

Sequential Logic Design

CS 64: Computer Organization and Design Logic

Lecture #14

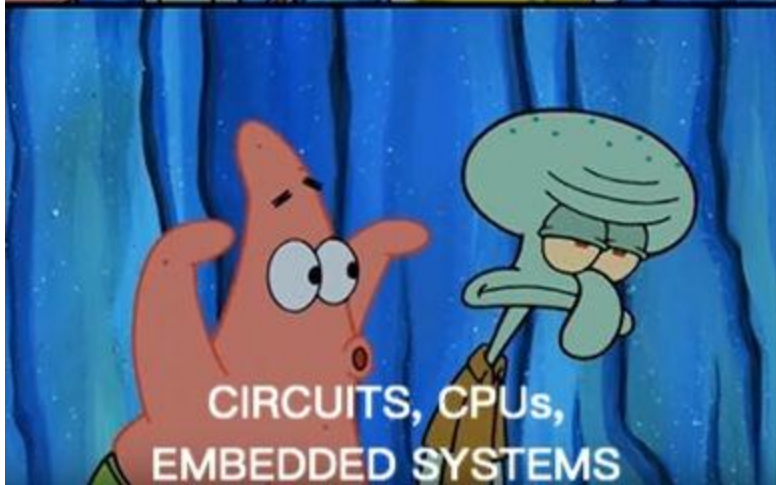
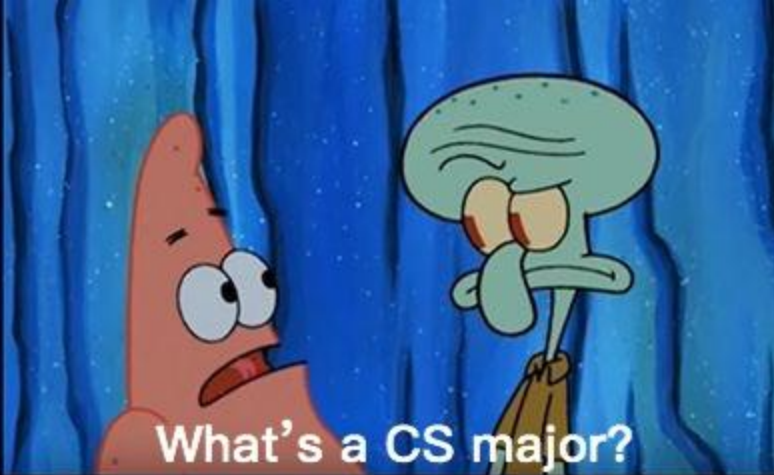
Winter 2020

Ziad Matni, Ph.D.

Dept. of Computer Science, UCSB

Administrative

- Lab 7 is due tomorrow
- New Lab 8 will be posted on Wednesday
- Final Exam Info:
 - **Tuesday, March 17th at 7:30 PM** in this classroom
 - Arrive 10 mins early – randomized seating...
 - Cumulative Exam
 - Study guide/example Qs will be issued by this weekend
 - More details to follow

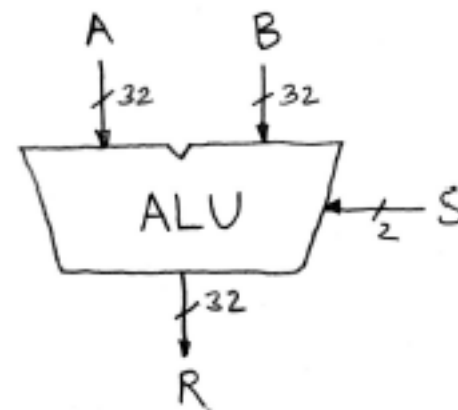


Lecture Outline

- **S-R Latch**
- Gated S-R Latch
- Gated D-Latch
- The D-Flip Flop (D-FF)

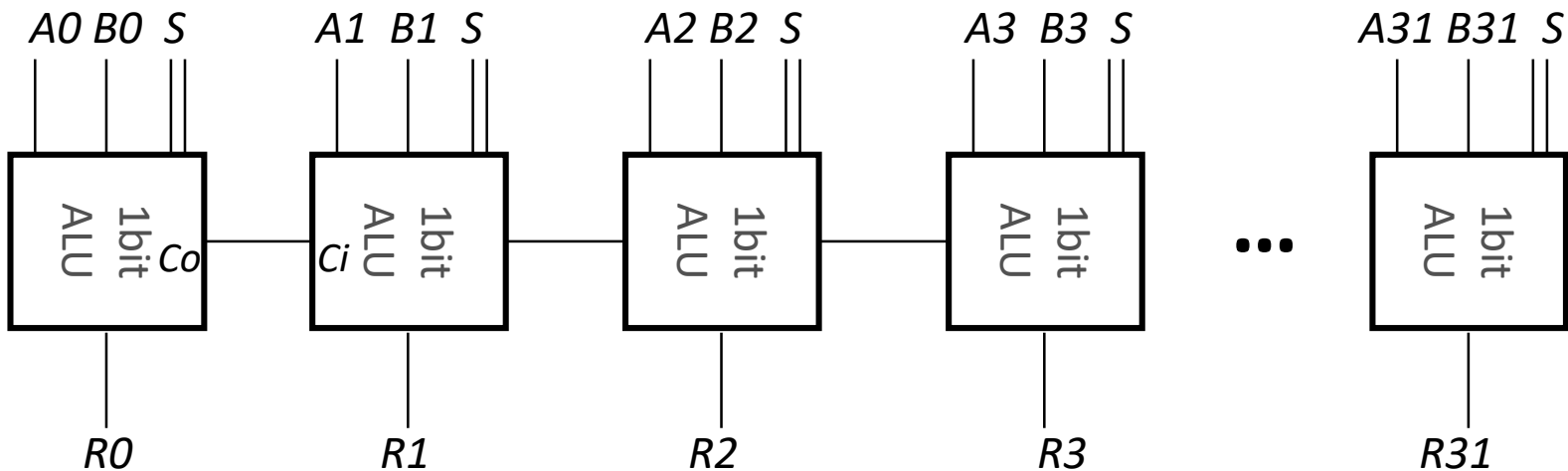
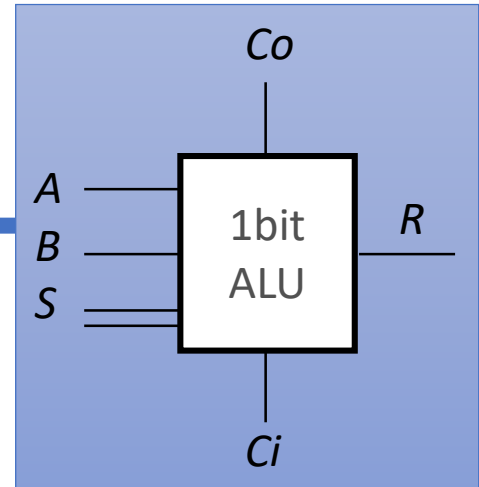
Arithmetic-Logic Unit (ALU)

