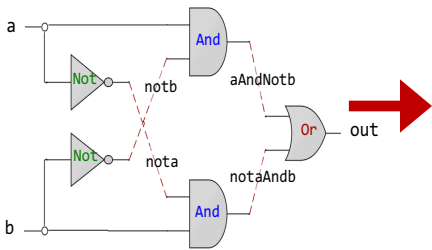
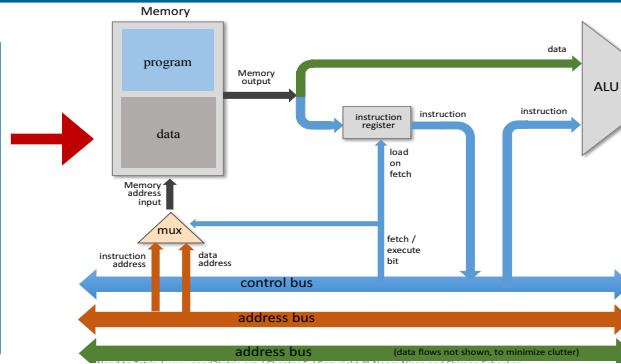




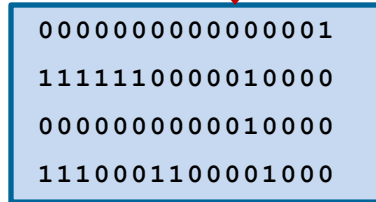
# Digital Logic Design



```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
  Not(in=a, out=nota);
  Not(in=b, out=notb);
  And(a=nota, b=b, out=w1);
  And(a=a, b=notb, out=w2);
  Or(a=w1, b=w2, out=out);
}
```



```
@R1
D=M
@temp
M=D
```

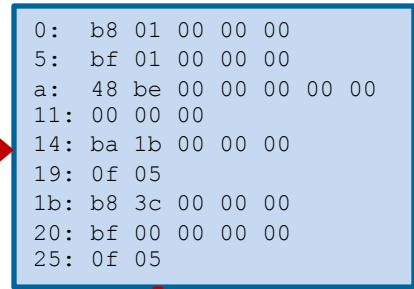


## Lecture # 15-16

# An Overview of Sequential Circuits

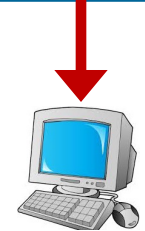
```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
  msg: db "Learning is fun with Arif", 0Ah, 0h
  len_msg: equ $ - msg
SECTION .text
main:
  mov rax,1
  mov rdi,1
  mov rsi,msg
  mov rdx,len_msg
  syscall
  mov rax,60
  mov rdi,0
  syscall
```



Slides of first half of the course are adapted from:  
<https://www.nand2tetrtris.org>  
 Download s/w tools required for first half of the course from the following link:  
<https://drive.google.com/file/d/0B9c0BdDz6XpZUh3X2dPR1o0MUE/view>

**Instructor: Muhammad Arif Butt, Ph.D.**





# Today's Agenda

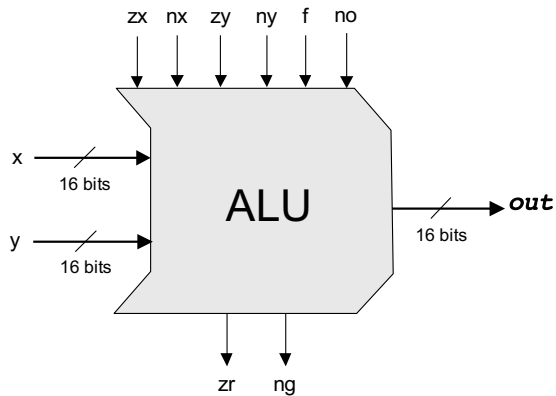
---

- Class Quiz
- Why Sequential Circuits?
- Understanding Time in Circuits
- Combinational vs Sequential Circuits
- Flip Flops
  - D flip Flop
  - SR Flip Flop
  - JK Flip Flop
  - T Flip Flop





# Class Quiz:



Pre-setting the x input		Pre-setting the y input		Selecting op + or &	Post-setting o/p	ALU output
$zx$	$nx$	$zy$	$ny$	$f$	$no$	$out$
if $zx$ then $x=0$	if $nx$ then $x=!x$	if $zy$ then $y=0$	if $ny$ then $y=!y$	if $f$ then $out=x+y$ else $out=x&y$	if $no$ then $out=!out$	$out(x,y)$ = $out$

- Problem 1:** Suppose  $x=123_{10}$  and  $y=141_{10}$  and the six control inputs are 000111. What is the value of **out** in hex. Also mention the values of **zr** and **ng** flags
- Problem 2:** Suppose  $x=5A63_{16}$  and  $y=1CF4_{16}$  and the six control inputs are 010101. What is the value of **out** in hex. Also mention the values of **zr** and **ng** flags



