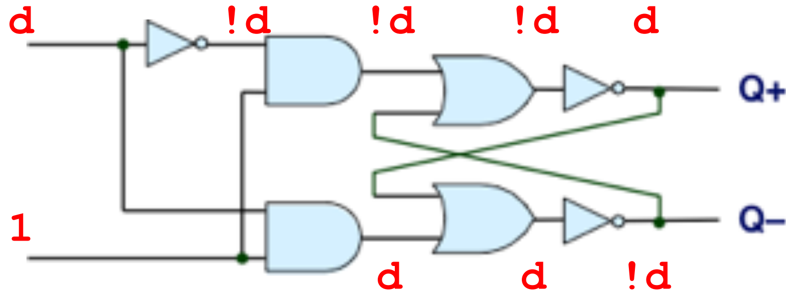


Changing D

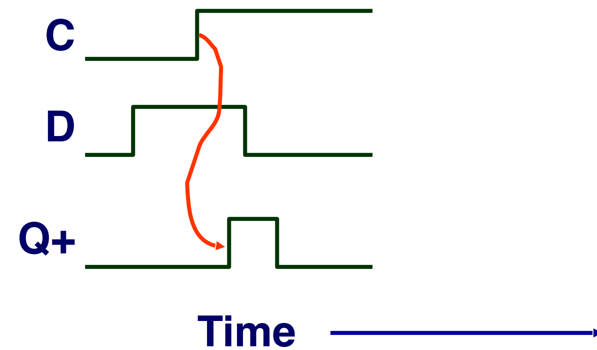


D-Latch is “Transparent”

Latching

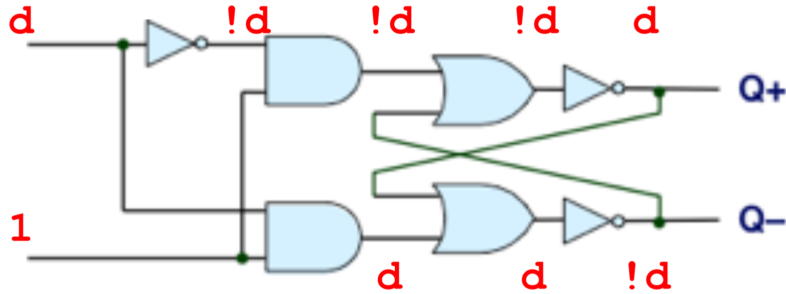


Changing D

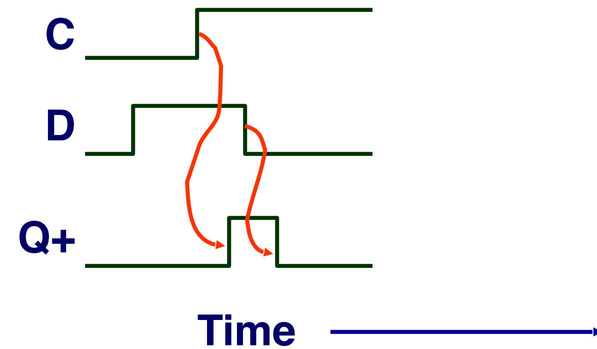


D-Latch is "Transparent"

Latching

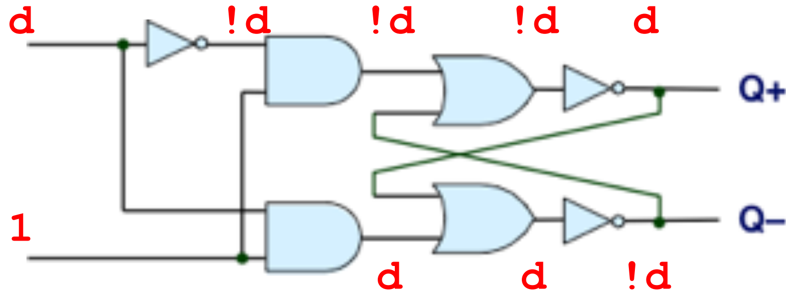


Changing D

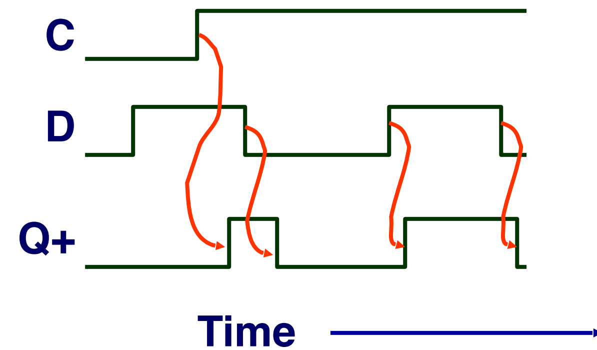


D-Latch is “Transparent”

Latching

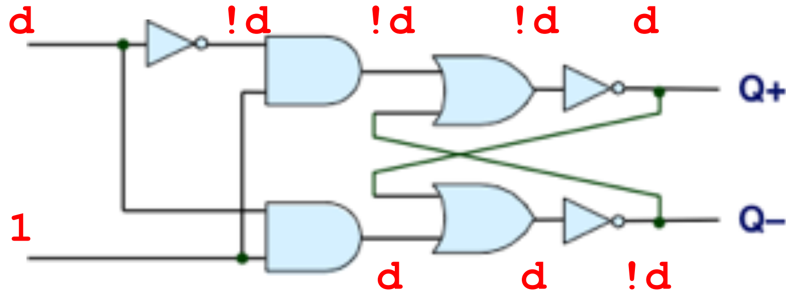


Changing D

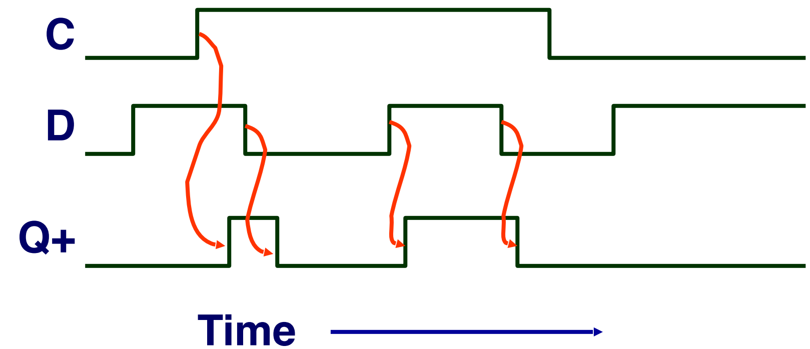


D-Latch is “Transparent”

Latching

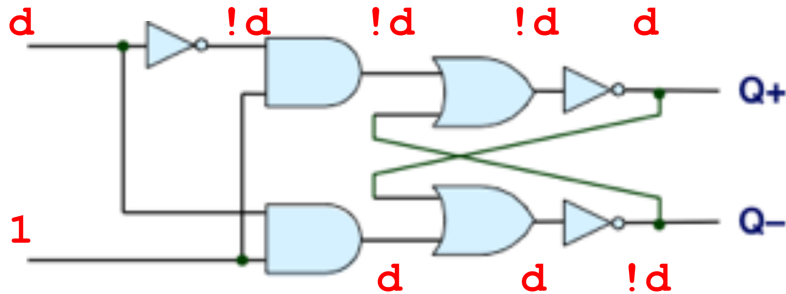


Changing D

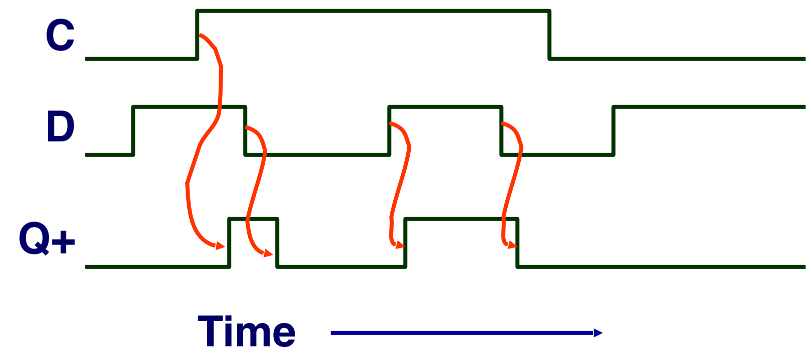


D-Latch is “Transparent”

Latching



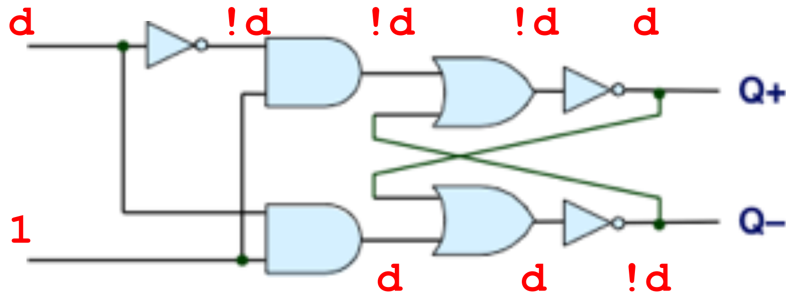
Changing D



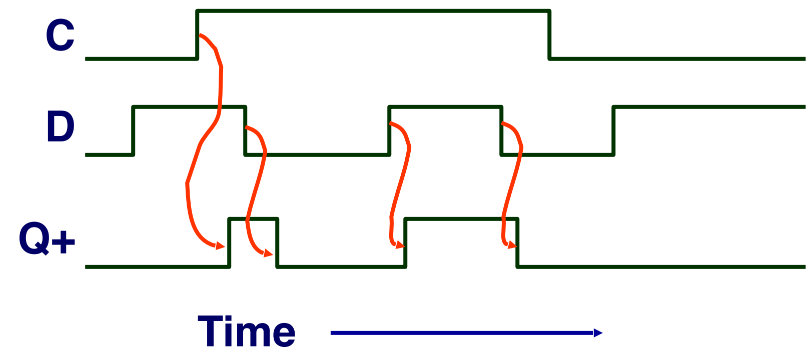
- When you want to store **d**, you have to first set **C** to 1, and then set **d**

D-Latch is “Transparent”

Latching



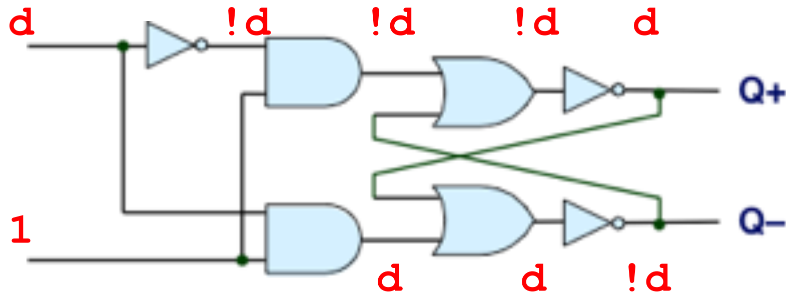
Changing D



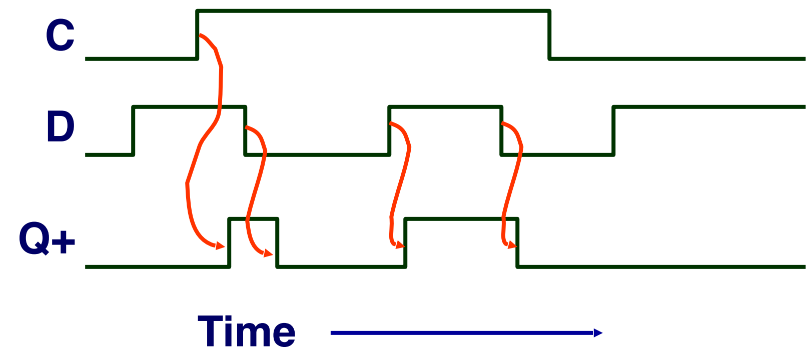
- When you want to store **d**, you have to first set **C** to 1, and then set **d**
- There is a propagation delay of the combinational circuit from **D** to **Q+** and **Q–**. So hold C for a while until the signal is fully propagated

D-Latch is “Transparent”

Latching



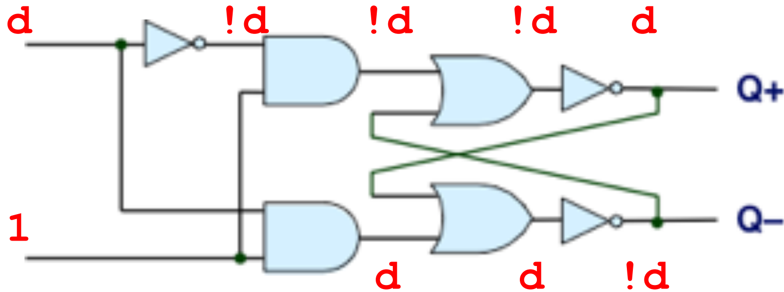
Changing D



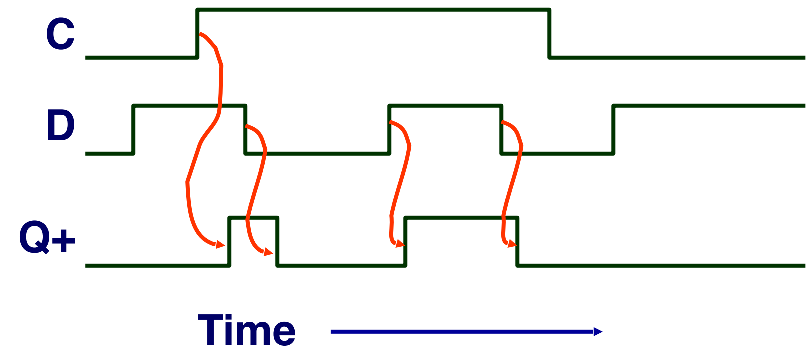
- When you want to store **d**, you have to first set **C** to 1, and then set **d**
- There is a propagation delay of the combinational circuit from **D** to **Q+** and **Q–**. So hold C for a while until the signal is fully propagated
- Then set **C** to 0. Value latched depends on value of **D** as **C** goes to 0

D-Latch is "Transparent"

Latching



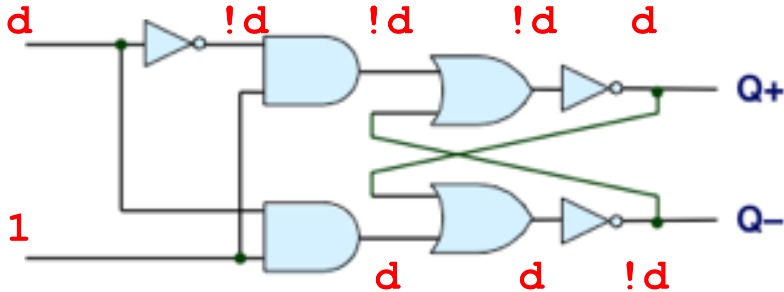
Changing D



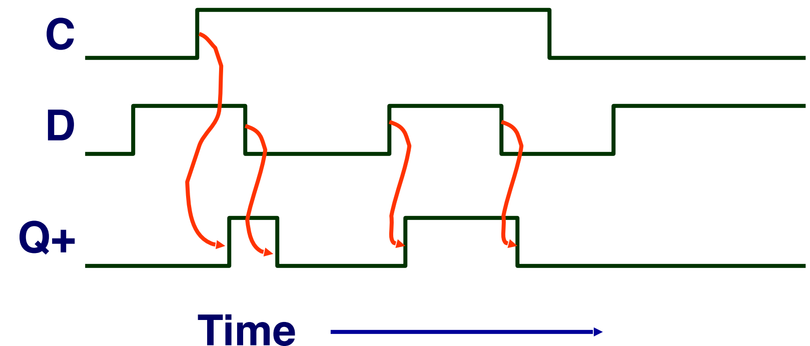
- When you want to store **d**, you have to first set **C** to 1, and then set **d**
- There is a propagation delay of the combinational circuit from **D** to **Q+** and **Q–**. So hold C for a while until the signal is fully propagated
- Then set **C** to 0. Value latched depends on value of **D** as **C** goes to 0
- D-latch is *transparent* when **C** is 1

D-Latch is “Transparent”

Latching

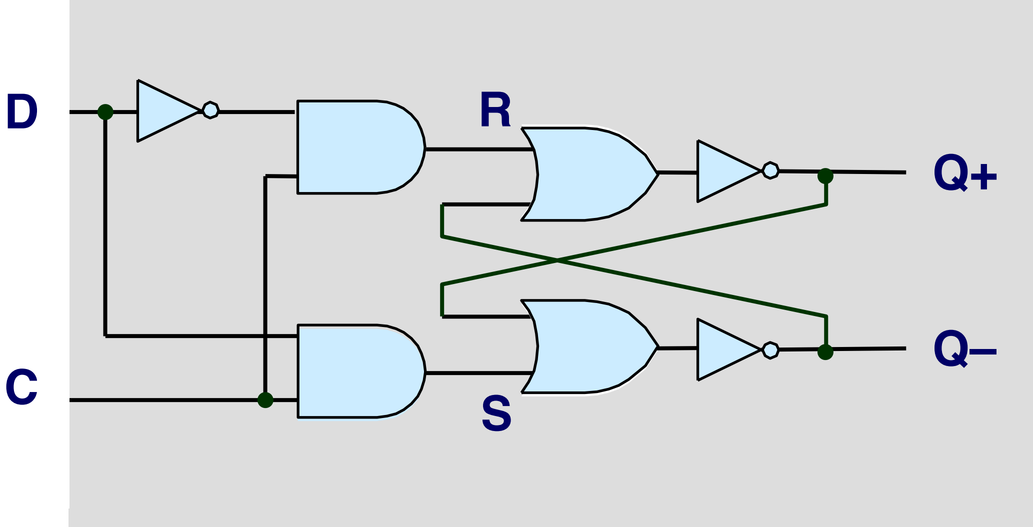


Changing D

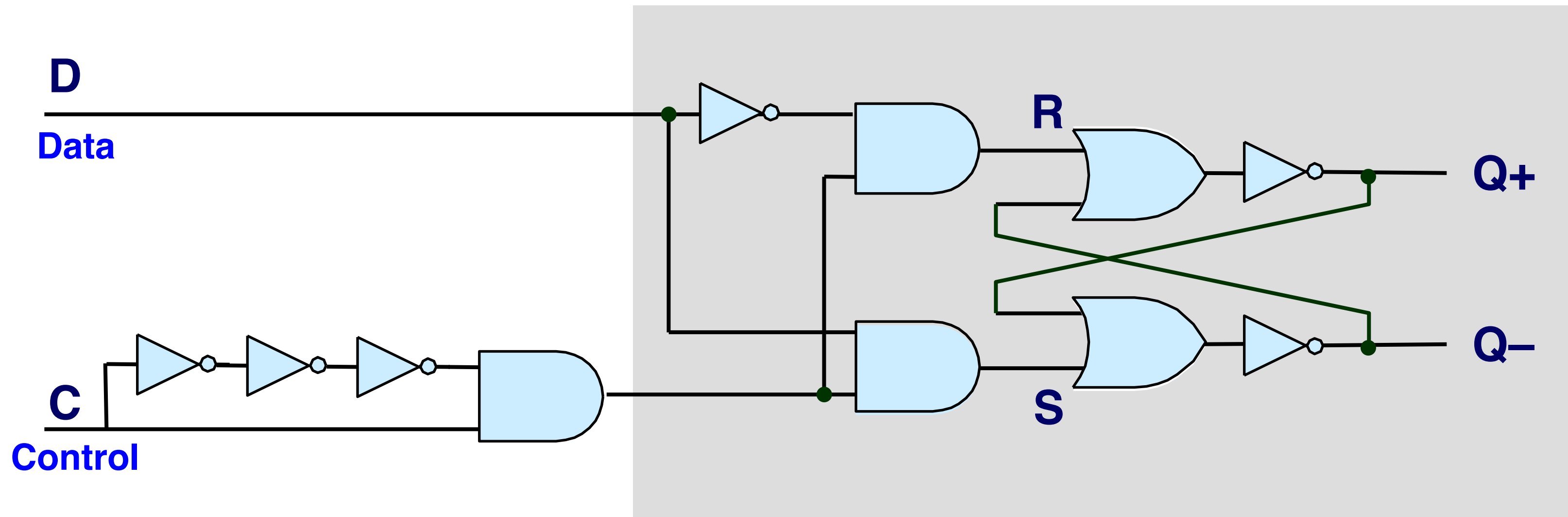


- When you want to store **d**, you have to first set **C** to 1, and then set **d**
- There is a propagation delay of the combinational circuit from **D** to **Q+** and **Q–**. So hold C for a while until the signal is fully propagated
- Then set **C** to 0. Value latched depends on value of **D** as **C** goes to 0
- D-latch is *transparent* when **C** is 1
- D-latch is “*level-triggered*” b/c **Q** changes as the voltage level of **C** rises.

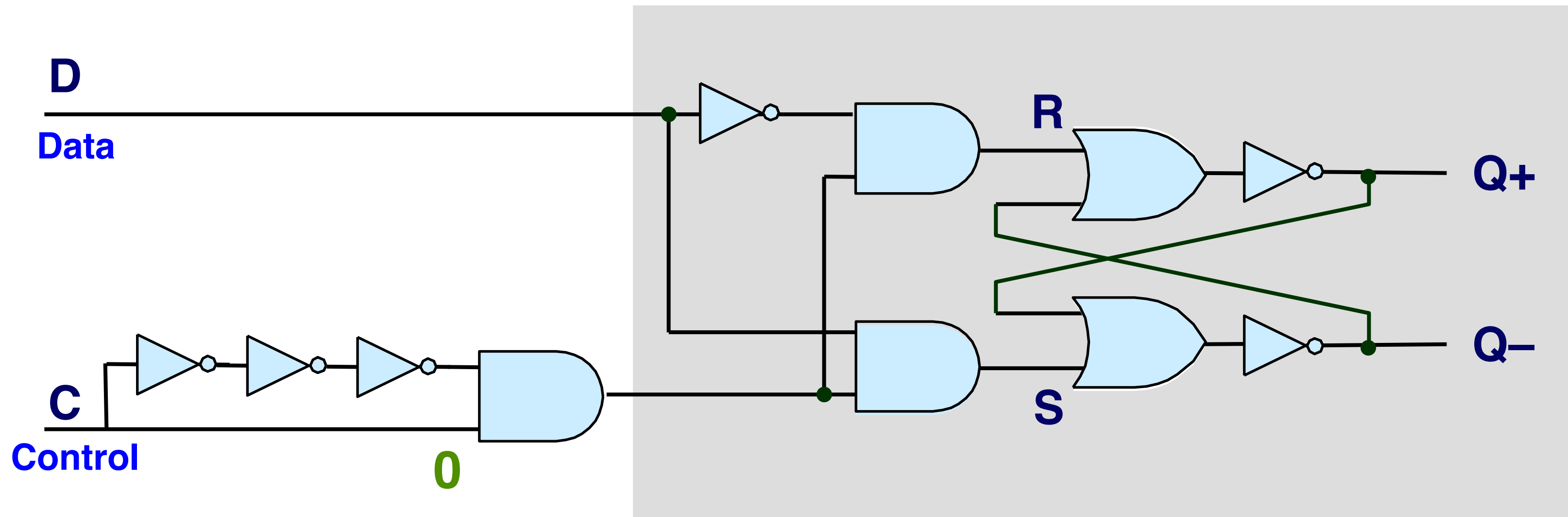
Edge-Triggered Latch (Flip-Flop)



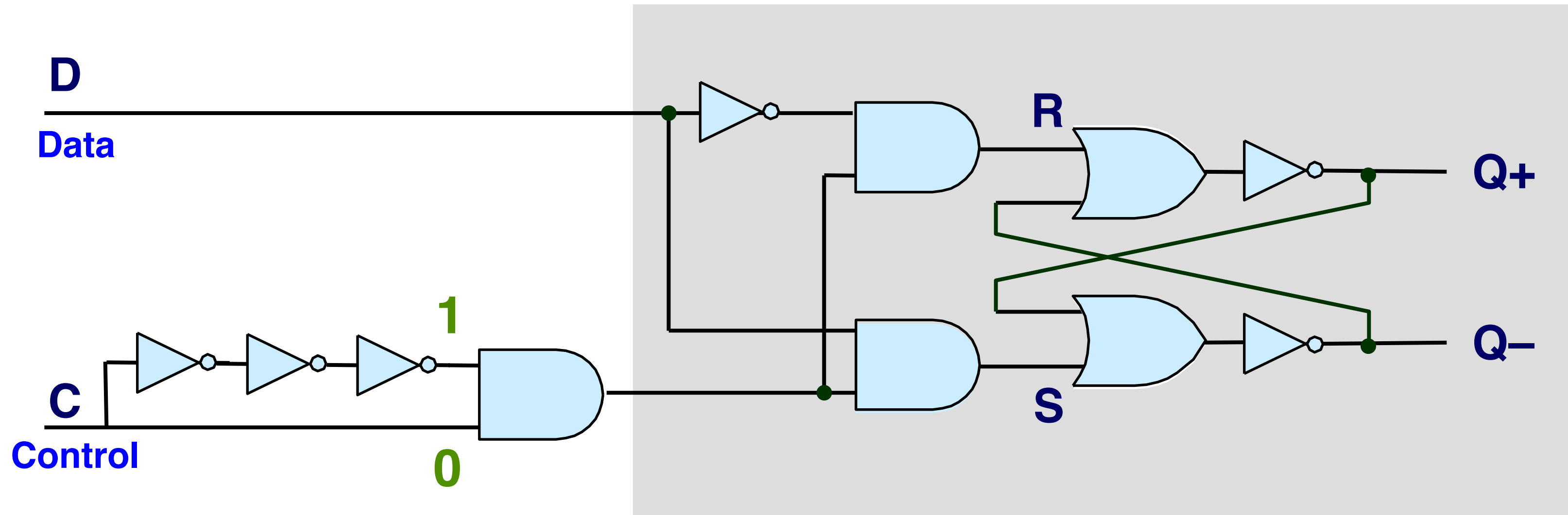
Edge-Triggered Latch (Flip-Flop)



