CS250: Foundations of Computer Systems [Computer Architecture]

The Stored Program Concept
A limited repertoire you say?
Well … it's just like Lego
Only with a surfeit of pieces
Mix, match and combine in ways
That are infinite
Literally and metaphorically
All leading to a machine
With limits
constrained only by creativity

Shrideep Pallickara
Computer Science
Colorado State University

1

Frequently asked questions from the previous class survey

- DNS
  - Do individual pages have mappings?
- Does DHCP i.e., addresses assigned by providers, change from ISP to ISP?
  - Renewal periods? 30-120 minutes
- Who assigns a static IP address?
- When does a server and a client connect to each other? When you send data? When you read?
- OSI: Did it ever take off?

2

Topics covered in this lecture

- Stored Program Concept
- The von Neumann architecture
- Midterm-II

3

4

Etymology Tidbit: Program "execution"

- Execute
  - Verb
  - Carry out or put into affect a plan, order, or course of action
- Contrary to some lore
  - No bits are actually "killed" when a program "executes"

The Stored Program Concept [1/2]

- Compared to all the machines around us, the most remarkable feature of the digital computer is its amazing **versatility**
- Here is a machine with a **finite and fixed hardware** that can perform an **infinite number of tasks**
  - From playing games to typesetting books to driving cars
- This remarkable versatility (a boon that we have come to take for granted) is the fruit of a brilliant early idea called the **stored program concept**

The Stored Program Concept [2/2]

- Formulated independently by several scientists and engineers in the **1930s**
- The stored program concept is still considered the **most profound invention** in, if not the very foundation of, modern computer science

The Stored Program Concept: Like many scientific breakthroughs, the basic idea is simple

- The computer is based on a **fixed hardware** platform capable of executing a fixed repertoire of simple instructions
- At the same time, these instructions can be **combined like building blocks**, yielding *arbitrarily* sophisticated programs

Also, the logic of these programs is not embedded in the hardware

- As was customary in mechanical computers predating 1930
- Instead, the program's code is **temporarily stored** in the computer's memory, like data
  - Becoming what is known as **software**

Since the computer's operation manifests itself to the user through the currently executing software …

- The same hardware platform can be made to **behave completely differently** each time it is loaded with a different program

The stored program concept in models of computing

- The stored program concept is the key element of both abstract and practical computer models
  - Notably the Turing machine (1936) and the von Neumann machine (1945)
- The Turing machine is an **abstract artifact** describing a deceptively simple computer
  - Is used mainly in *theoretical* computer science for analyzing the logical foundations of computation
- In contrast, the **von Neumann** machine is a **practical** model that informs the construction of almost all computer platforms today

Make everything as simple as possible, but not simpler.
—Albert Einstein (1879–1955)

THE VON NEUMANN ARCHITECTURE

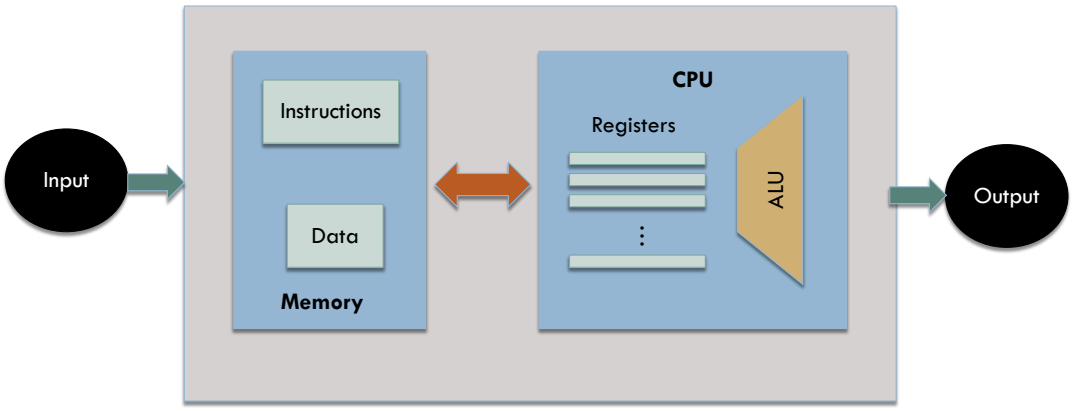
13

The von Neumann architecture

- The von Neumann architecture is based on a Central Processing Unit (CPU)
  - Interacting with a memory device
  - Receiving data from some input device, and
  - Emitting data to some output device
- At the heart of this architecture lies the stored program concept:
  - The computer's **memory stores** not only the **data** that the computer manipulates
  - But also, the **instructions** that tell the computer what to do