# Why Sequential Logic?



# Combinational vs Sequential Logic

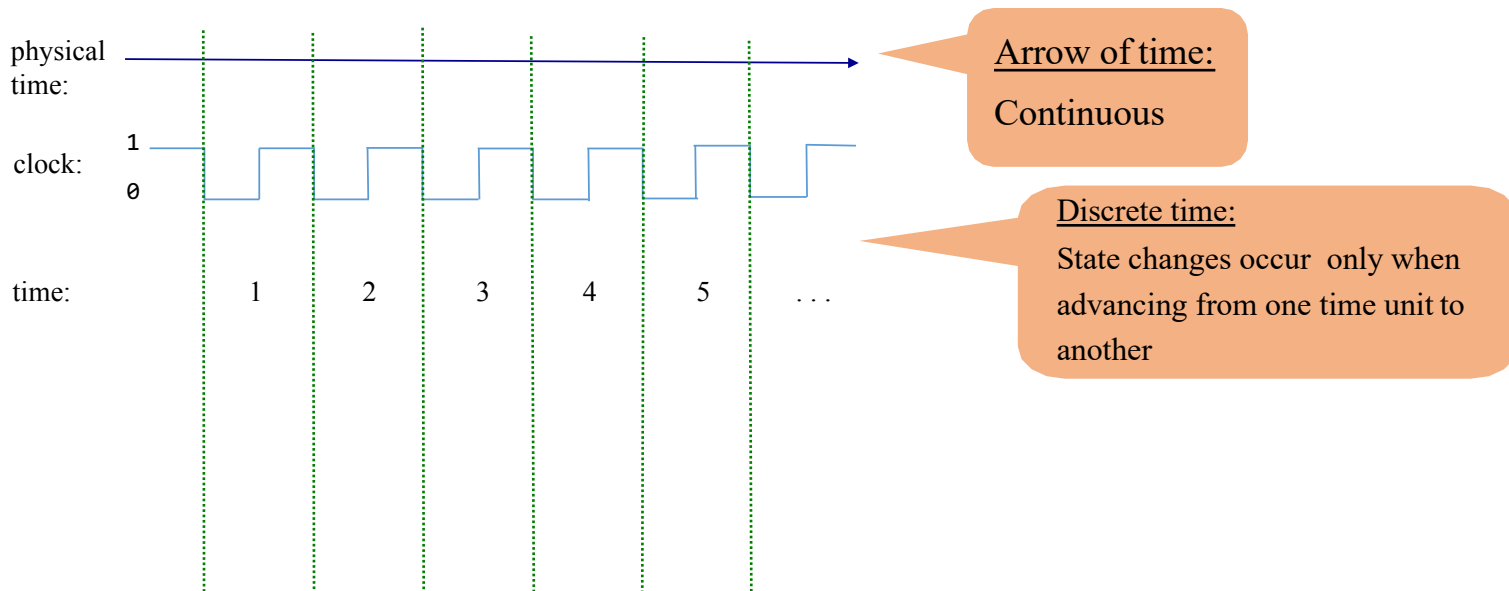
---

- So far we have designed and implemented our own ALU using different *combinational chips* whose output at any time depend on the current input. The combinational chips we have designed so far cannot maintain state
- Since computers must be able to not only compute values but also store and recall values, they must be equipped with memory elements that can preserve data over time. These memory elements are built from *sequential chips*
- The act of “remembering something” is inherently time-dependent. Thus, in order to build chips that “remember” previously stored information, we must first develop some standard means for representing the progression of **time**



# Physical Time / Clock Time

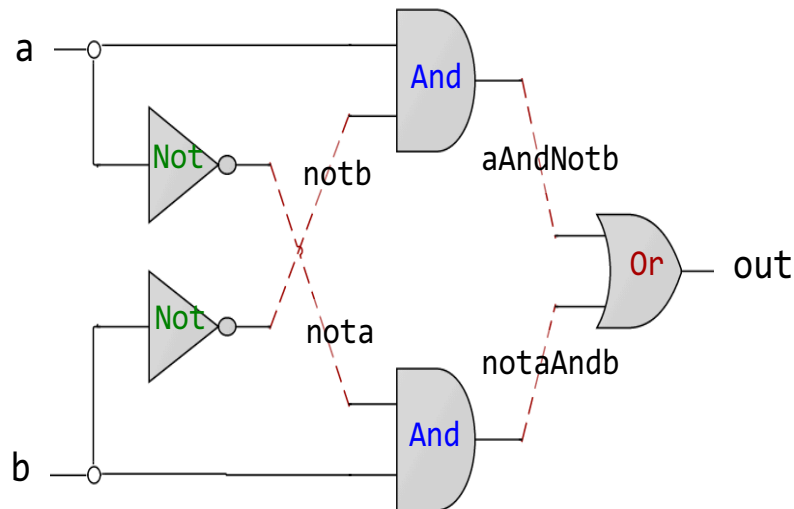
- In most computers, the passage of time is represented by a master clock that delivers a continuous train of alternating signals using a hardware called an oscillator, which alternates continuously between two phases 0 and 1
- The elapsed time between the beginning of a “tick” and the end of the subsequent “tock” is called clock cycle
- The current clock phase (tick or tock) is represented by a binary signal. Using the hardware’s circuitry, this signal is simultaneously broadcasted to every sequential chip throughout the computer platform





# Time-independent Logic

- So far we ignored the issue of time and the chip's inputs were just “sitting there” – fixed and unchanging
- The chip's output was a pure function of the current inputs, computed instantaneously. It did not depend on anything that happened previously (previous state)
- This style of gate logic is sometimes called time-independent logic or combinational logic





# Hello, Time

## Abstraction Issues:

- The hardware must support maintaining state, for example

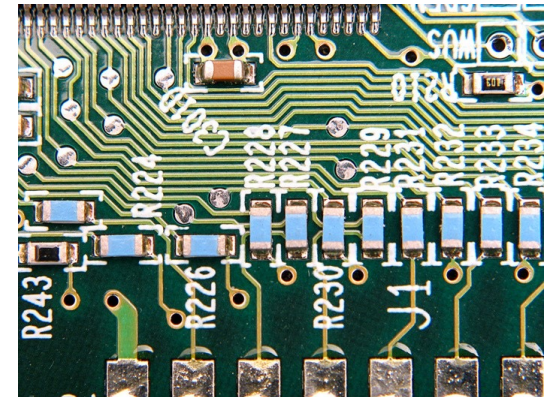
```
x = 17
```

- The hardware must support computations over time, for example

```
for i = 0 ... 99:  
    sum = sum + a[i]
```

## Implementation Issues:

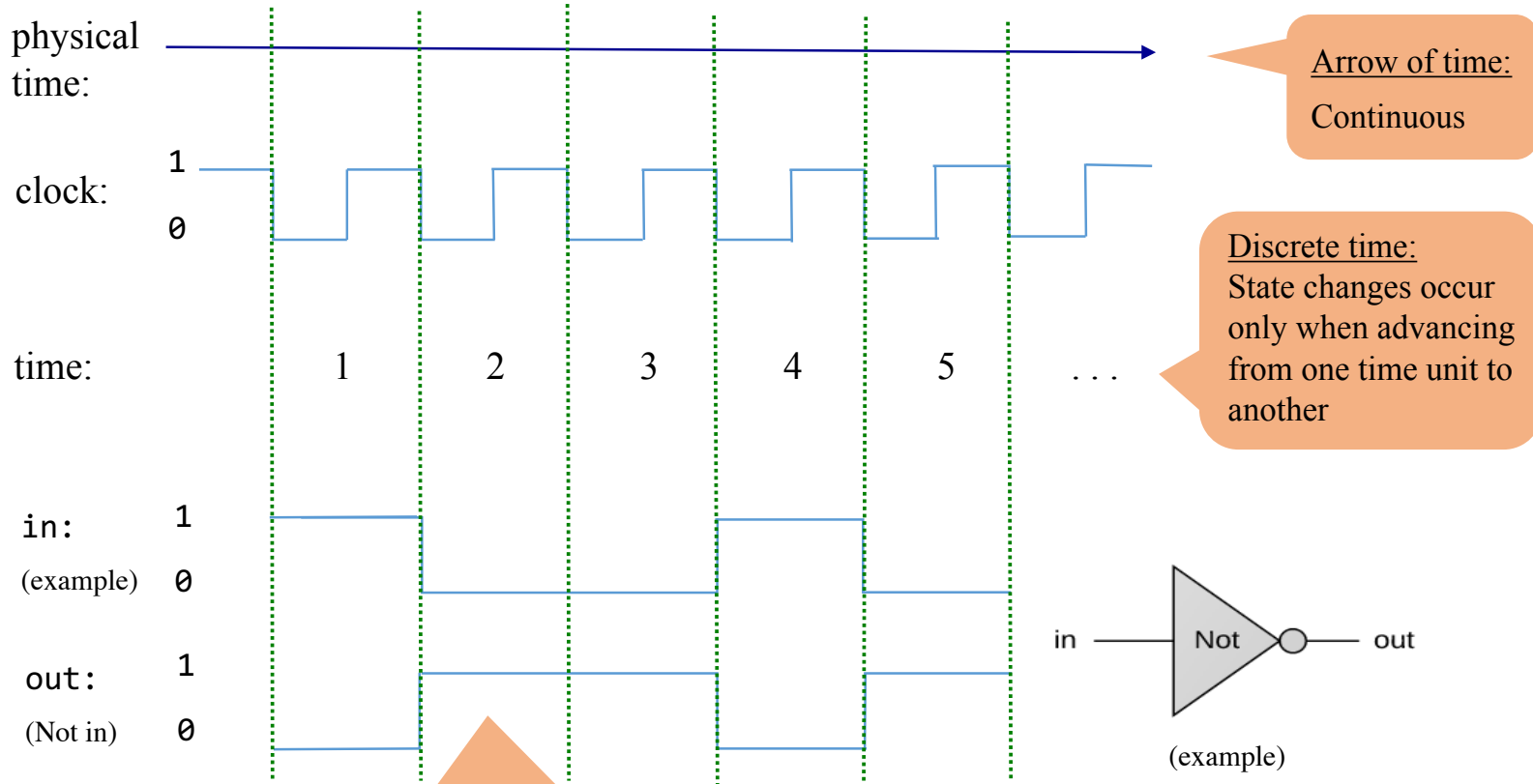
- The hardware must handle the physical time delays associated with calculating and moving data from one chip to another