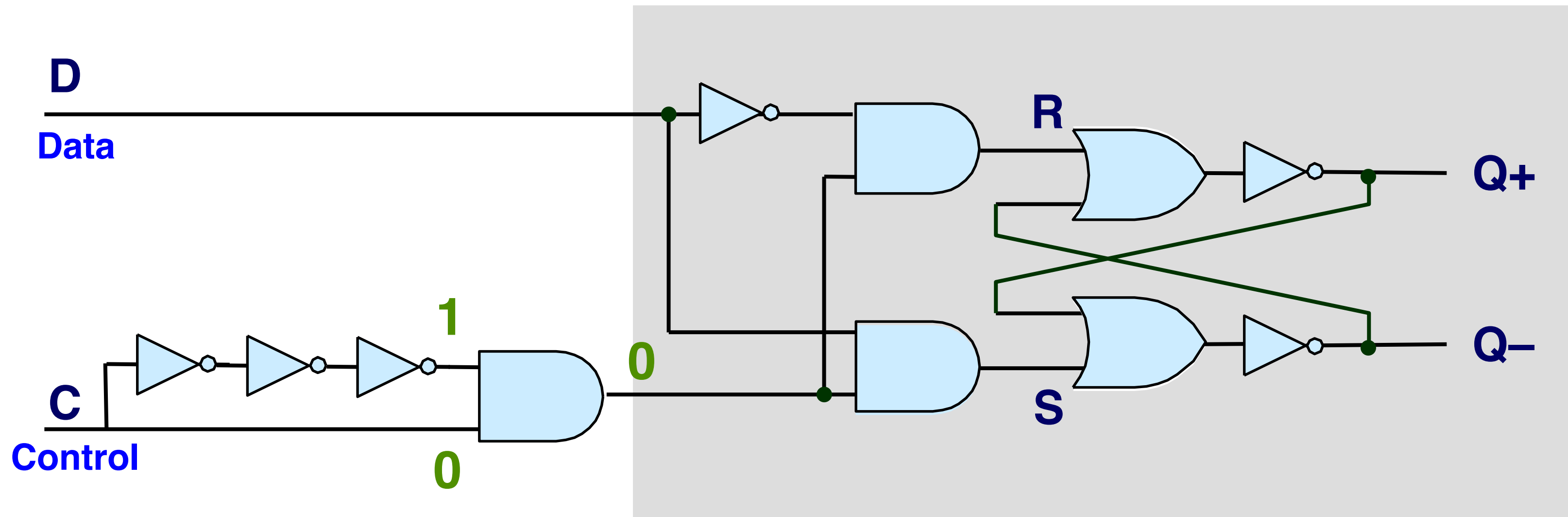
Edge-Triggered Latch (Flip-Flop)



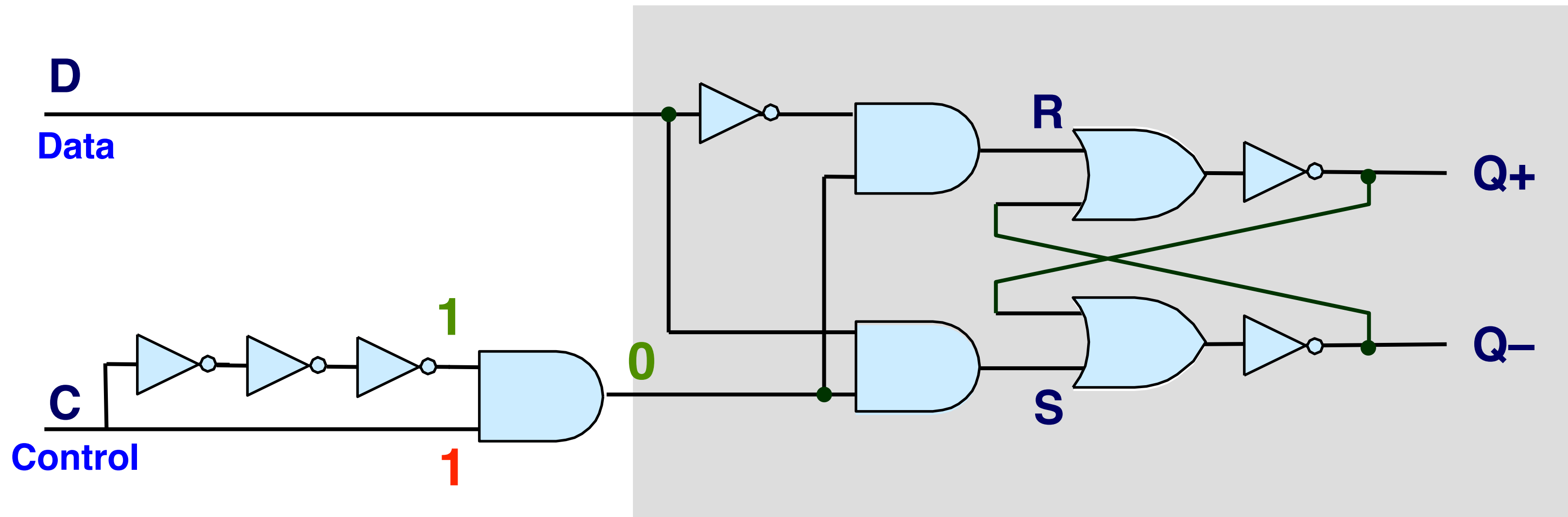
Edge-Triggered Latch (Flip-Flop)



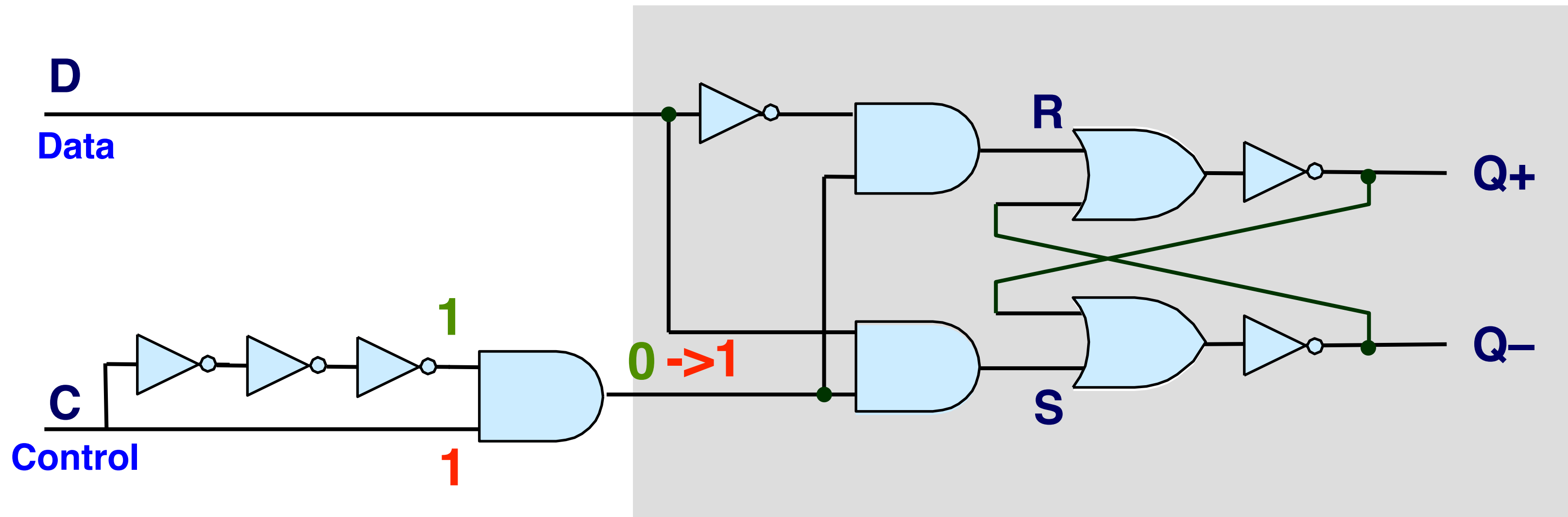
Edge-Triggered Latch (Flip-Flop)



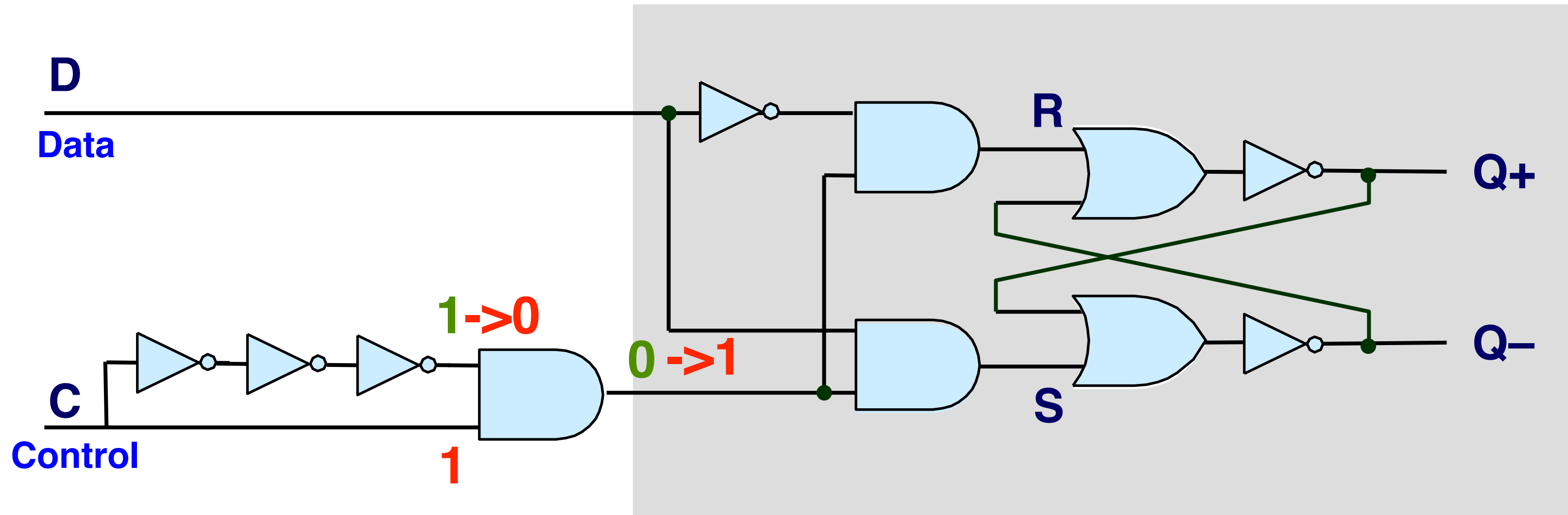
Edge-Triggered Latch (Flip-Flop)



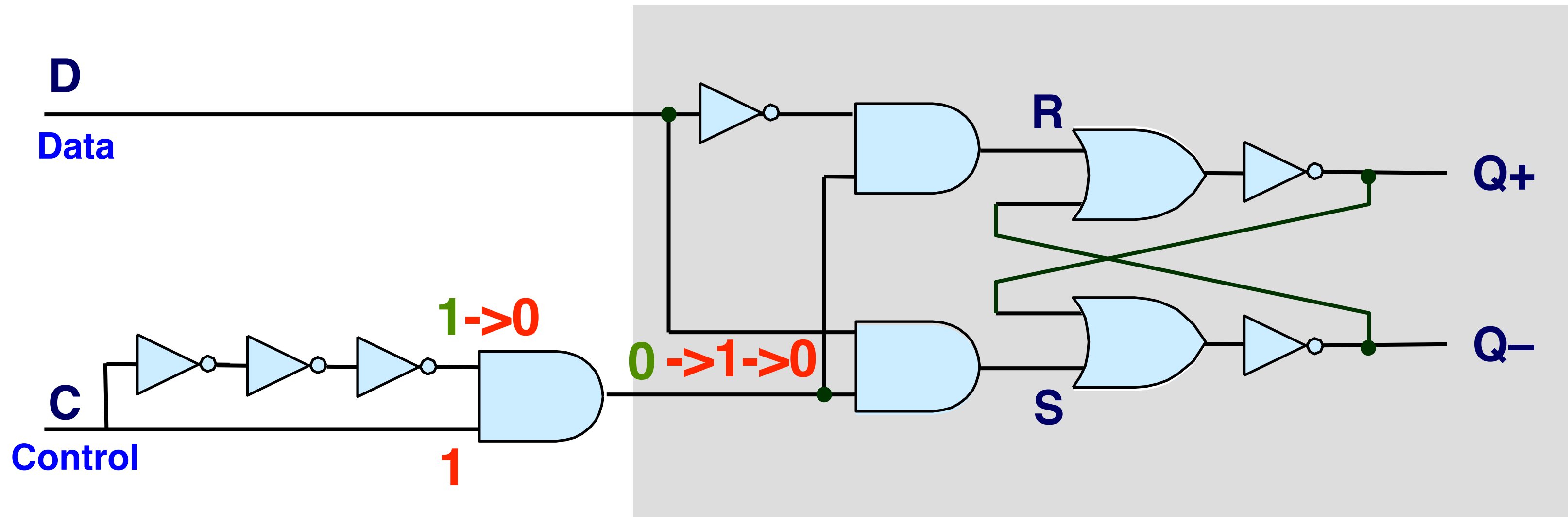
Edge-Triggered Latch (Flip-Flop)



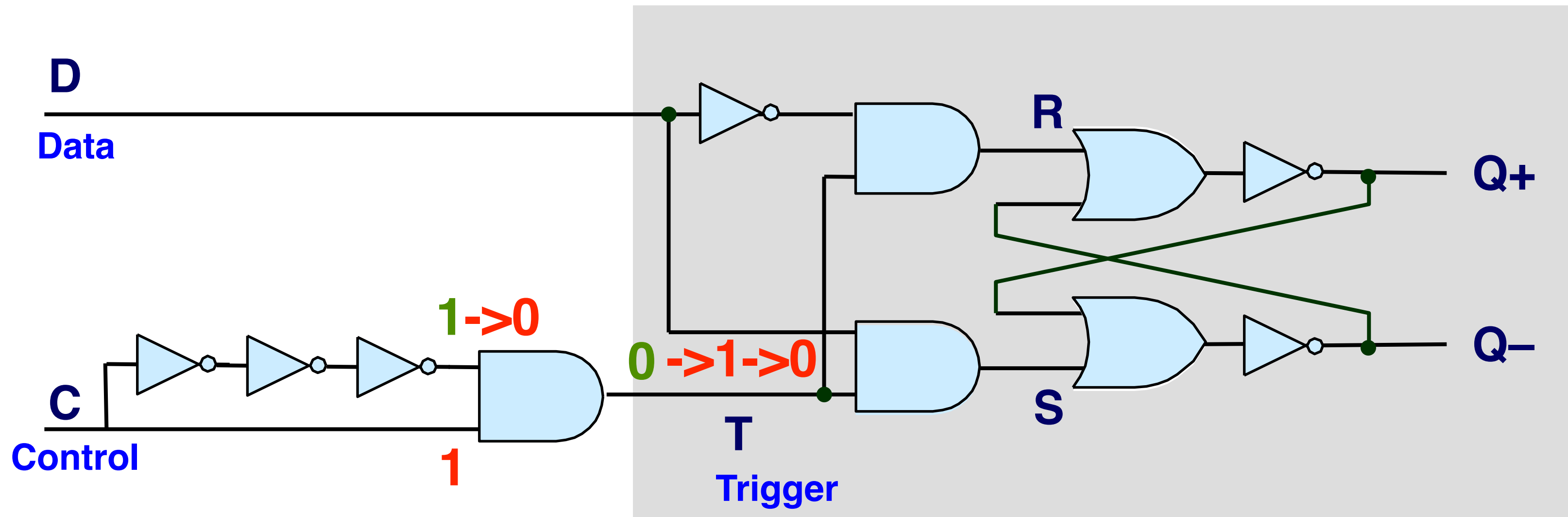
Edge-Triggered Latch (Flip-Flop)

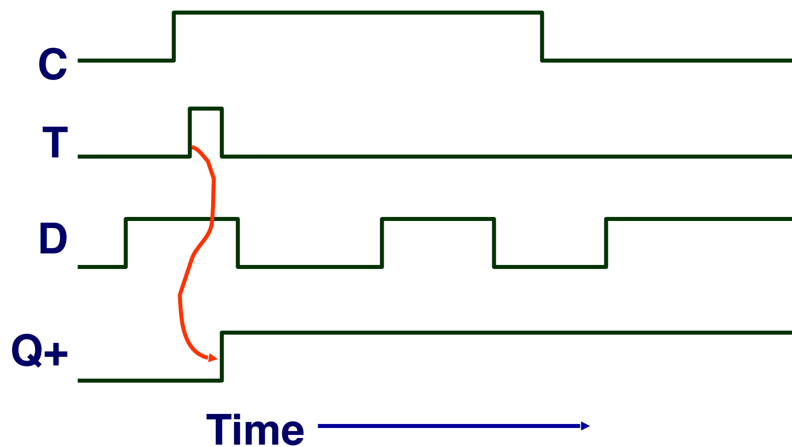


Edge-Triggered Latch (Flip-Flop)

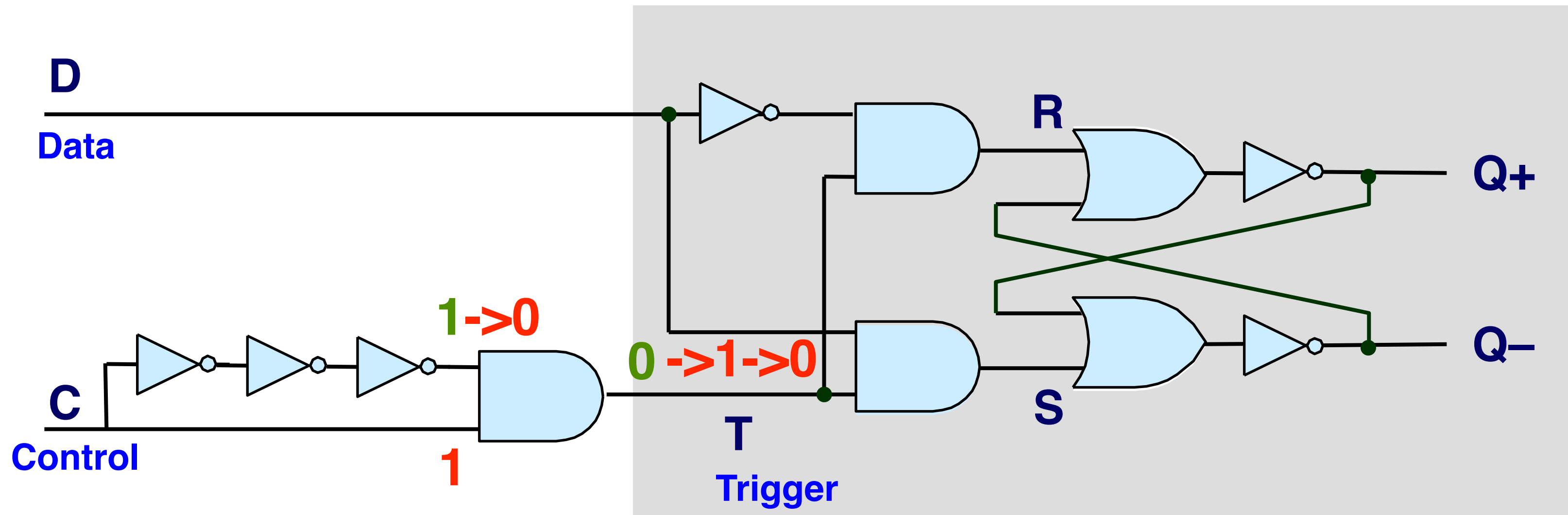


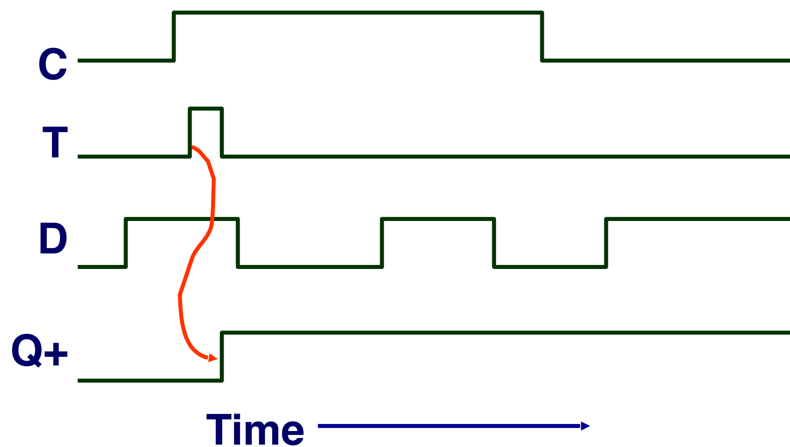
Edge-Triggered Latch (Flip-Flop)





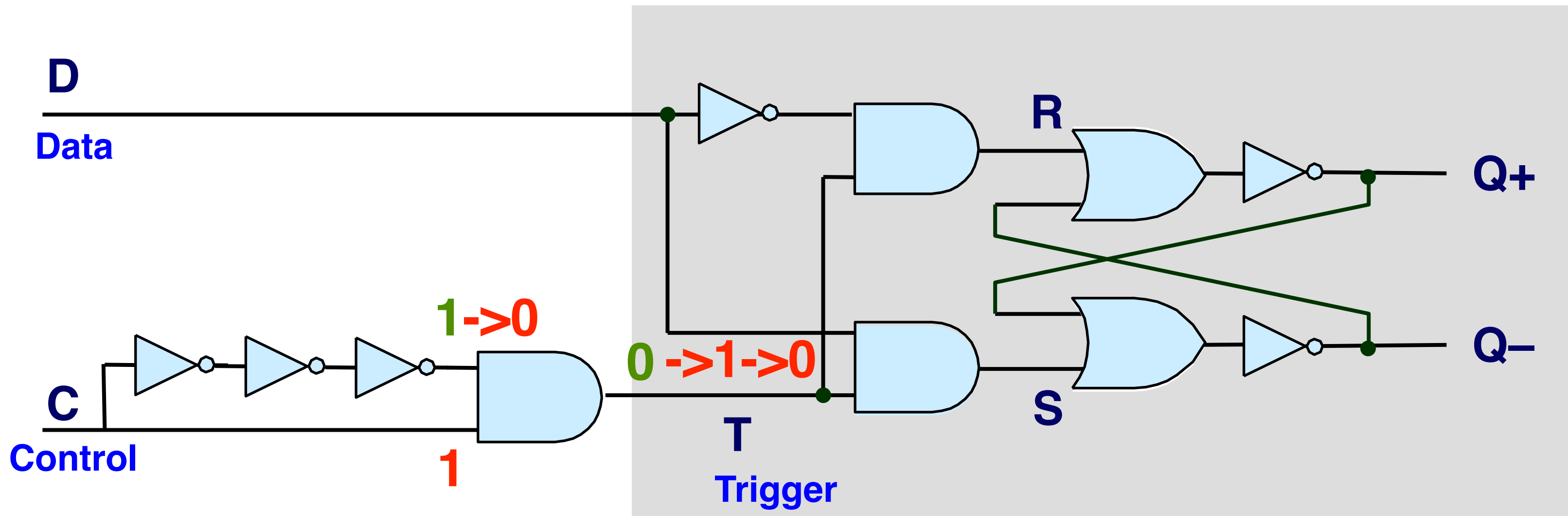
Edge-Triggered Latch (Flip-Flop)

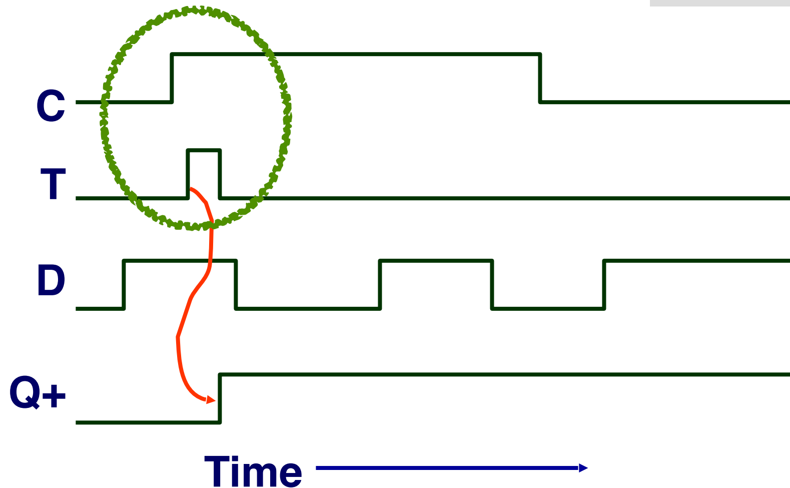




- Flip-flop: Only latches data for a brief period

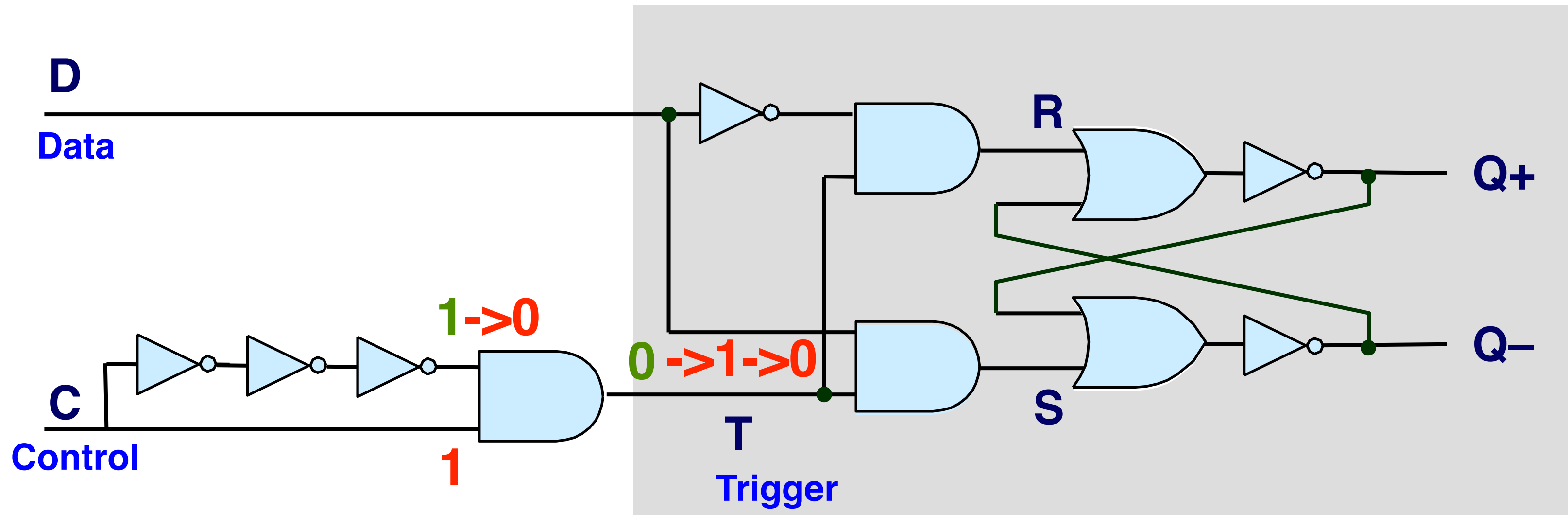
Edge-Triggered Latch (Flip-Flop)

