CS250: Foundations of Computer Systems [Memory]

Slicing and dicing time
The smaller you cut
The faster you are
While doing more

Cut too small
And you run into the Physics wall
With the legions of light and heat
Conspiring to lead a mutiny

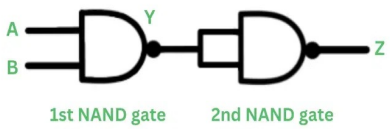
Use the Goldilocks rule
And the CPU keeps its cool

Shrideep Pallickara
Computer Science
Colorado State University

1

Frequently asked questions from the previous class survey

- Difference between Boolean expressions and Boolean functions
  - A Boolean function is a mathematical relationship (a mapping) that defines an output for *every* possible combination of binary inputs
- Quizzes are being handed back during the recitations
- NAND gate implementations: Are you using AND and then a NOT gate? No!
  - Most efficient production implementations have NAND gates with just 2 transistors. Few 10s of nanometers
  - NOR? ~4 transistors
- AND gates from NAND gates?

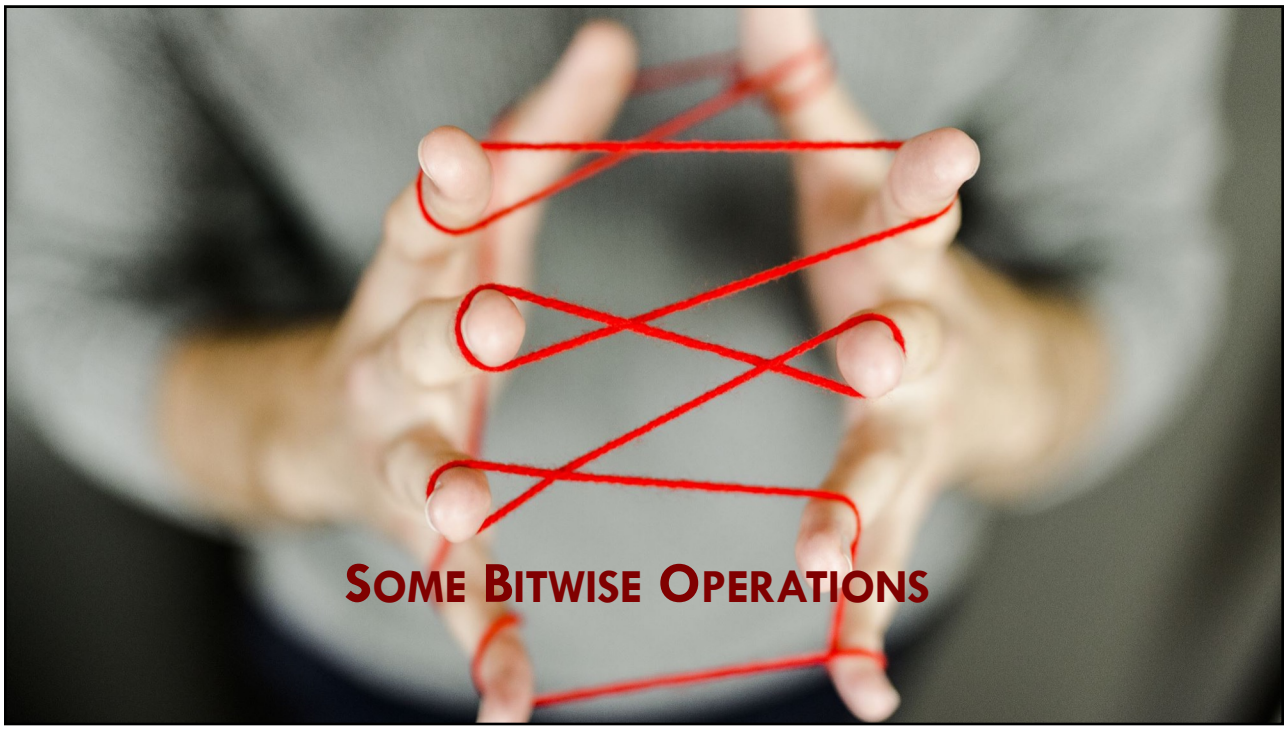


2

Topics covered in this lecture

- Bitwise Operators
- Memory

3



4

NOT (~ in Java)

- The bitwise NOT is a **unary** operation
  - Performs logical negation of each bit

	1	0	1	1	0	0	1	1
NOT	0	1	0	0	1	1	0	0

- What's another way of thinking about the NOT operation?
  - **1s complement**!