# The Clock



- Desired / idealized behavior of the in and out signals
- That's how we want the hardware to handle time-dependent behavior



# The Clock

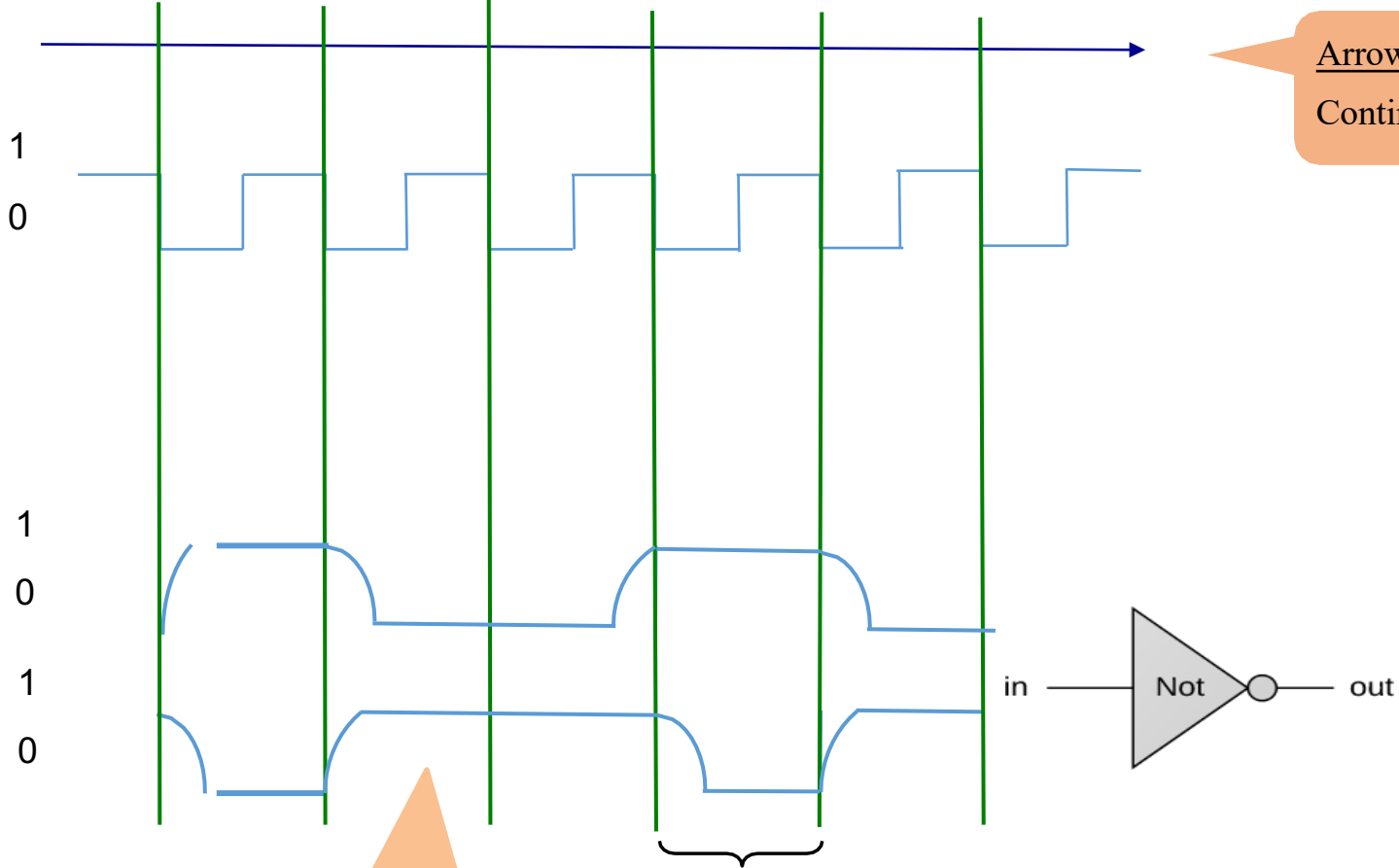
physical  
time:

clock:

time:

in:  
(example)

out:  
(Not in)

