CS250: Foundations of Computer Systems [Memory]

Memory Addressing
The bigger the memory
  The more bits that you need
  To address each byte
For each byte's unique

Add a humble bit   and
  What you can address
   Doubles
Add another? and see it double again

The final length
  for those trying to leapfrog
  depends on the log
Of the   how much

Shrideep Pallickara
Computer Science
Colorado State University

1

Frequently asked questions from the previous class survey

- Difference between & and &&: & (bitwise) and && (loops, conditionals, etc.)
- Why quartz oscillators? Cheap, robust, exhibits very low phase noise, and frequency dependence on temperature can be very low
- Anything other than quartz for oscillators?
  - Silicon-based MEMS (Micro-Electro-Mechanical Systems), ceramic resonators
- How do fans know when to start cooling down?
- Computers in Antarctica: what do folks use there?

2

Topics covered in this lecture

- Combinational logic
- Sequential logic
- Data flip flop
- Random access memory

3

The Data Flip flop

Oh, it was just a glimpse of you, like looking through a window
Or a shallow sea
Could you see me?
And after all this time
It's like nothing else we used to know
And after all the hangers-on are done hanging on to the dead light
Of the afterglow
I've gotta know

Afterlife, Arcade Fire

4

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

Data flip flop

- 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 elements 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 elements in the system

What I want back is what I was.
Sylvia Plath

COMBINATIONAL & SEQUENTIAL LOGIC

Combinational

- Circuitry encompassing the elementary logic gates we have discussed previously were designed to respond only to changes that occur during the **current** clock cycle
- Such circuitry are called time-independent, or **combinational**
- The combinational name alludes to the fact that these circuitry respond only to different combinations of their input values
  - While paying **no attention** to the *progression of time*

Suppose we instruct the ALU to compute $x + y$ [1/2]

- x is the output of a nearby register, and y is the output of a remote RAM register
- Because of physical constraints like distance, resistance, and interference, the electric signals representing x and y will likely arrive at the ALU at different times

Suppose we instruct the ALU to compute $x + y$ [2/2]

- However, being a combinational chip, the ALU is insensitive to the concept of time
  - It continuously and happily adds up whichever data values happen to lodge at its inputs
- Thus, it will take some time before the ALU's output stabilizes to the correct $x+ y$ result

13

Until then, the ALU will generate garbage

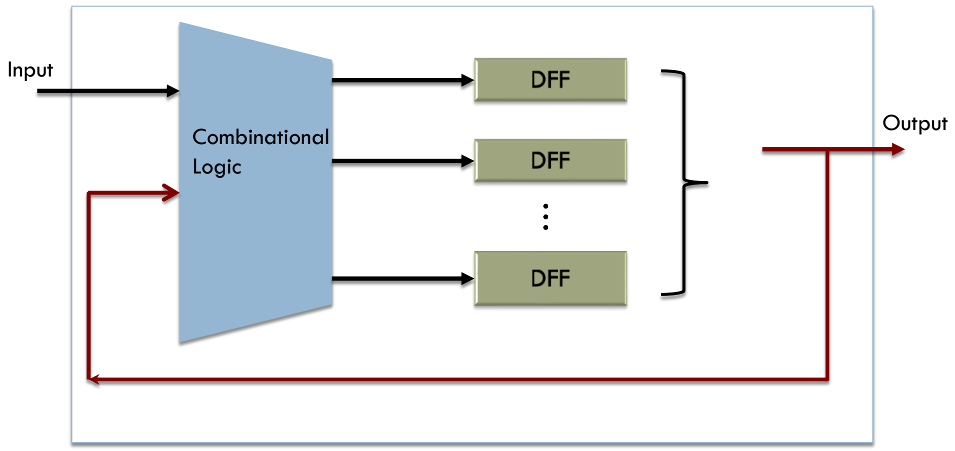
- How can we overcome this difficulty?
- Well, if we use a discrete representation of time, we simply don't care
- All we have to do is set the clock cycle duration effectively
  - Slightly longer than the time it takes a bit to travel the longest distance, plus the time it takes to complete the most time-consuming within-chip calculation
  - This way, we are guaranteed that by the end of the clock cycle, the ALU's output will be valid