Bitwise AND (& in Java) operates on two equal length binary representations

	1	0	1	1	0	0	1	1
	0	1	1	1	1	0	1	0
AND	0	0	1	1	0	0	1	0

- Can be used to detect if a particular bit is set (1) or cleared (0)
  - If you wish to detect if position x is set, do a bitwise AND with only that position set to 1

Check position 5	1	0	1	1	0	0	1	1
AND with:	0	0	0	1	0	0	0	0

Also called a MASK

Non-zero result if the bit position was set; zero otherwise

Bitwise OR (| in Java) operates on two equal length binary representations

	1	0	1	1	0	0	1	1
	0	1	1	1	1	0	1	0
OR	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**

- Can be used to set selected bit(s) to 1

Set positions 4 and 5	1	0	1	1	0	0	1	1
OR with	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**

Bitwise XOR (^ in Java) operates on two equal length binary representations [1/2]

	1	0	1	1	0	0	1	1
	0	1	1	1	1	0	1	0
XOR	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**

- Can be used to **toggle** (invert or flip) the bits
  - Any bit can be toggled by XOR-ing it with 1

Toggle 5 and 7	1	0	1	1	0	0	1	1
XOR with:	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**

Bitwise XOR (^ in Java) operates on two equal length binary representations [2/2]

- Performing an XOR on itself?
  - Always leads to a Zero

	1	**0**	**1**	**1**	**0**	**0**	**1**	**1**
	1	0	1	1	0	0	1	1
XOR	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Java code snippet for these binary operations …

```
public static void main(String[] args) {
        int a = 0b10110011; // 179
        int b = 0b01111010; // 122

        // NOT (~): flips all bits
        (Nota Bene: in Java it's 32-bit for int)
        int notA = ~a;

        // AND (&): masking / testing bits
        int andAB = a & b;

        // OR (|): setting bits
        int orAB = a | b;

        // XOR (^): toggling bits / differences
        int xorAB = a ^ b;
        continued in the right panel …
```

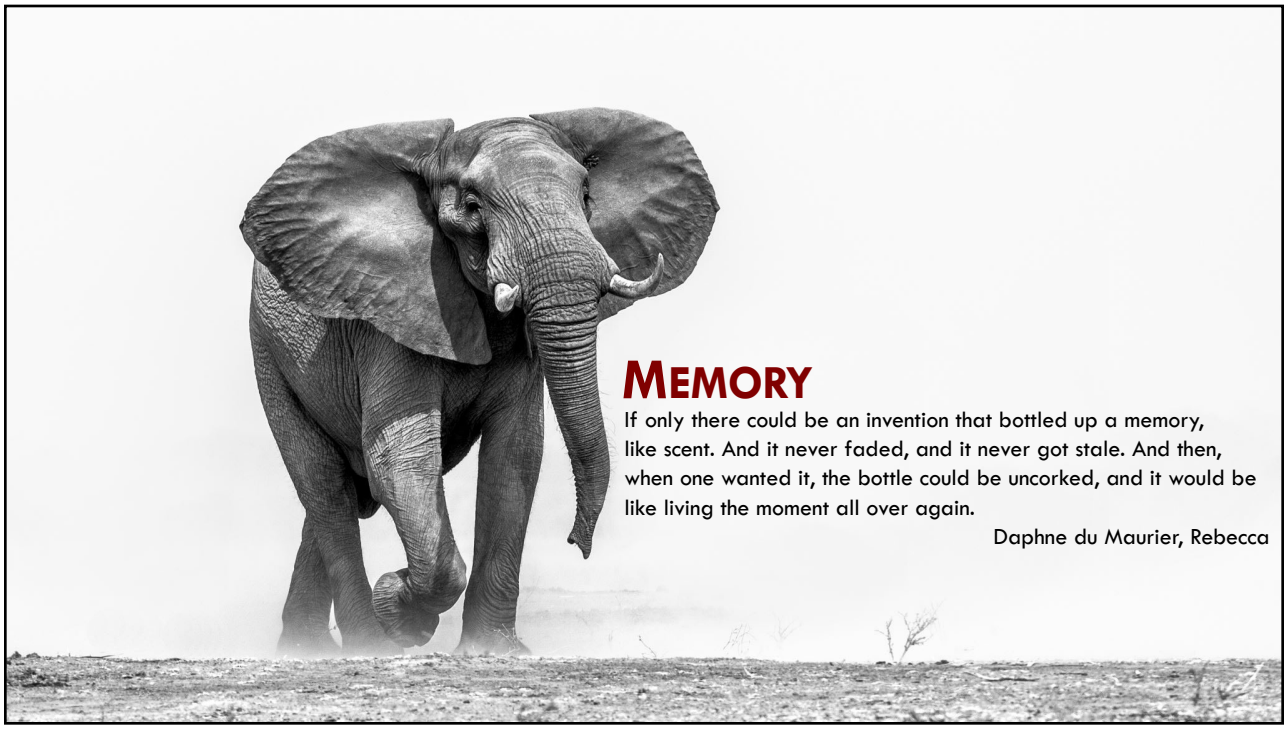
```
        // Mask example: test if bit position k is set (0 = LSB)
        int k = 5;
        int mask = 1 << k;
        boolean isSet = (a & mask) != 0;
        System.out.println("bit " + k + " of a is " + (isSet ? "SET" : "CLEAR") );

        // Set bits (turn bit k ON)
        int setK = a | mask;

        // Clear bits (turn bit k OFF) using AND with inverted mask
        int clearK = a & ~mask;

        // Toggle bits (flip bit k) using XOR
        int toggleK = a ^ mask;
}
```