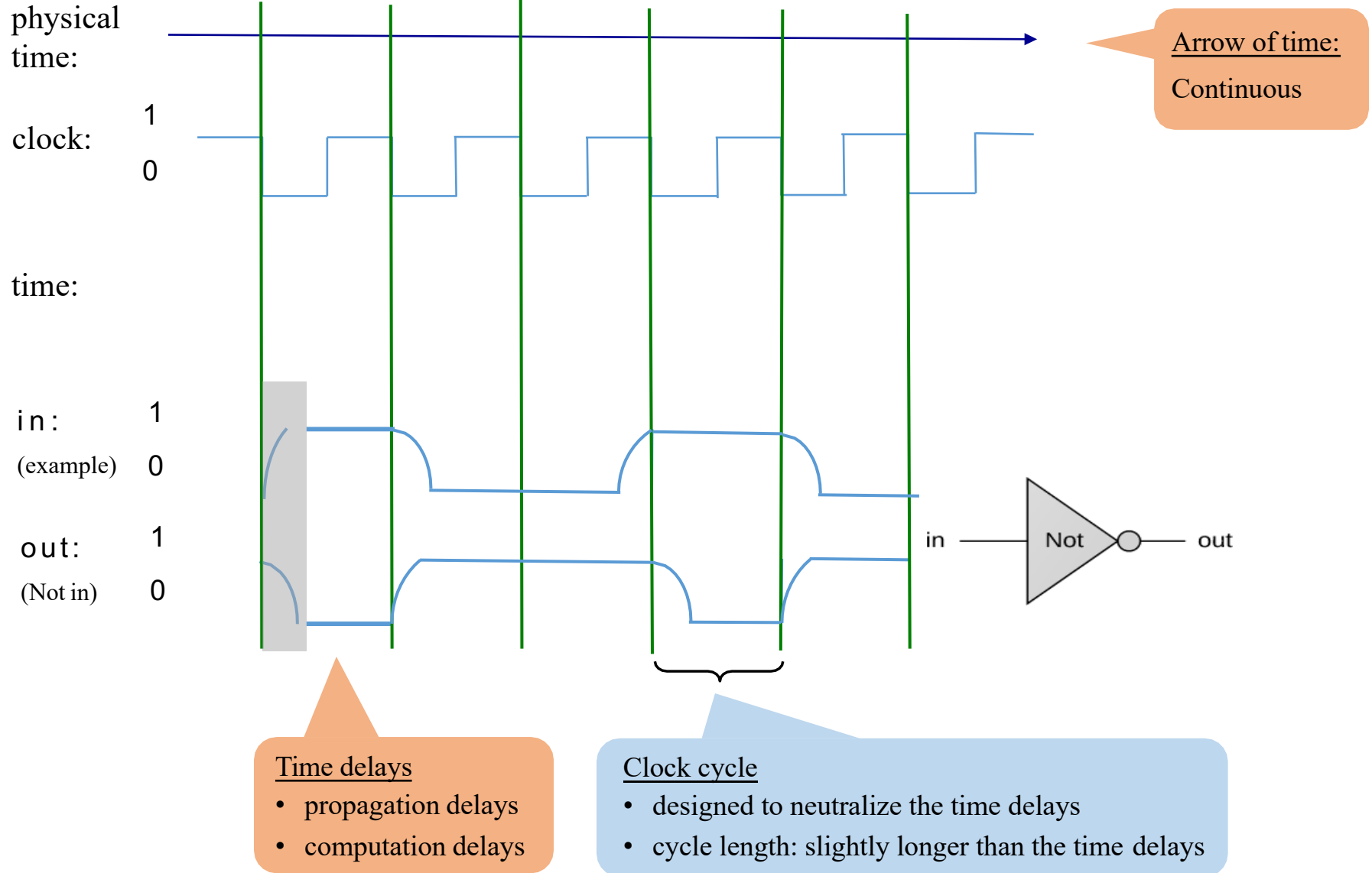


Arrow of time:  
Continuous

actual behavior of the  
in and out signals, due  
to physical time delays

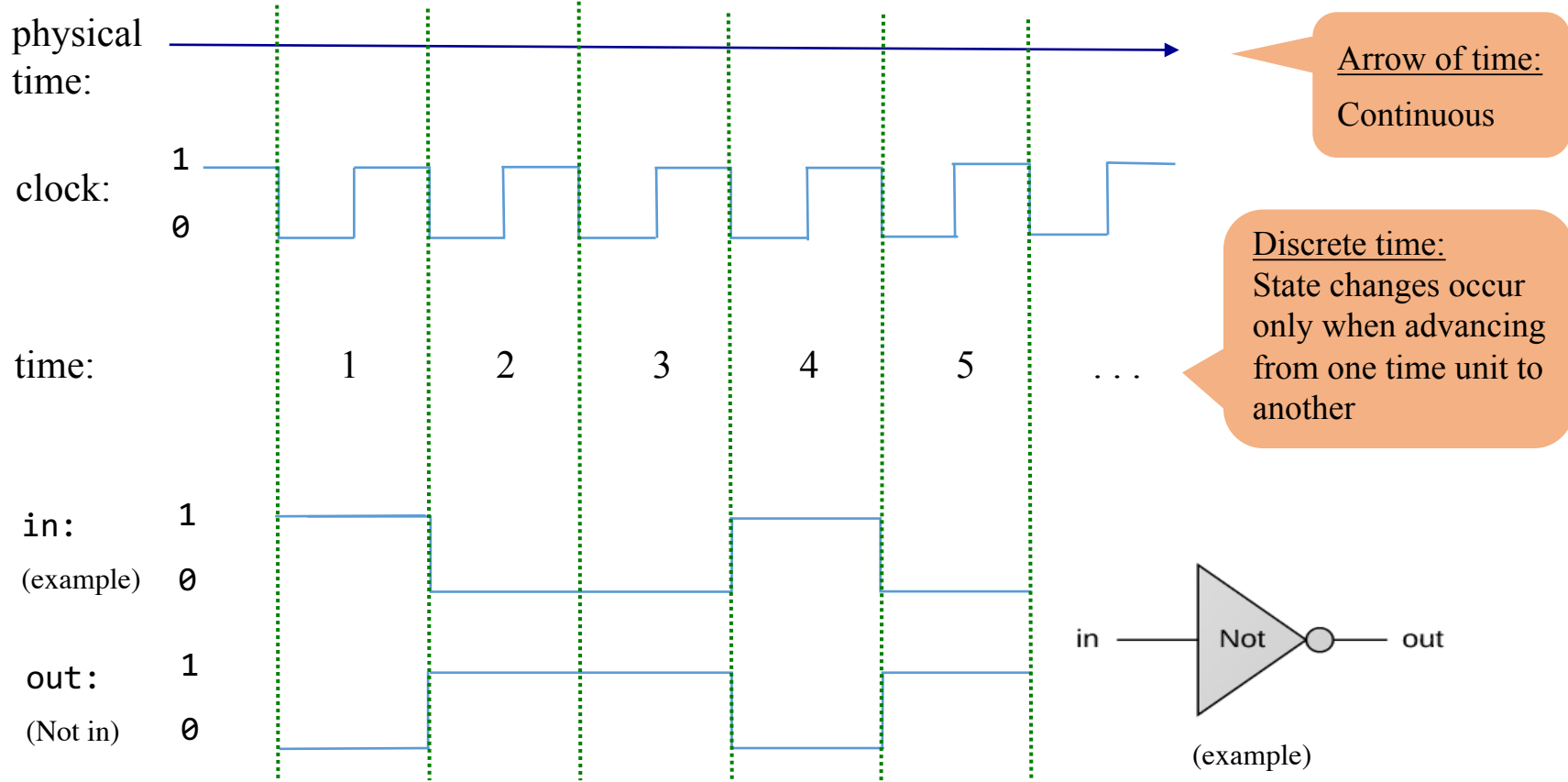


# The Clock





# The Clock



## An example of a combinational chip (NOT gate):

- The gate reacts “immediately” to the inputs
- Well, not really, but the clock’s behavior creates this effect.