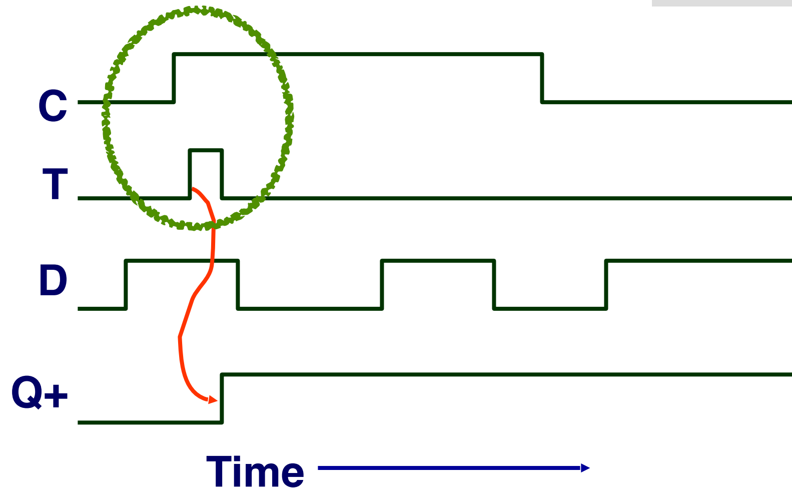




- Flip-flop: Only latches data for a brief period
- Value latched depends on data as C **rises** (i.e., 0–>1); usually called at the **rising edge** of C

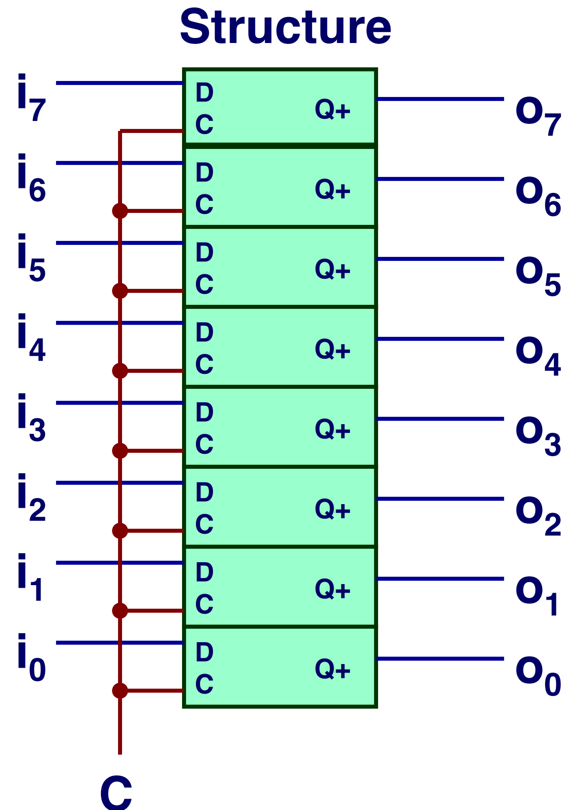
Edge-Triggered Latch (Flip-Flop)





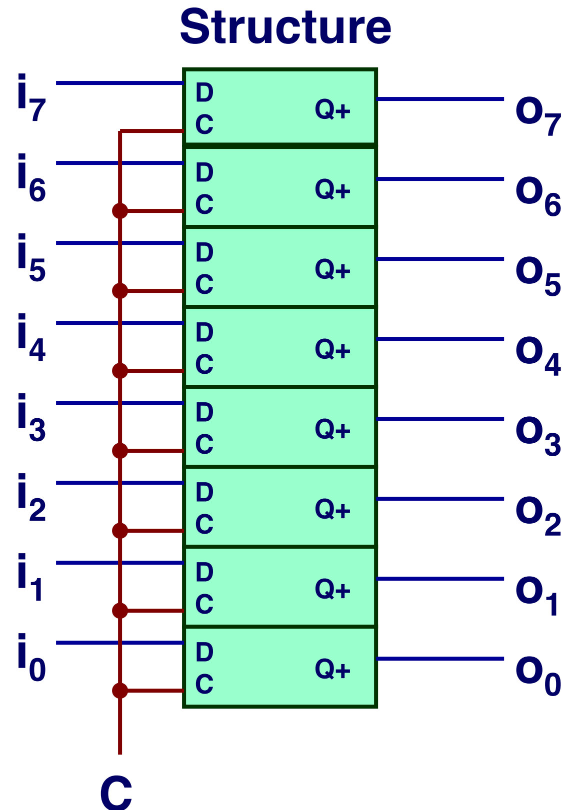
- Flip-flop: Only latches data for a brief period
- Value latched depends on data as C **rises** (i.e., 0–>1); usually called at the **rising edge** of C
- Output remains stable at all other times

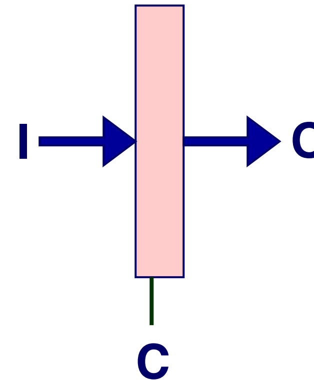
Registers



- Stores several bits of data
- Collection of edge-triggered latches (D Flip-flops)
- Loads input on rising edge of the C signal

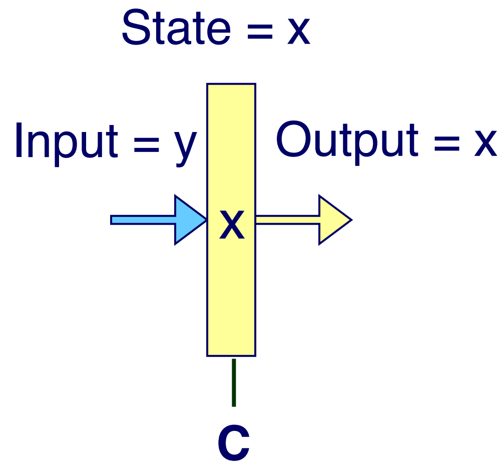
Registers



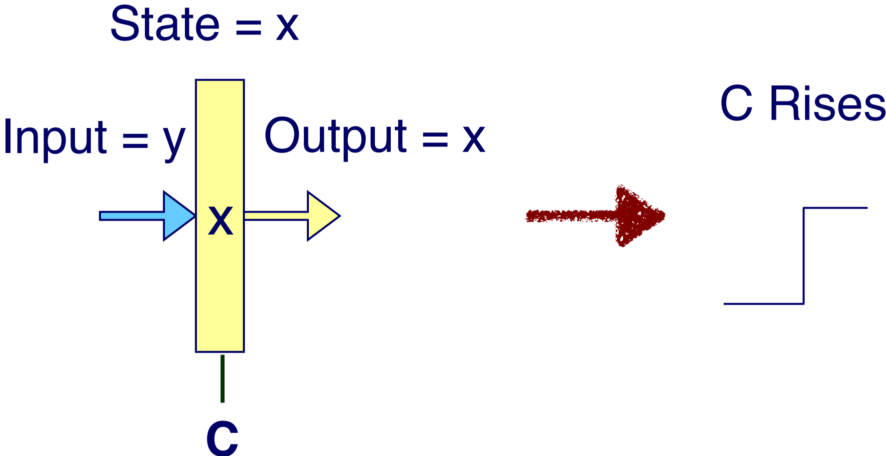


- Stores several bits of data
- Collection of edge-triggered latches (D Flip-flops)
- Loads input on rising edge of the C signal

Register Operation

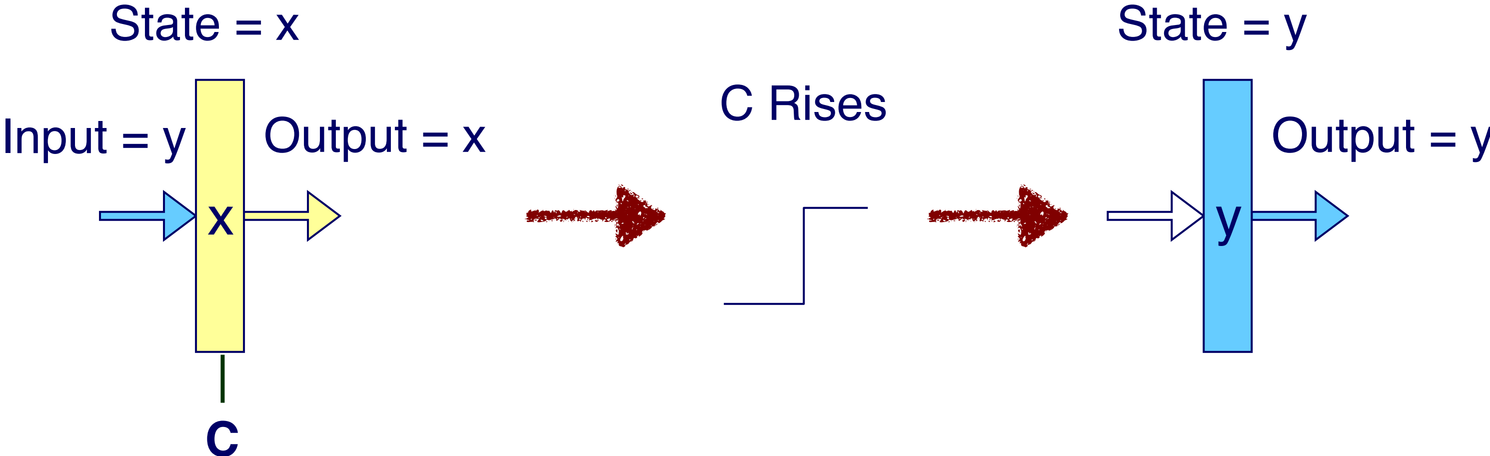


Register Operation

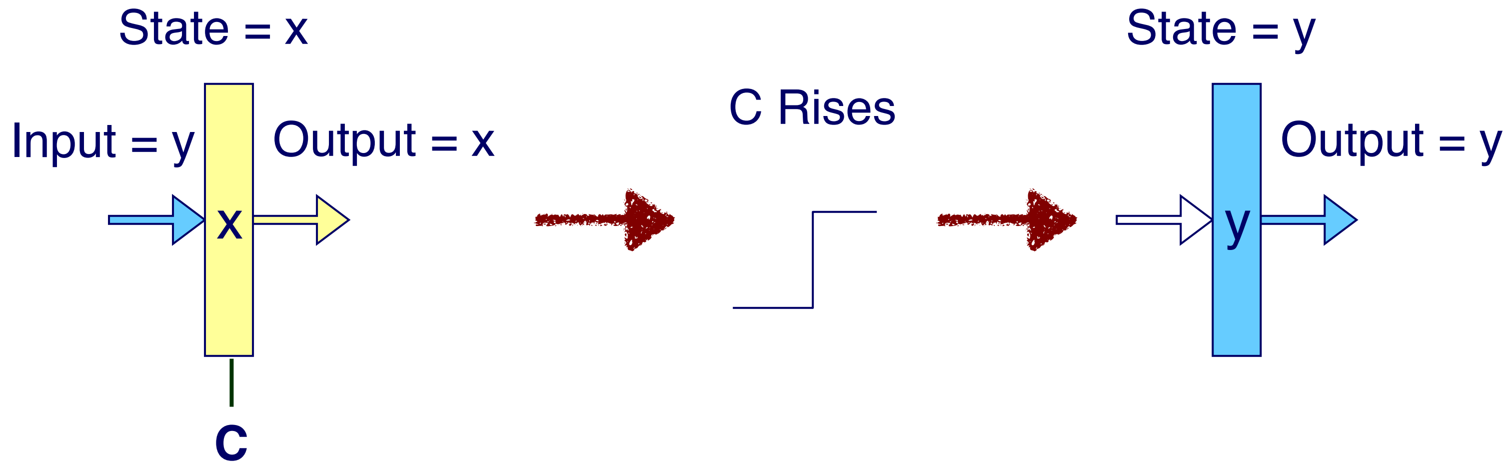


C Rises

Register Operation

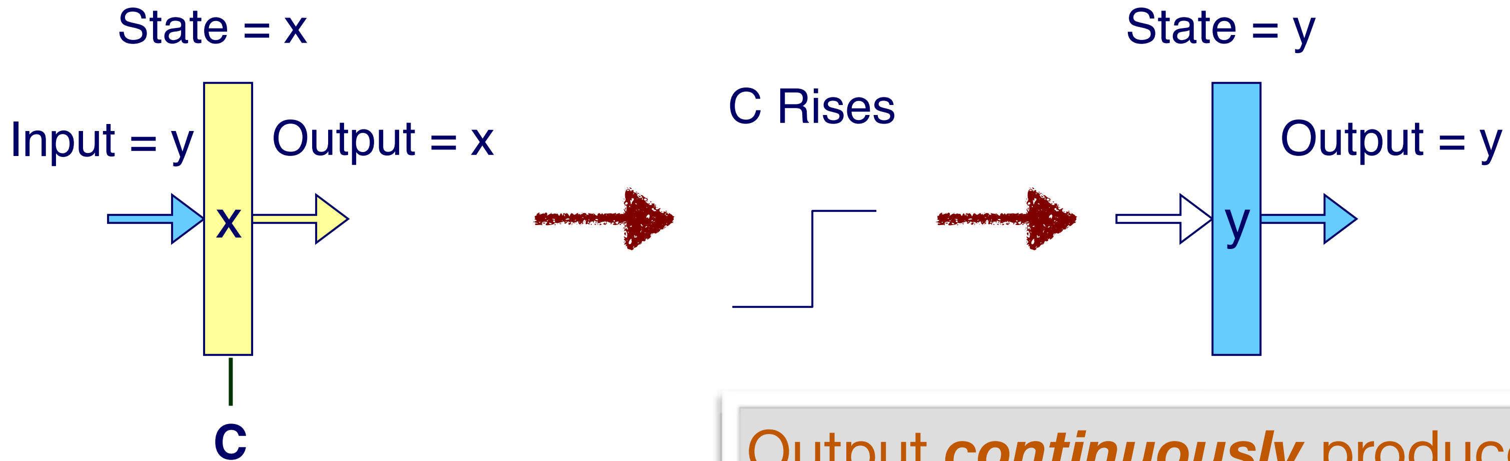


Register Operation



- Stores data bits
- For most of time acts as barrier between input and output
- As C rises, loads input
- So you'd better compute the input before the C signal rises if you want to store the input data to the register

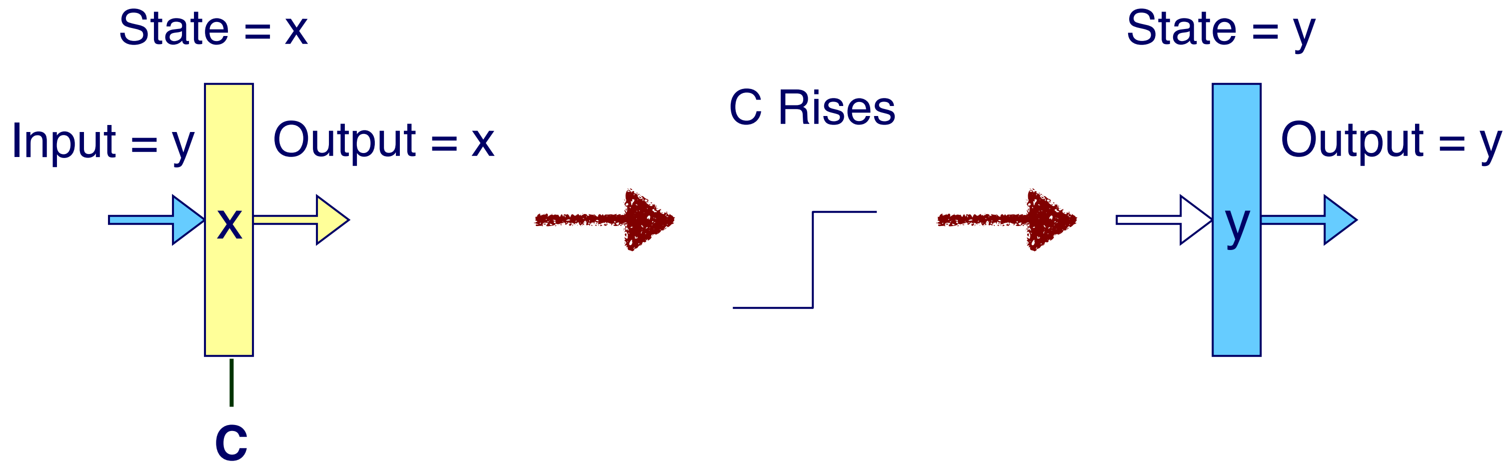
Register Operation



Output ***continuously*** produces y after the rising edge unless you cut off power.

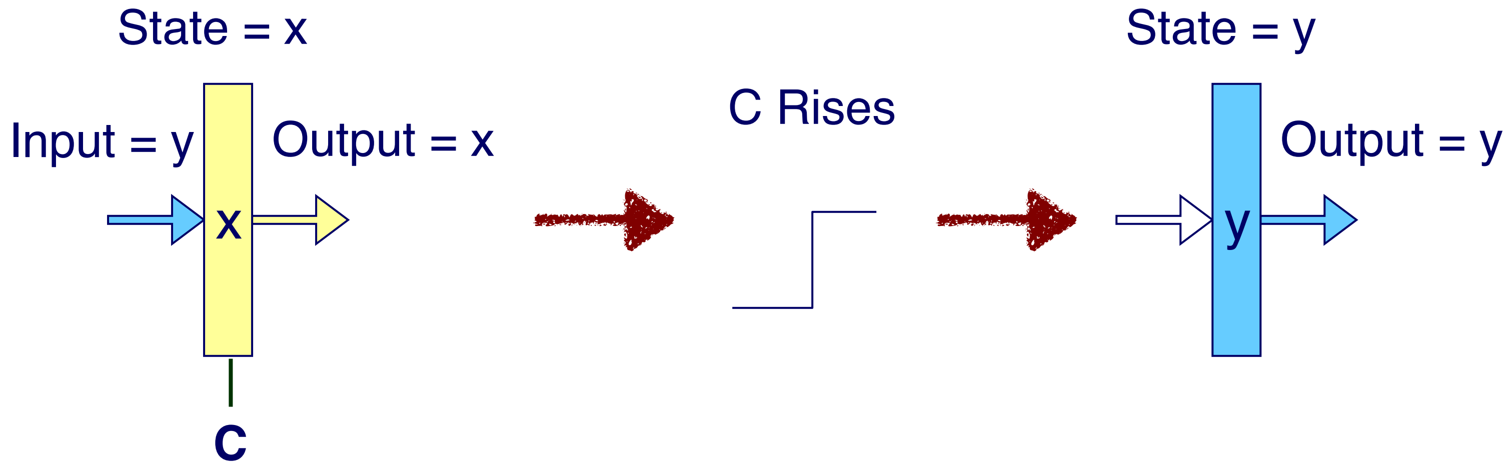
- Stores data bits
- For most of time acts as barrier between input and output
- As C rises, loads input
- So you'd better compute the input before the C signal rises if you want to store the input data to the register

Clock Signal



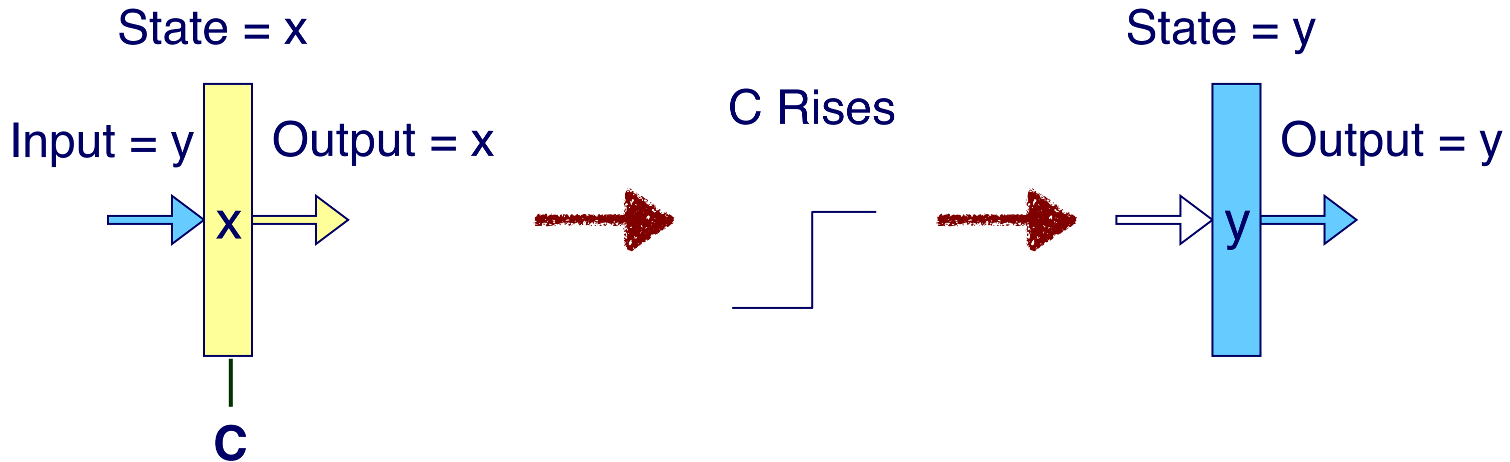
- A special C: periodically oscillating between 0 and 1
- That's called the **clock** signal. Generated by a crystal oscillator inside your computer

Clock Signal

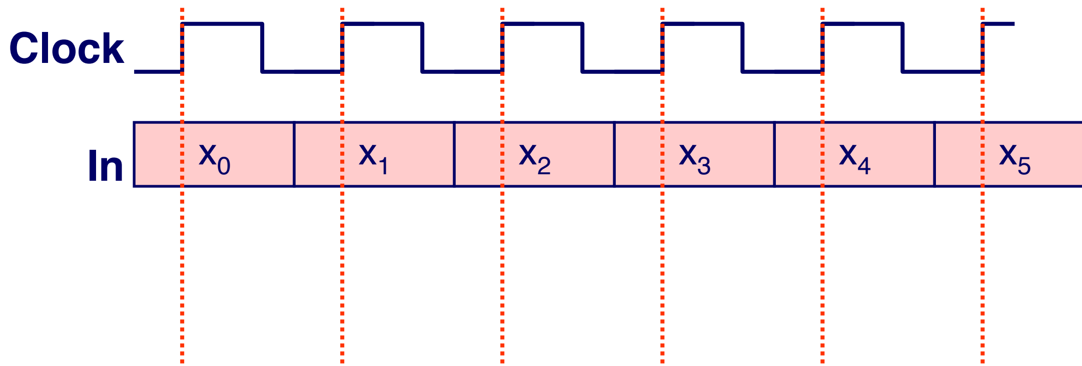


- A special C: periodically oscillating between 0 and 1
- That's called the **clock** signal. Generated by a crystal oscillator inside your computer

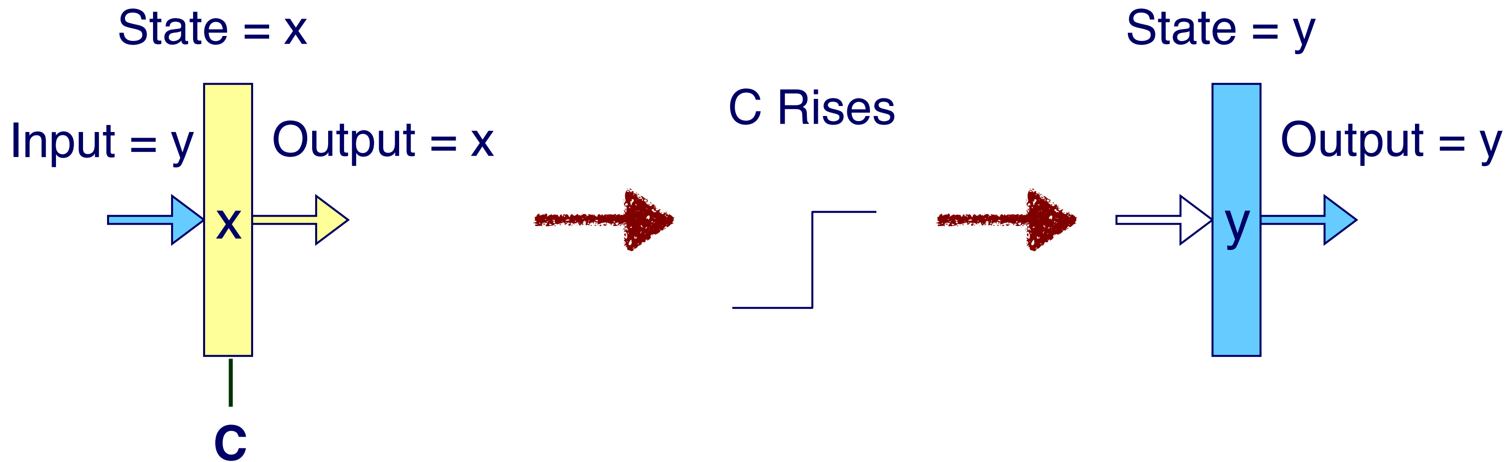
Clock Signal



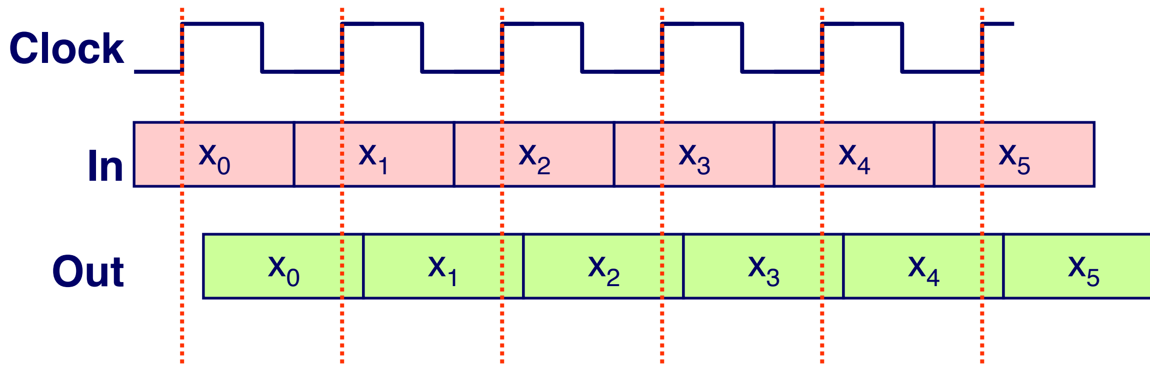
- A special C: periodically oscillating between 0 and 1
- That's called the **clock** signal. Generated by a crystal oscillator inside your computer



Clock Signal

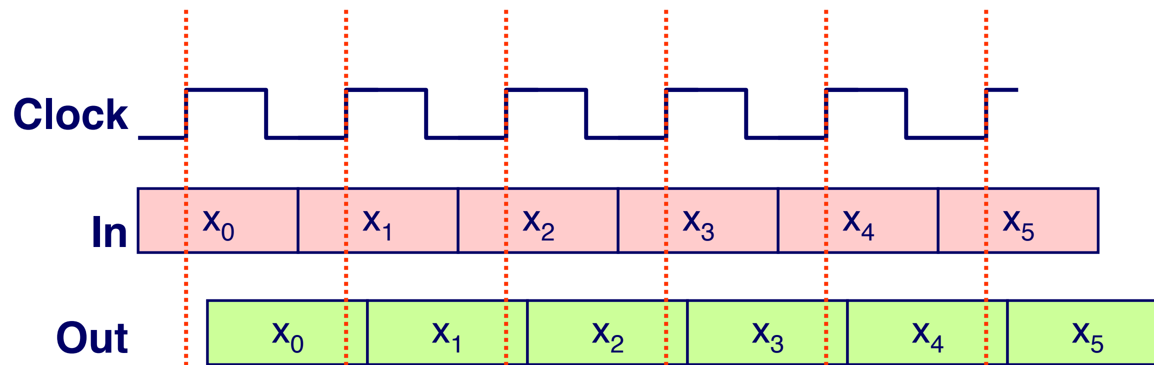


- A special C: periodically oscillating between 0 and 1
- That's called the **clock** signal. Generated by a crystal oscillator inside your computer



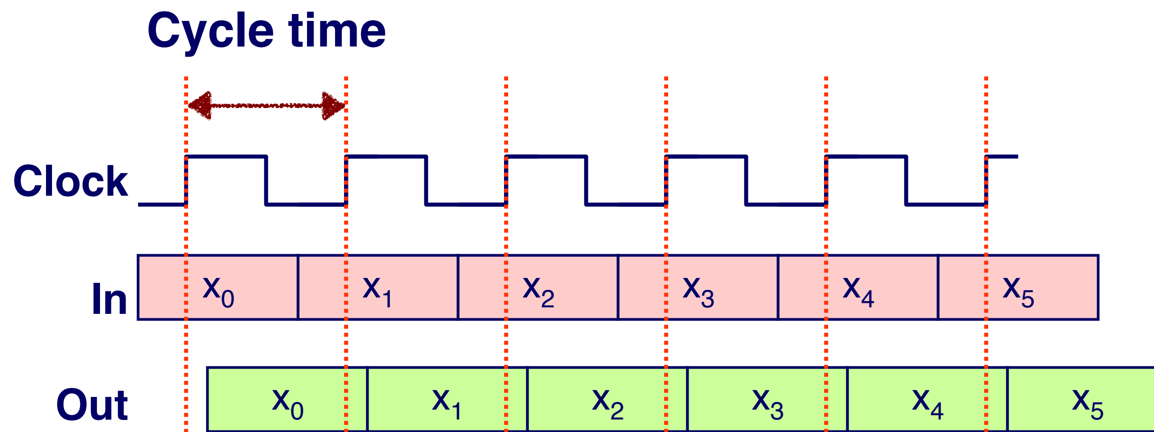
Clock Signal

- Cycle time of a clock signal: the time duration between two rising edges.