DO NOT USE these operators in HW1



11

Memory Management: Topics we will cover over the next few lectures

- Data flip flop
- The speed differential across the memory hierarchy
- Why caches are needed
- Registers and L1, L2, and L3 caches
- Direct-mapped caches and fully-associative caches
- Main memory
- Memory addressing

12

Computer programs use variables, arrays, and objects

- These are abstractions that **persist** data over time
- Hardware platforms support this ability by offering memory devices that know *how* to maintain **state**
- Because evolution gave humans a phenomenal electro-chemical memory system
  - We tend to *take for granted* the ability to remember things over time
  - However, this ability is hard to implement in classical logic, which is aware of neither time nor state

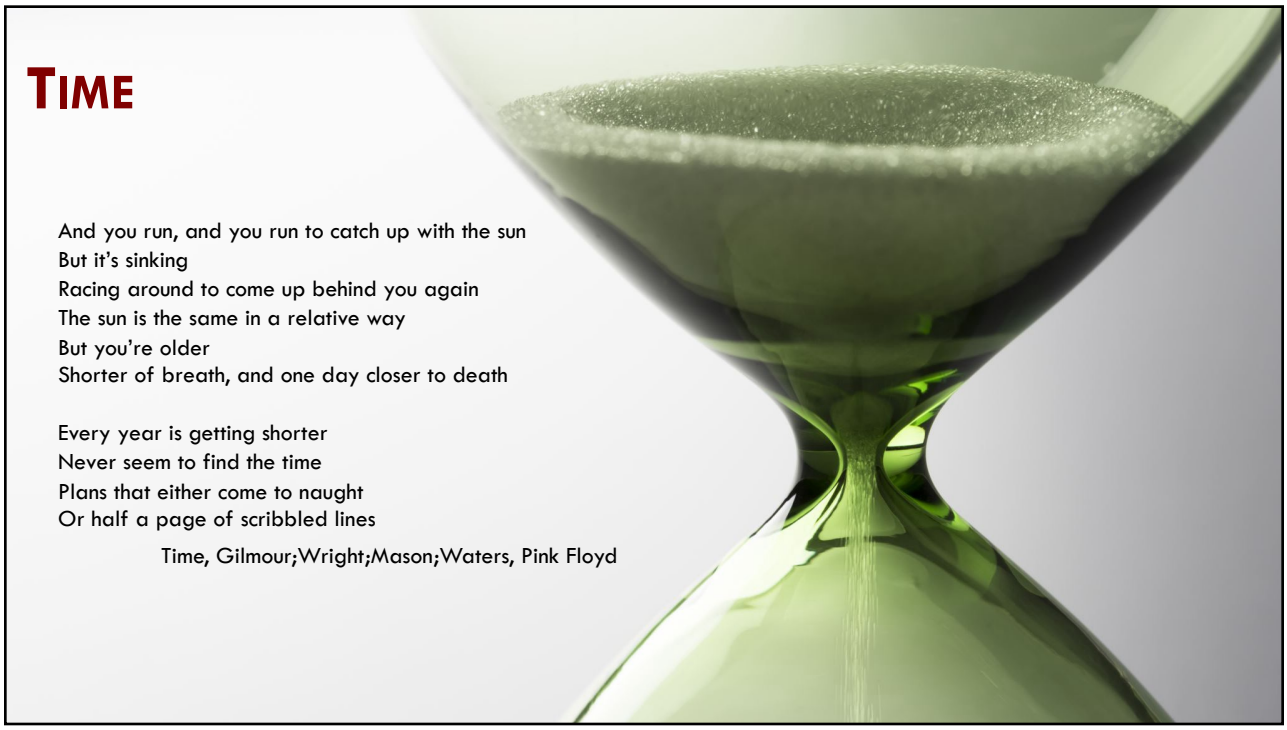
To get started, we need to …

- First find a way to
  - **Model** the progression of time and
  - **Endow** logic gates with the ability to maintain state & respond to time changes
- Done by introducing a clock and an elementary, time-dependent logic gate that can flip and flop between two stable states:
  - Representing 0 and representing 1
- This **data flip-flop** (DFF) is the fundamental building block

Data flip flop (DFF)

- Despite its **central role** the DFF is low-profile and inconspicuous
- DFFs are **used implicitly**, as low-level chip-parts embedded deep within other memory devices
  - Unlike registers, RAM devices, and counters, which play prominent roles in computer architectures

TIME



And you run, and you run to catch up with the sun
But it's sinking
Racing around to come up behind you again
The sun is the same in a relative way