- Recall: the ALU does all the computations necessary in a CPU
- The previous circuit was a simplified ALU:
 - When $S = 00$, $R = A + B$
 - When $S = 01$, $R = A - B$
 - When $S = 10$, $R = A \text{ AND } B$
 - When $S = 11$, $R = A \text{ OR } B$



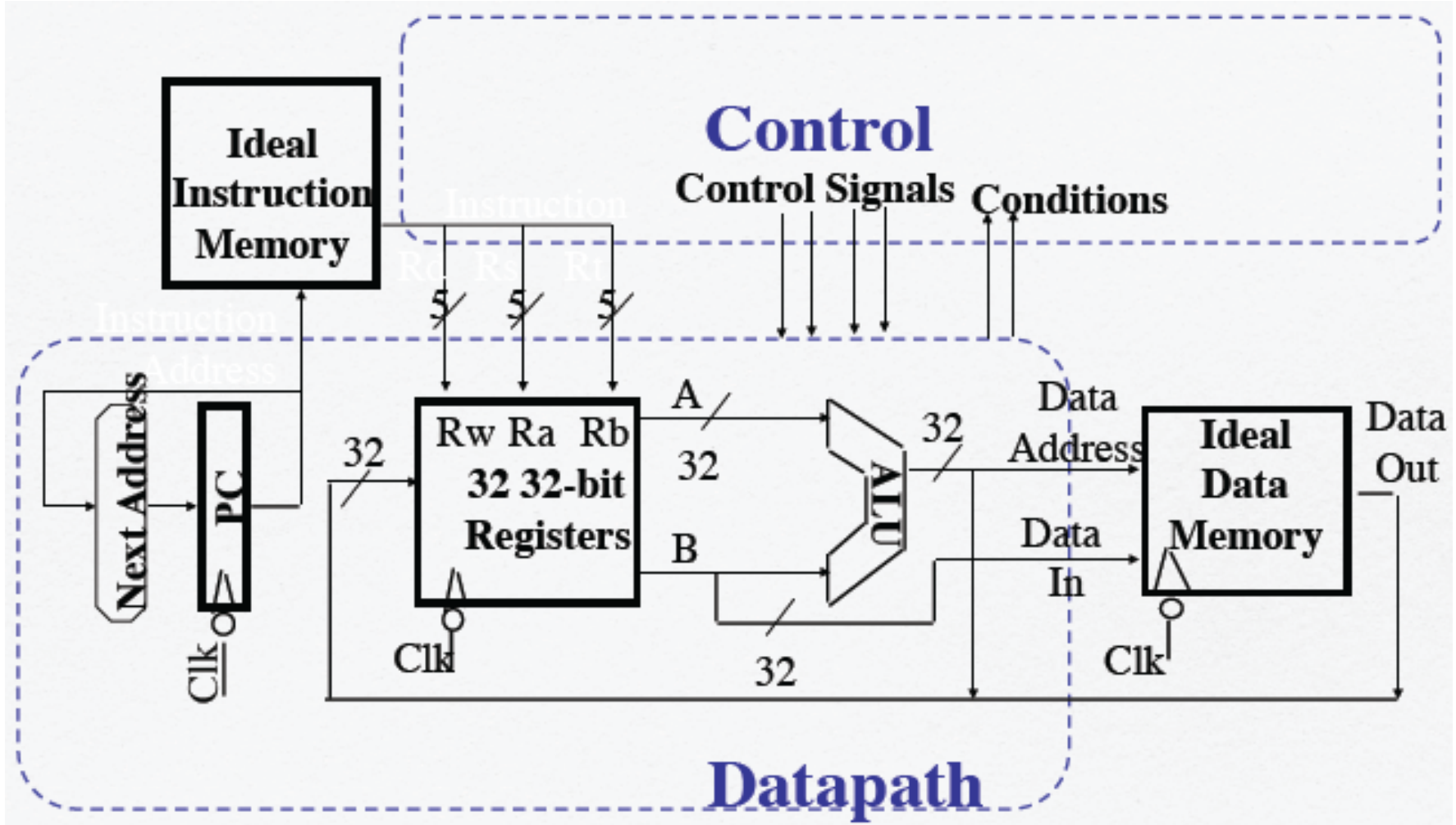
Simplified ALU

- We can string 1-bit ALUs together to make bigger-bit ALUs (e.g. 32b ALU)



Abstract Schematic of the MIPS CPU

Relevant to a future lab...