14

Sequential circuits

- Circuits that are designed to respond to changes that occurred during **previous time units** (and possibly during the current time unit as well) are called **sequential**, or clocked
- The most fundamental sequential gate is the DFF, and any chip that includes it, either directly or indirectly, is also said to be sequential

Typical sequential logic configuration



Sequential logic configuration

- The main element in this configuration is a set of one or more circuits that include DFF chip-parts, either directly or indirectly
- These sequential circuits may also interact with combinational circuits
- The **feedback loop** enables the sequential circuit to respond to inputs and outputs from the previous time unit

In combinational chips, where time is neither modeled nor recognized

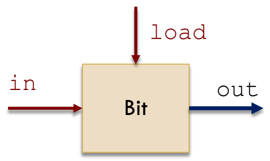
- The introduction of feedback loops is **problematic**
- The output of the chip would depend on its input, which itself would depend on the output, and thus the output would depend on itself

However …

- There is no difficulty in feeding outputs back into inputs, as long as the feedback loop goes through a DFF
- The DFF introduces an **inherent time delay** so that the output at time t does not depend on itself but rather on the output at time $t-1$

DFF in operation [1/2]

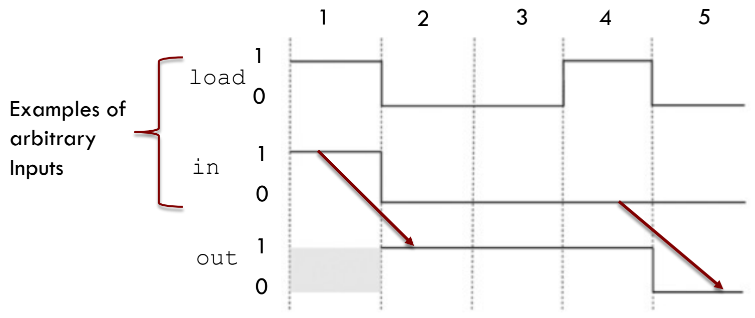
- A DFF has a single-bit data input, a single-bit data output, a clock input, and a simple time-dependent behavior:
  - `out(t) = in(t-1)`
- Usage:
  - If we put a one-bit value in the DFF's input, the DFF's state will be set to this value, and the DFF's output will emit it in the next time unit

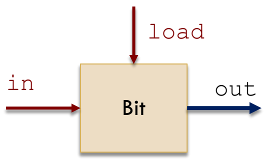


Function:

```
If load(t)
  out(t+1) = in(t)
else
  out(t+1) = out(t)
```

DFF in operation [2/2]





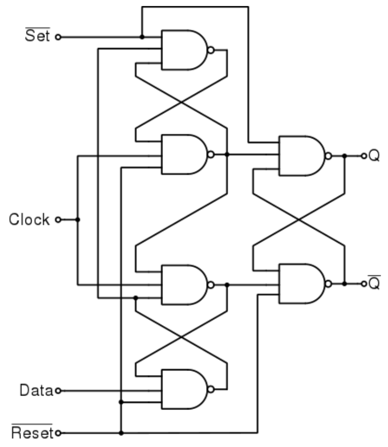
Function:

```
If load(t)
  out(t+1) = in(t)
else
  out(t+1) = out(t)
```

DFF implementation

- A DFF is designed to be able to "flip-flop" between two stable states, representing 0 and representing 1
- This functionality can be implemented in several different ways, including ones that use **Nand** gates only
- The Nand-based DFF implementations are elegant, yet intricate

A data flip-flop: Under the hood



23

RANDOM ACCESS MEMORY

24

The primary memory used in a computer is known as **main memory**

- Often, it's referred to as just memory or random access memory (**RAM**)
- It's **volatile**
  - Meaning it only retains data while powered
- The "random access" part of RAM means that any **arbitrary memory location** can be accessed
  - In *roughly the same amount of time* as any other location

There are two common types of computer memory

- Static random access memory (SRAM)
- Dynamic random access memory (DRAM)
- In both types, the basic unit of memory storage is a **memory cell**, a circuit that can store a single bit

SRAM vs DRAM [1/3]

- In SRAM, memory cells are a type of **data flip-flop**
- SRAM is **static** because its flip-flop memory cells *retain* their bit values while power is applied
- SRAM is **faster** but **more expensive**
  - So it's used in scenarios where speed is critical, such as in **cache memory**
- Space considerations
  - It takes six transistors for each bit
  - SRAM isn't a great choice for storing billions or trillions of bits