14

A generic von Neumann computer architecture



15

The computer's Memory can be discussed from both physical and logical perspectives

- Physically, the memory is a **linear sequence of addressable bytes**, each having a unique address and a value
- Logically, this address space serves two purposes: **storing data and storing instructions**
  - Both the "instruction words" and the "data words" are implemented exactly the same way: as sequences of bits

16

Instruction Memory [1/2]

- Before a high-level program can be executed on a target computer, it must first be **translated into the machine language** of the target computer
- Each high-level statement is translated into one or more low-level instructions
  - Written as binary values to a file called the binary, or **executable**, version of the program
- Before running a program:
  - **First load** its binary version from a mass storage device, and
  - **Serialize its instructions** into the computer's instruction memory

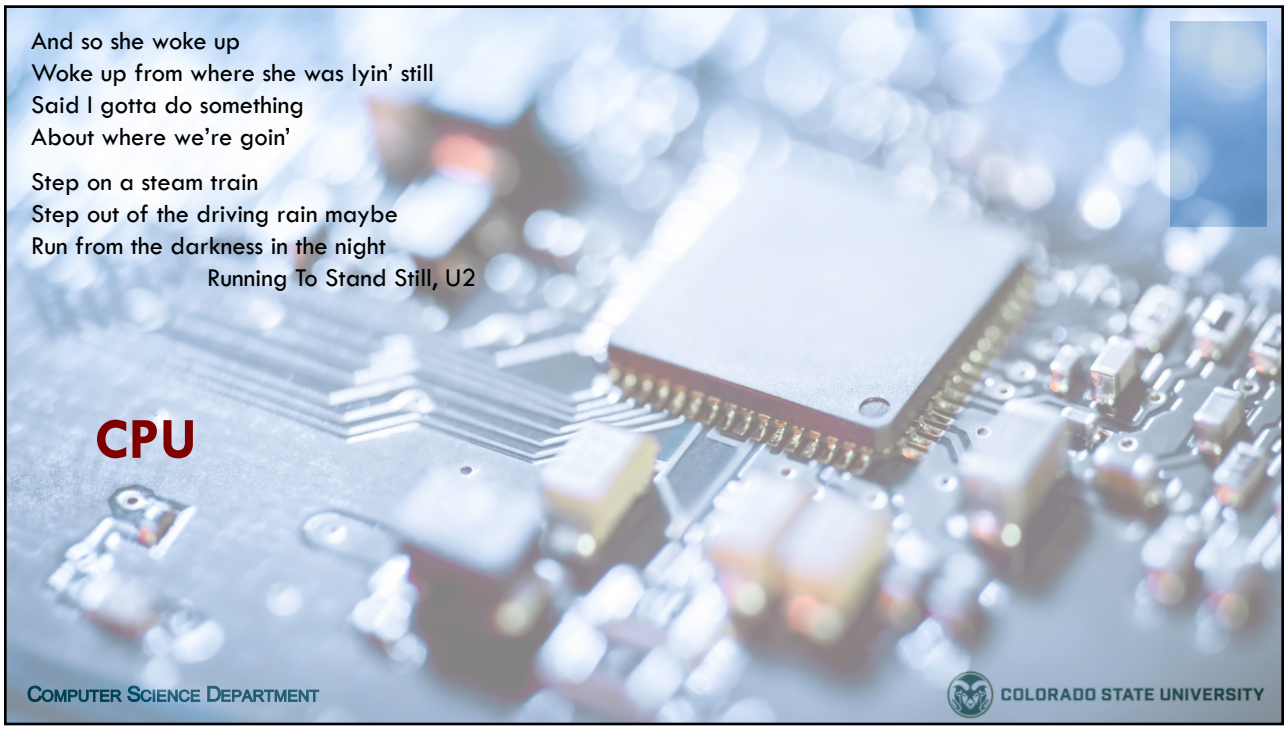
Instruction Memory [2/2]