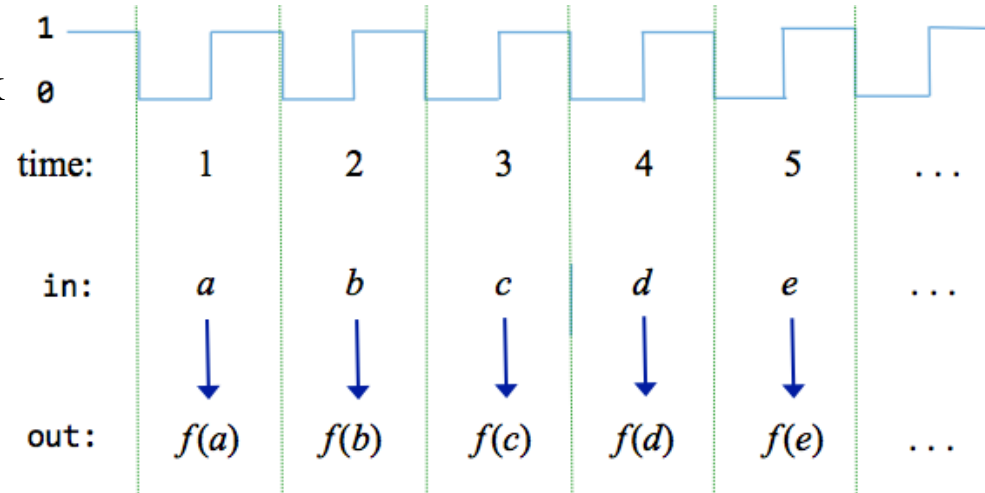


# Combinational Logic vs. Sequential Logic

## Combinational logic:

The output at time  $t$  is completely dependent on input at time  $t$

$$\text{out}[t] = \text{function}(\text{in}[t])$$



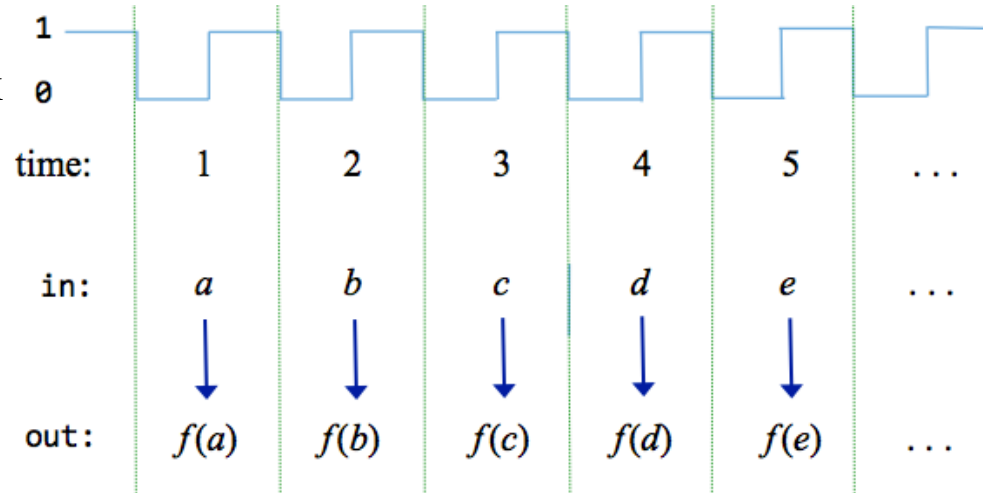


# Combinational Logic vs. Sequential Logic

## Combinational logic:

The output at time  $t$  is completely dependent on input at time  $t$

$$\text{out}[t] = \text{function}(\text{in}[t])$$

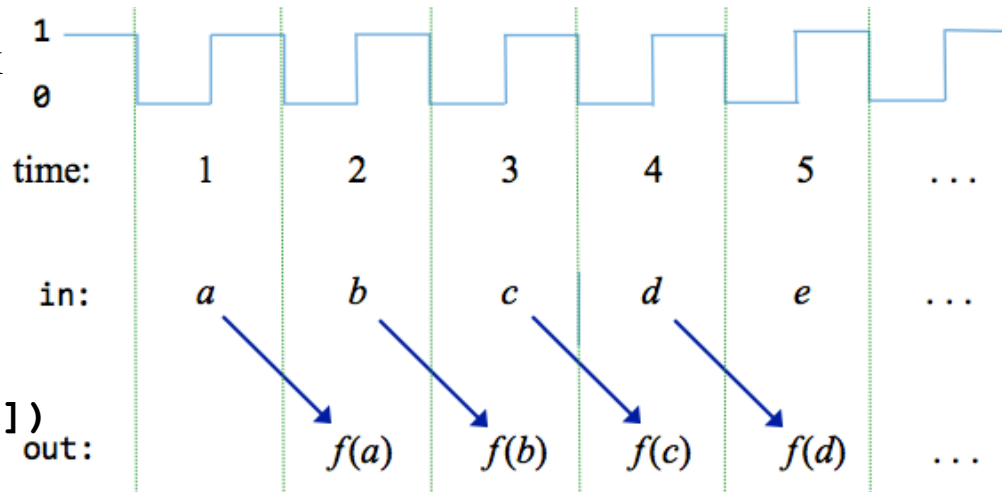


## Sequential logic:

The output at time  $t$  is dependent on input at time  $t-1$  (previously stored state) and optionally on current input. The history of the input creates a memory effect