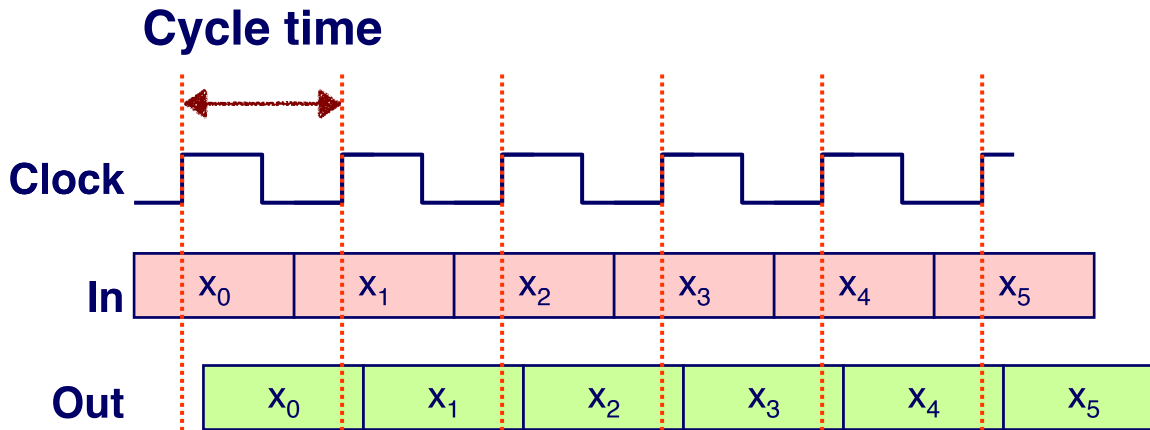
Clock Signal

- Cycle time of a clock signal: the time duration between two rising edges.



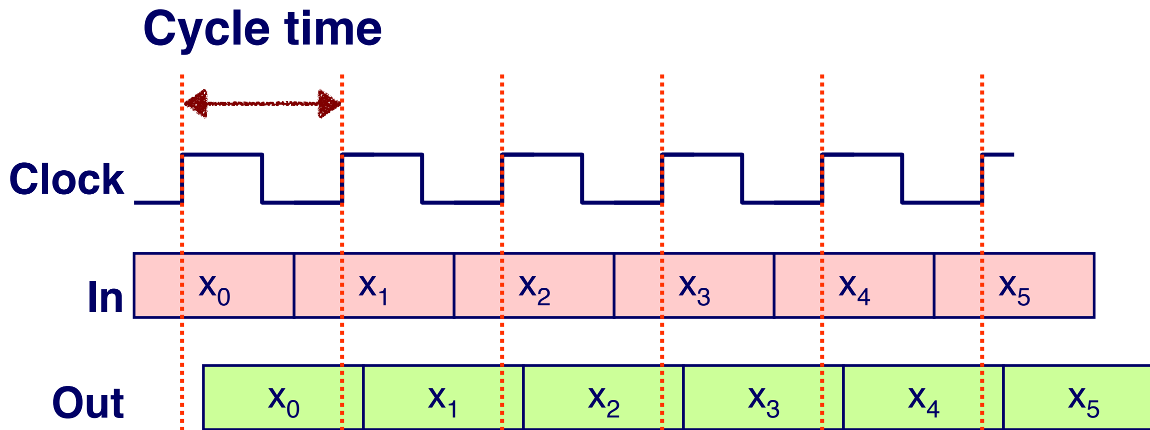
Clock Signal

- Cycle time of a clock signal: the time duration between two rising edges.
- Frequency of a clock signal: how many rising (falling) edges in 1 second.



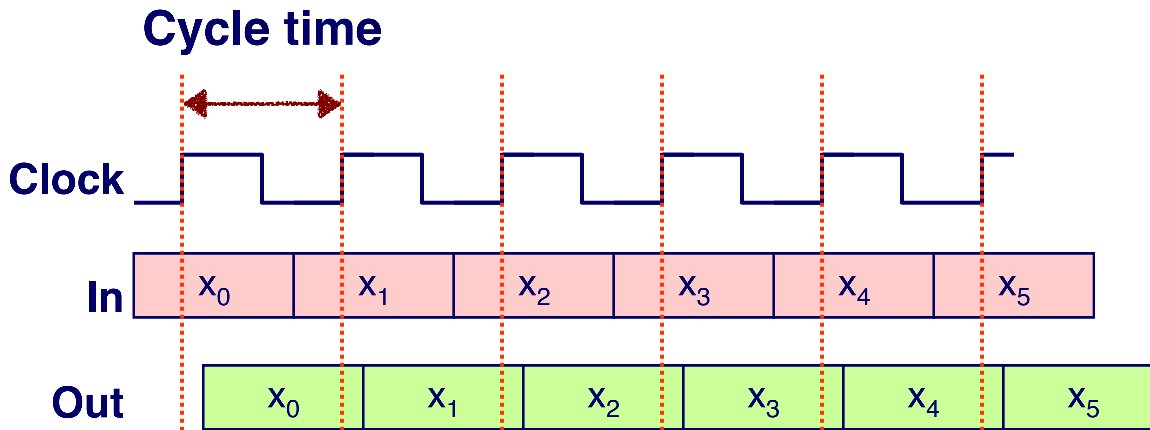
Clock Signal

- Cycle time of a clock signal: the time duration between two rising edges.
- Frequency of a clock signal: how many rising (falling) edges in 1 second.
- 1 GHz CPU means the clock frequency is 1 GHz



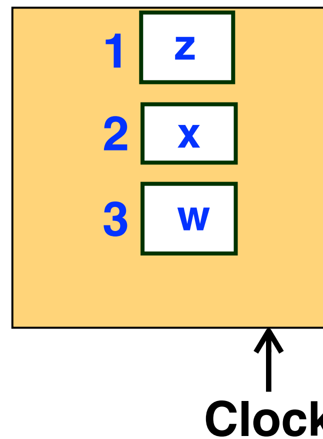
Clock Signal

- Cycle time of a clock signal: the time duration between two rising edges.
- Frequency of a clock signal: how many rising (falling) edges in 1 second.
- 1 GHz CPU means the clock frequency is 1 GHz
    - The cycle time is 1/10^9 = 1 ns



Register File

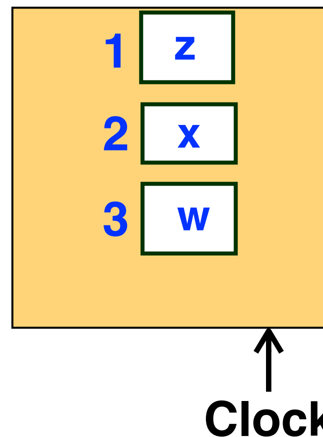
- A register file consists of a set of registers that you can individual read from and write to.



Register File

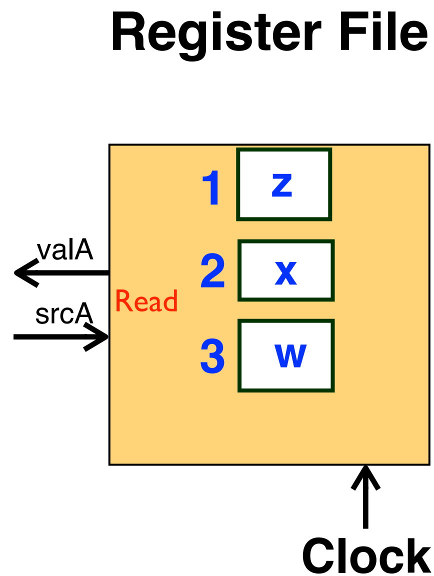
- A register file consists of a set of registers that you can individual read from and write to.
- To read: give a register file ID, and read the stored value out

Register File



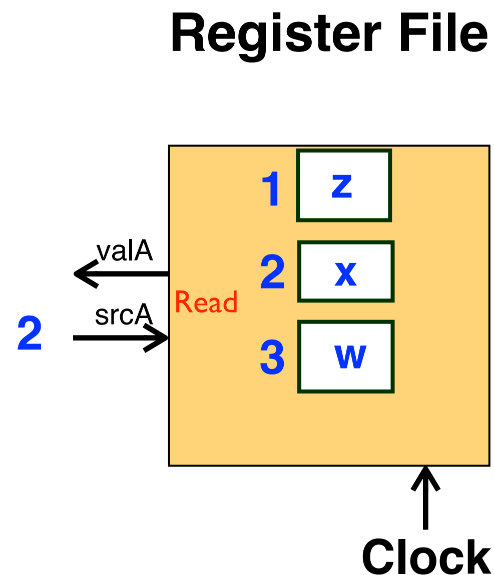
Register File

- A register file consists of a set of registers that you can individual read from and write to.
- To read: give a register file ID, and read the stored value out



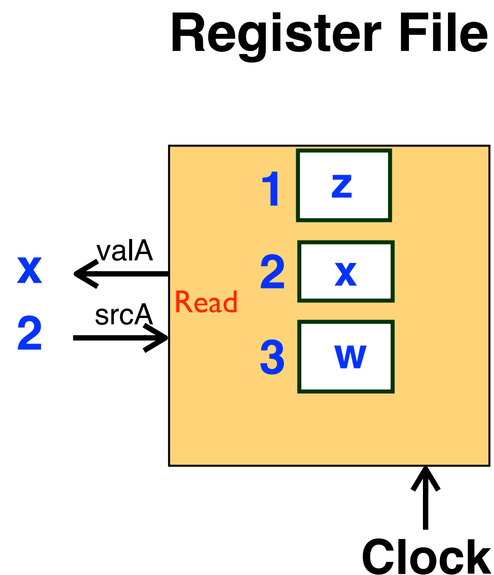
Register File

- A register file consists of a set of registers that you can individual read from and write to.
- To read: give a register file ID, and read the stored value out



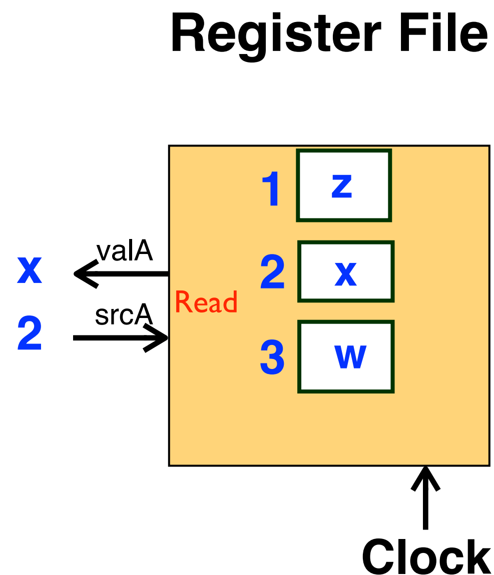
Register File

- A register file consists of a set of registers that you can individual read from and write to.
- To read: give a register file ID, and read the stored value out



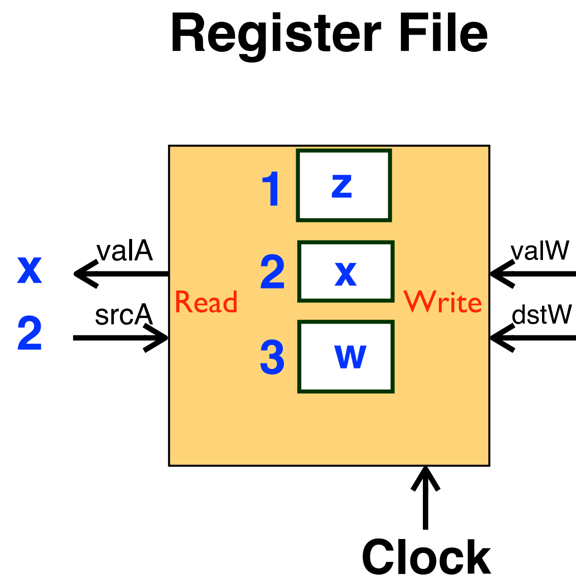
Register File

- A register file consists of a set of registers that you can individual read from and write to.
- To read: give a register file ID, and read the stored value out
- To write: give a register file ID, a new value, overwrite the old value



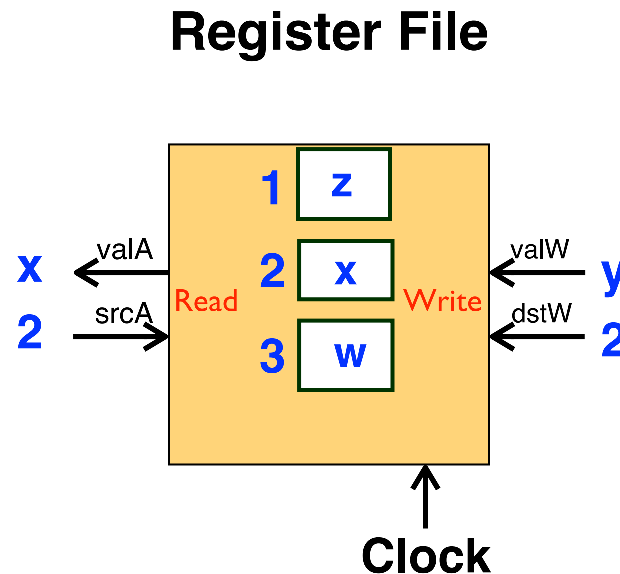
Register File

- A register file consists of a set of registers that you can individual read from and write to.
- To read: give a register file ID, and read the stored value out
- To write: give a register file ID, a new value, overwrite the old value



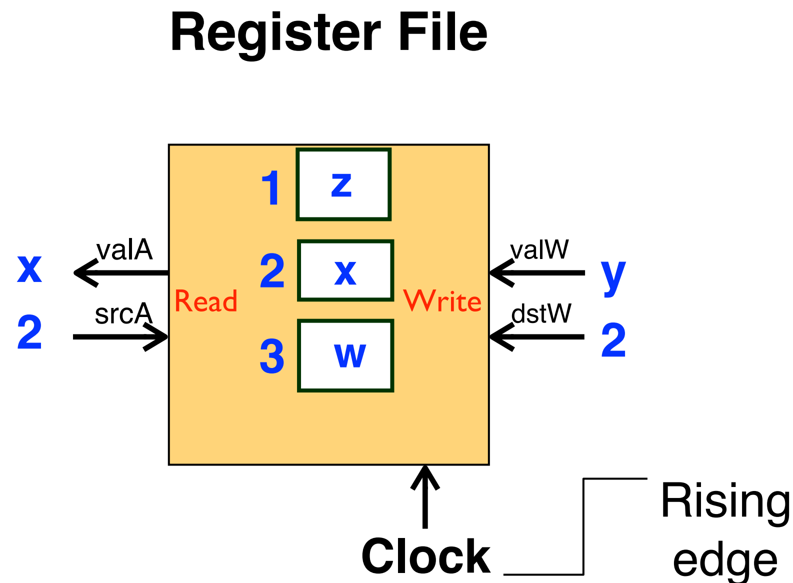
Register File

- A register file consists of a set of registers that you can individual read from and write to.
- To read: give a register file ID, and read the stored value out
- To write: give a register file ID, a new value, overwrite the old value



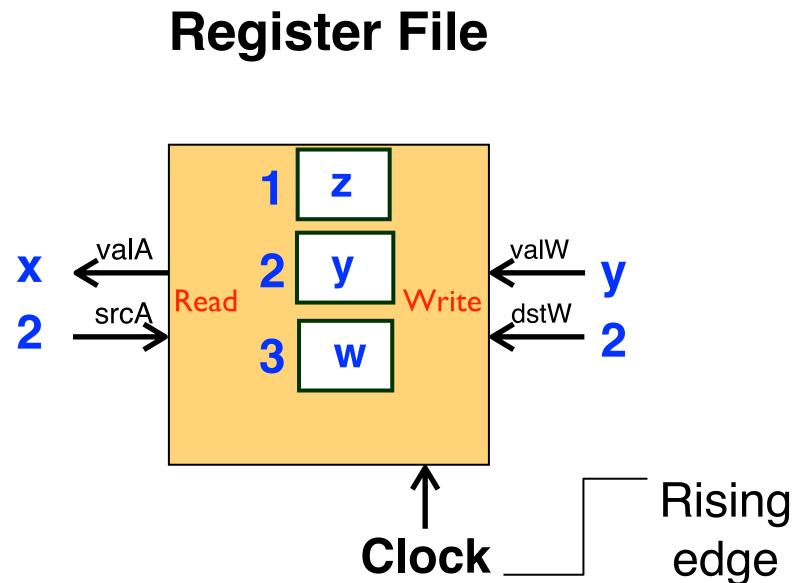
Register File

- A register file consists of a set of registers that you can individual read from and write to.
- To read: give a register file ID, and read the stored value out
- To write: give a register file ID, a new value, overwrite the old value



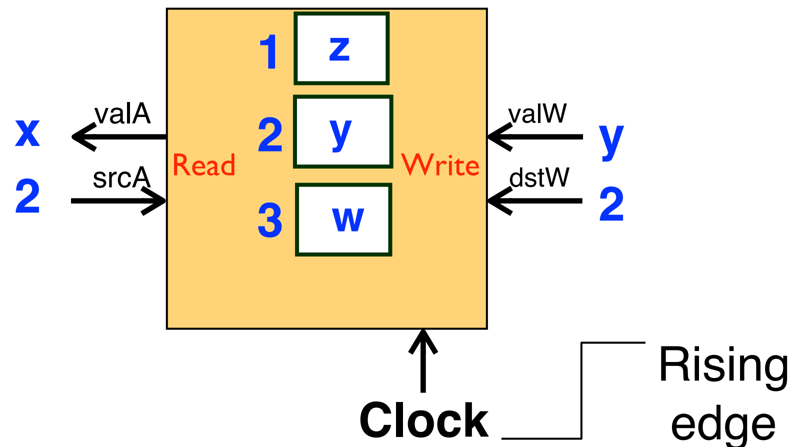
Register File

- A register file consists of a set of registers that you can individual read from and write to.
- To read: give a register file ID, and read the stored value out
- To write: give a register file ID, a new value, overwrite the old value



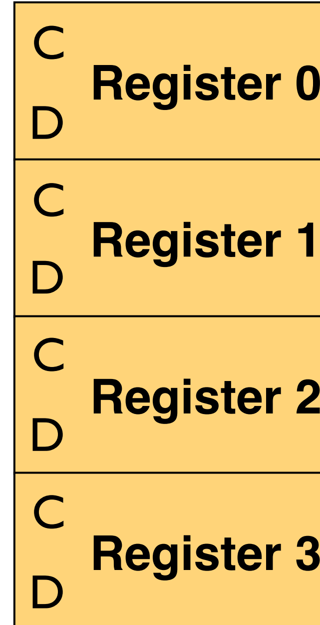
Register File

- A register file consists of a set of registers that you can individual read from and write to.
- To read: give a register file ID, and read the stored value out
- To write: give a register file ID, a new value, overwrite the old value
- How do we build a register file out of individual registers??



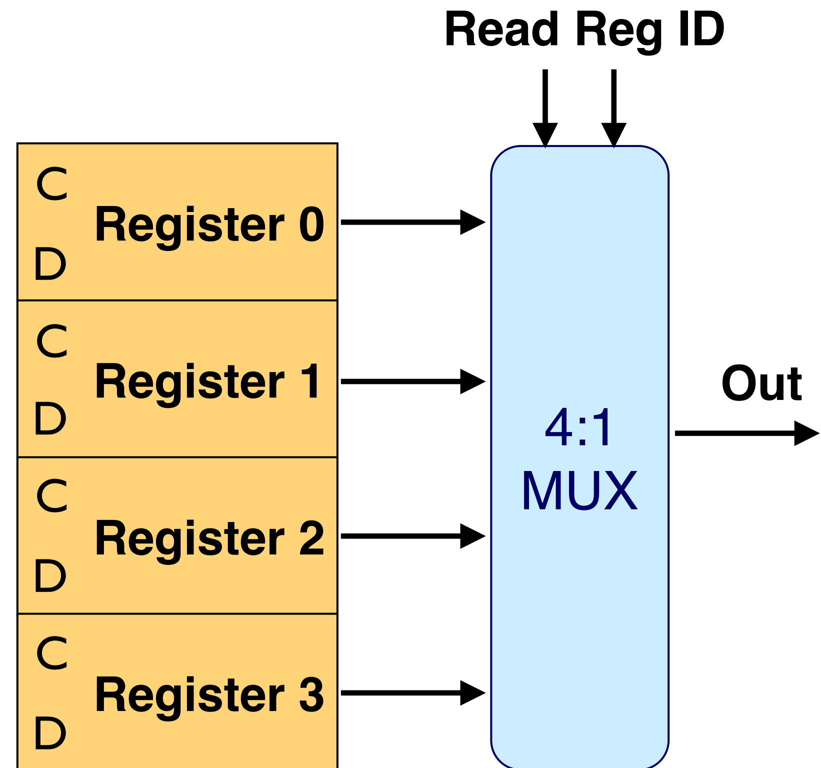
Register File Read

- Continuously read a register independent of the clock signal

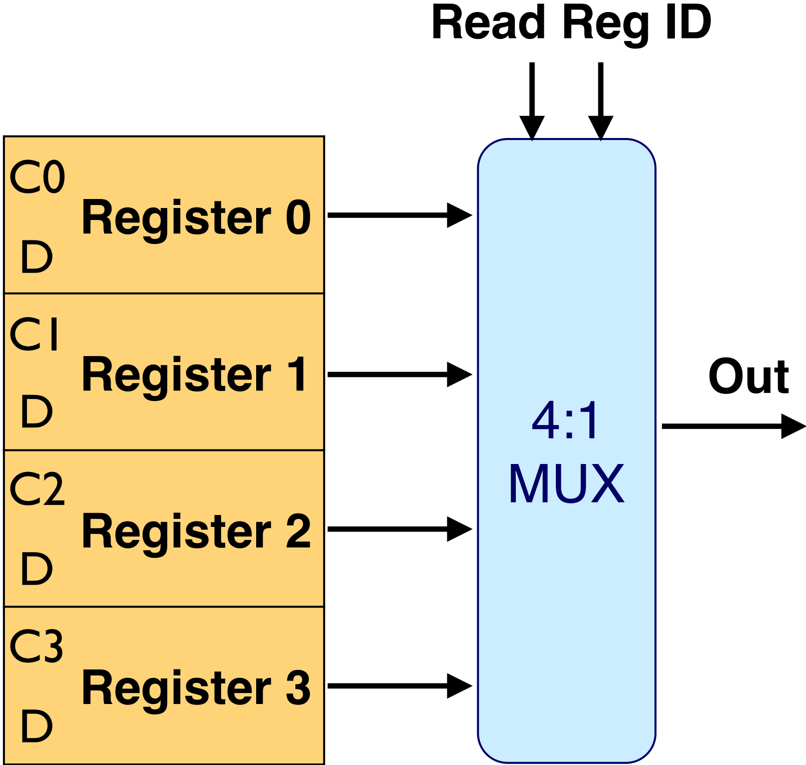


Register File Read

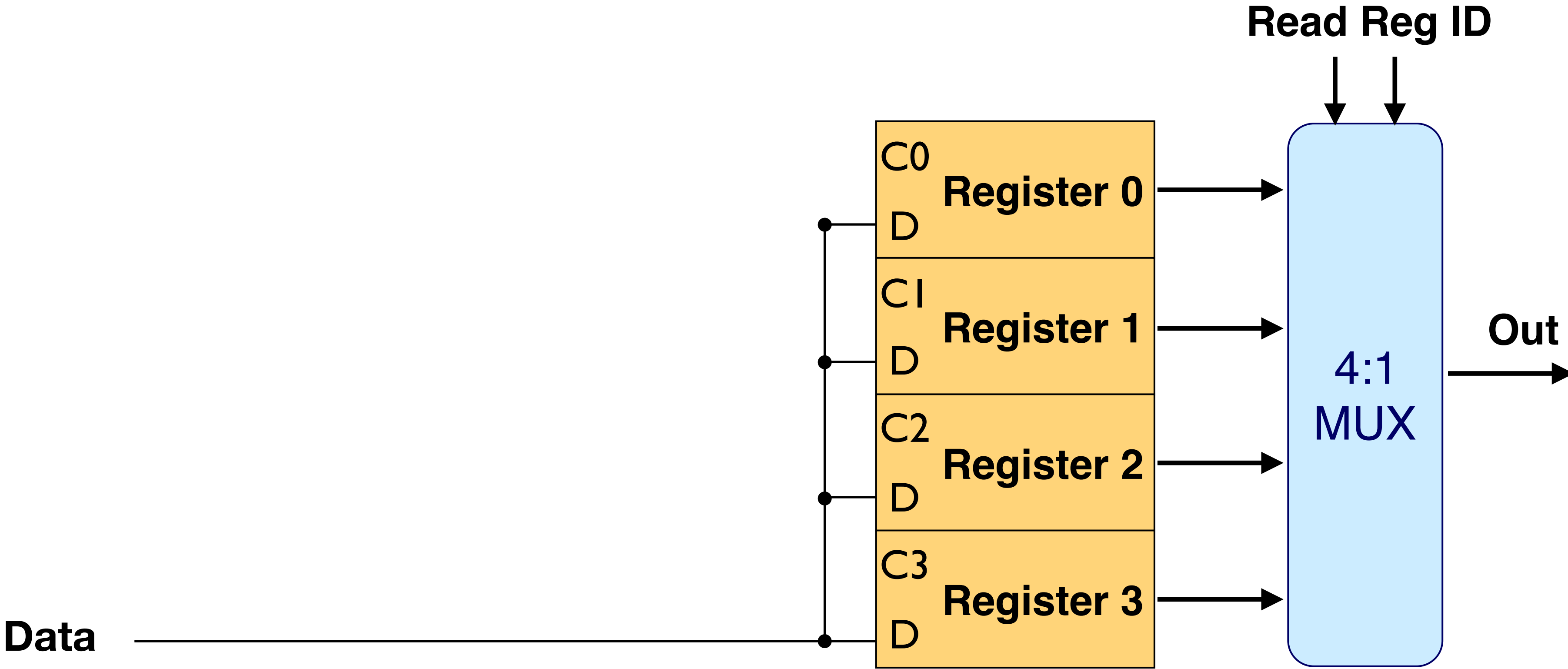
- Continuously read a register independent of the clock signal