- From the pure focus of computer architecture, how a program is loaded into the computer's memory is considered an external issue
- What's important is that *when* the CPU is called upon to execute a program
  - The program's code will **already reside in the computer's memory**

Data Memory

- High-level programs are designed to manipulate abstract artifacts like variables, arrays, and objects
- Yet at the hardware level, these data abstractions are realized by binary values stored in memory
- In particular, following translation to machine language
  - Abstract array processing and get/set operations on objects are reduced to reading and writing selected locations in memory

19



20

The CPU is the centerpiece of the computer's architecture

- The Central Processing Unit (CPU) is in charge of **executing the instructions** of the currently running program
- *Each* instruction tells the CPU which computation to perform, which registers to access, and which instruction to fetch and execute next
- The CPU executes these tasks using three main elements:
  - An Arithmetic Logic Unit (ALU)
  - A set of registers, and
  - A control unit

Arithmetic Logic Unit (ALU)

- The ALU chip is built to perform all the low-level arithmetic and logical operations featured by the computer
- A typical ALU can add two given values, compute their bitwise `And`, compare them for equality, and so on …

How much functionality should be packed into the ALU is a design decision

- In general, any function not supported by the ALU can be *realized later*, using system software running on top of the hardware platform
- The **trade-off** is simple:
  - Hardware implementations are typically more efficient but result in more expensive hardware
  - While software implementations are inexpensive and less efficient

Why we use registers [1/2]

- While performing computations, the CPU is often required to store **interim values** temporarily
- In theory, we could have stored these values in the RAM, but this would entail **long-distance trips** between the CPU and the RAM
  - The CPU and the RAM are two separate chips

Why we use registers [2/2]

- Long distance trips between the CPU and RAM result in **delays**
- These delays would frustrate the CPU-resident ALU, which is an ultra-fast combinational calculator
- The result will be a condition known as **starvation**
  - Happens when a fast processor depends on a **sluggish data store** for supplying its inputs and consuming its outputs

To avert starvation and boost performance

- CPUs are equipped with a **small set** of high-speed (and relatively expensive) registers, acting as the processor's immediate memory
- These registers serve **various purposes**:
  - Data registers store interim values
  - Address registers store values that are used to address the RAM
  - The program counter stores the address of the instruction that should be fetched and executed next
  - The instruction register stores the current instruction

27

Control

- A computer instruction is a structured package of agreed-upon **micro-codes**
  - Sequences of one or more bits that signal to different circuitry what to do
- Thus, before an instruction can be executed, it must first be decoded into its micro-codes
- Next, each micro-code is **routed** to its designated hardware circuitry (ALU, registers) within the CPU
  - Where it tells the device *how to partake* in the overall instruction execution