But you're older
Shorter of breath, and one day closer to death

Every year is getting shorter
Never seem to find the time
Plans that either come to naught
Or half a page of scribbled lines

Time, Gilmour;Wright;Mason;Waters, Pink Floyd

So far …

- We have assumed that chips *respond* to their inputs **instantaneously**: you input 7, 2, and "add" into the ALU, and … poof! the ALU output becomes 9
- In reality, outputs are always **delayed**

Outputs are always delayed, due to at least two reasons

- The inputs don't appear out of thin air
  - Rather, the signals that represent them **travel** from the outputs of other chips, and this *travel takes time*
- The **computations** that chips perform also take time
  - The more chip-parts the chip has; the more elaborate its logic
    - The more time it will take for the outputs to **emerge fully formed** from the chip's circuitry

Time is typically viewed as a metaphorical arrow that progresses relentlessly forward

- This progression is taken to be **continuous**
  - Between every two time-points there is another time-point, and changes in the world can be *infinitesimally small*
- This notion of time, which is popular among philosophers and physicists, is too deep and mysterious for computer scientists

Instead of viewing time as a continuous progression

- We prefer to break it into *fixed-length intervals*, called **cycles**
- This representation is **discrete**
  - Resulting in cycle 1, cycle 2, cycle 3, and so on
- Unlike the continuous arrow of time, which has an infinite granularity, the **cycles are atomic and indivisible**
- **Changes** in the world occur only *during cycle transitions*; within cycles, the world stands still