27

SRAM vs DRAM [2/3]

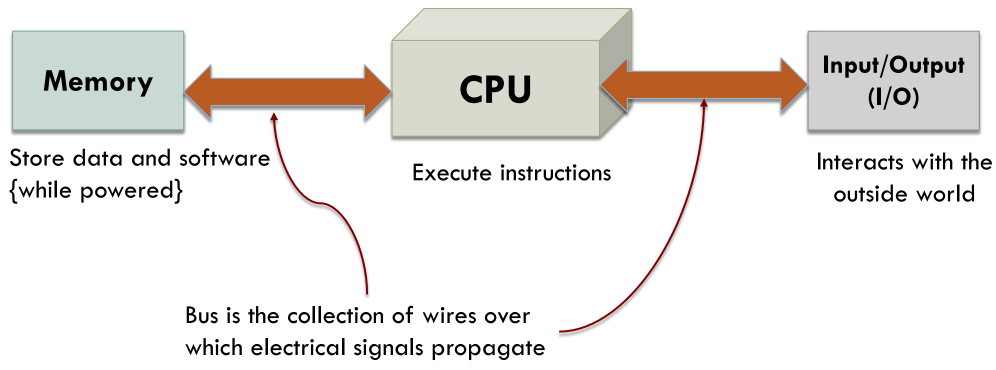
- DRAM memory cells are implemented using a transistor and a capacitor
- The capacitor's charge **leaks** over time, so data must be *periodically rewritten to the cells*
- This **refreshing** of the memory cells is what makes DRAM *dynamic*

28

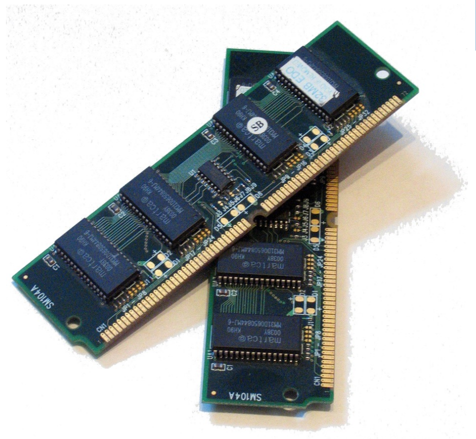
SRAM vs DRAM [3/3]

- Today, DRAM is commonly used for large main memory chips
  - Due to its **relatively low price**
  - Because of its **high density** (number of bits per area)
- Typically packaged as dual inline memory module (DIMM)
  - i.e., pins on both sides of the chip
  - 204 pins for DDR3 and 260 pins for DDR4
  - Refresh rate 32-64 milliseconds

A high-level view of the computer (von Neumann architecture)


Memory
Store data and software {while powered}
CPU
Execute instructions
Input/Output (I/O)
Interacts with the outside world
Bus is the collection of wires over which electrical signals propagate

Time moves in one direction, memory in another.
— William Gibson



MAIN MEMORY

COMPUTER SCIENCE DEPARTMENT COLORADO STATE UNIVERSITY

31

Memory is byte-addressable

- Accessing a single bit at a time isn't very efficient, so the RAM accesses **multiple grids** of 1-bit memory cells *in parallel*
  - Allowing for reads or writes of multiple bits at once – for e.g., a whole byte
- The location of a set of bits in memory is known as a **memory address**
  - a numeric value that *identifies* a **memory location**
- It's common for memory to be **byte-addressable**, meaning a single memory address refers to 8 bits of data

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT MEMORY L8.32

32

Let's consider a fictitious computer system that can address up to 64KB of memory [1/4]

- A tiny amount for a computer, but it's still useful for us as an example
- Let's also imagine that our fictitious computer's memory is byte-addressable; each memory address represents a single byte
- We need *one unique address* for **each byte** of memory
  - Since 64KB is 64 × 1024 = 65,536 bytes → we need 65,536 unique addresses

Let's consider a fictitious computer system that can address up to 64KB of memory [2/4]

- Each address is just a number, and memory addresses usually start with 0
- Our range of addresses is 0 to 65,535 (or 0xFFFF)

Let's consider a fictitious computer system that can address up to 64KB of memory [3/4]