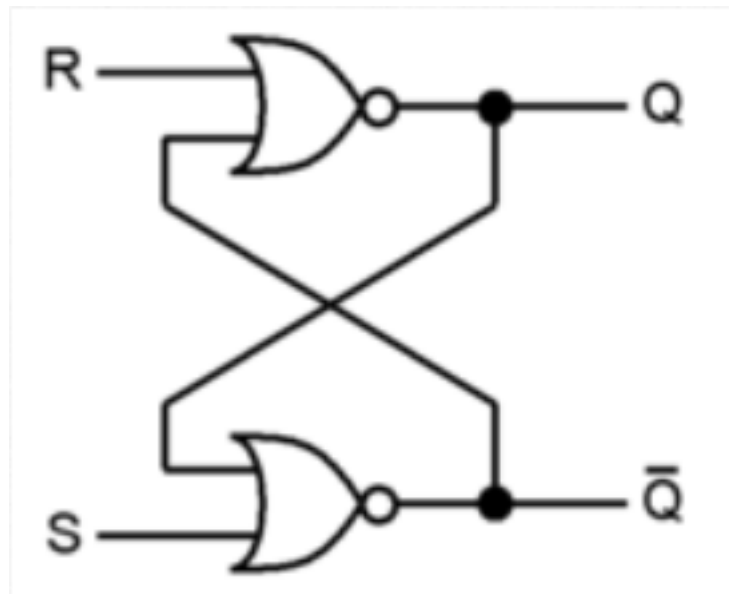


The Wonderfully Weird World Of...

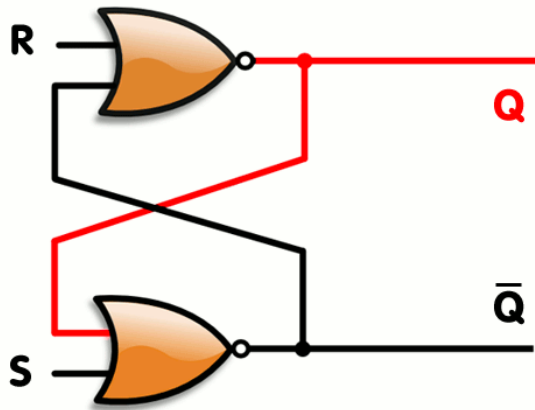
Latches

The S-R Latch

- Only involves 2 NORs
- The outputs are ***fed-back*** to the inputs
- The result is that the output state (either a 1 or a 0) is ***maintained*** *even if the input changes!*



How a S-R Latch Works



- Note that if one NOR input is **0**, the output becomes the inverse of the other input
- So, if output Q already exists and if $S = 0, R = 0$, then Q will remain at whatever it was before! (**hold output state**)

- If $S = 0, R = 1$, then Q becomes 0 (**reset output**)

- If $S = 1, R = 0$, then Q becomes 1 (**set output**)

- Making $S = 1, R = 1$ is not allowed (**gives an undetermined output**)

S	R	Q_0	Comment
0	0	Q^*	Hold output
0	1	0	Reset output
1	0	1	Set output
1	1	X	Undetermined

Consequences?

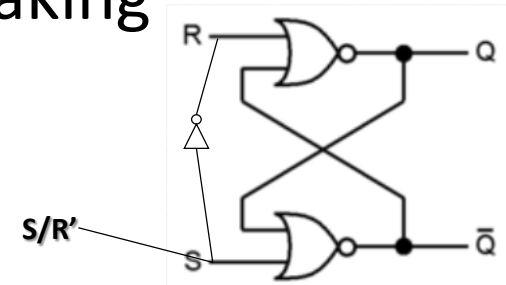
- As long as **S = 0** and **R = 0**, the circuit output holds memory of its prior value (state)
- To change the output, just make
S = 1 (but also R = 0) to make the output 1 (set) **OR**
S = 0 (but also R = 1) to make the output 0 (reset)
- Just avoid S = 1, R = 1...

S	R	Q ₀	Comment
0	0	Q*	Hold output
0	1	0	Reset output
1	0	1	Set output
1	1	X	Undetermined

About that $S = 1, R = 1$ Case...

S	R	Q_0	Comment
0	0	Q^*	Hold output
0	1	0	Reset output
1	0	1	Set output
1	1	X	Undetermined

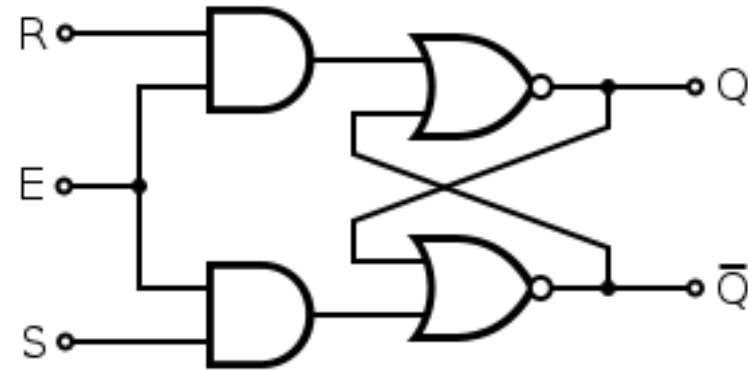
- What if we avoided it on purpose by making $R = \text{NOT}(S)$?
 - Where's the problem?