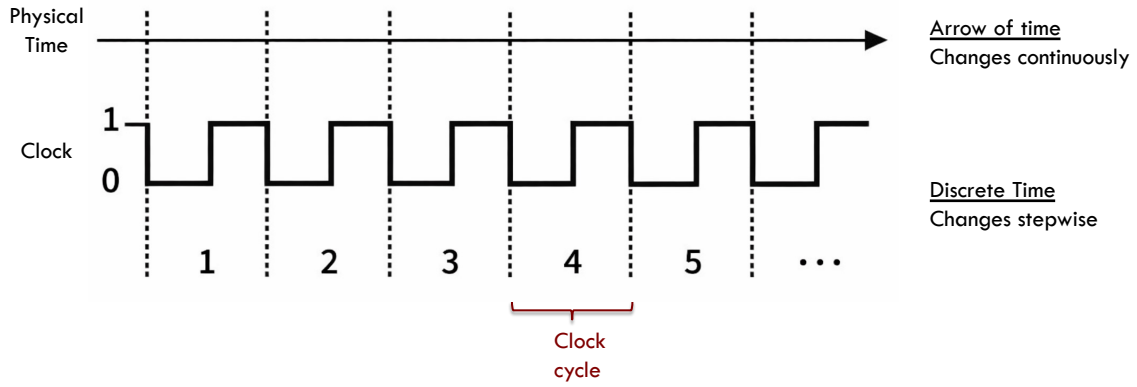
Of course, the world never stands still

- However, by treating time **discretely**, we make a conscious decision to *ignore continuous change*
- We are content to know the state of the world in cycle n, and then in cycle (n+1) but ***during*** each cycle the state is assumed to be …
  - Well, we don't care

This discrete view of time serves two important design objectives

- First, it can be used for **neutralizing the randomness** associated with communications and computation time delays
- Second, it can be used for **synchronizing** the operations of different chips across the system

Discrete time representation



Ticktock [1/2]

- Typically, the cycles are realized by an **oscillator** that alternates continuously between two phases labeled 0—1, low-high, or **ticktock**
- The elapsed time between the beginning of a tick and the end of the subsequent tock is called a **cycle**
  - Each cycle is taken to model *one discrete time unit*

Ticktock [2/2]

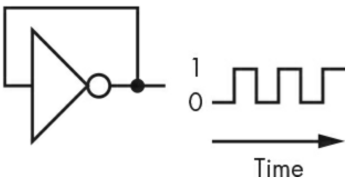
- The current clock phase (tick or tock) is represented by a binary signal
- Using the hardware's circuitry, the same **master clock signal is simultaneously broadcast** to every memory chip in the system
- In every such chip, the clock input is funneled to the lower-level DFF gates
  - Where it serves to ensure that the chip will commit to a new state, and output it, only at the end of the clock cycle

How we measure time …

- We measure time using some sort of **periodic function**
  - E.g., the rotation of the Earth; we call one full rotation a day
  - A day can be subdivided (e.g., hours, minutes, and seconds)
  - A second? 1/86,400th of an Earth rotation – 86,400 seconds in a day
- In addition to using an external event like the rotation of the Earth, we can also **generate our own periodic functions**
  - Using physics: the time that it takes for a pendulum to swing
    - This produced the "tick tock" sound in old grandfather clocks
    - To be useful, the **pendulum has to be calibrated** to the measured length of a second

A simple oscillator

- This produces feedback, just like what you get when a microphone is too close to a loudspeaker
- The output of the inverter bounces back and forth, or **oscillates**, between 0 and 1
- The speed at which it oscillates is a function of the *propagation delay*



Frequency and oscillation

- Useful to have an oscillator with a stable frequency: Why?
  - So that we could generate an accurate time reference
- A cost-effective way to do this is with a **crystal**

Crystals, like magnets, have a relationship with electricity

- If you attach electrodes (wires) to a crystal and give it a squeeze, it'll generate electricity
- And if you put some electricity on those wires, the crystal will bend
- This is called the **piezoelectric effect**
  - Discovered by brothers Paul-Jacques (1855–1941) and Pierre (1859–1906) Curie in the late 1800s

But how would that work?

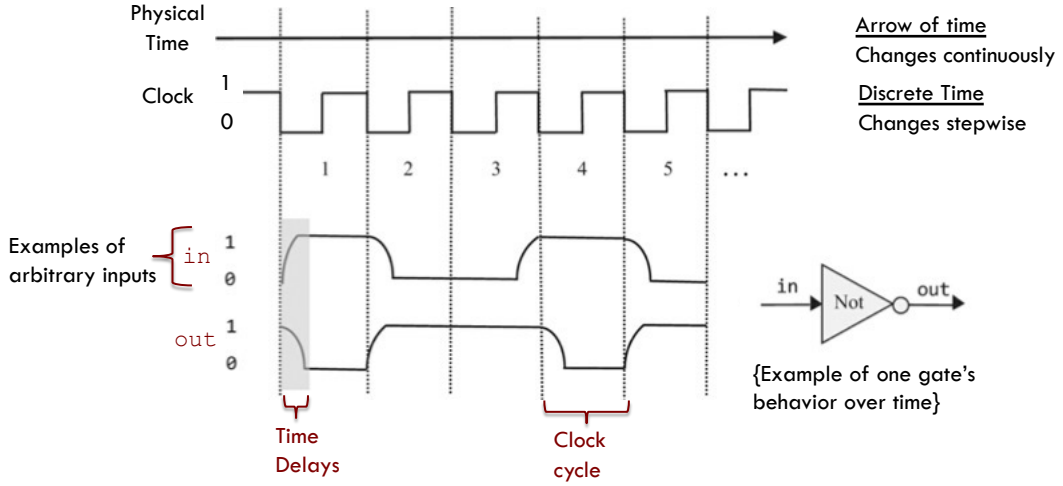
- A crystal oscillator alternately
  1. Applies electricity to a crystal and
  2. Receives electricity back