- Memory addresses are ultimately represented in binary
- How many bits do we need to represent a memory address on this system?
  - The number of unique values that can be represented by a binary number with n bits is equal to 2^n
- So, we want to know the value of n for $2^n = 65{,}536$

Let's consider a fictitious computer system that can address up to 64KB of memory [4/4]

- We wish to know the value of n for $2^n = 65{,}536$
- The **inverse** of raising 2 to a certain power is the base-2 logarithm
- Therefore $\log_2(2^n) = n$ and $\log_2(65{,}536) = 16$
- Stated another way, $2^{16} = 65{,}536$
  - Therefore, a 16-bit memory address is needed to address 65,536 bytes

Why does the number of bits matter?

- The number of bits used to represent a memory address is a key part of a computer system's design
- It **limits the amount of memory** that a computer can access, and it impacts how programs deal with memory at a low level

Modern computing systems have 64-bit addressing

- A 64-bit address range should *suffice for a long time* as far as memory is concerned
- Unlikely you will ever put 2^{64} bytes of memory into a computer system and feel that you need more
  - 16,000,000 Terabytes or 16,000 Petabytes!
- Of course, people have made claims like this in the past
  - A few years ago, no one thought a computer would need 1GB of memory, yet computers with 64GB of memory (or more) are very common today

Let's say you end up needing even more addressing

- You would increase in powers of two
- You would first double to 128– and then …
  - 256– bits of addressing

Why 256–bits is effectively infinity …

- 2^{256} is effectively infinity for one simple reason
  - It's **physically impossible** to build that much memory based on estimates of the *current size* of the universe
    - About 2^{246} different elementary particles
- Unless you can attach 1 byte of memory to every elementary particle on the planet
  - You won't even come close to approaching 2^{256} bytes of memory on a given computer system
- Maybe we really will use whole planets as computer systems one day, as Douglas Adams predicted in *The Hitchhiker's Guide to the Galaxy*

MEMORY ADDRESSING

Suburbia is where the developer bulldozes out the trees, then names the streets after them.
— Bill Vaughan

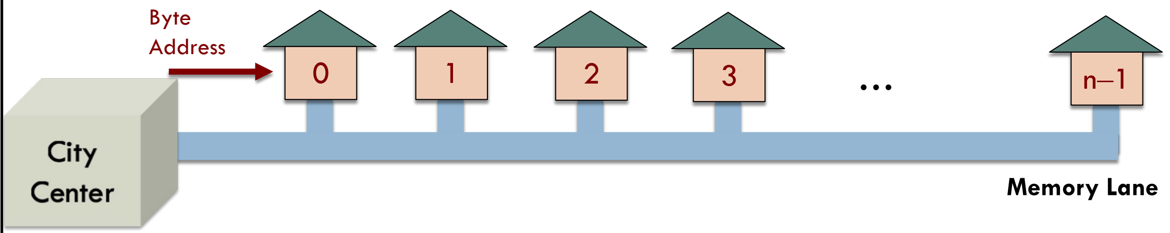
41

42

Memory is like a long street full of houses [1/2]

- Each house is exactly the same size and has room for a certain number of bits
- Building codes have pretty much settled on 1 byte per house
- And just like on a real street, each house has an address, which is just a number
- It's pretty common to refer to a **memory location**, which is just memory at a particular address, such as 3 Memory Lane

Memory is like a long street full of houses [2/2]