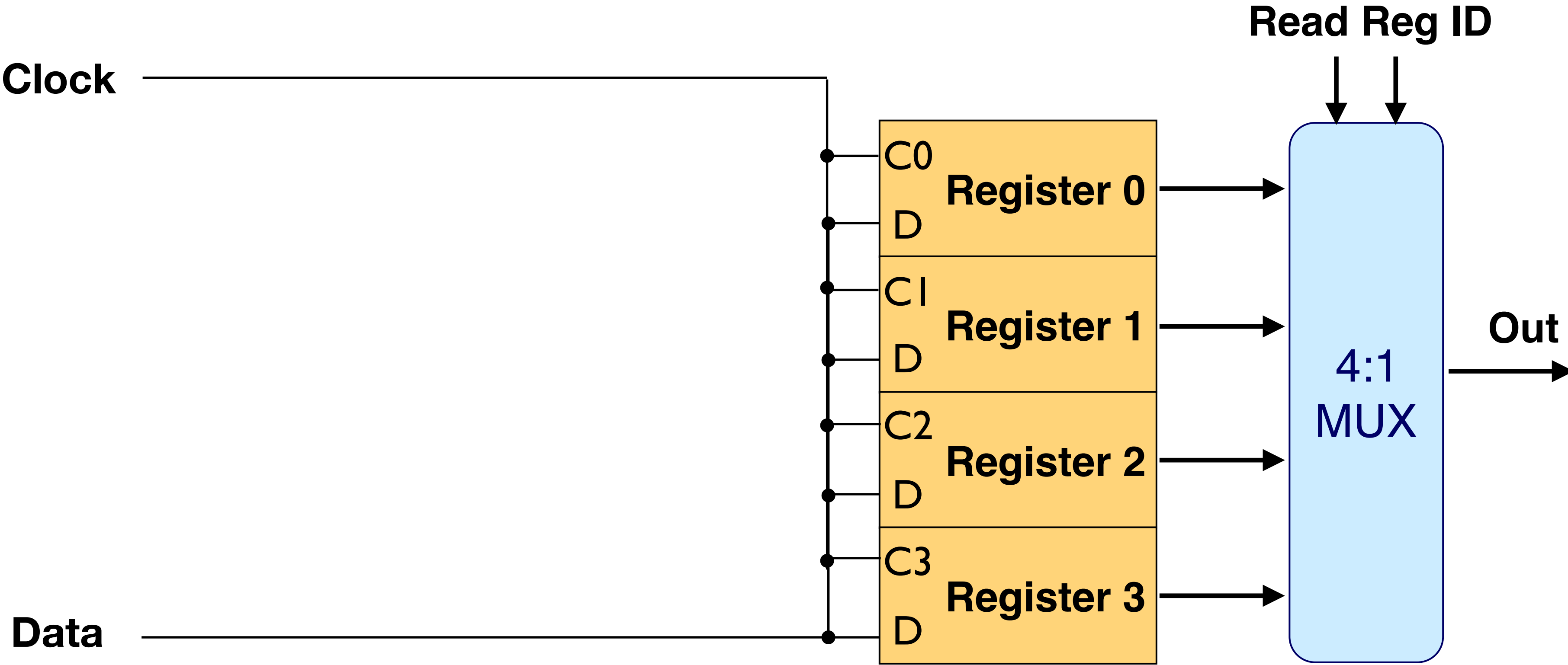
Register File Write



Register File Write

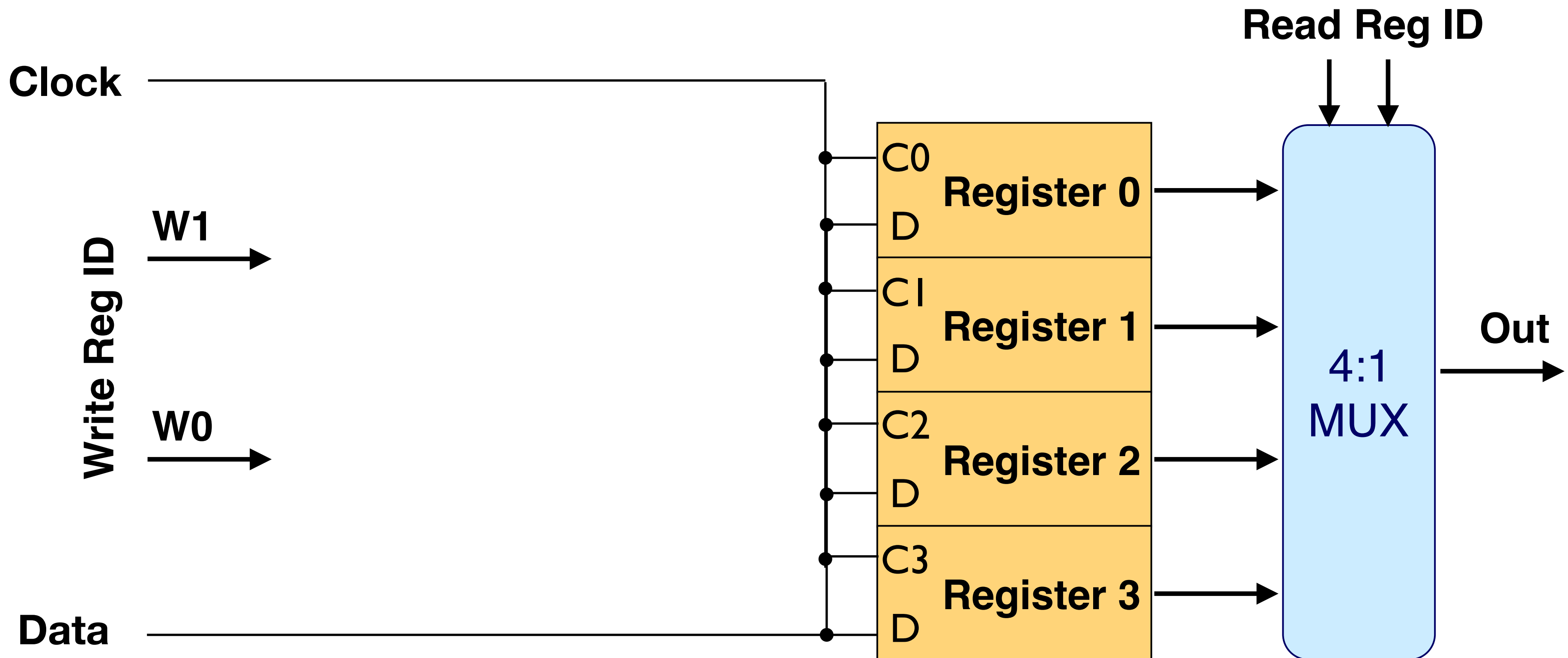


Register File Write



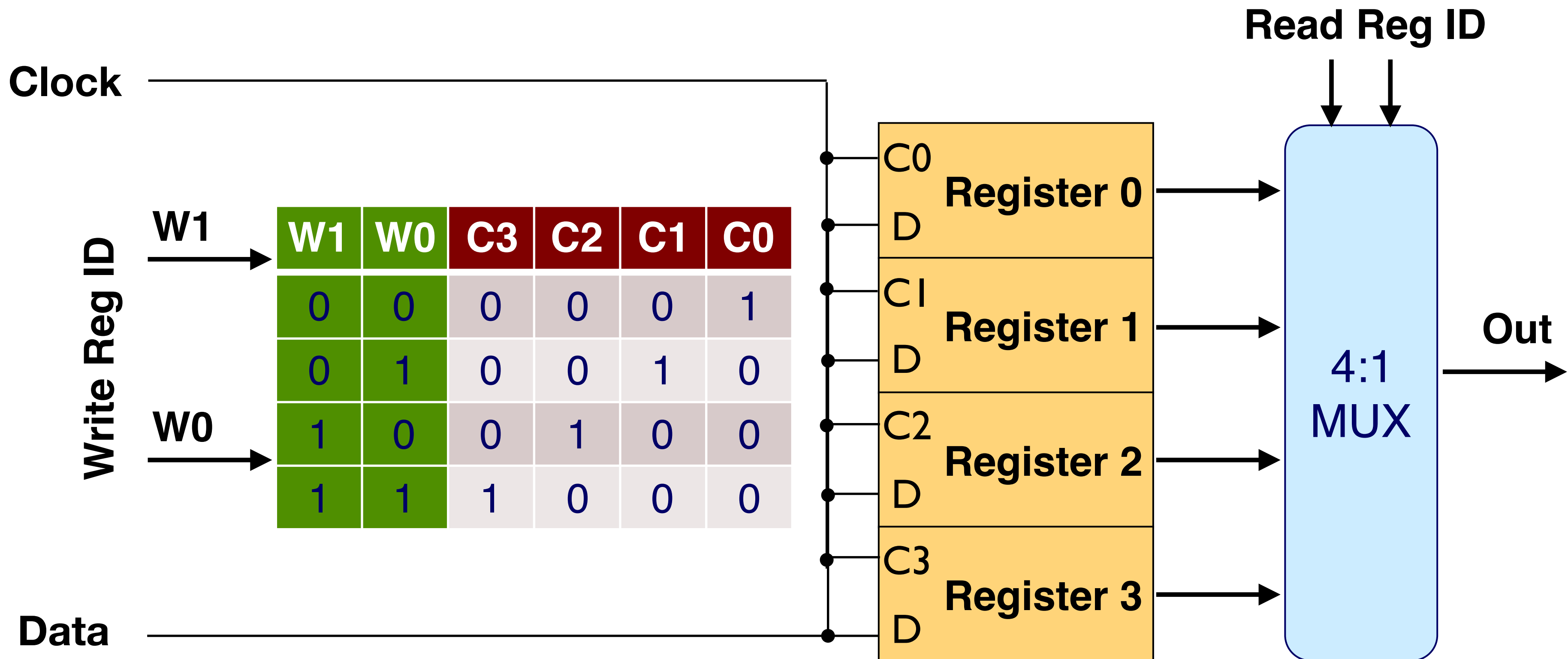
Register File Write

- Only write the a specific register when the clock rises. How??



Register File Write

- Only write the a specific register when the clock rises. How??