- Done using electronic single-pole, double-throw switches
- The time it takes a crystal to do this is **predictable and very accurate**
- Quartz is one of the best crystal materials to use
  - You typically see ads for accurate quartz timepieces
  - But a really good crystal retails for only about 25 cents

Oscillators and clocks

- Oscillators supply **clocks** to computers
- A computer's clock is like the drummer in a marching band; it sets the pace for the circuitry
- The maximum clock speed or fastest tempo is determined by the propagation delays

Discrete time representation



Example: Considering the Not gate [1/2]

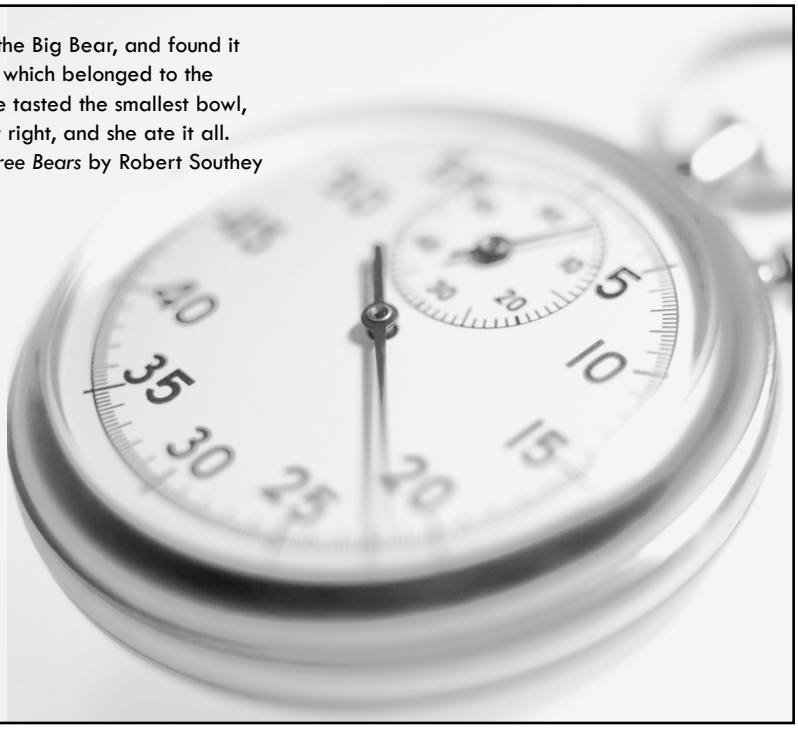
- When we feed the gate with 1, it *takes a while* before the gate's output stabilizes on 0
- However, since the cycle duration is (*by design*) longer than the time delay
  - When we reach the cycle's end, the gate output has already stabilized on 0

Example: Considering the Not gate [2/2]

- Since we *probe* the state of the world only at cycle ends, we *don't get to see* **the interim time delays**
  - Rather, it appears as if we fed the gate with 0, and poof! the gate responded with 1
- If we make the same observations at the end of each cycle?
  - We can generalize that when a Not gate is fed with some binary input x, it instantaneously outputs Not (x)

She tasted the largest bowl, which belonged to the Big Bear, and found it too cold; then she tasted the middle-sized bowl, which belonged to the Middle-sized Bear, and found it too hot; then she tasted the smallest bowl, which belonged to the Little Bear, and it was just right, and she ate it all.