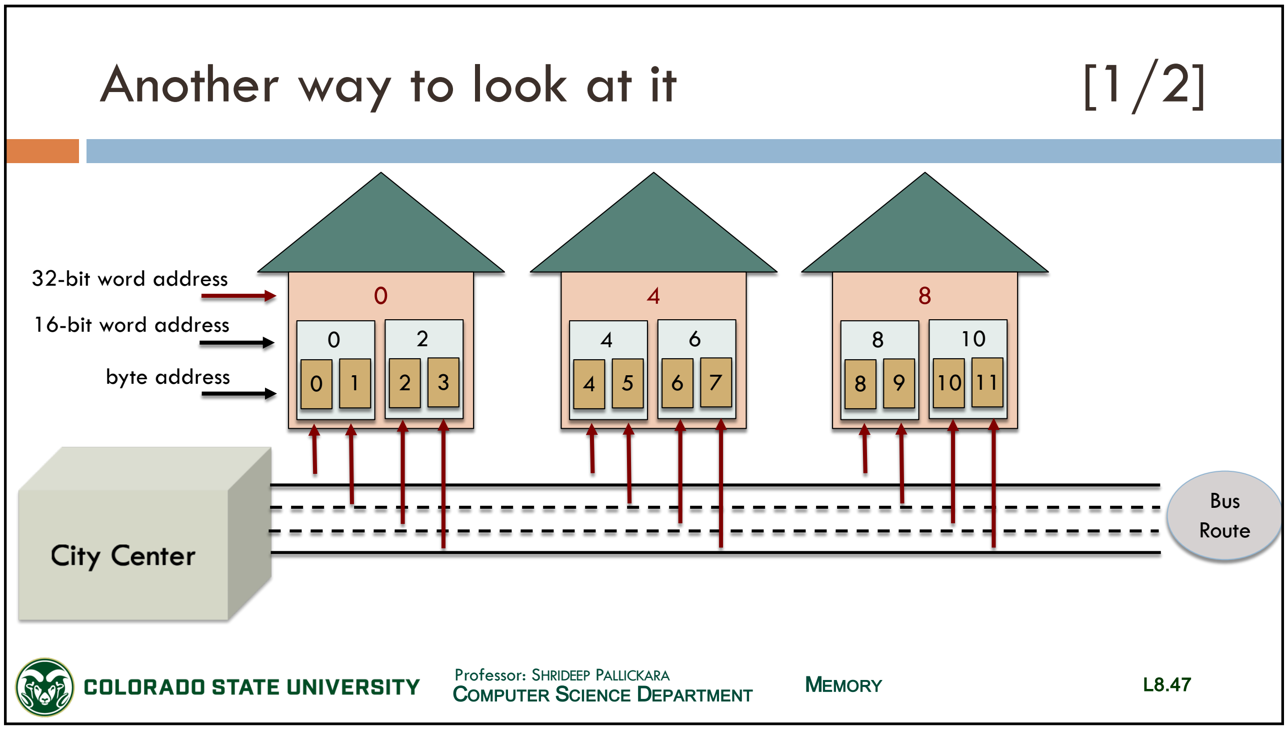


Just because the basic unit of memory is a byte doesn't mean we always look at it that way

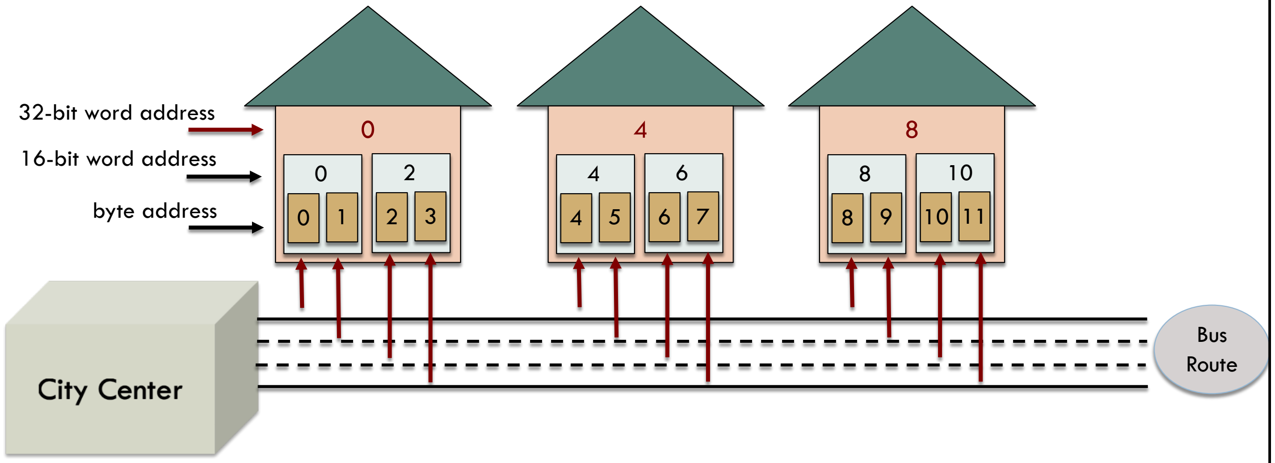
- For e.g., 32-bit computers usually organize their memory in 4-byte **chunks**
  - While 64-bit computers usually organize their memory in 8-byte chunks
- Why does that matter?
  - It's like having a four- or eight-lane highway instead of a one-lane road
  - Huge, positive impact on **throughput**