W1	W0	C3	C2	C1	C0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Decoder

W1	W0	C3	C2	C1	C0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

W0

W1

C0

C1

C2

C3

Decoder

W1	W0	C3	C2	C1	C0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

W0

W1

C0

C1

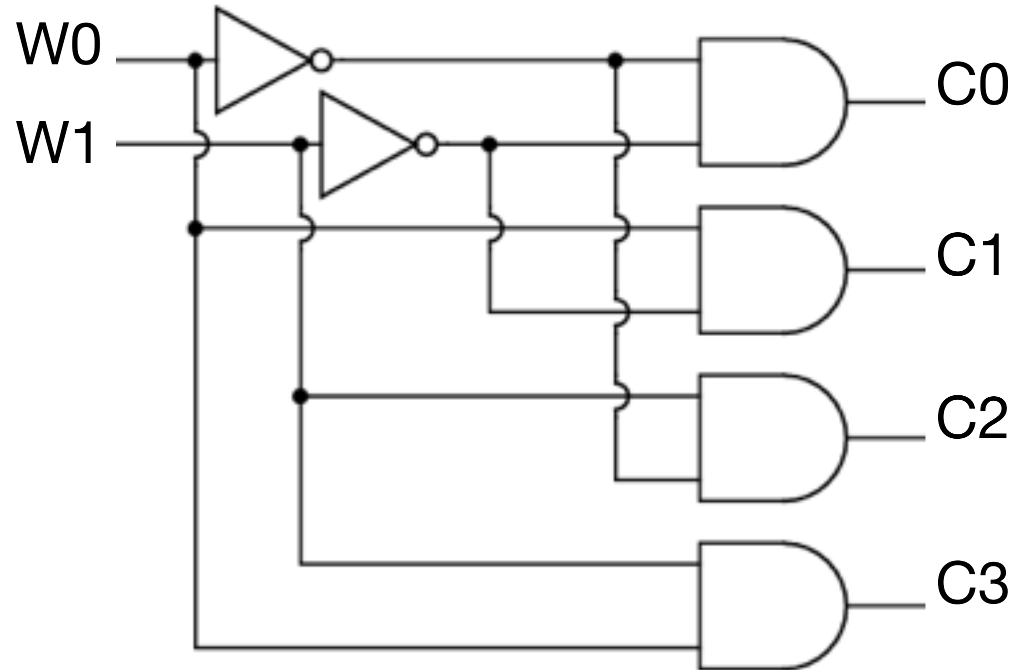
C2

C3

C0 = !W1 & !W0
C1= !W1 & W0
C2 = W1 & !W0
C3 = W1 & W0

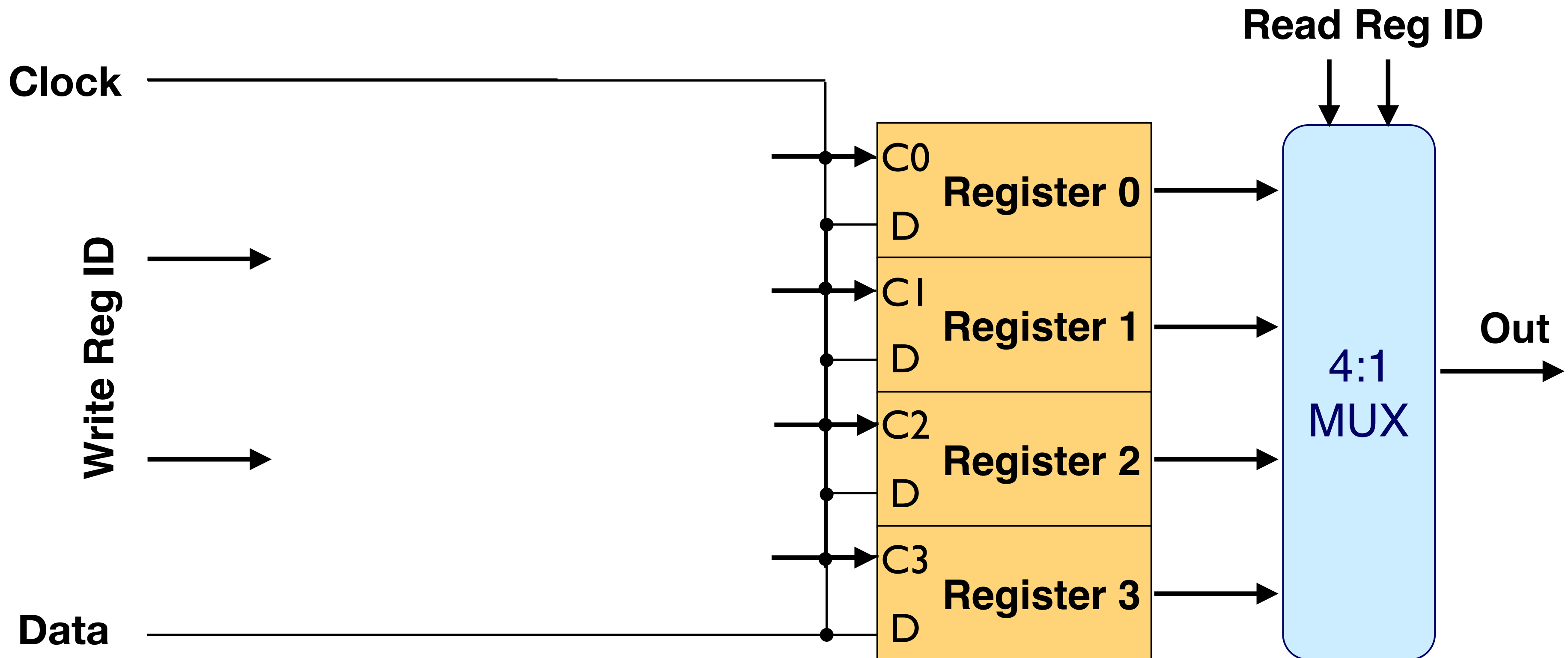
Decoder

W1	W0	C3	C2	C1	C0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

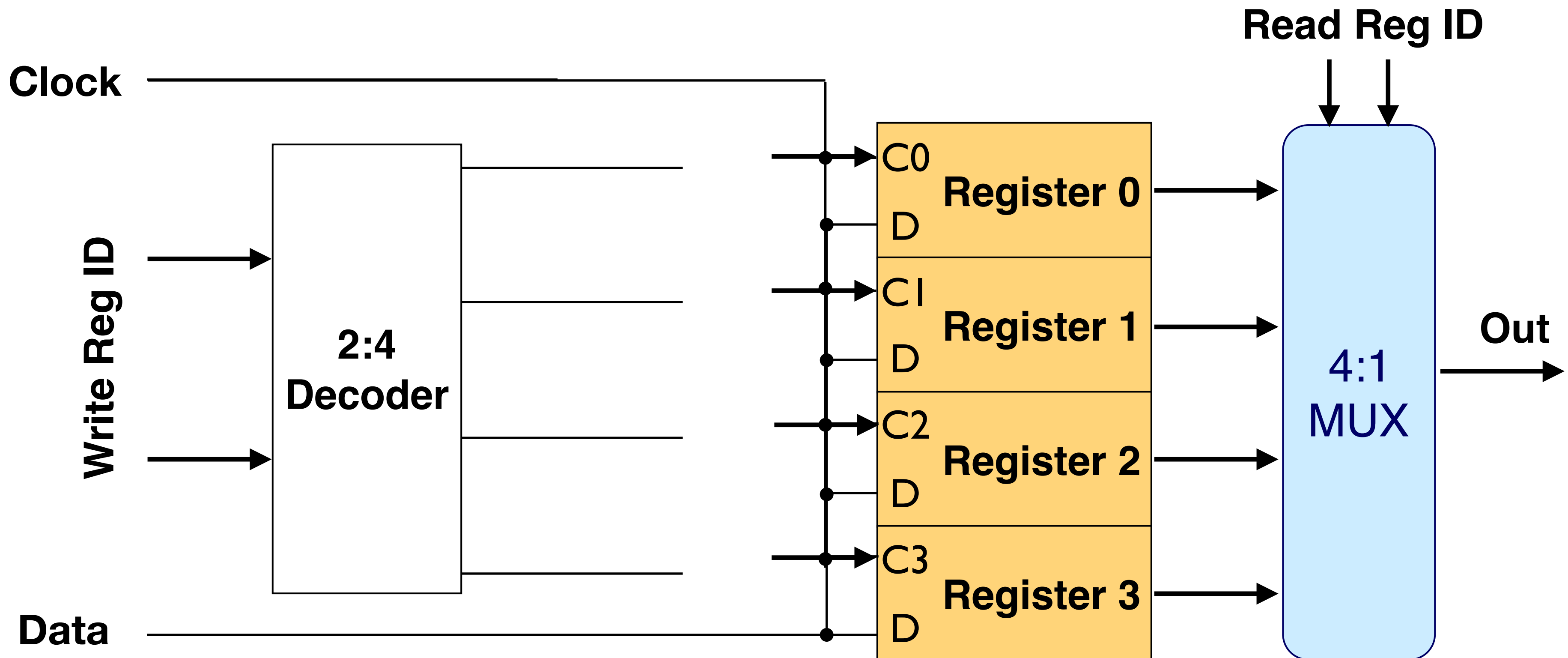


C0 = !W1 & !W0
C1= !W1 & W0
C2 = W1 & !W0
C3 = W1 & W0

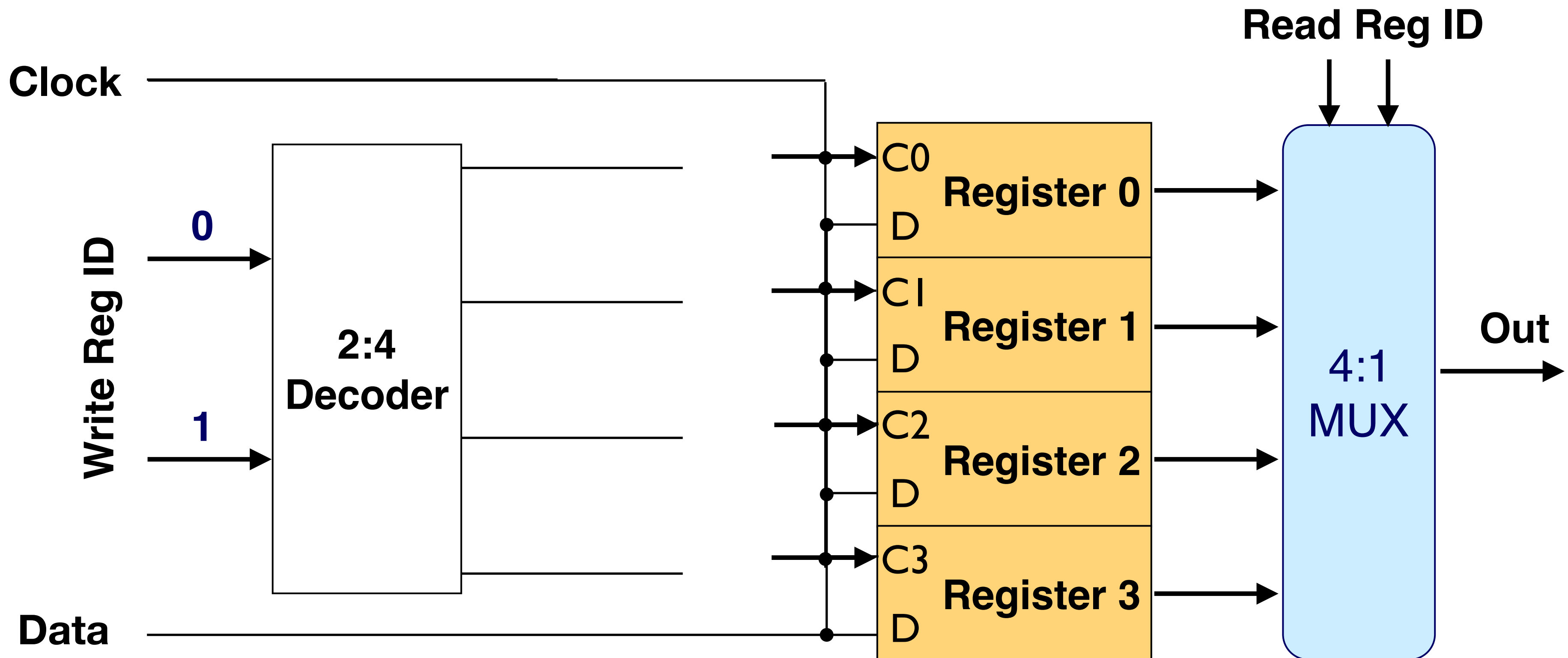
Register File Write



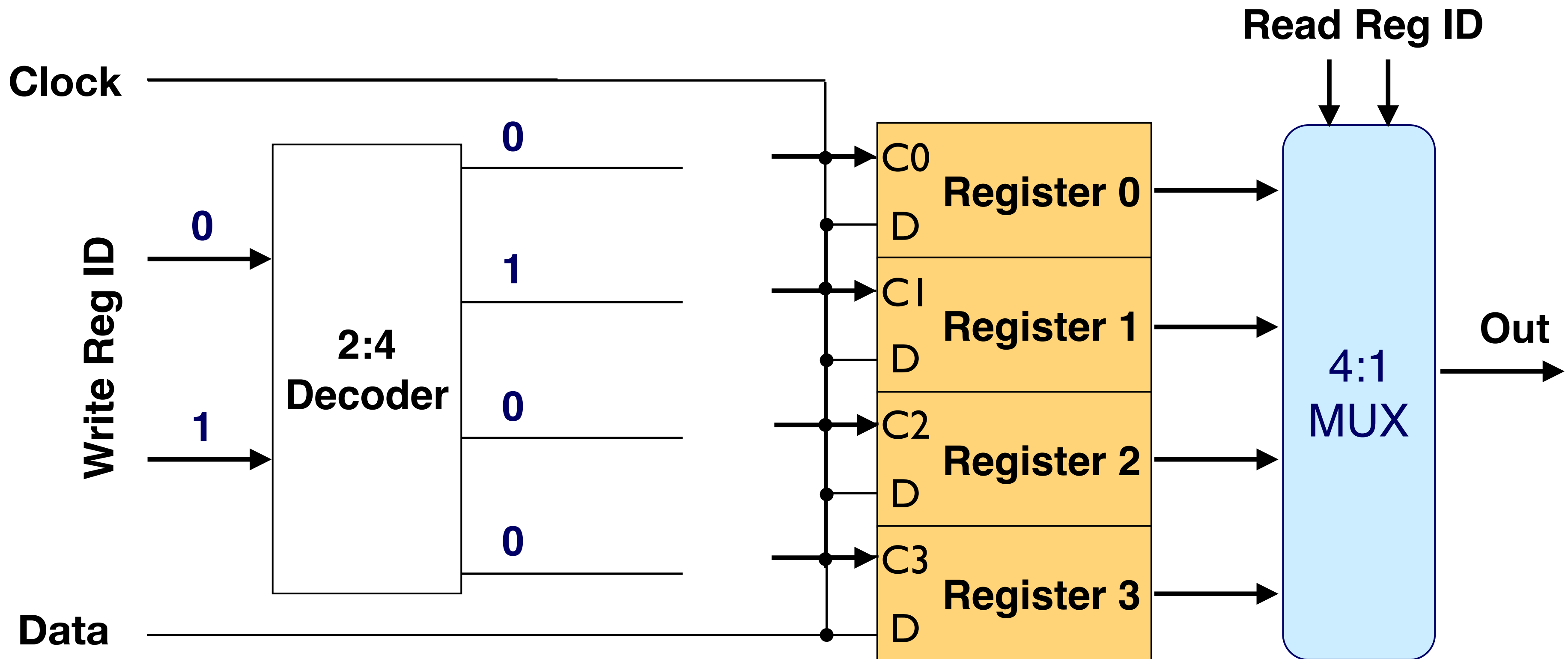
Register File Write



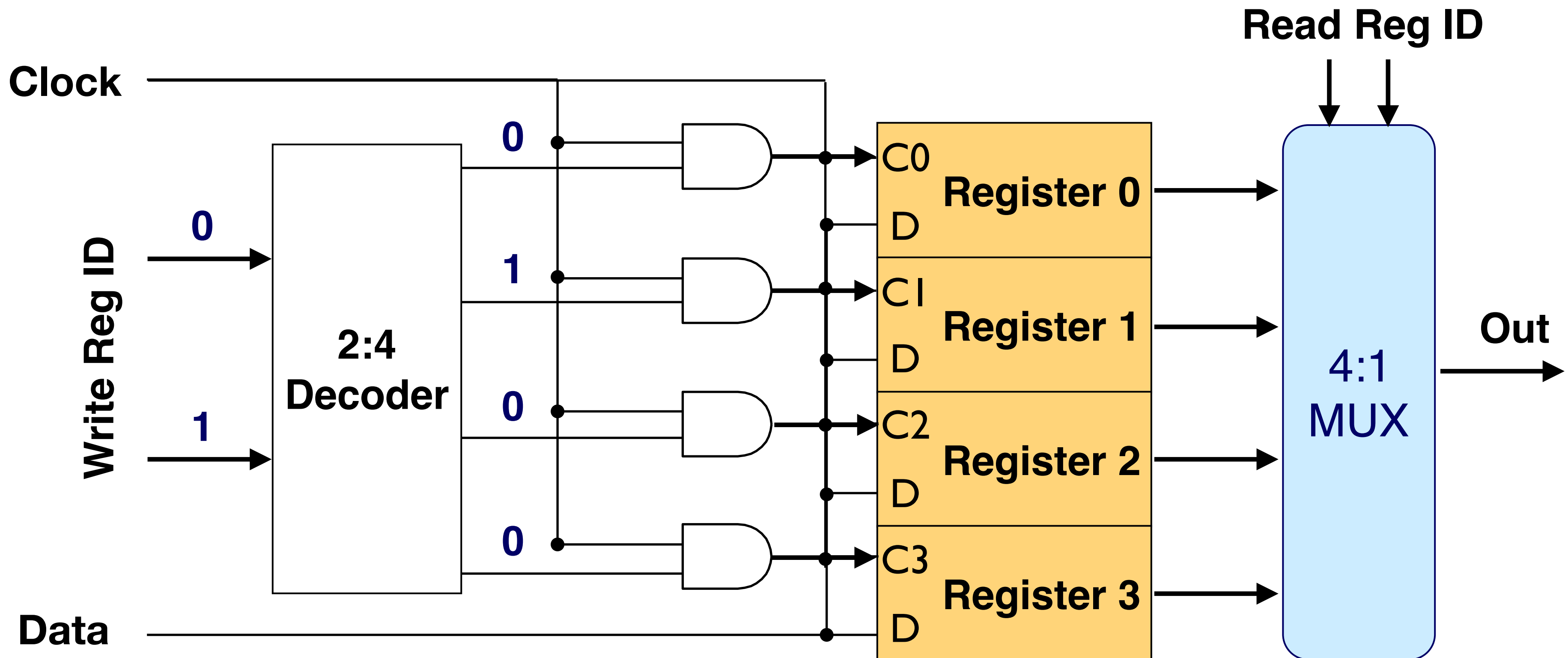
Register File Write



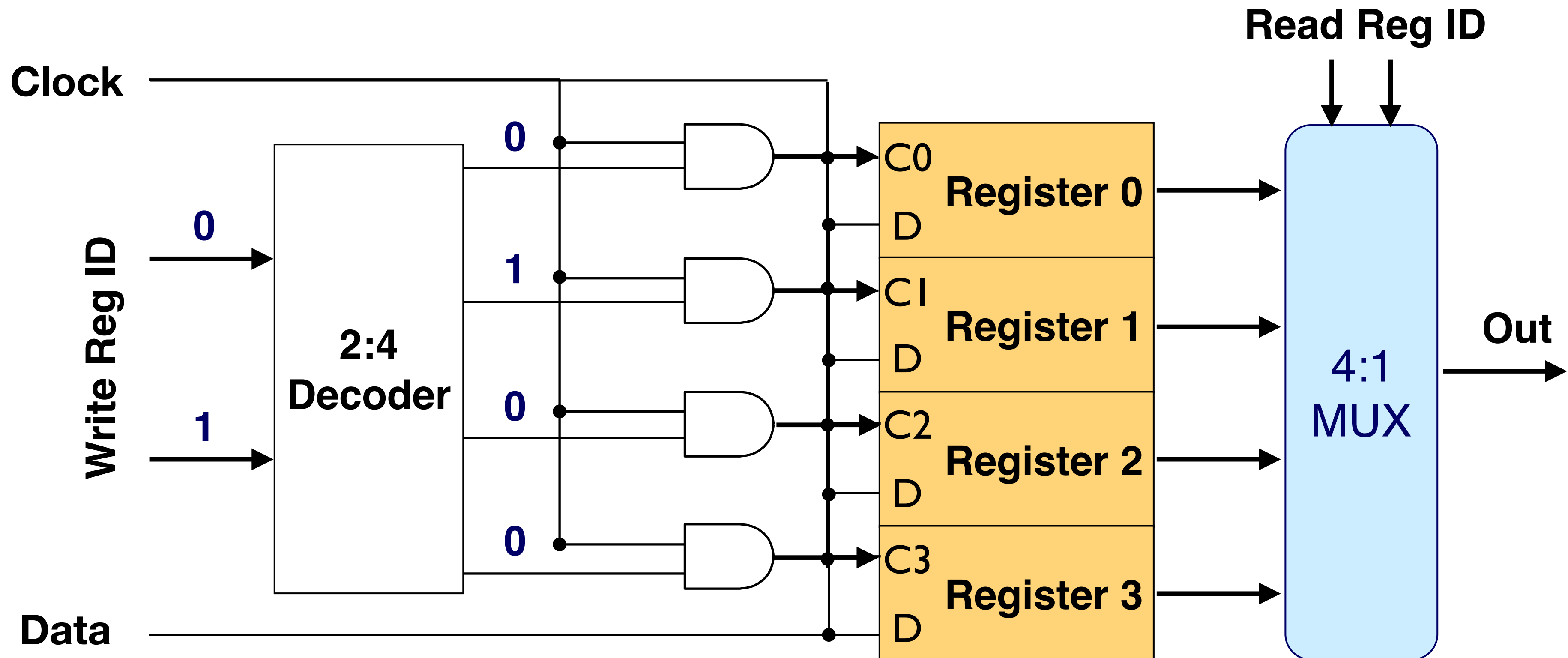
Register File Write



Register File Write



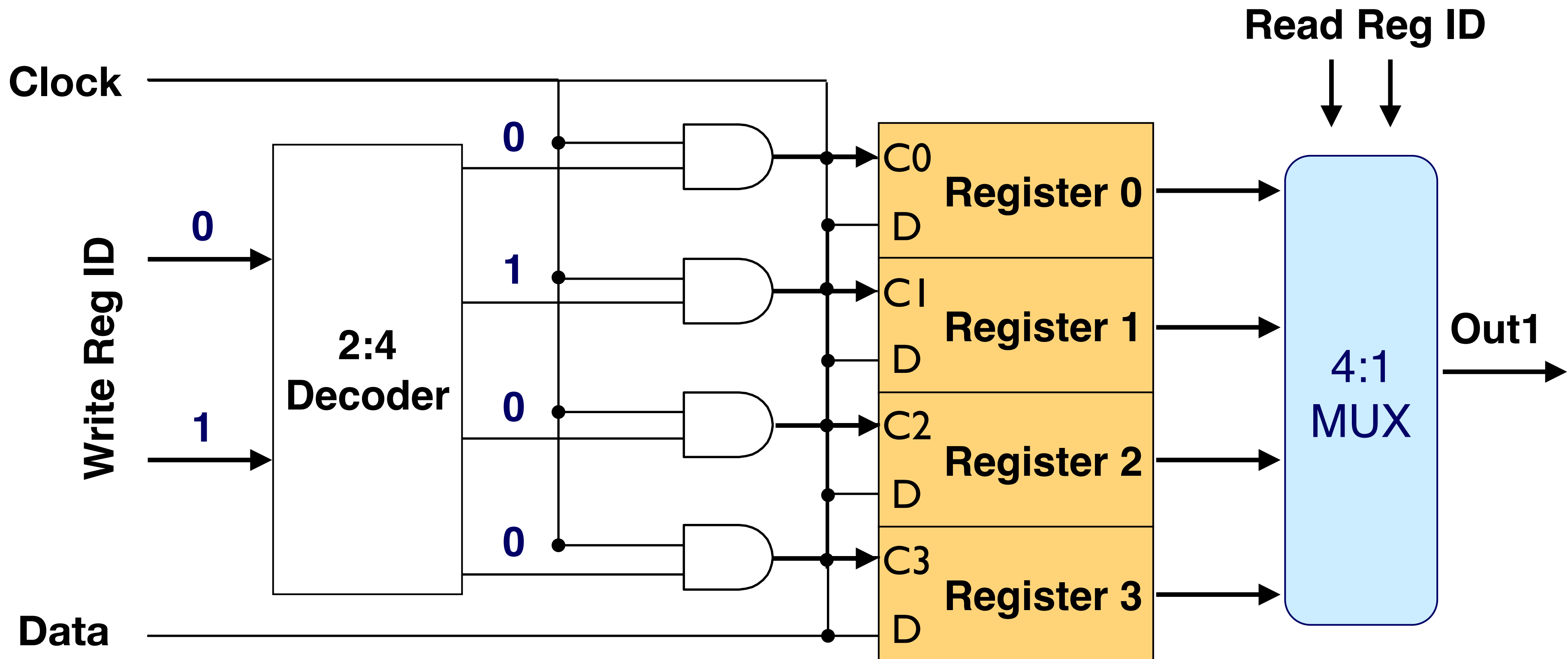
Register File Write



- This implementation can read 1 register and write 1 register at the same time: 1 read port and 1 write port

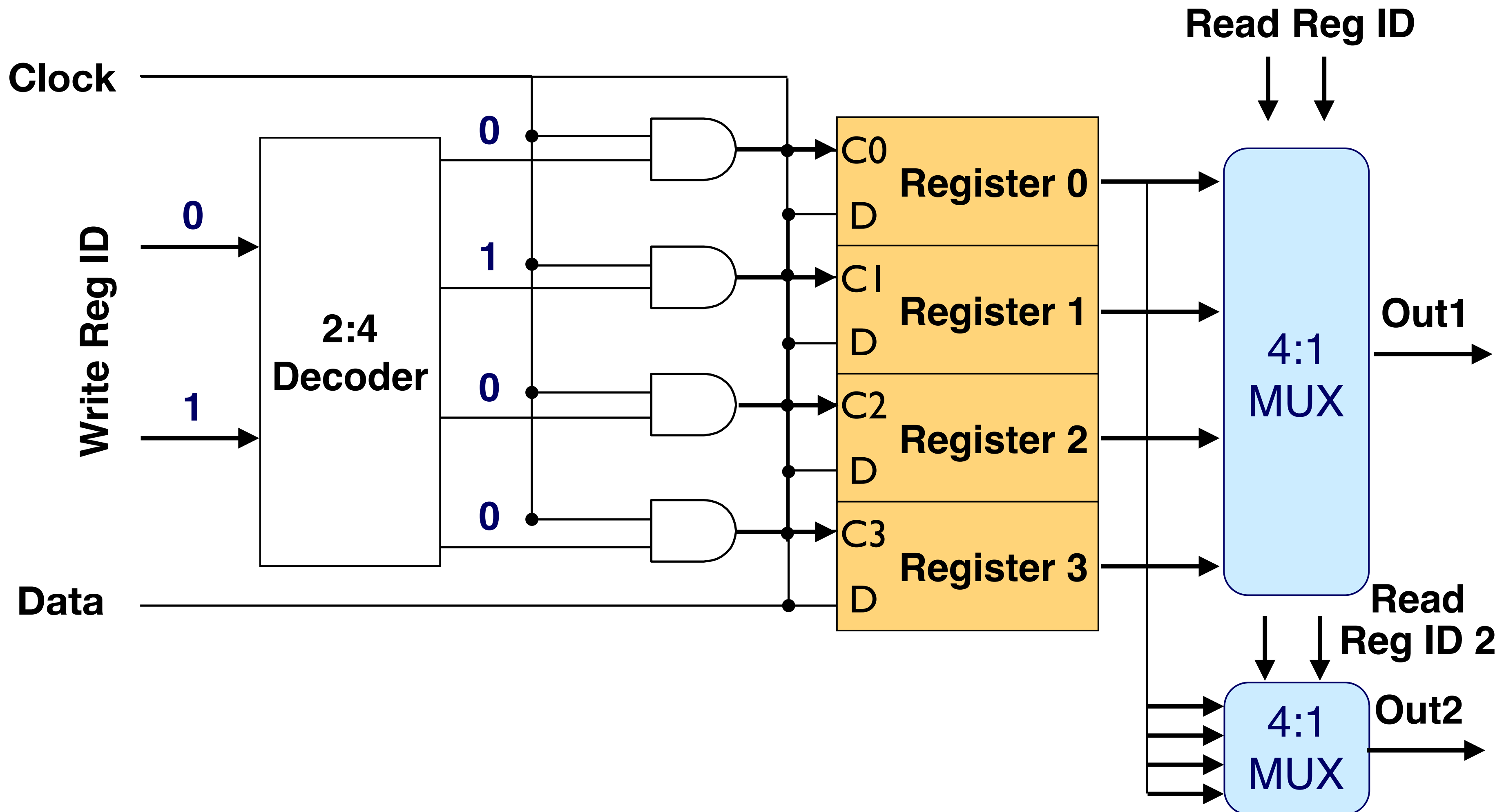
Multi-Port Register File

- What if we want to read multiple registers at the same time?



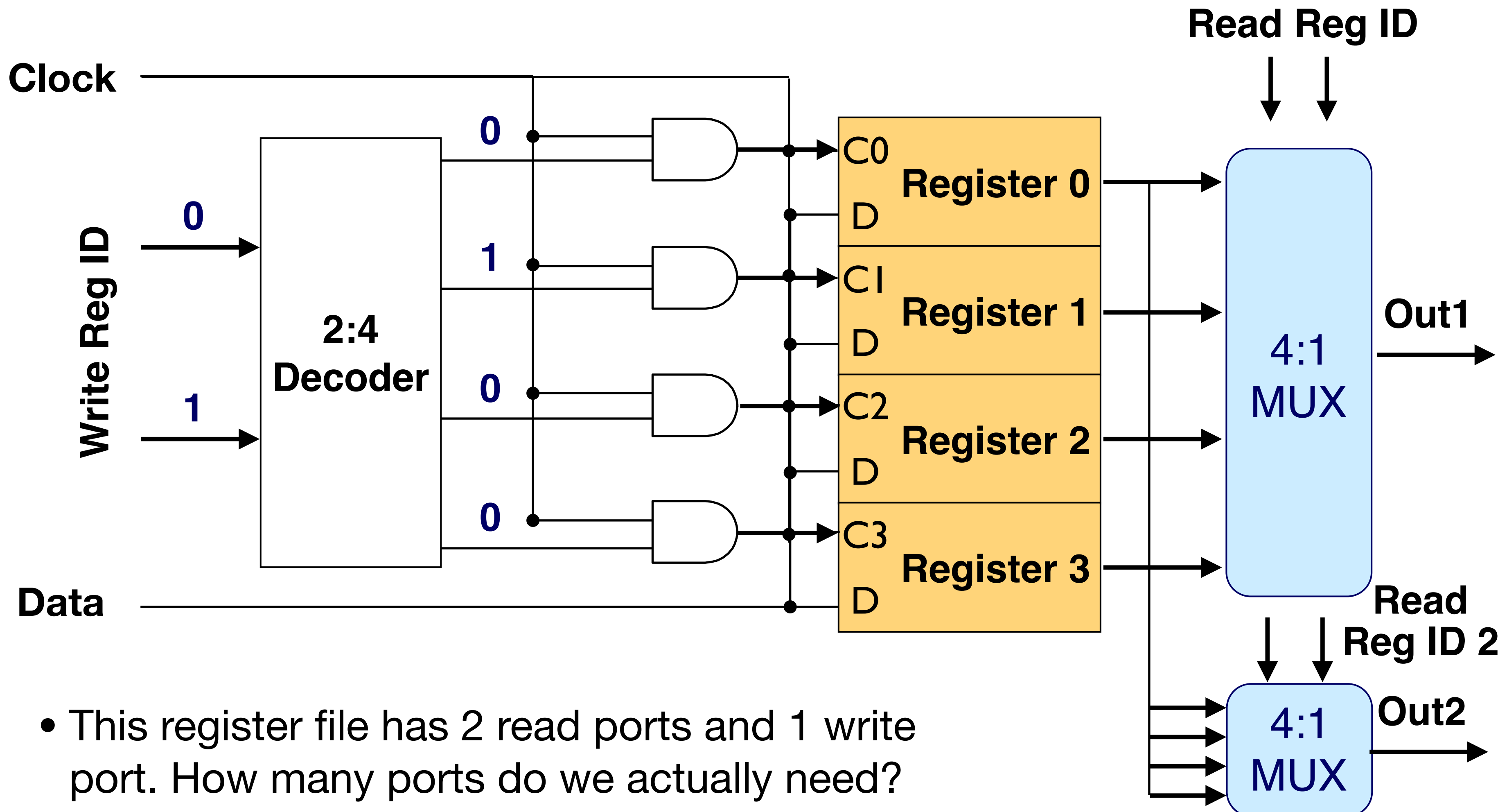
Multi-Port Register File

- What if we want to read multiple registers at the same time?



Multi-Port Register File

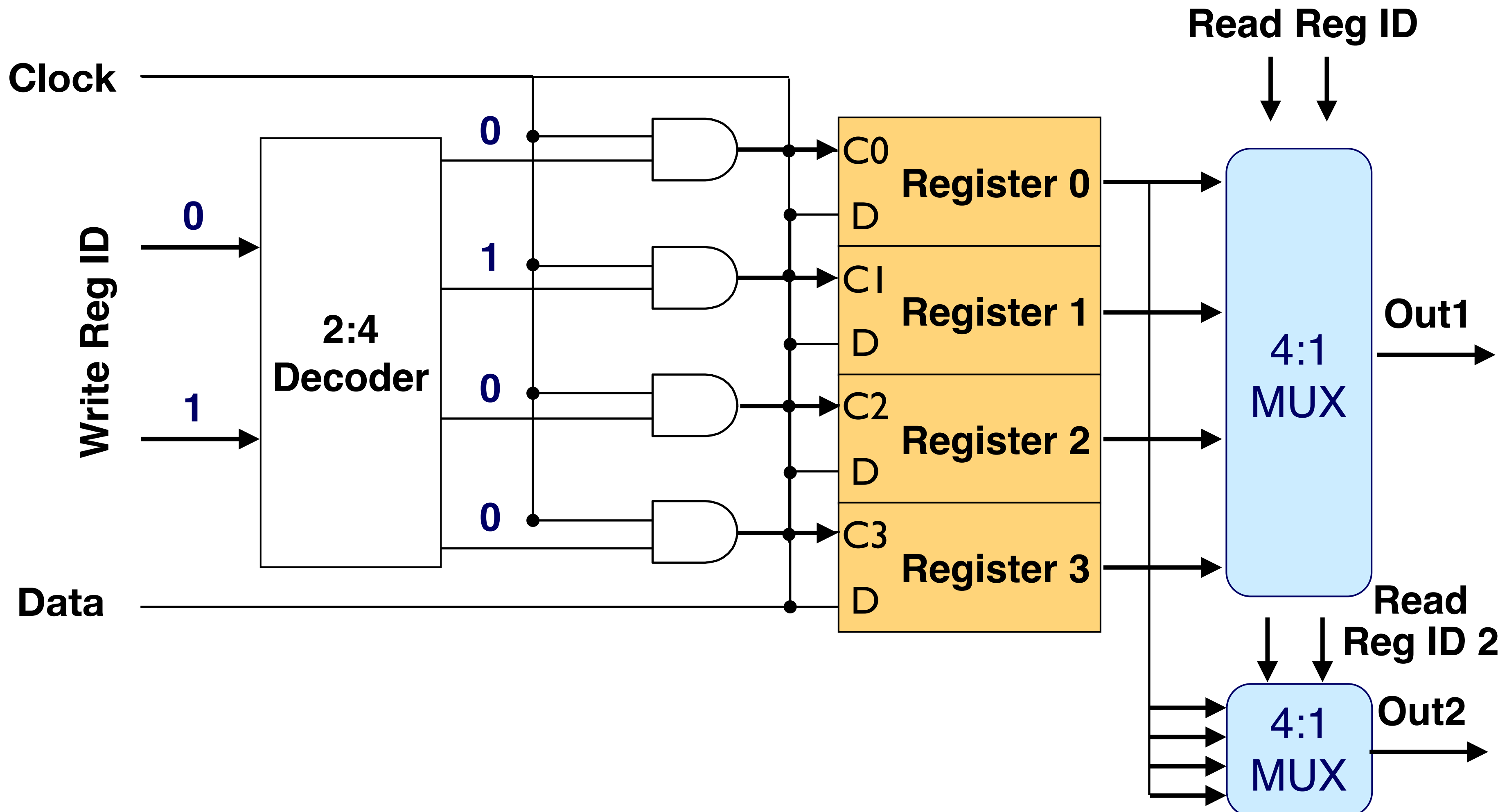
- What if we want to read multiple registers at the same time?



- This register file has 2 read ports and 1 write port. How many ports do we actually need?

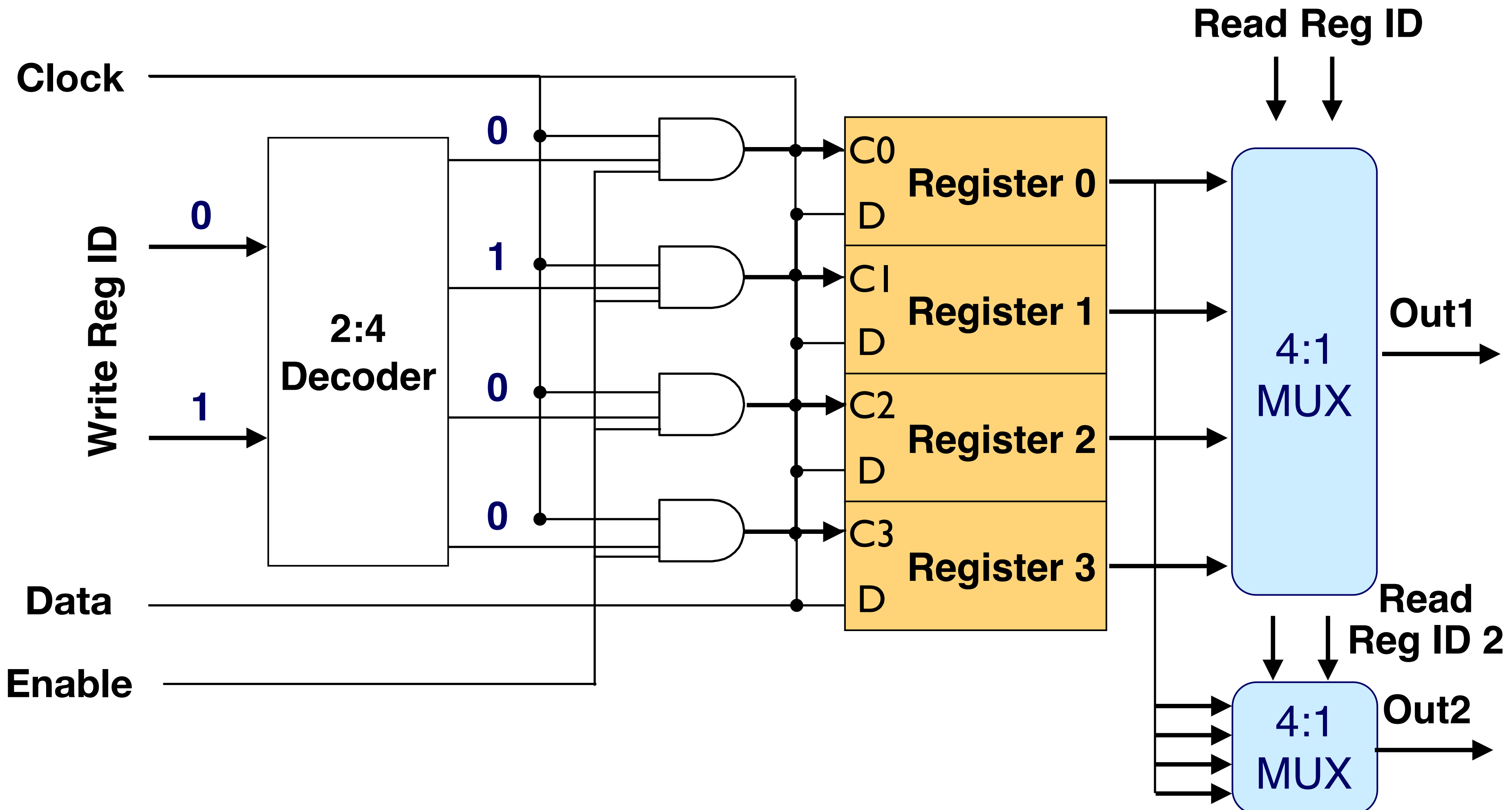
Multi-Port Register File

- Is this correct? What if we don't want to write anything?



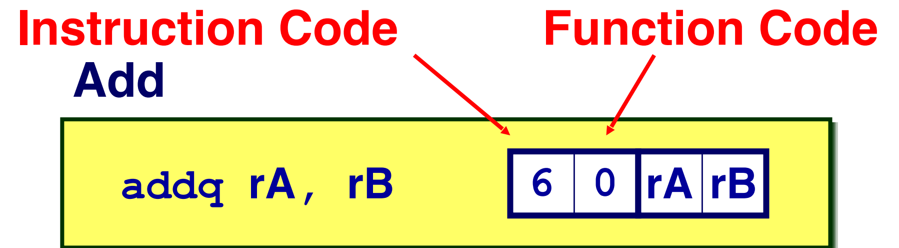
Multi-Port Register File

- Is this correct? What if we don't want to write anything?