- This, by itself, precludes a case when $R = S = 0$
 - **You will need that** if you want to preserve the previous output state!
 - So this isn't the best solution...
- Actual Solution: the ***clocked latch***
and ***the flip-flop (aka the register)***

Adding an “Enable” Input: The Gated S-R Latch

- Create a way to “gate” the inputs
 - R/S inputs go through *only if* an “enable input” (E) is 1
 - If E is 0, then the S-R latch gets $SR = 00$ and it hold the state of previous outputs



- So, the truth table would change from a “normal” S-R Latch:

S	R	Q_0	Comment
0	0	Q^*	Hold output
0	1	0	Reset output
1	0	1	Set output
1	1	X	Undetermined

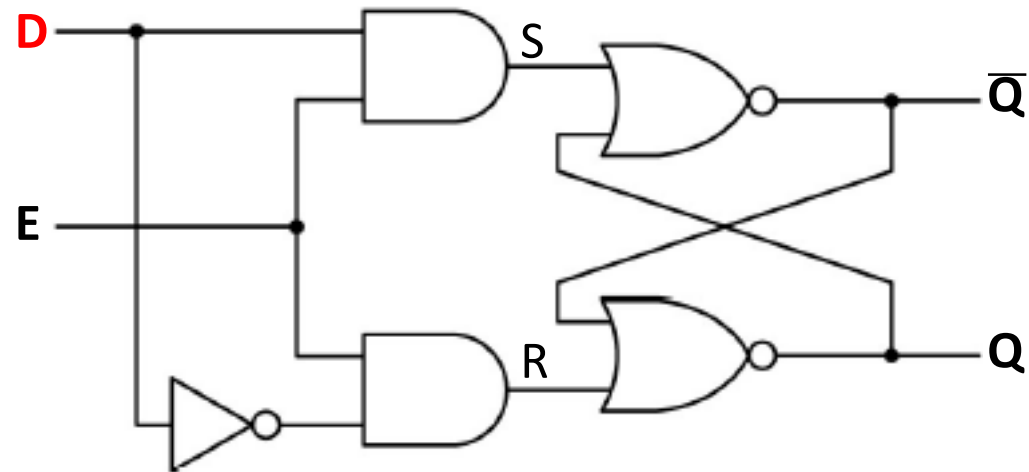


S	R	E	Q_0	Comment
X	X	0	Q^*	Hold output
0	1	1	0	Reset output
1	0	1	1	Set output

We got rid of the “undetermined” state!!! 😊😊😊

Combining R and S inputs into One: The Gated D Latch

- Force S and R inputs to *always be opposite of each other*
 - Make them the same as an input D, where $D = S$ and $\neg D = R$.



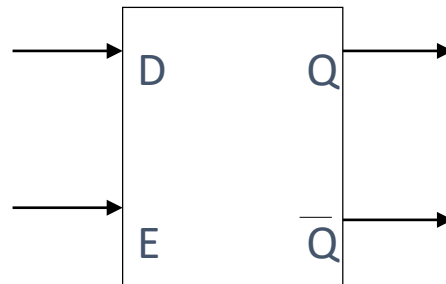
- Create a way to “gate” the D input
 - D input goes through only if an enable input (E) is 1
 - If E is 0, then hold the state of the previous outputs

D	E	Q ₀	Comment
X	0	Q*	Hold output
0	1	0	Reset output
1	1	1	Set output

We got rid of an extra input!!! 😊😊😊

The Gated D Latch

- The gated D-Latch is very commonly used in electronic circuits in computer hardware, especially as a register because it's a circuit that holds memory!



Whatever data you present to the input D,

the D-Latch will **hold** that value (*as long as input E is 0*)

You can **present** this value to output Q *as soon as input E is 1*.

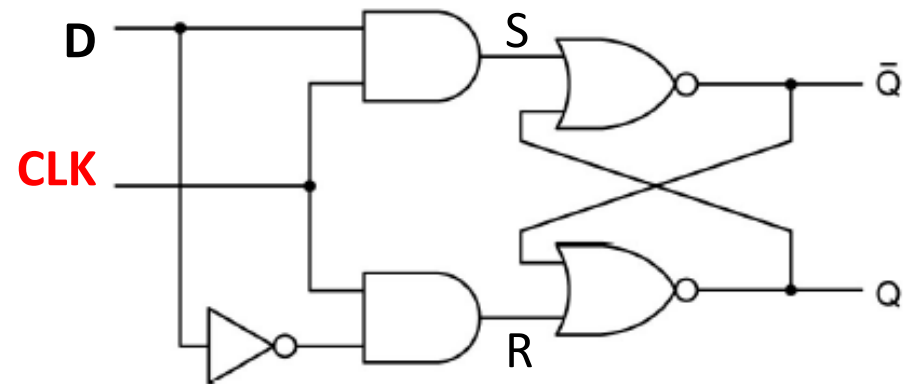
Enabling the Latch Synchronously: The Clocked D Latch

- If you apply a **synchronous clock** on input E, you get a **clocked D latch**.