More lanes …

- More lanes can handle more traffic because **more bits can get on the data bus**
- When we address memory, we need to know what we're addressing
- Addressing 32-bit words is different from addressing bytes because there are 4 bytes to that word on a 32-bit computer
  - And 8 bytes to a long word on a 64-bit computer

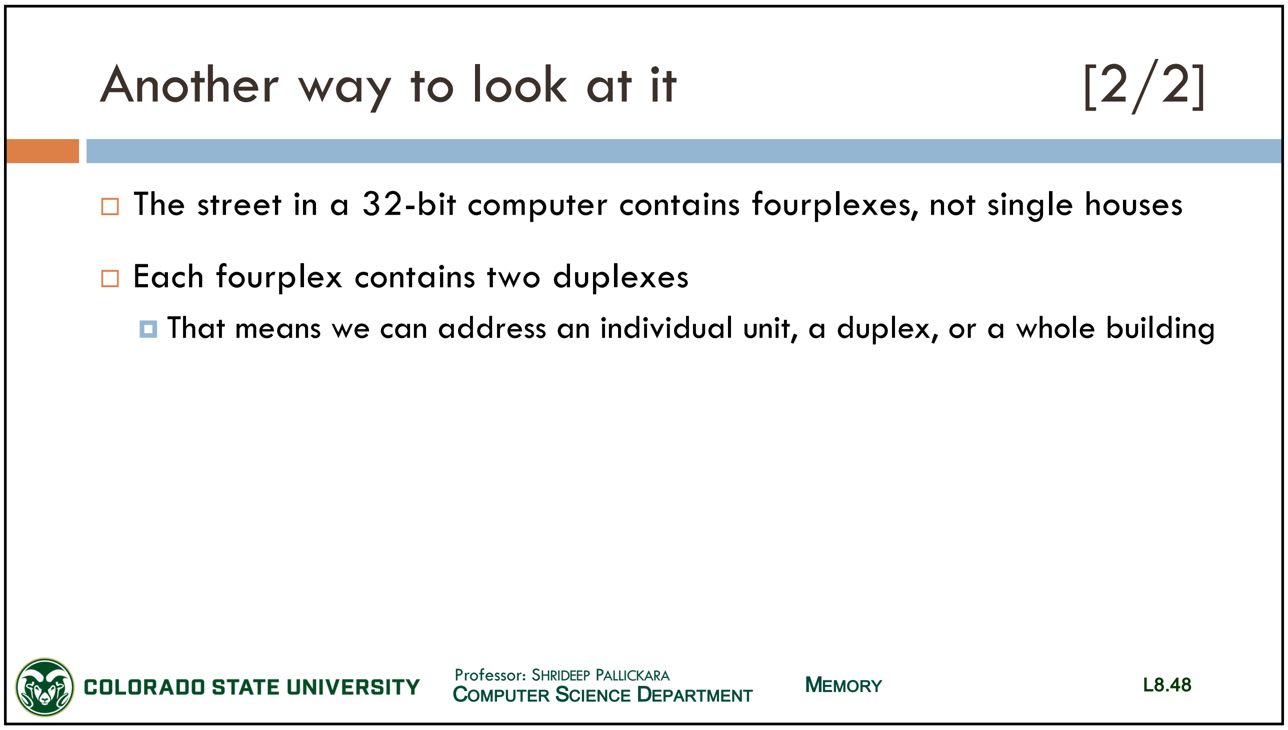


Another way to look at it [1/2]



COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT MEMORY L8.47

47



Another way to look at it [2/2]

- The street in a 32-bit computer contains fourplexes, not single houses
- Each fourplex contains two duplexes
  - That means we can address an individual unit, a duplex, or a whole building