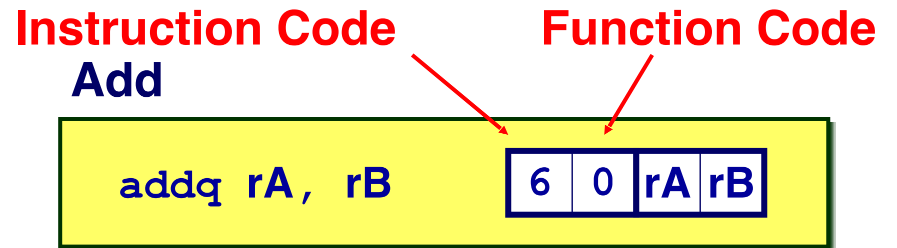
Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`



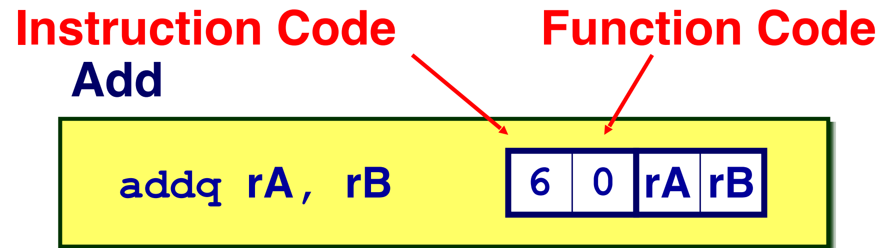
Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`



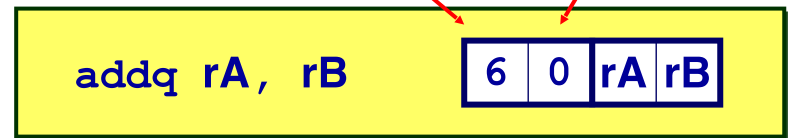
Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`



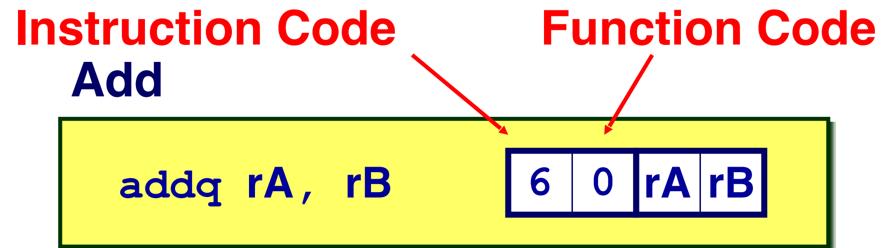
Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`



Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`



Memory (Later…)

Clock

PC

Read Reg. ID 1

Read Reg. ID 2

Register File

Reg 1 Data

ALU

Enable

Clock

Flags

Z S O

Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`

Memory (Later…)

Clock

PC

Read Reg. ID 1

Read Reg. ID 2

Register File

Reg 1 Data

Reg 2 Data

ALU

Enable

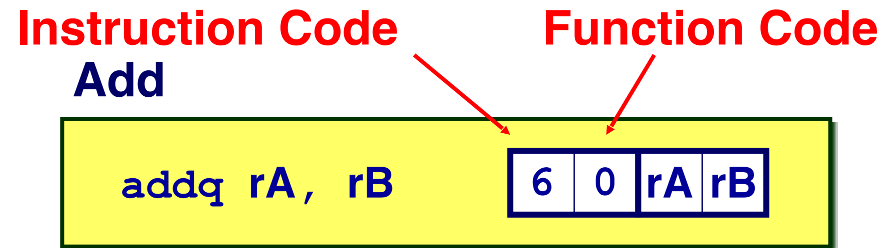
Clock

Flags

Z S O

Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`



Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`

Instruction Code
Function Code
Add

Memory (Later…)

Clock
PC

newData

Write Reg. ID
Read Reg. ID 1
Read Reg. ID 2

Register File

Reg 1 Data
Reg 2 Data

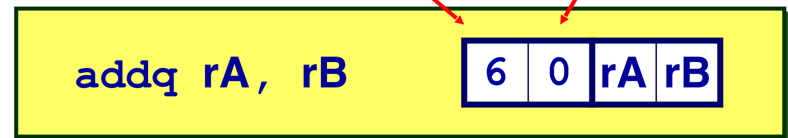
Select
ALU

Enable
Clock

Flags
Z S O

Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`



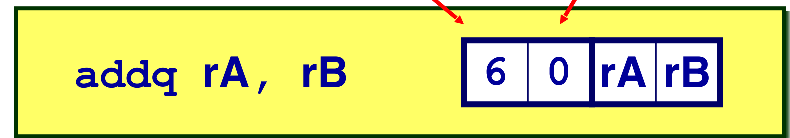
Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`

Instruction Code

Function Code

Add



Executing an ADD instruction

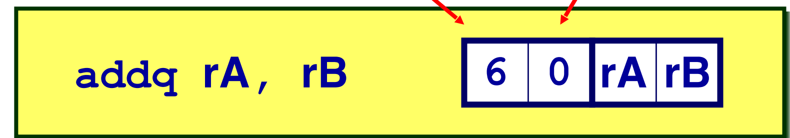
- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`

Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`

Instruction Code

Function Code



Memory (Later…)

Clock

PC

s0 6

s1 0

s2 0

s3 6

…

Write Reg. ID

Read Reg. ID 1

Read Reg. ID 2

newData

Register File

Enable

Clock

Reg 1 Data

Reg 2 Data

Select

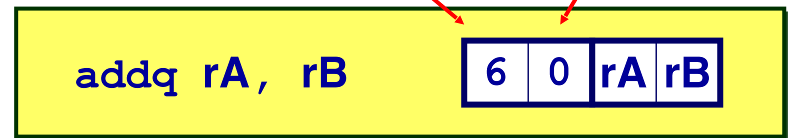
A L U

Flags

Z S O

Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`



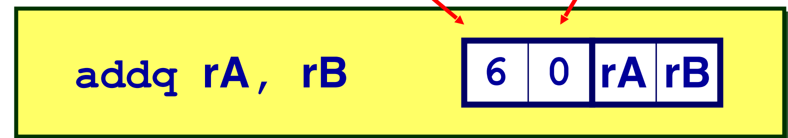
Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`

Instruction Code

Function Code

Add



Memory (Later…)

Clock

PC

s0 6
s1 0
s2 0
s3 6
…

Write Reg. ID

Read Reg. ID 1

Read Reg. ID 2

newData

Register File

Enable

Clock

Reg 1 Data

Reg 2 Data

What Logic?

Select

A L U

Flags

Z S O

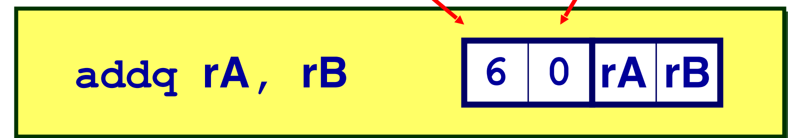
Executing an ADD instruction

- How does the processor execute `addq %rax,%rsi`
- The binary encoding is `60 06`

Instruction Code

Function Code

Add



Memory (Later…)

Clock

PC

s0 6

s1 0

s2 0

s3 6

…

newData

Write Reg. ID

Read Reg. ID 1

Read Reg. ID 2

Register File

Enable

Clock

Reg 1 Data

Reg 2 Data

What Logic?

Select

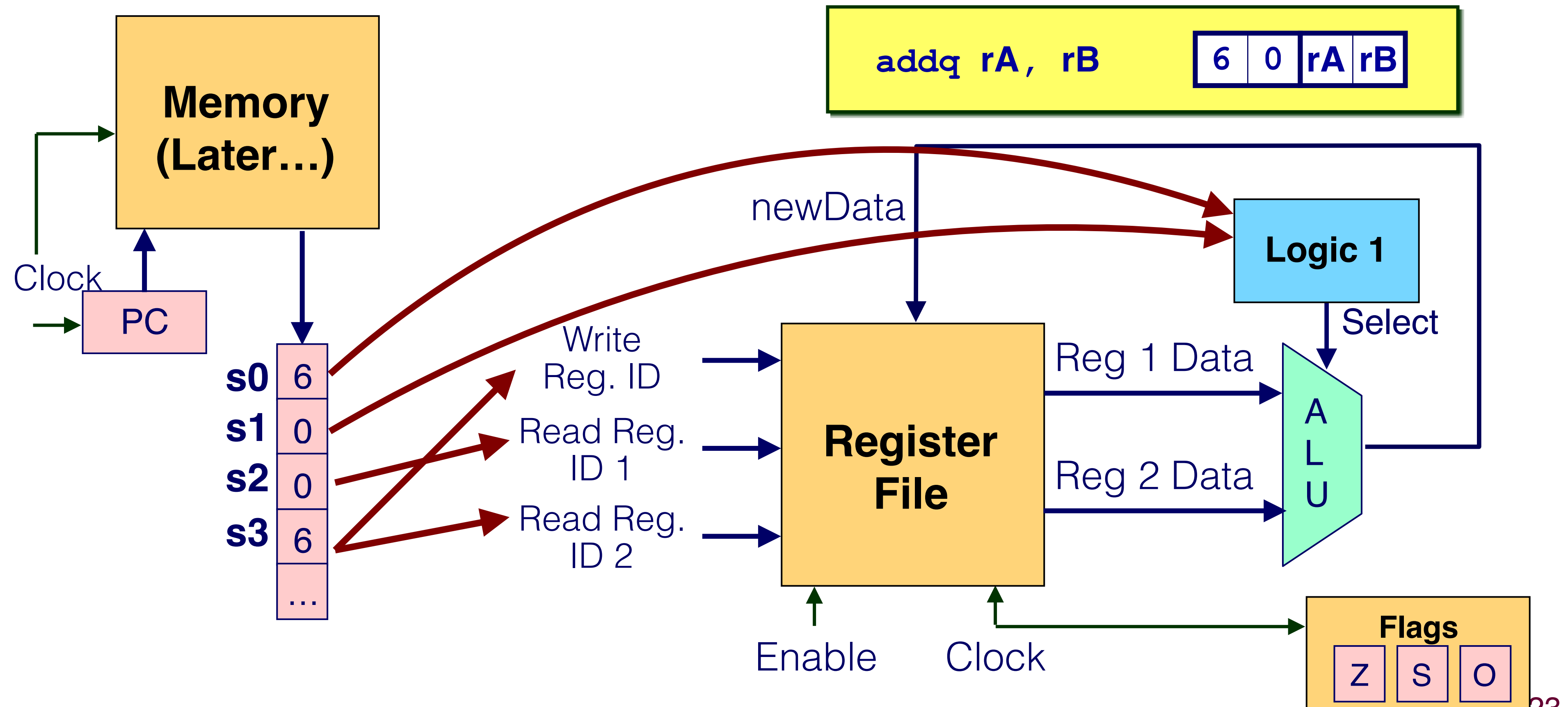
A L U

Flags

Z S O

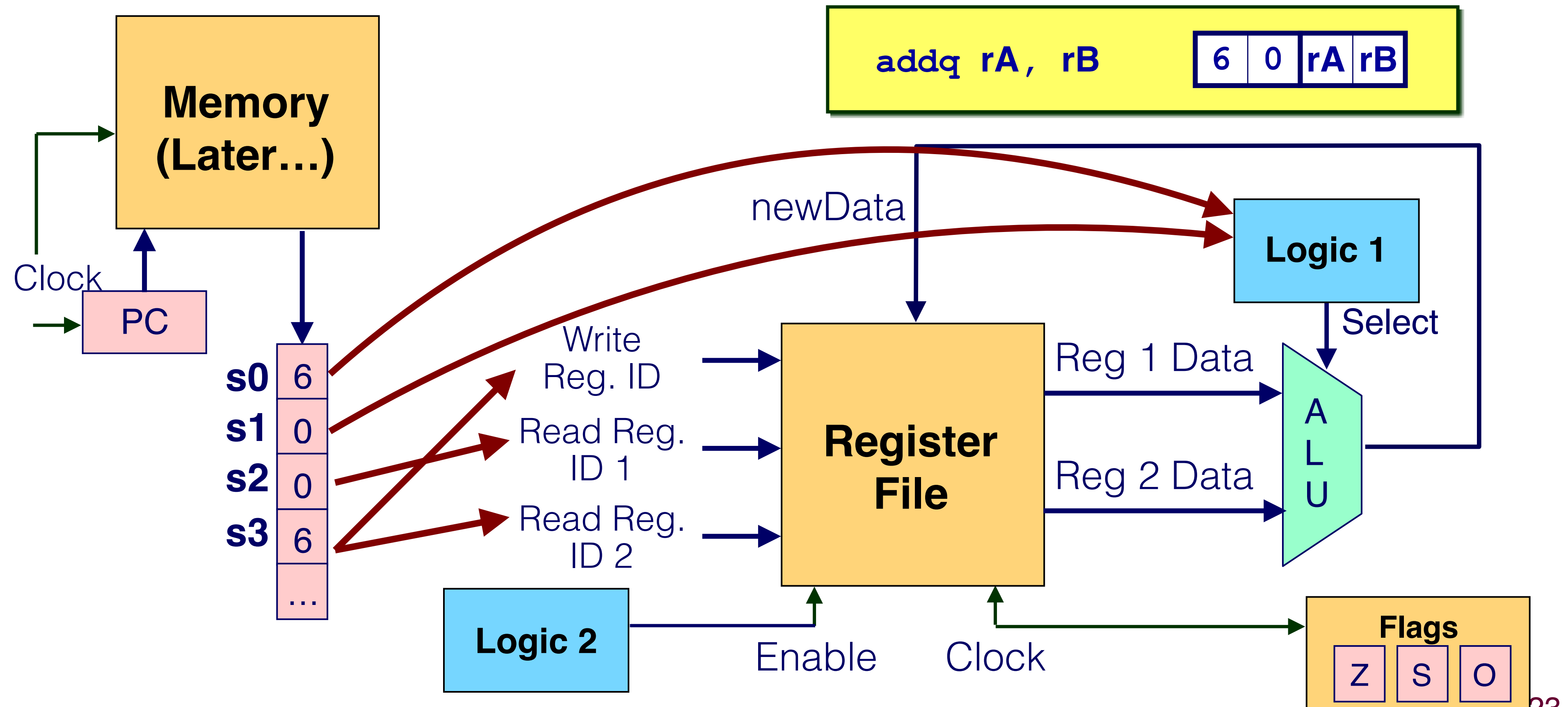
Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;



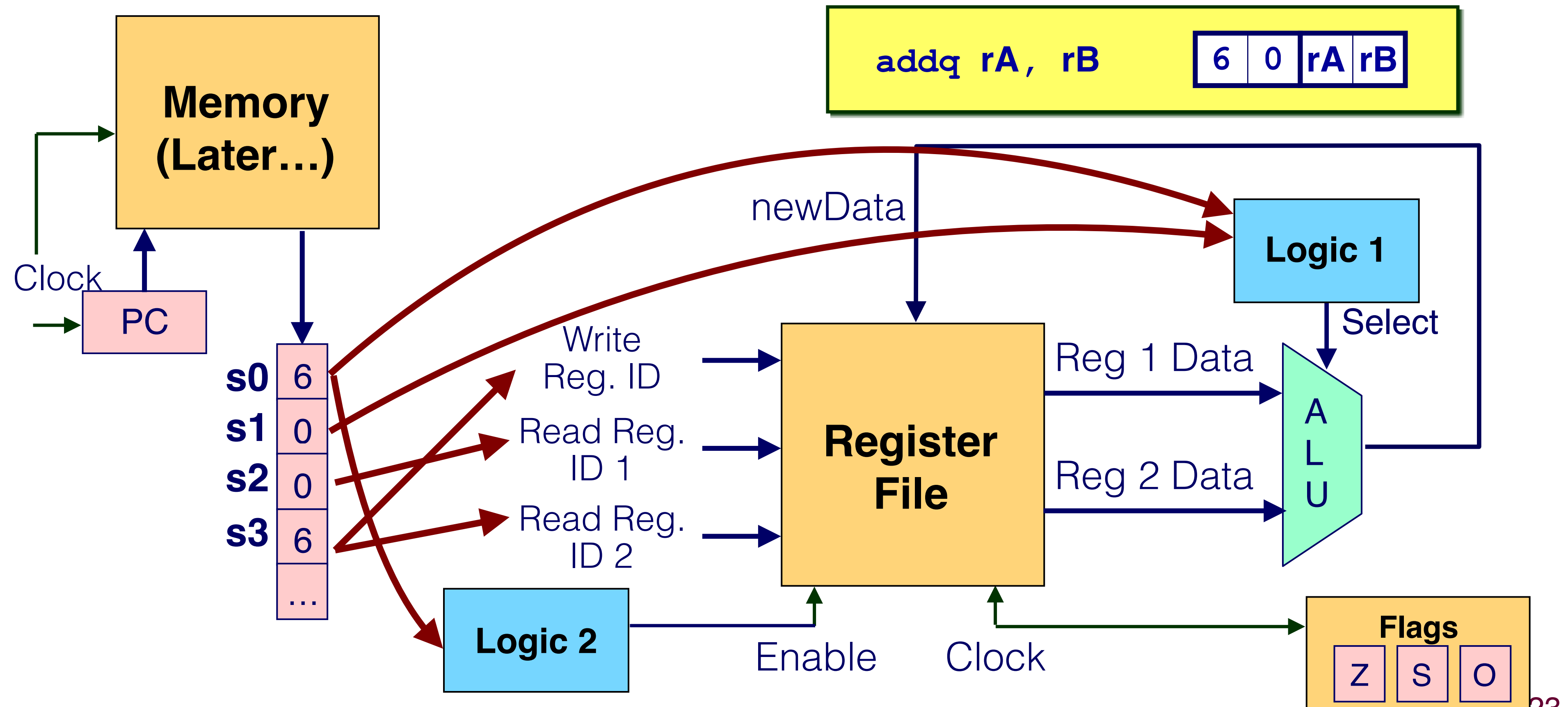
Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;



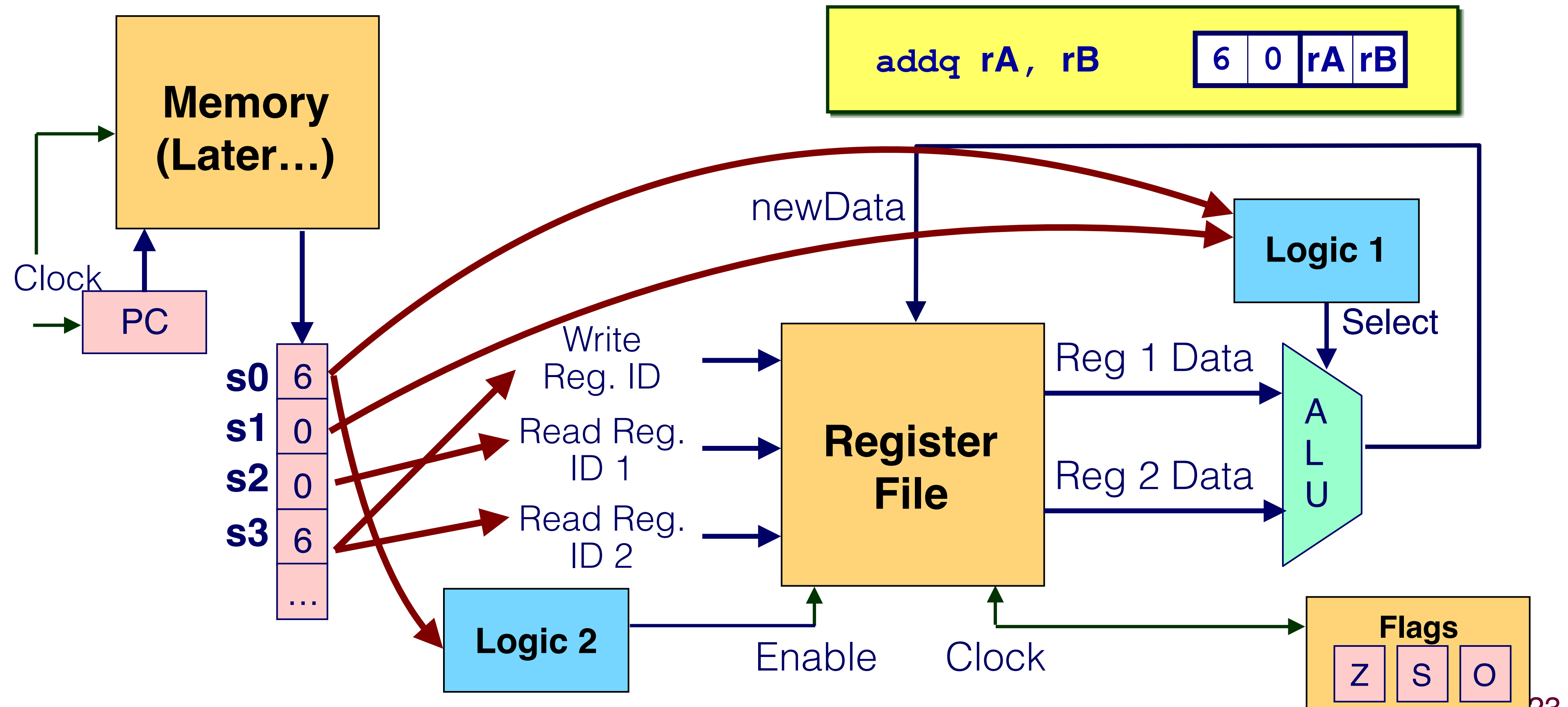
Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;



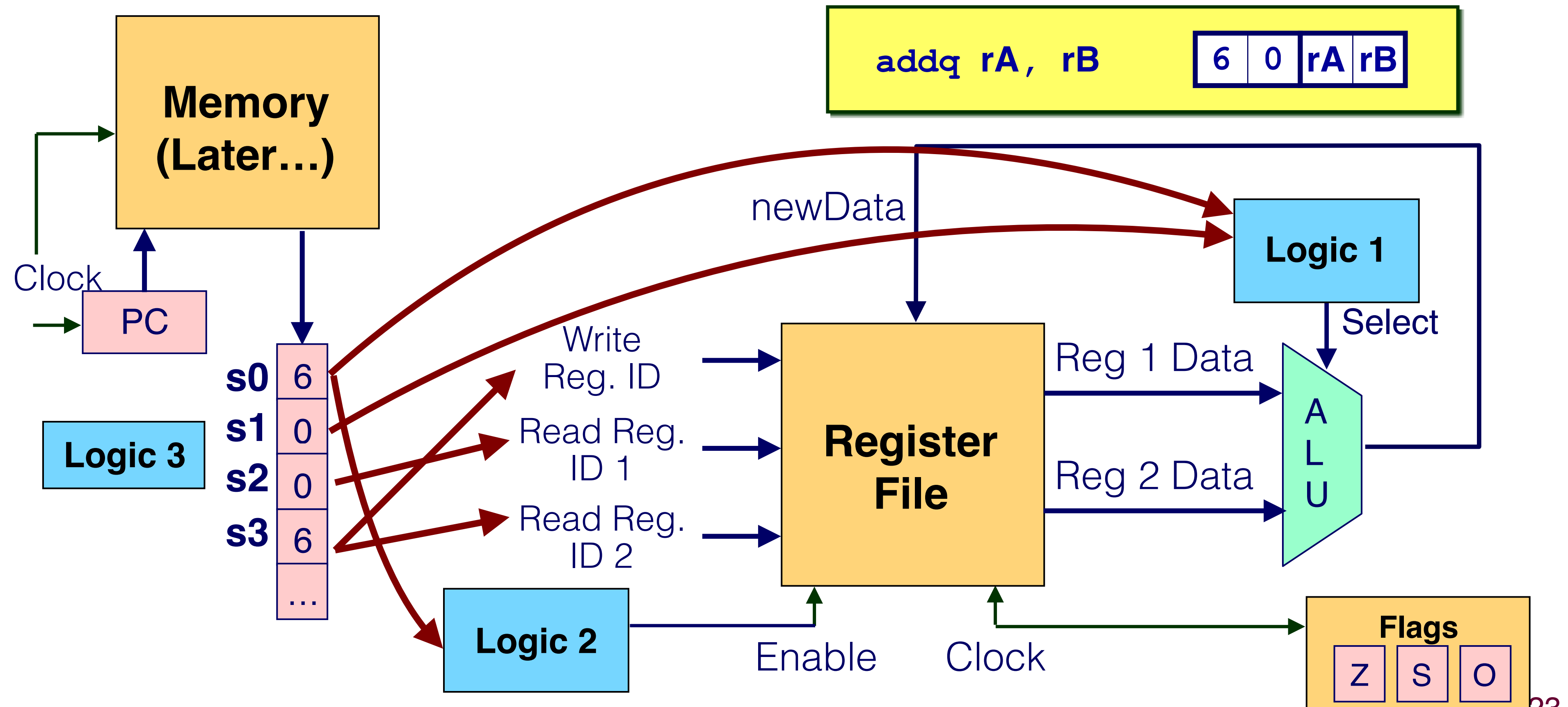
Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;
- Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;



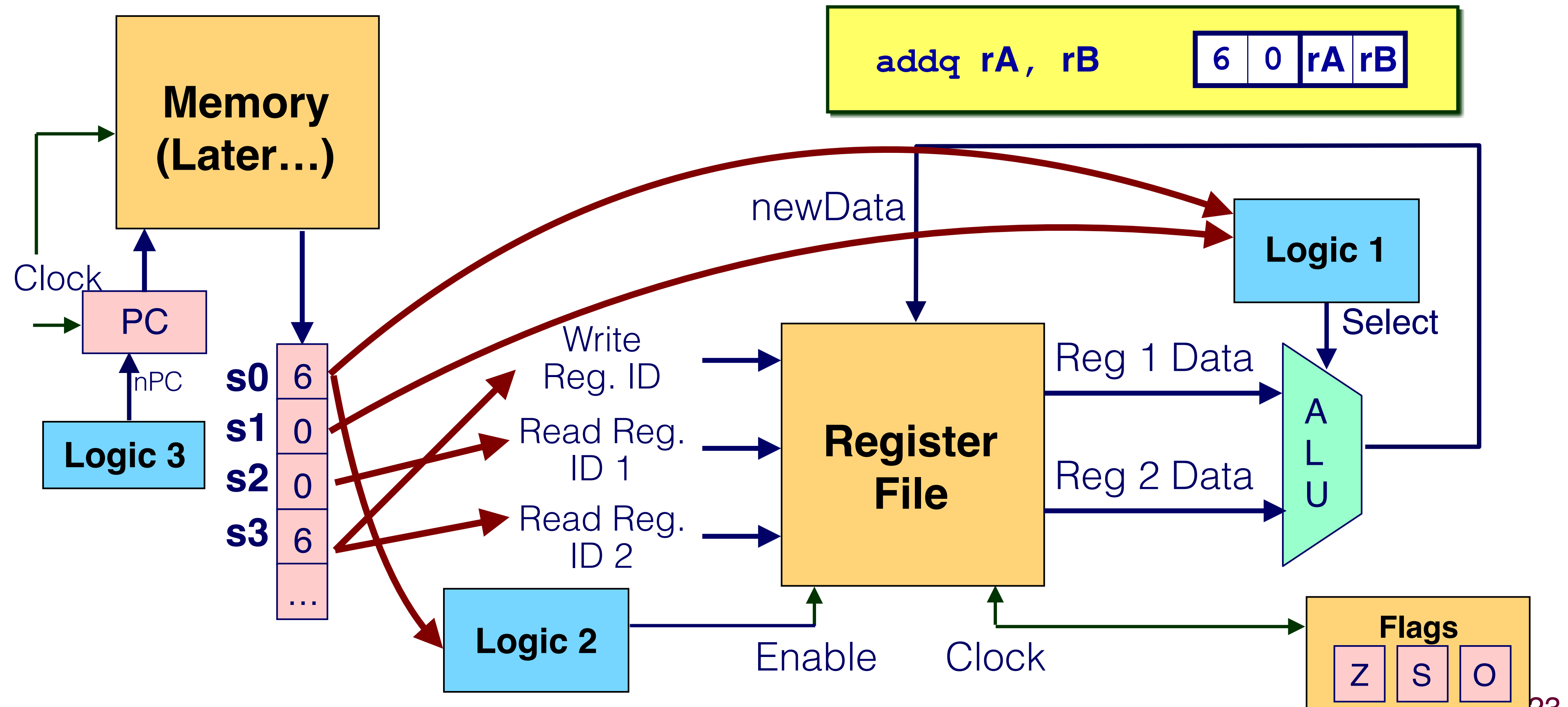
Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;
- Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;