- A clock is an input that cycles from 1 to 0, then back to 1 again in a set time period

- e.g.: if a clock input cycles this in a period of 1 ms, we call it a 1 MHz clock (1 Hz = 1 / 1 second)

- **Note 1:** When CLK is 0, both S and R inputs to the latch are 0 too, so the Q output holds its value whatever it is ($Q = Q_0$)



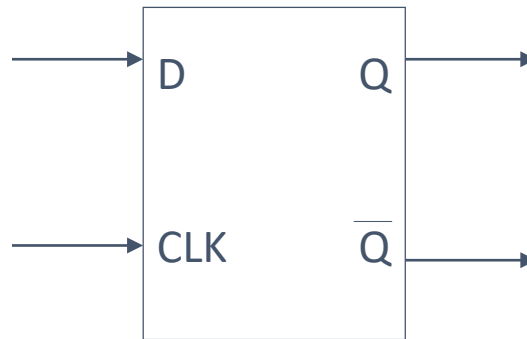
- **Note 2:** When CLK is 1:
if $D = 1$, then $Q = 1$,
if $D = 0$, then $Q = 0$

Truth table

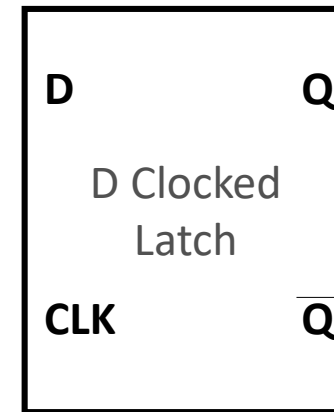
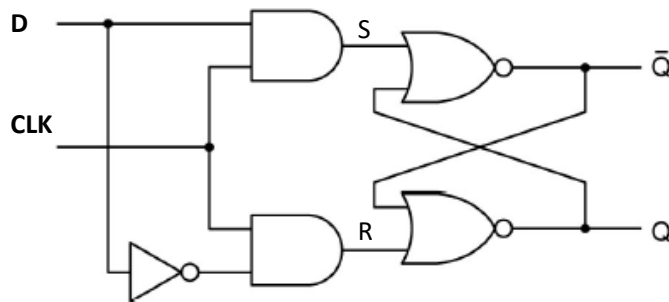
D	CK	Q
0	1	0
1	1	1
X	0	Q_0

Clocked D Latch as Digital Sampler

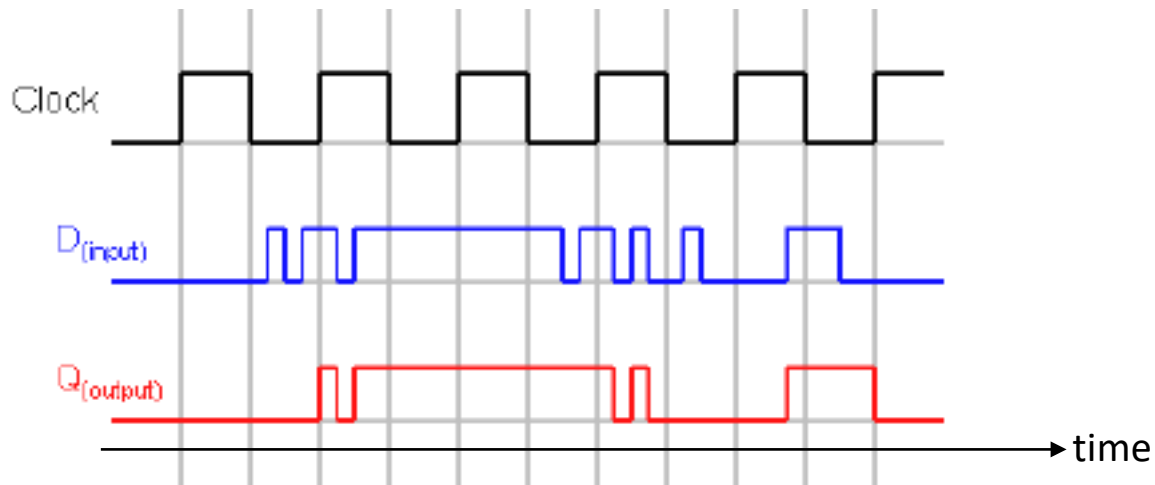
- This clocked latch can be used as a “programmable” memory device that “samples” an input on a regular basis



The Clocked D Latch By Any Other Name...



- Observing input and output “waveforms”