$$\text{out}[t] = \text{function}(\text{in}[t-1])$$

clock

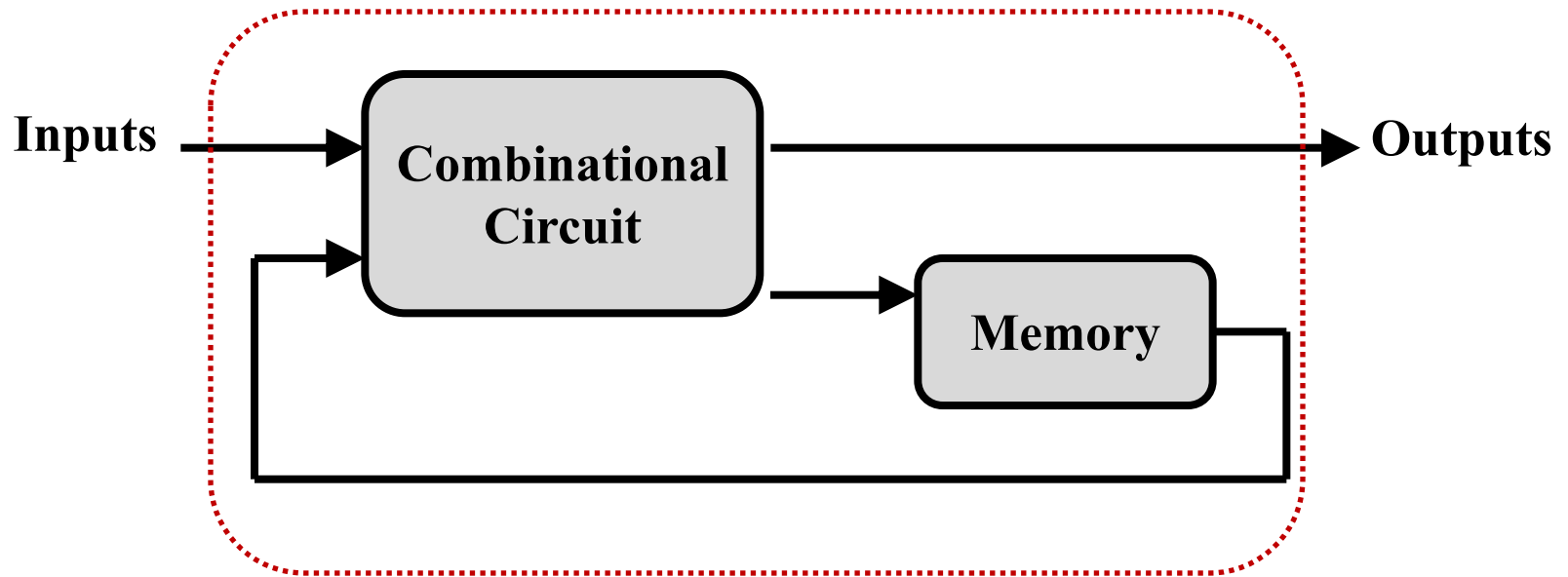




# Types of Sequential Circuits



# Sequential Circuits



- A sequential circuit is a circuit whose output at any time  $t$  depends on the input at time  $t$  as well as the stored state at time  $t-1$
- Examples:
  - TV channel
  - Program counter register
  - A loop construct that accesses memory

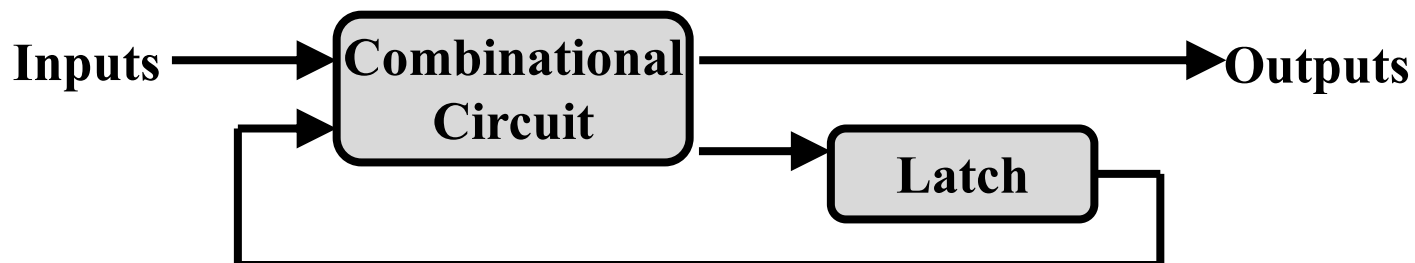




# Types of Sequential Circuits

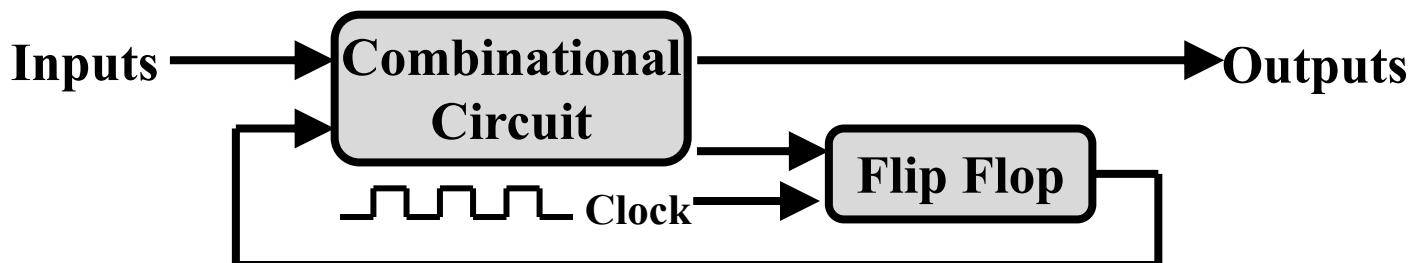
## Asynchronous

- Immediately responds to input(s) as they change
- No clock input
- Storage/Memory elements used are called Latches



## Synchronous

- Responds to the input(s) only at discrete time intervals
- It has an additional input called clock
- Storage/Memory elements used are called Flip-Flops



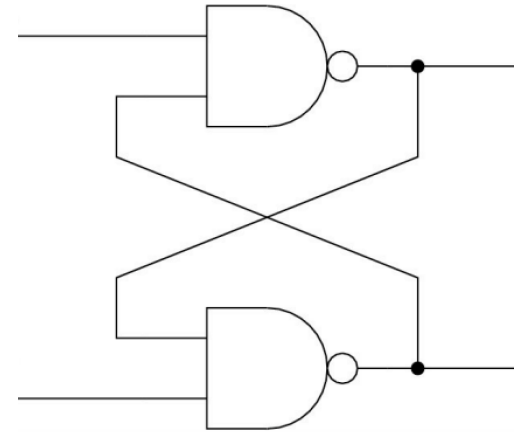


# Flip Flops



# Latches and Flip Flops

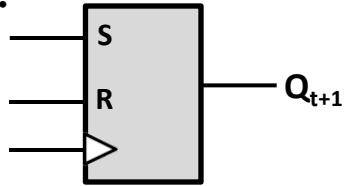
- To design sequential circuits we need a chip that can remember/store the state of the circuit, i.e., remember one bit of information from time  $t-1$ , so that it can be used at time  $t$
- This bit of information can either be 0 or 1. So the state that we want to store is a zero or a one
- The gate/chip that can flip between two stable states i.e., “remembering 0” or “remembering 1” is called a latch. It can be designed using two cross coupled NAND gates
- The limitation of a latch is that its state changes for the duration the clock input remains at logic 1 level. Due to this unreliable operation the o/p of a latch cannot be applied directly to some other latch when all the latches are triggered by a common clock pulse
- Solution to this problem are Flip Flops, which are constructed in such a way that they trigger only at signal transition. Comes in two flavors: Positive/Rising Edge Triggered, and Negative/Falling Edge Triggered





# RS Flip-Flop

**Graphic Symbol:**



**Behavior:**

```

if R=S=0 then
    Q_{t+1} = Q_t
elseif R=S=1 then
    Q_{t+1} = Indeterminate
else
    Q_{t+1} = S

```

**Characteristic Table:** A characteristic table defines the next state of a flip-flop as a function of its inputs and previous state  $Q_t$

R	S	$Q_t$	$Q_{t+1}$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	x
1	1	1	x

**Characteristic Equation:** A characteristic equation is the representation of the next state of flip-flop in terms of present state and current input(s)

$$Q_{t+1} = S + QR'$$

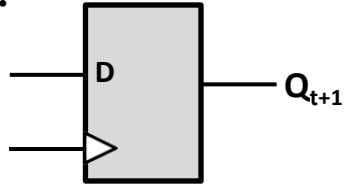
**Excitation Table:** During design process, we usually know the transition from present state to next state, and wish to find the input of the flip-flop that will cause the required transition. Excitation table gives us this information.