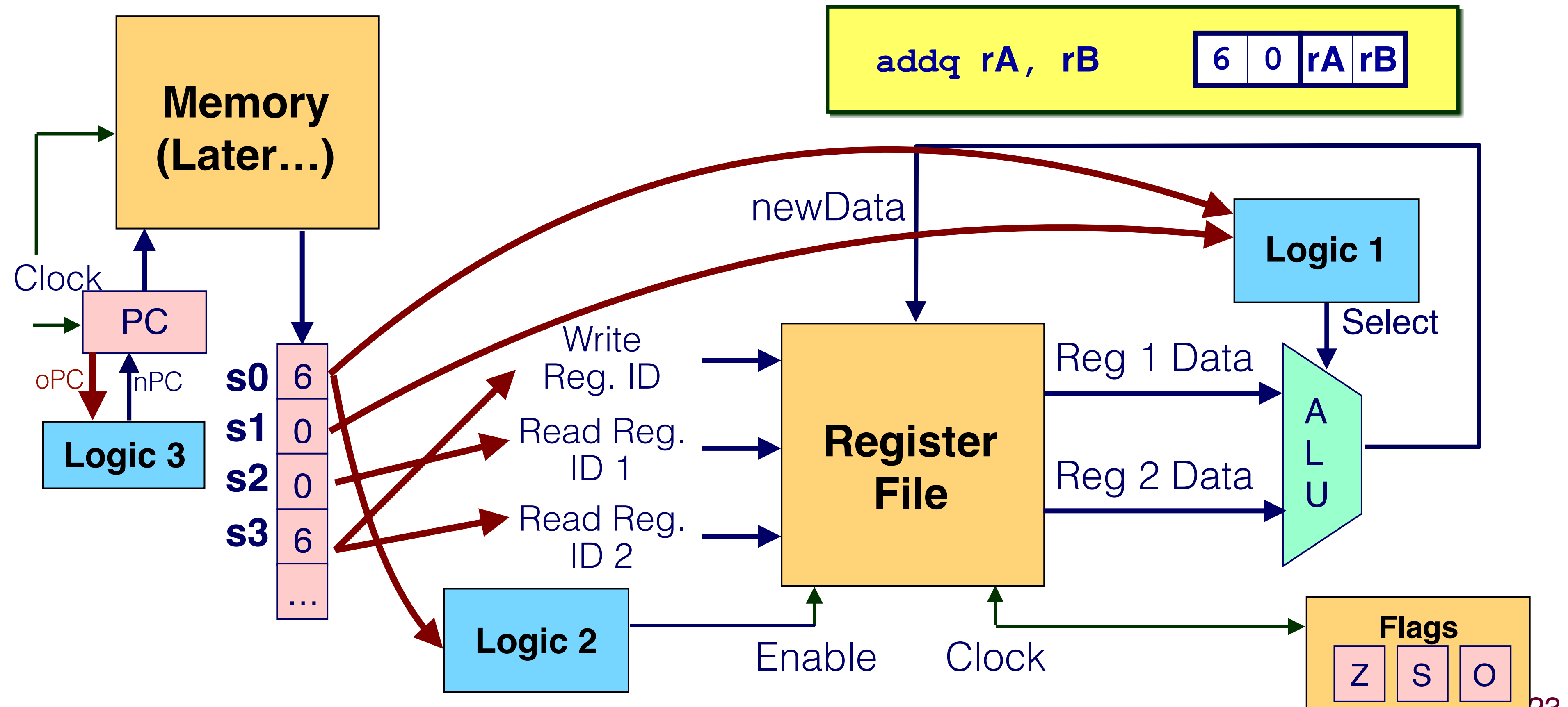
Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;
- Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;



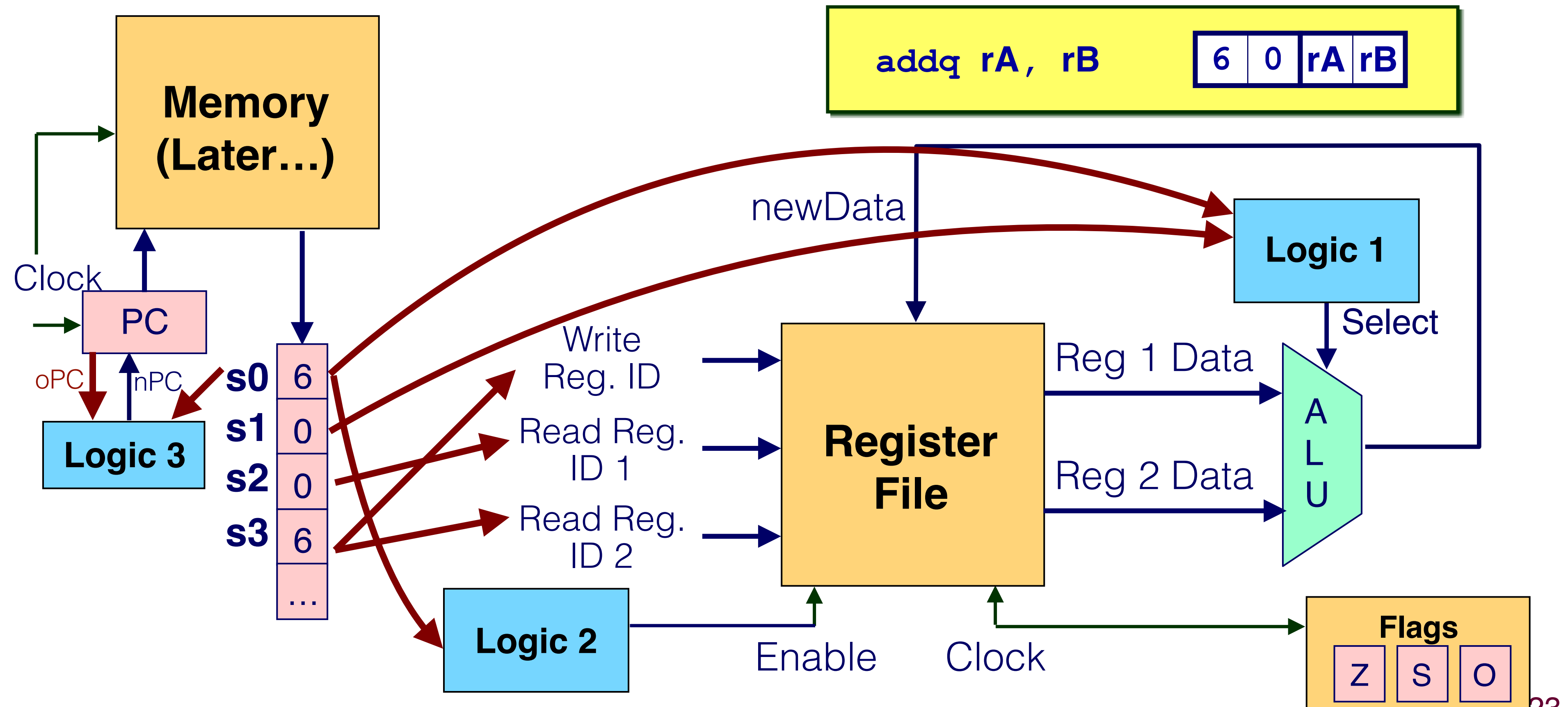
Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;
- Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;



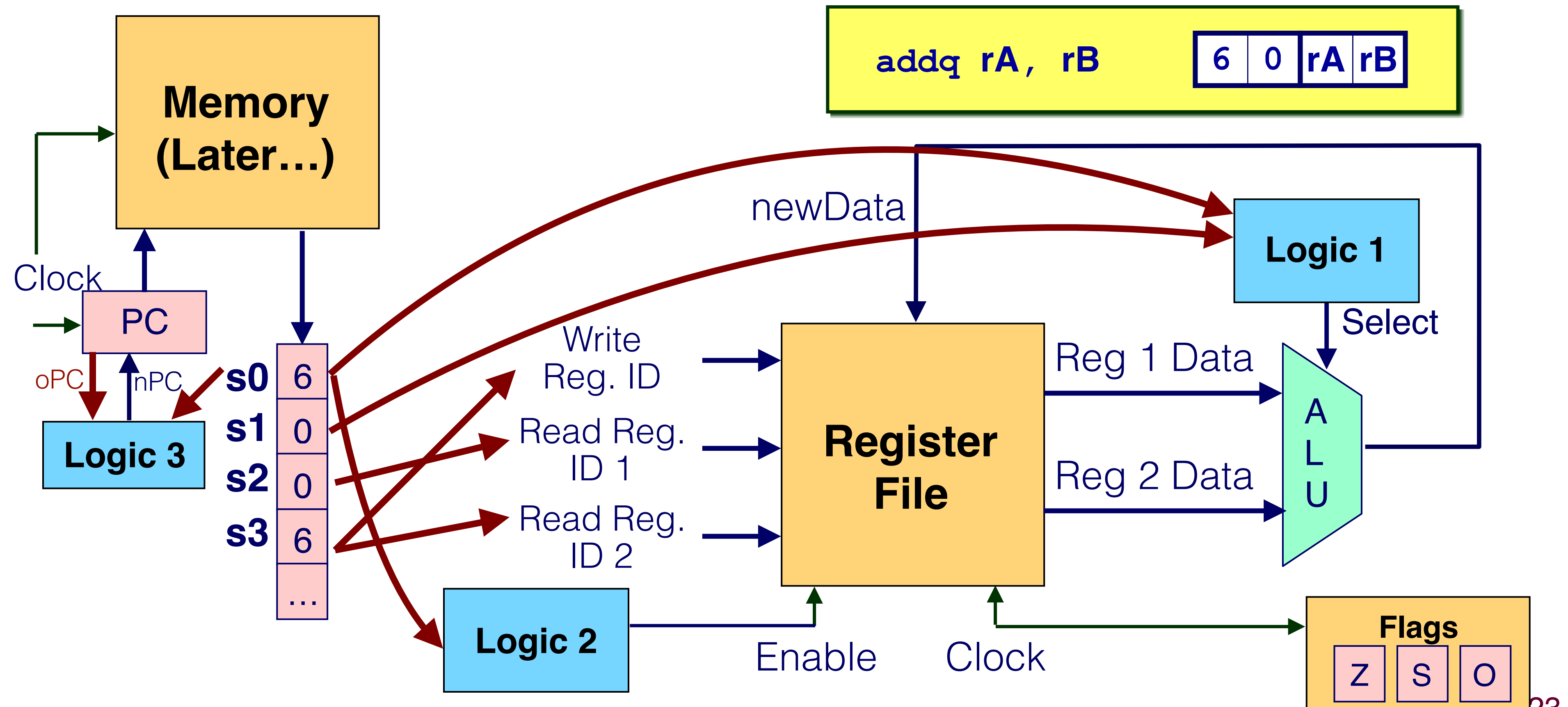
Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;
- Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;



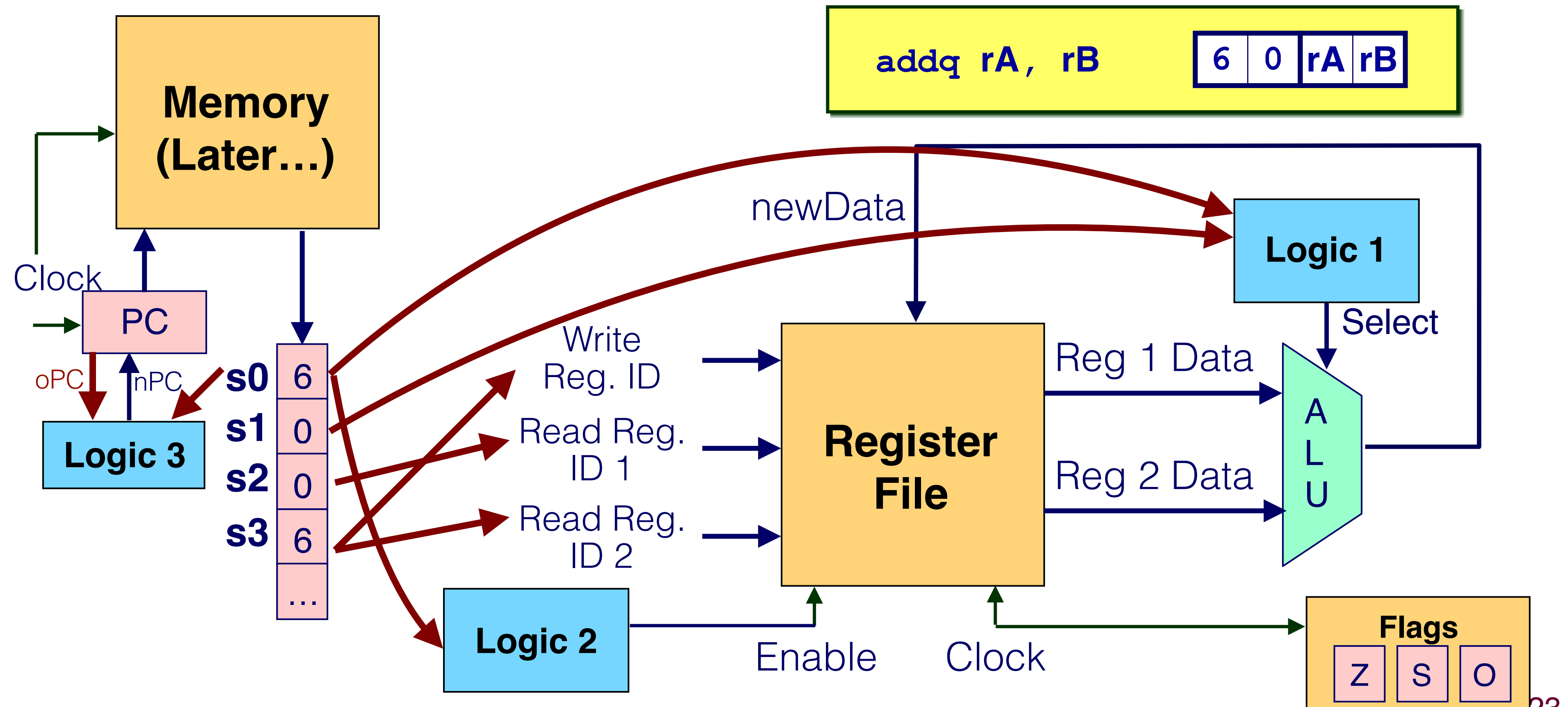
Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;
- Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;
- Logic 3: if (s0 == 6) nPC = oPC + 2;



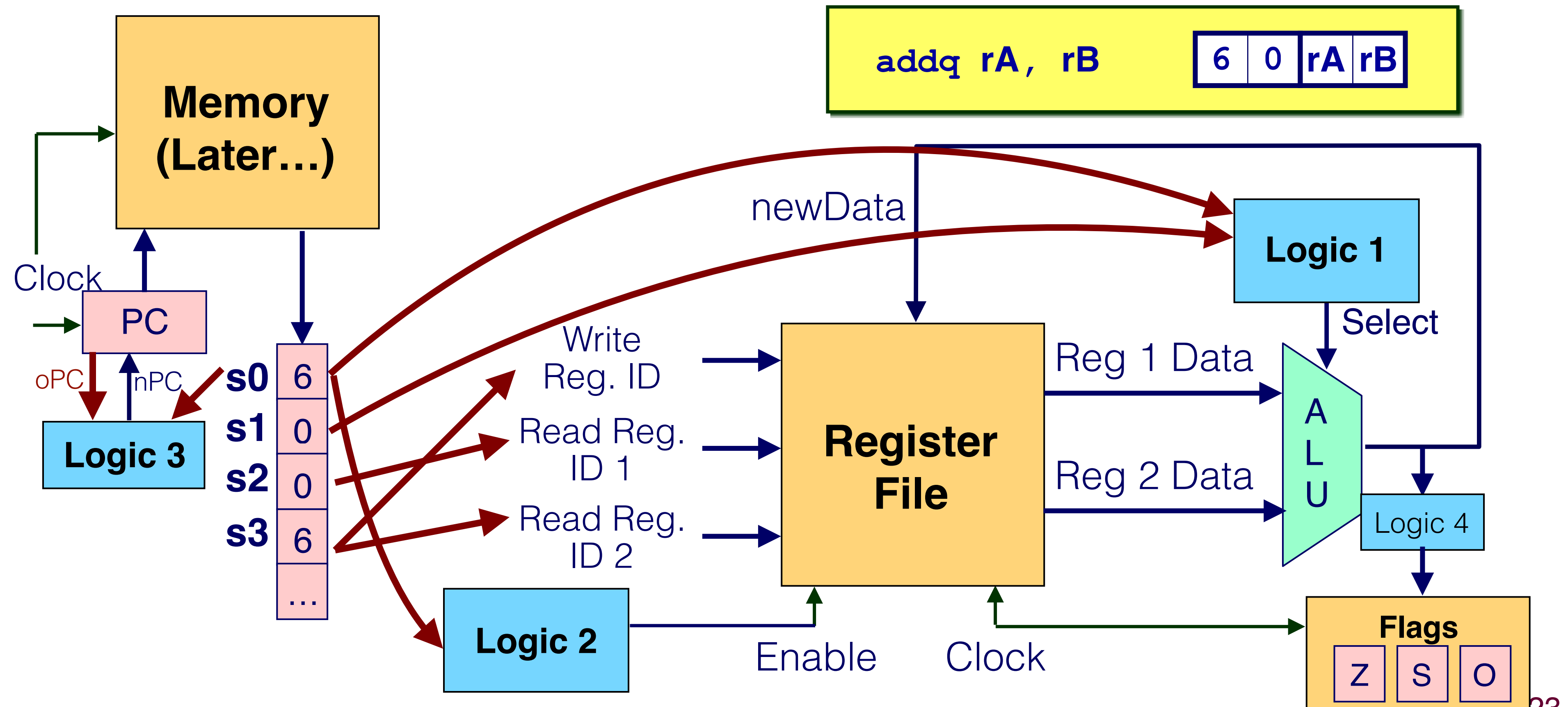
Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;
- Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;
- Logic 3: if (s0 == 6) nPC = oPC + 2;
- How about Logic 4?



Executing an ADD instruction

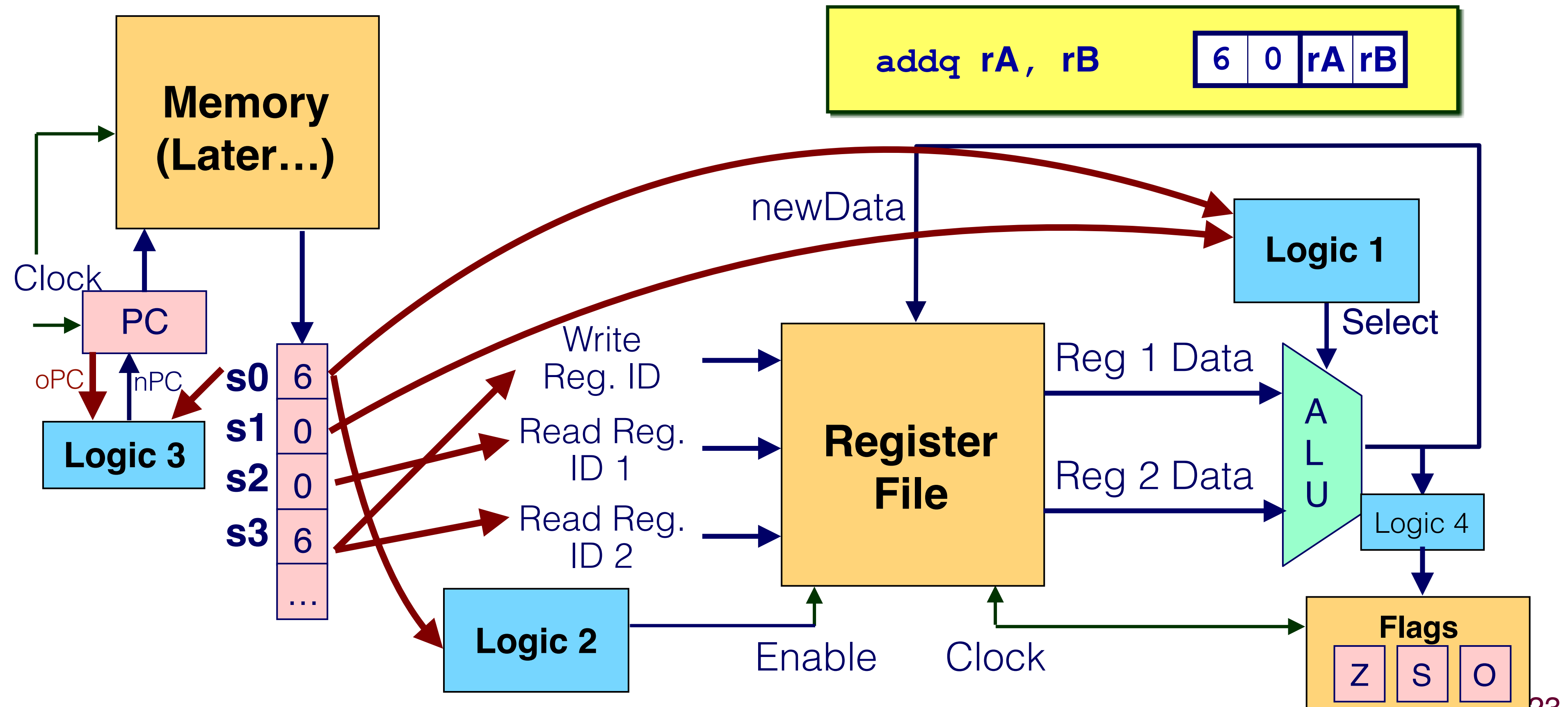
- Logic 1: if (s0 == 6) select = s1;
- Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;
- Logic 3: if (s0 == 6) nPC = oPC + 2;
- How about Logic 4?



Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;
- Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;
- Logic 3: if (s0 == 6) nPC = oPC + 2;
- How about Logic 4?

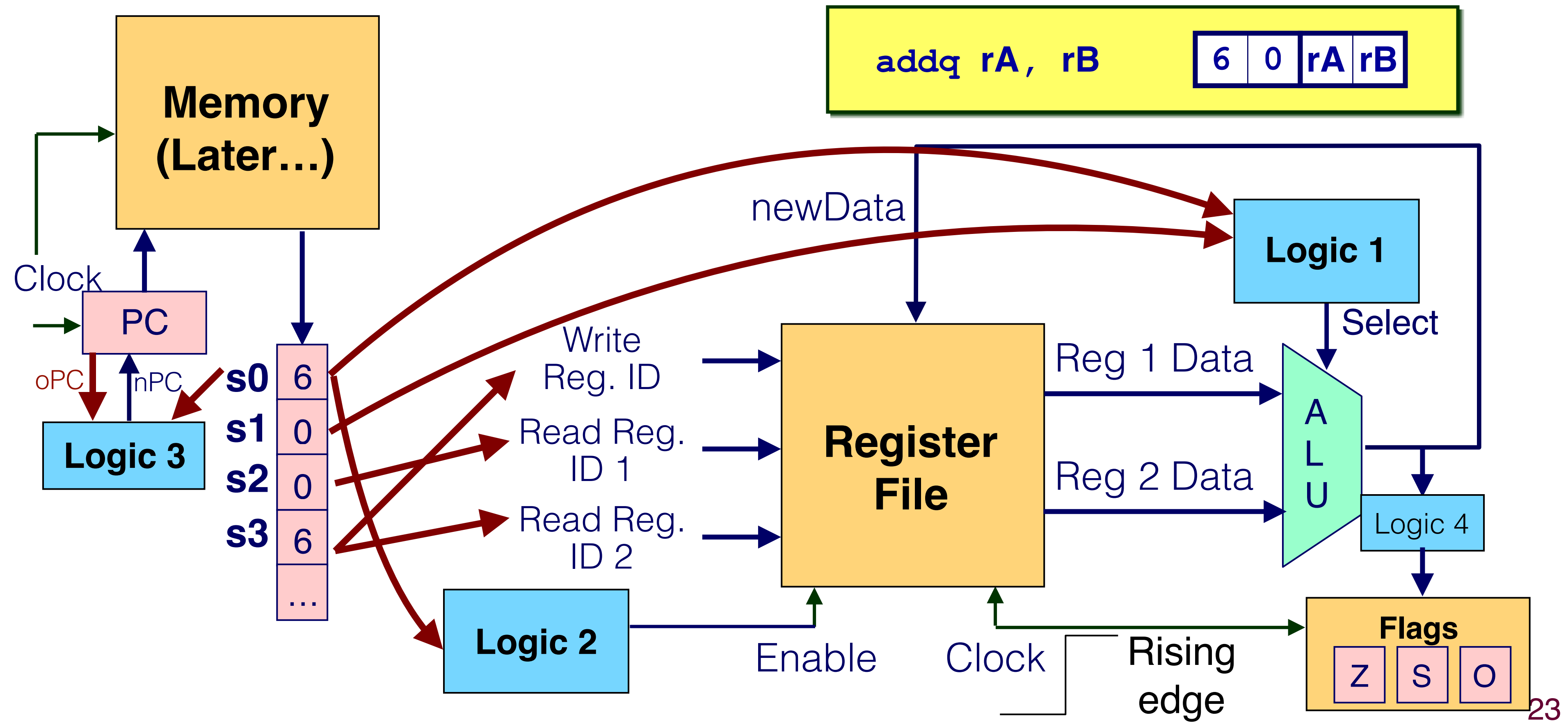
How do these logics get implemented?



Executing an ADD instruction

- Logic 1: if (s0 == 6) select = s1;
- Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;
- Logic 3: if (s0 == 6) nPC = oPC + 2;
- How about Logic 4?

How do these logics get implemented?



Executing an ADD instruction

- When the rising edge of the clock arrives, the RF/PC/Flags will be written.
- So the following has to be ready: newData, nPC, which means Logic1, Logic2, Logic3, and Logic4 has to finish.