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT MEMORY L8.48

48

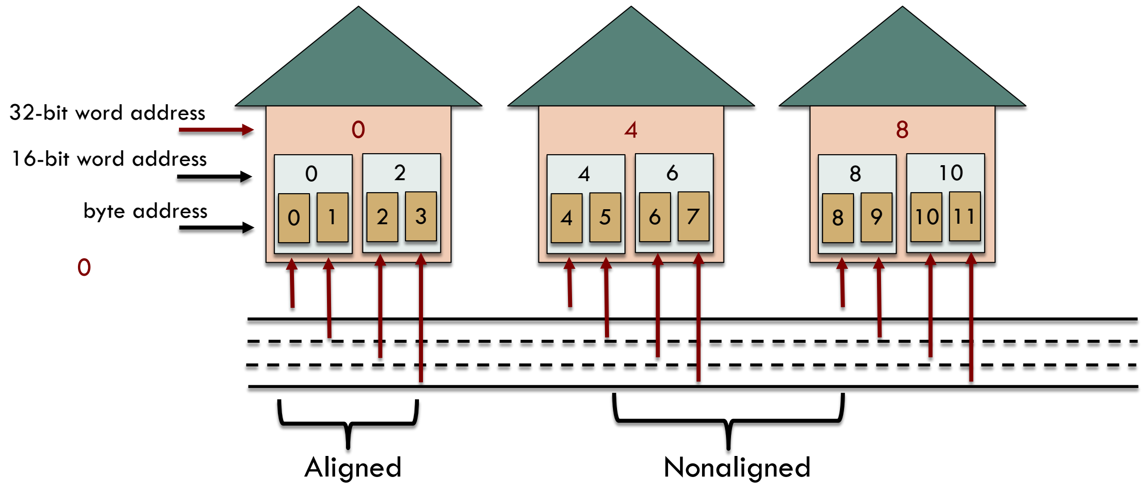
How commutes occur …

- Notice that each building straddles the highway such that **each byte has its own assigned lane**
  - And a 32-bit word takes up the whole road
- Bits commute to and from City Center on a bus that has four seats, *one for each byte*
- The doors are set up so that there's **one seat for each lane**
- On most modern computers, the bus stops only at one building on each trip from City Center

On most modern computers, the bus stops only at one building on each trip from City Center [1/2]

- This means we can't do things like form a long word from bytes 5, 6, 7, and 8
  - Because that would mean that the bus would have to make two trips: one to building 4 and one to building 8
- Older computers contained a complicated loading dock that allowed this, but planners noticed that it wasn't all that useful
  - Cut it out of the budget on newer models
- Trying to get into two buildings at the same time is called a **nonaligned access**

On most modern computers, the bus stops only at one building on each trip from City Center [2/2]



51

Who gets to sit in which seat when commuting on the bus? [1/2]

- Does byte 0 or byte 3 get to sit in the **leftmost** seat when a 32-bit word heads into town?
- It depends on the processor you're using, because designers have made them both ways
- Both work, so it's pretty much a theological debate
  - The **big-endian** and **little-endian** debate

52

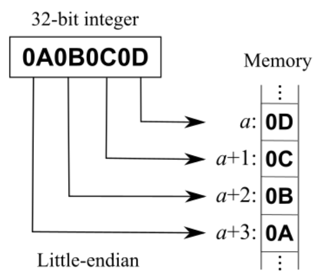
Endian

- The term endian is based on the royal edicts in Lilliput and Blefuscu in Jonathan Swift's *Gulliver's Travels*
  - Regarding which was the proper end on which to crack open a soft-boiled egg
- Is used to describe the difference
  - Byte 0 goes into the rightmost seat in little-endian machines like Intel processors

Little-endian vs Big-endian

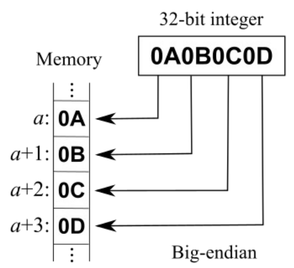
Byte 3	Byte 2	Byte 1	Byte 0

Little-endian



Byte 0	Byte 1	Byte 2	Byte 3

Big-endian



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]
- Randall Hyde. Write Great Code, Volume 1, 2nd Edition: Understanding the Machine 2nd Edition. ASIN: B07VSC1K8Z. No Starch Press. 2020. [Chapter 11]
- Matthew Justice. *How Computers Really Work: A Hands-On Guide to the Inner Workings of the Machine*. ISBN-10/ISBN-13 : 1718500661/ 978-1718500662. No Starch Press. 2020. [Chapter 7]