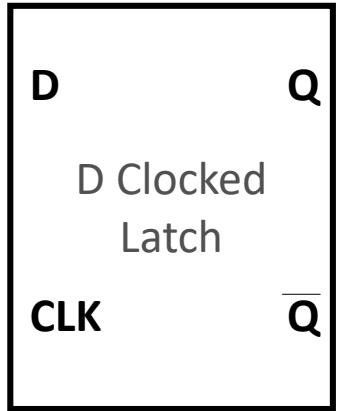
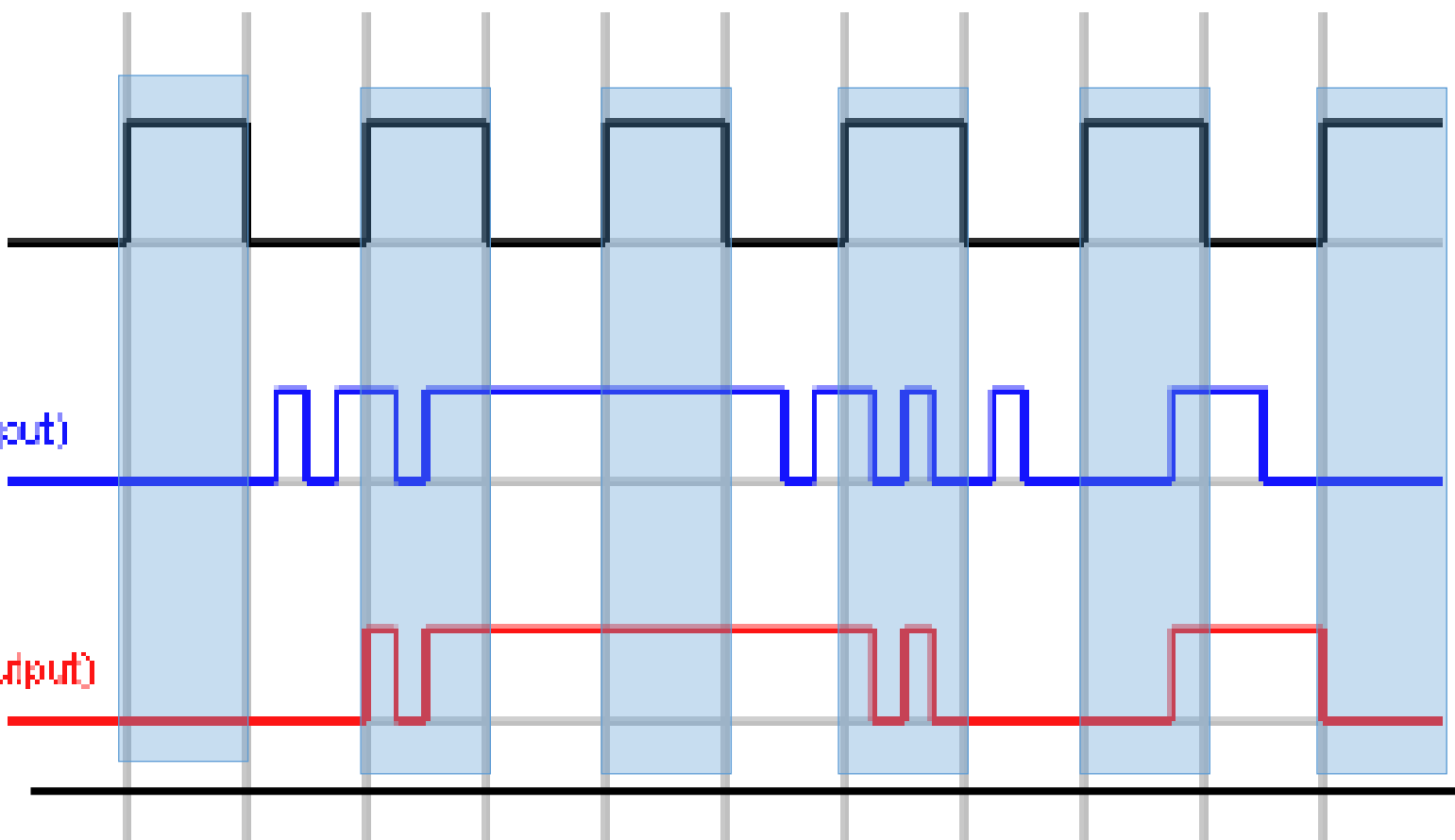


Clock

D (input)

Q (output)

time



The Joys of Sampling...

- Sampling data in a periodic way is advantageous
 - I can start designing more complex circuits that can help me do *synchronous* logical functions
 - *Synchronous*: in-time
- Very useful in *pipelining* designs used in CPUs
 - Pipelining: a technique that allows CPUs to execute instructions more efficiently – in parallel

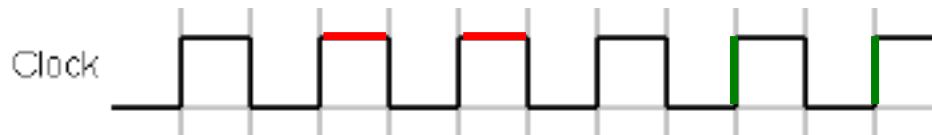
Instr. No.	Pipeline Stage						
	IF	ID	EX	MEM	WB		
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Instruction fetch, decode, execute, memory access, register write

The Most Efficient Way to Sample Inputs

Instead of sampling the latch input using a *level* of the clock...

- That is, when the clock is “1” (or “0”)



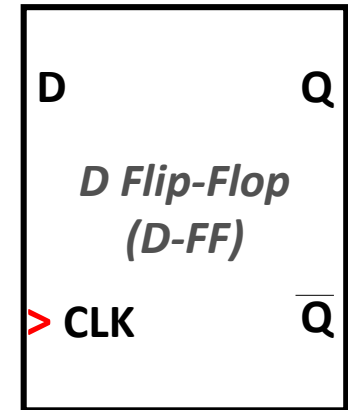
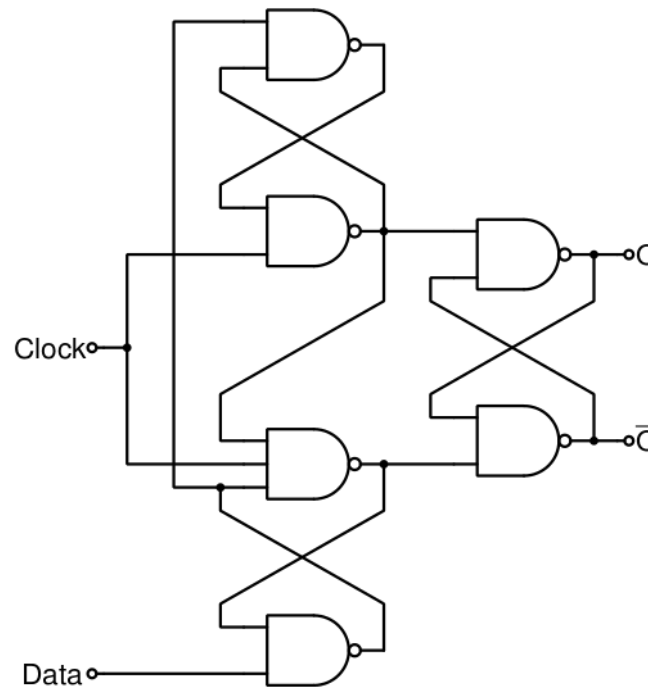
... sample the input at the *edge* of the clock

- That is,
 - when the clock is transitioning from $0 \rightarrow 1$, called a *rising* or *positive* edge (or it could be done from $1 \rightarrow 0$, the *falling* edge a.k.a *negative* edge)
- Why is this more efficient??