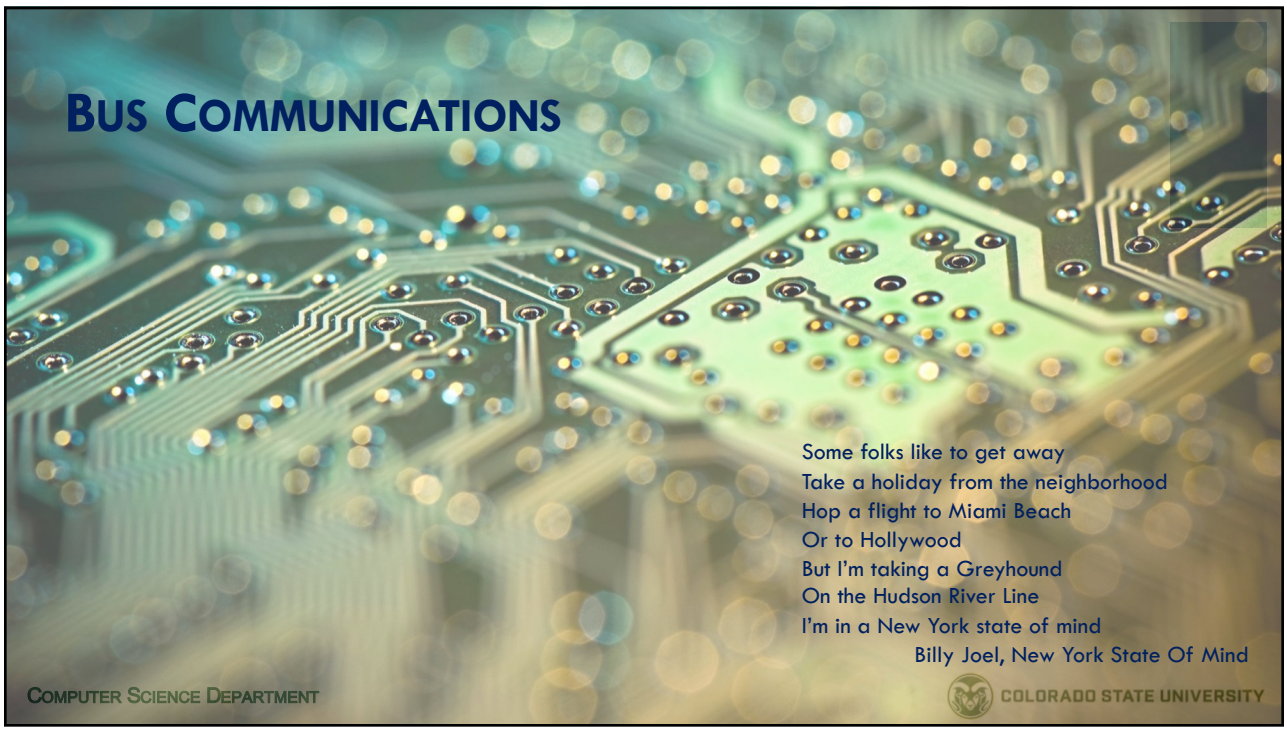
28

Fetch-Execute

- In each step (cycle) of the program's execution:
  - The CPU fetches a binary machine instruction from the instruction memory
  - Decodes it, and
  - Executes it
- As a side effect of the instruction's execution, the CPU also **figures out** *which instruction* to fetch and execute next
- This *repetitive process* is sometimes referred to as the **fetch-execute cycle**

CPU trends

- Multiprocessor systems with multiple CPUs debuted in the 1980s to get higher performance than could be achieved with a single CPU
- As it turns out, though, it's not that easy
- Dividing up a single program so that it can be parallelized to make use of multiple CPUs is an **unsolved problem in the general case**
  - E.g.: loops with complex dependencies, irregular algorithms (depth-first or breadth-first), some recursive algorithms, stochastic programs
  - Although it **works extremely well for <u>several</u> classes** of problems



31

A bus is a hardware communication system used by computer components

- In the early days of computers, a bus was simply a set of **parallel wires**, each carrying an electrical signal
  - This allowed multiple bits of data to be transferred in parallel; the voltage on each wire represented a single bit
- Today's bus designs aren't always that simple, but the **intent is similar**

32

There are **3** common bus types used in communication between the CPU, memory, and I/O devices [1/2]