Goldilocks And The Three Bears by Robert Southey



GOLDILOCKS & THE CLOCK CYCLE

COMPUTER SCIENCE DEPARTMENT

For this scheme of discrete time representations to work?

- The cycle's length must be *longer than* **the maximal time delays** that can occur in the system
- Indeed, **cycle length** is one of the most important design parameters of any hardware platform!

The hardware engineer chooses the cycle length that meets two design objectives

- On the one hand, the cycle should be sufficiently long to contain, and **neutralize, any possible time delay**
- On the other hand, the **shorter the cycle, the faster the computer**
  - If things happen only during cycle transitions, then obviously things happen faster when the cycles are shorter
- To sum up: the cycle length is chosen to be slightly longer than the maximal time delay in any chip in the system

39

Once you see cycle length as the 'just right' parameter

- You'll immediately understand the two temptations of modern CPUs
  - Make it *shorter when you can*, and
  - *pretend it's safe* when you can't

40

A cutting-edge view of the clock cycle

- Following the tremendous progress in switching technologies, we are now able to create cycles as tiny as a billionth of a second
  - Achieving remarkable computer speed
- 9.13 GHz (9,130 MHz), achieved with a Core i9-14900KF
  - Using extreme liquid helium cooling in 2025
  - For consumer settings, the Intel Core i9-14900KS holds the record with a max turbo frequency of 6.2 GHz

Turbo Boost

- Several Intel chips support this
- Enhance clock speed **dynamically** based on *thermal headroom* in the processor to the maximum safe level
  - For example, number of cores are accounted for
    - Intel® Turbo Boost Technology

Overclocking

- AMD has always supported this feature
- Possible on Intel CPUs with a "K" in the name (signifying unlocked)
- Signifies an unlocked "**multiplier**" when used in tandem with a motherboard chipset that supports overclocking
- Make sure you have proper cooling!
  - Probably use water cooled systems
  - Overclocking typically not done on laptops

Overclocking: Disadvantages

- When you overclock, you are pushing more electricity through it
- CPU **heat**
  - You are using the CPU to do something you aren't supposed to do
    - Outside typical operating conditions
  - Introduces wear and tear
    - Shelf-life reduces
- While complete failure does not typically occur
  - Leads to a lot of **instability**

45

Memory

- Memory chips are designed to "**remember**", or store, information over time
- The low-level devices that facilitate this storage abstraction are named **flip-flops**, of which there are several variants

46

Data flip flop (DFF)

- DFF's interface includes a single-bit data input and a single-bit data output
- The DFF has a clock input that feeds from the **master clock's signal**
- Taken together, the data input and the clock input enable the DFF to implement the simple time-based behavior **`out(t) = in(t-1)`**
  - Where `in` and `out` are the DFF's input and output values
  - `t` is the current time unit

Like Nand gates, DFF gates lie deep in the hardware hierarchy [1/2]

- All the memory chips in the computer (registers, SRAM units, and counters) are based, at bottom, on DFF gates
- All these DFFs are connected to the same master clock, forming a huge **distributed "chorus line"**

Like Nand gates, DFF gates lie deep in the hardware hierarchy [2/2]

- At the end of each clock cycle, the outputs of all the DFFs in the computer commit to their inputs from the *previous* cycle
- At all other times, the DFFs are **latched**, meaning that changes in their inputs have no immediate effect on their outputs

DFF conduction operations

- This conduction operation effects any one of the system's numerous DFF gates many times per second
  - Depending on the computer's **clock frequency**
- Hardware implementations realize the time dependency using a dedicated clock bus that **feeds the master clock signal** *simultaneously* to all the DFF gates in the system

The contents of this slide-set are based on the following references

- Noam Nisan and Shimon Schocken. *The Elements of Computing Systems: Building a Modern Computer from First Principles*. 2nd Edition. ISBN-10/ ISBN-13: 0262539802 / 978-0262539807. MIT Press. [Chapter 3]
- Jonathan E. Steinhart. *The Secret Life of Programs: Understand Computers -- Craft Better Code*. ISBN-10/ ISBN-13 : 1593279701/ 978-1593279707. No Starch Press. [Chapter 3]
- https://en.wikipedia.org/wiki/Bitwise_operation