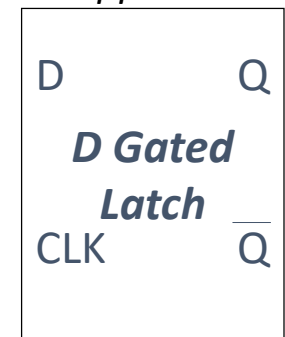
An Improvement on the Latch: The D Flip-Flop

Don't worry about the circuit implementation details (not going to be quizzed on this), but understand the use!

The **D Flip-Flop** only changes the output (Q) into the input (D) at the **positive edge** (the 0 → 1 transition) of the clock



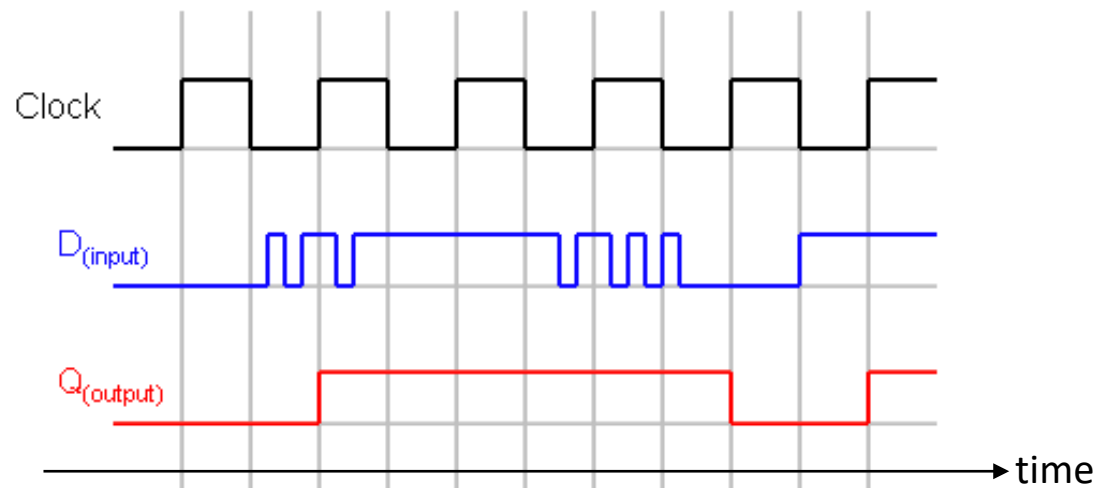
As opposed to:

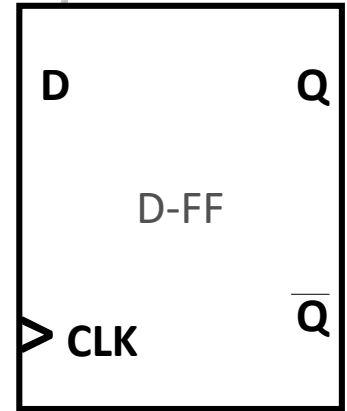
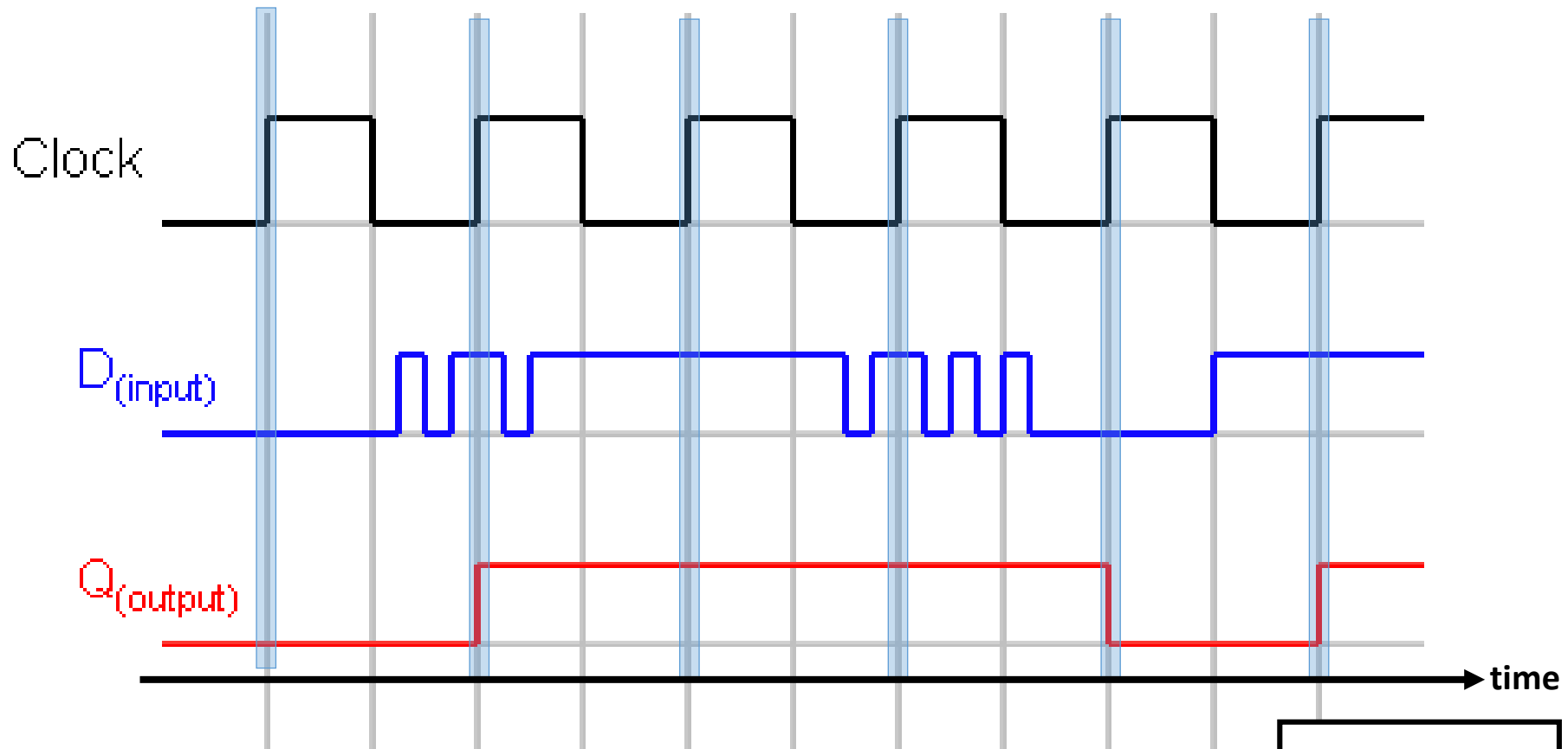