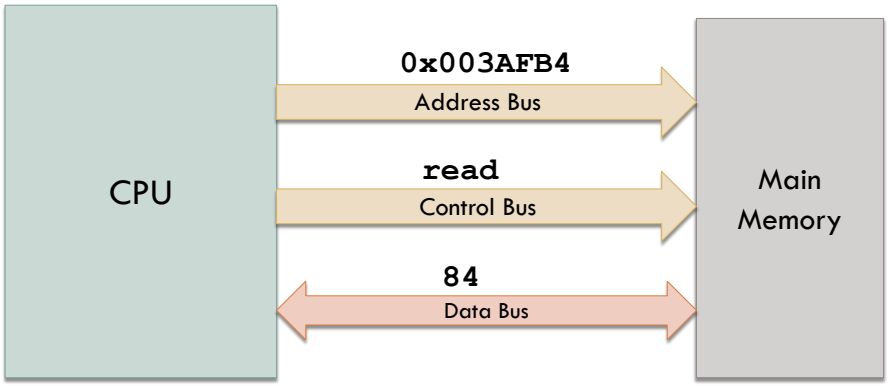
- An **address bus** selects the memory address that the CPU wishes to access
  - For example, if a program wishes to write to address 0x2FE, the CPU writes 0x2FE to the address bus
- The **data bus** *transmits* a value read from (or to be written to) memory
  - If the CPU is reading data from memory, that value is read from the data bus
  - If the CPU wishes to write 25 to memory, then 25 is written to the data bus

There are **3** common bus types used in communication between the CPU, memory, and I/O devices [2/2]

- A **control bus** manages the operations happening over the other two buses
  - For e.g., the CPU uses the control bus to indicate that a write operation will occur
  - Or the control bus can carry a signal indicating the status of an operation

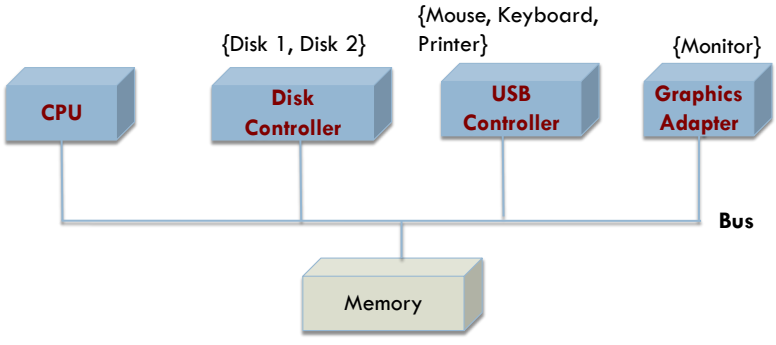
Bus Interactions: Example

The CPU requests a read of the value at memory location **0x003AFB4** which returns the value **84**



35

A simple bus-based structure

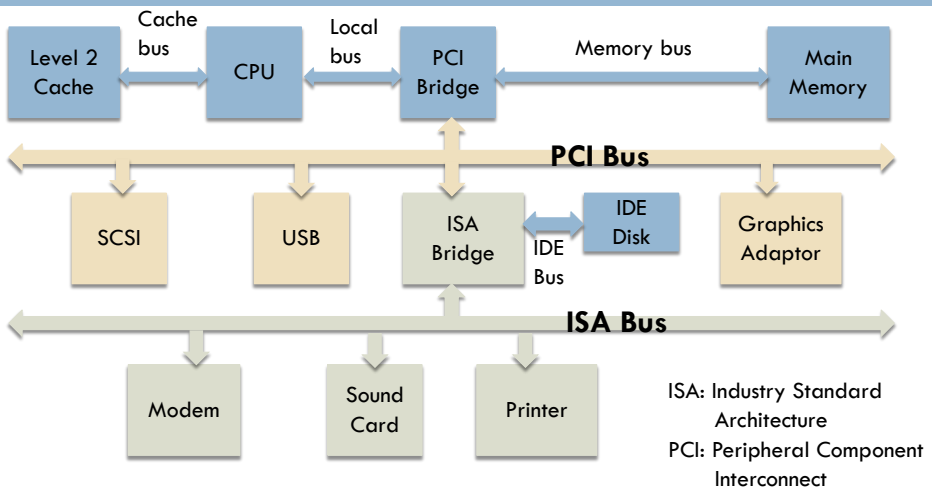


36

Limitations of the bus structure from the earlier slide

- As processors and memories got faster
  - Ability of a single bus to handle *all traffic* strained considerably
- Result?
  - Additional buses were added
  - For faster I/O devices and CPU-memory traffic