$Q_t$	$Q_{t+1}$	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0



# D Flip-Flop

**Graphic Symbol:**



**Behavior:**  $Q_{t+1} = D$

**Characteristic Table:** A characteristic table defines the next state of a flip-flop as a function of its inputs and previous state  $Q_t$

D	$Q_t$	$Q_{t+1}$
0	0	0
0	1	0
1	0	1
1	1	1

**Characteristic Equation:** A characteristic equation is the representation of the next state of flip-flop in terms of present state and current input(s)

$$Q_{t+1} = D$$

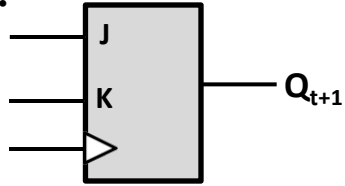
**Excitation Table:** During design process, we usually know the transition from present state to next state, and wish to find the input of the flip-flop that will cause the required transition. Excitation table gives us this information.

$Q_t$	$Q_{t+1}$	D
0	0	0
0	1	1
1	0	0
1	1	1



# JK Flip-Flop

**Graphic Symbol:**



**Behavior:**

```

if J=K=0 then
    Q_{t+1} = Q_t
elseif J=K=1 then
    Q_{t+1} = Q'_t
else
    Q_{t+1} = J

```

**Characteristic Table:** A characteristic table defines the next state of a flip-flop as a function of its inputs and previous state  $Q_t$

J	K	$Q_t$	$Q_{t+1}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

**Characteristic Equation:** A characteristic equation is the representation of the next state of flip-flop in terms of present state and current input(s)

$$Q_{t+1} = Q_t'J + Q_tK'$$

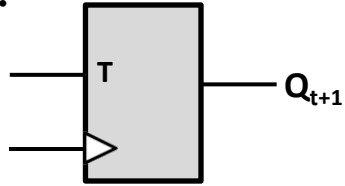
**Excitation Table:** During design process, we usually know the transition from present state to next state, and wish to find the input of the flip-flop that will cause the required transition. Excitation table gives us this information.

$Q_t$	$Q_{t+1}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0



# T Flip-Flop

**Graphic Symbol:**



**Behavior:** if  $T=0$  then  
 $Q_{t+1} = Q_t$   
elseif  $T=1$  then  
 $Q_{t+1} = Q'_t$

**Characteristic Table:** A characteristic table defines the next state of a flip-flop as a function of its inputs and previous state  $Q_t$

T	$Q_t$	$Q_{t+1}$
0	0	0
0	1	1
1	0	1
1	1	0

**Characteristic Equation:** A characteristic equation is the representation of the next state of flip-flop in terms of present state and current input(s)

$$Q_{t+1} = T \oplus Q_t$$

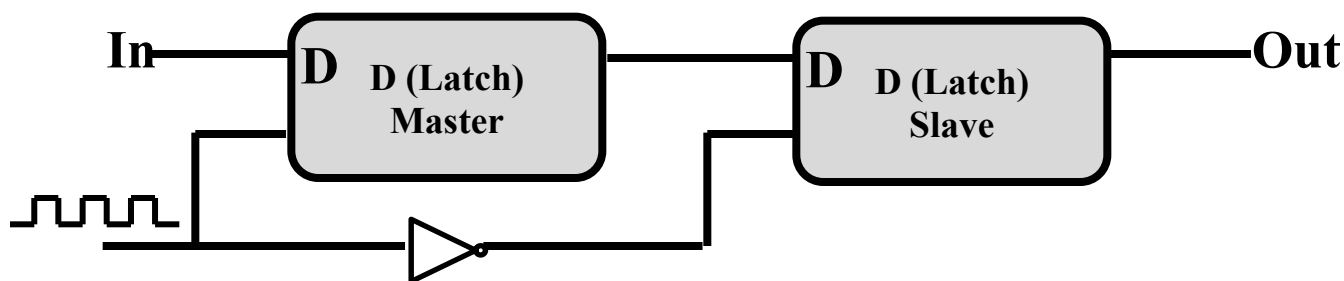
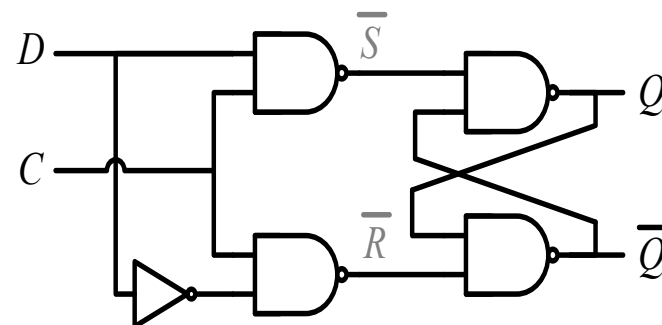
**Excitation Table:** During design process, we usually know the transition from present state to next state, and wish to find the input of the flip-flop that will cause the required transition. Excitation table gives us this information.

$Q_t$	$Q_{t+1}$	T
0	0	0
0	1	1
1	0	1
1	1	0



# D Flip Flop

- We can design a D-Flip Flop (DFF) using the built-in NAND gate available in our h/w simulator
  - Step 1: Create an un-clocked latch
  - Step 2: Use a master slave configuration to create a flip flop (-ve edge triggered)



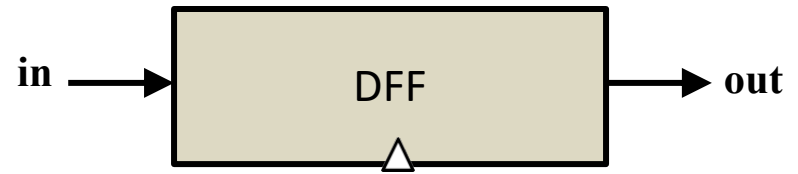
- Unfortunately, the hardware simulator which we are using does not permit combinational loops, so we cannot build DFF chip using HDL
- Fortunately, the way we have a **built-in NAND** gate, similarly we have a **built-in DFF** gate/chip available in our h/w simulator. So we will be designing and building all the sequential chips of our Hack computer by using build-in DFF chip