Note the (slight) difference in the 2 symbols...

The D-FF

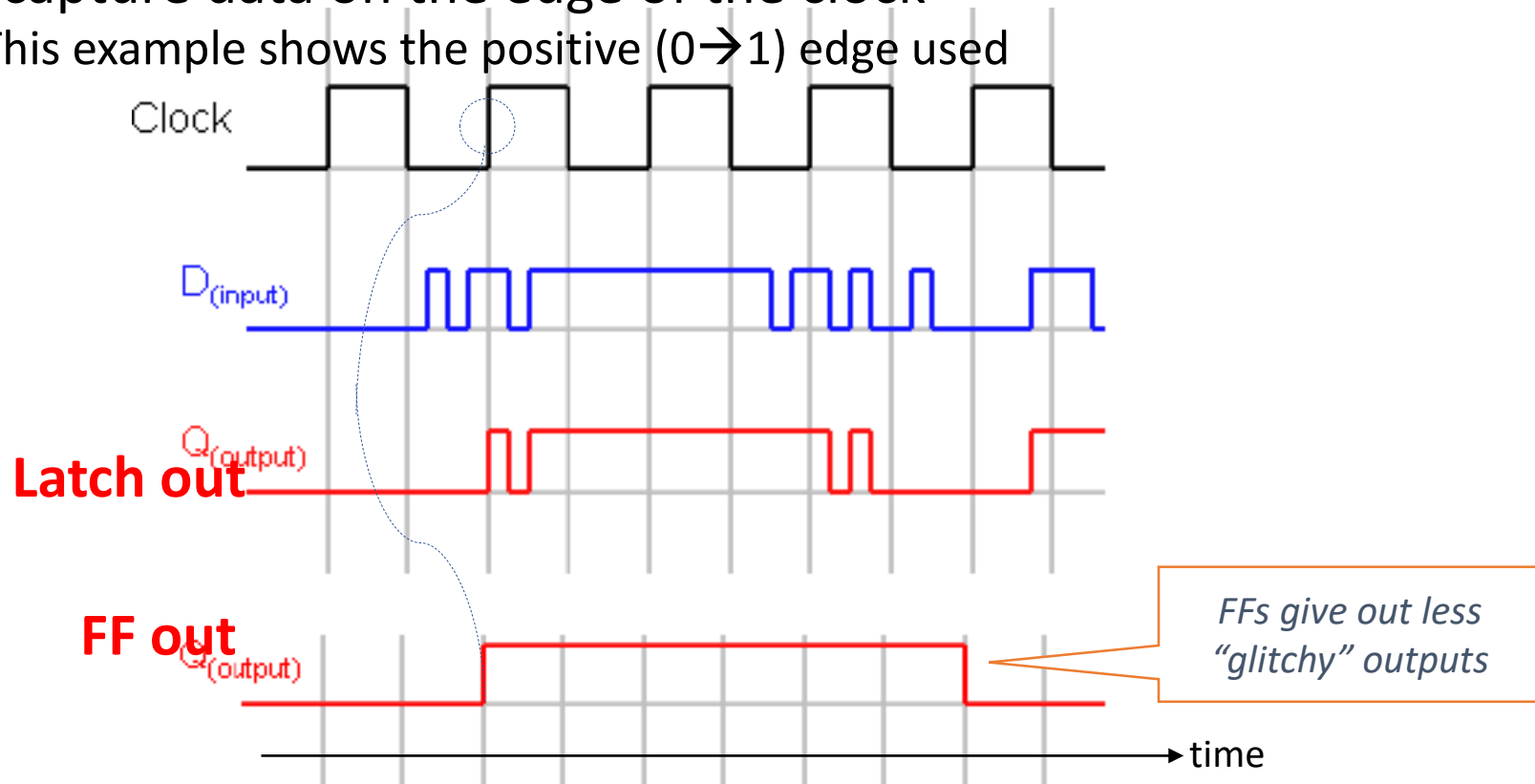
- When the input clock edge is *rising*, the input (D) is *captured* and placed on the output (Q)
 - Rising edge a.k.a positive edge FF
 - Some FF are negative edge FF (capture on the falling edge)





Latches vs. FFs

- Latches capture data on an entire 1 or 0 of the clock
- FFs capture data on the edge of the clock
 - This example shows the positive ($0 \rightarrow 1$) edge used



YOUR TO-DOs

- Lab 7 is due tomorrow
- Look for Lab 8 posted online by Wed.

</LECTURE>