# Built-in D Flip Flop in H/W Simulator

- The most elementary sequential device from which all memory elements will be designed is the data flip-flop
- Like Nand gate, our hardware simulator provides a built-in DFF implementation that can be readily used by other chips
- All the sequential chips in the computer (registers, memory, and counters) are based on numerous DFF gates. All these DFFs are connected to the same master clock. At the beginning of each clock cycle, the outputs of all the DFFs in the computer commit to their inputs during the previous time unit. At all other times, the DFFs are latched, meaning that changes in their inputs have no immediate effect on their outputs



```
/**
 * Data Flip-flop:
 * out(t) = in(t-1)
 * where t is the current
 * time unit, or clock cycle.
 */

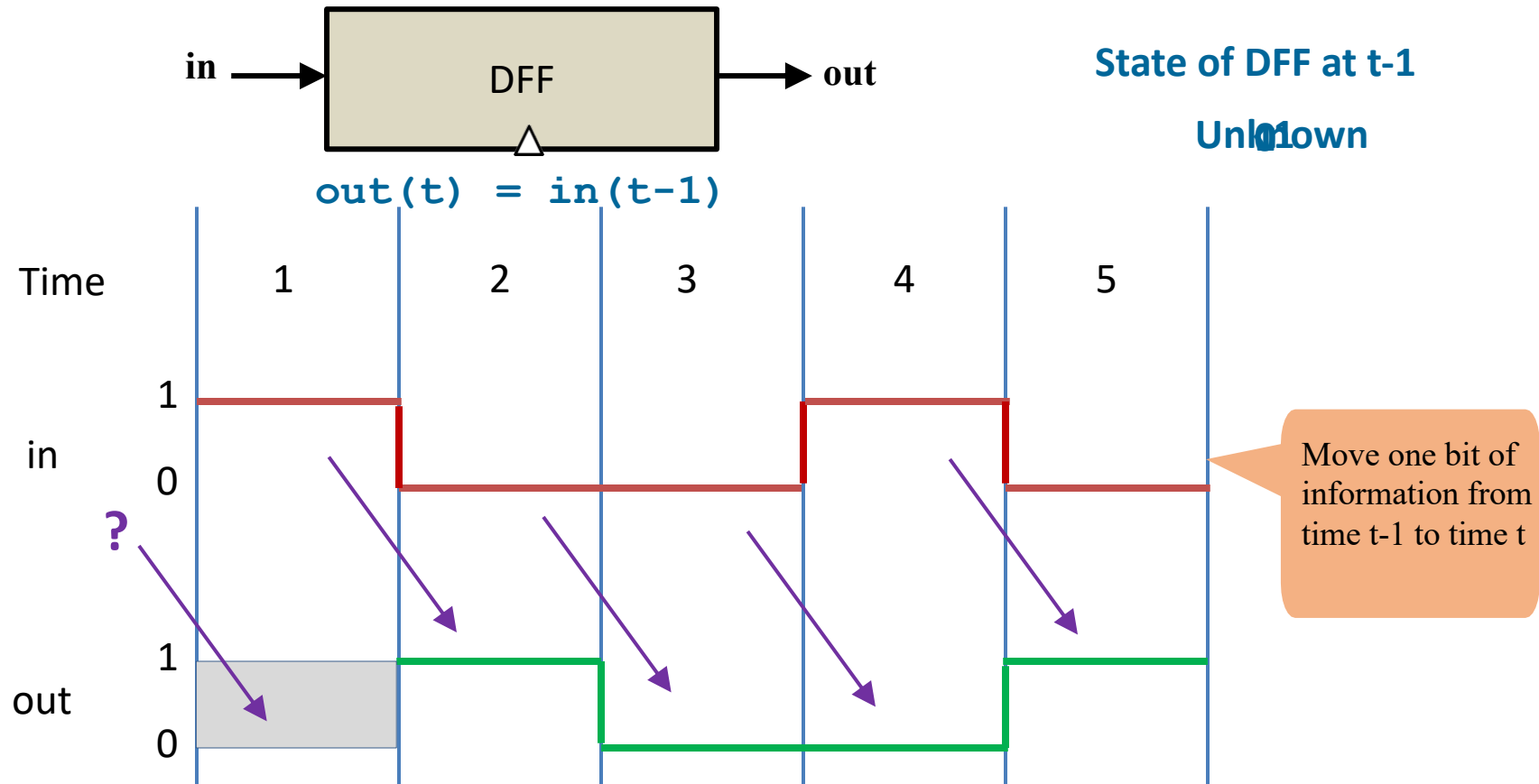
CHIP DFF {
    IN in;
    OUT out;

    BUILTIN DFF;
    CLOCKED in;
}
```



# Behavior of D - Flip Flop

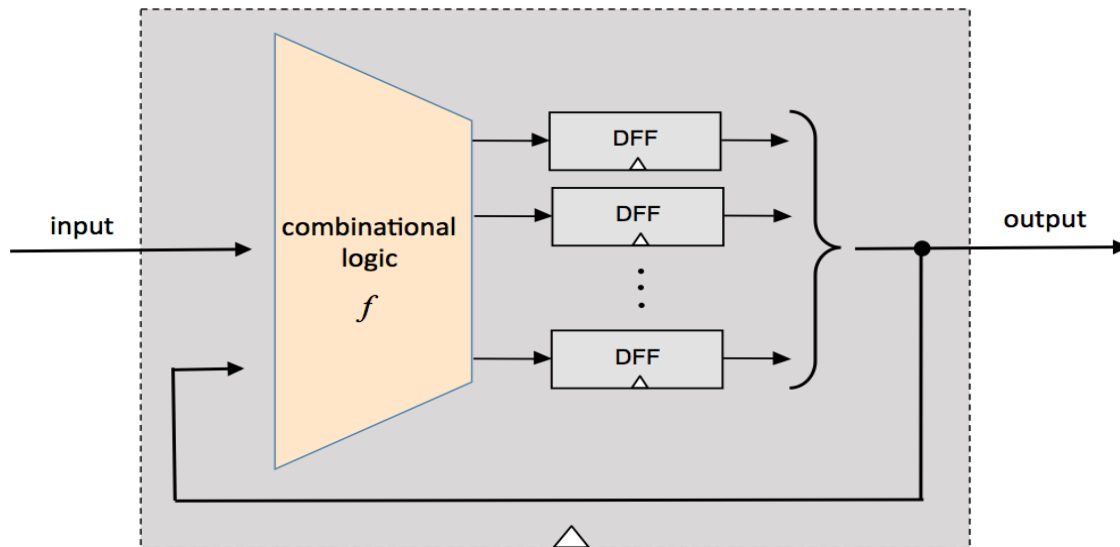
- D-Flip Flop (DFF) is the simplest state-keeping gate: 1-bit input, 1-bit output
- The gate outputs its previous input:  $out(t) = in(t - 1)$
- Due to its behavior it is also called Data Flip Flop





# Summary of Sequential Chips

$$\text{state}(t) = f(\text{state}(t-1), \text{input}(t))$$



- Sequential chips are capable of maintaining state, and, optionally acting on the state, and on the current input
- The simplest and most elementary sequential chip is DFF, which maintain a state, i.e., the value of the input from the previous time unit
- Using DFF we can design registers, and using registers we can design RAM, whose state is the current values of all its registers. Given an address, the RAM emits the value of the selected register
- All combinational chips are constructed from NAND gates, while all sequential chips are constructed from DFF gates, and combinational chips



# Things To Do

- Grasp the concept of all flip flops their graphic symbols, their behavior using the respective characteristic tables, characteristic equations, and excitation tables
- You should know as to how to perform analysis of a sequential circuit as well as how to design a sequential circuit
- Try running the built-in DFF chip in the hardware simulator to fully understand its behavior. You can download the .hdl, .tst and .cmp files of all the required chips from the course bitbucket repository:

<https://bitbucket.org/arifpucit/coal-repo/>



**Coming to office hours does NOT mean you are academically weak!**