What a modern bus architecture looks like



There are two main BUS standards

- Original IBM PC ISA (Industry Standard Architecture)
- PCI (Peripheral Component Interconnect)
  - From Intel

The IBM PC ISA bus

- Runs at 8.33 MHz
- Transfers 2 bytes at once
- Maximum speed = 16.67 MB/sec
- Included for **backward compatibility**
  - Older and slower I/O cards

The PCI bus

- Can run at 66 MHz
- Transfer 8 bytes at once
- Data transfer rate: 528 MB/sec
- Most high-speed I/O devices use PCI
- Newer computers have an updated version of PCI
  - **PCI Express**

Other specialized buses: IDE (Integrated Drive Electronics) bus

- For attaching peripheral devices
  - CD-ROMs and Disks
- Grew out of the disk controller interface

Other specialized buses: USB (Universal Serial Bus)

- Attach **slow** I/O devices to the computer
  - Keyboard, mouse etc
- Uses a small **4-wire** connector
  - **Two** supply electrical power to the USB devices
- Centralized bus
  - Root device polls I/O devices every millisecond
    - Check if they have any traffic

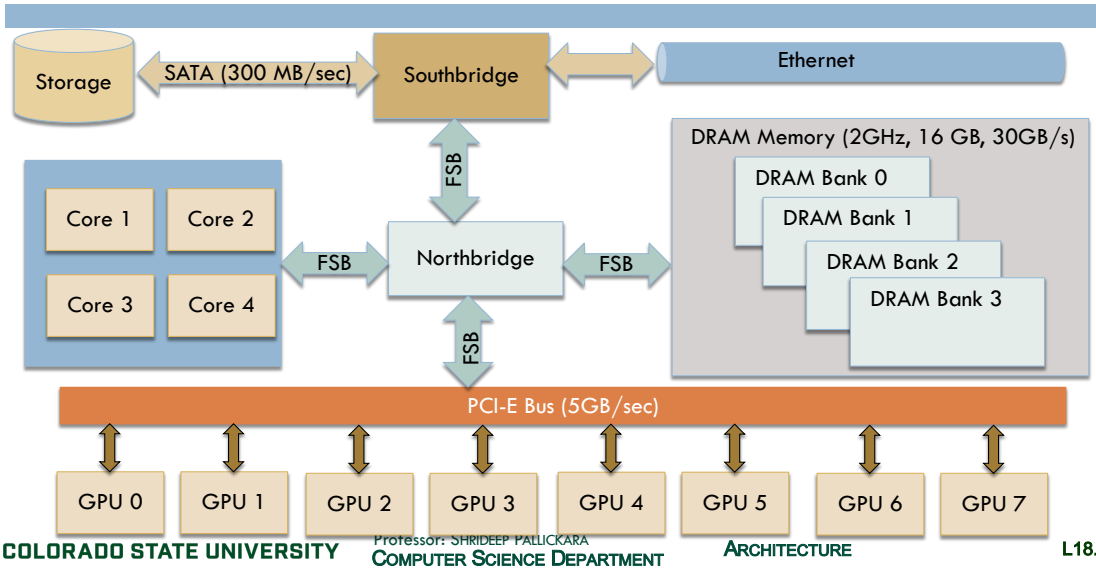
Some more information about USB

- All USB devices share a **single** USB device driver
  - *No need to install* a driver for each device
  - Can be added to computer *without need to reboot*
- USB 1.0 had a transfer rate of 1.5 MB/sec
- USB 2.0 went up to 60 MB/sec
- USB 3.0
  - Specification ready on 17 November 2008
  - Theoretical signaling rate: 600 MB/sec (4.8 Gbps)
  - USB 3.1: Jan 2013 goes up to 10 Gbps
  - US 3.2 released in September 2017 transfer modes 10 and 20 Gbps
- USB-C 24 pins: 16 for data transfer, 4 for power, and 4 ground
  - 40 Gbps (USB-4) and 80 Gbps (USB 4.2)

Other buses

- SCSI (Small Computer System Interface)
  - High performance bus
  - For devices that need high bandwidth
    - Fast disks, scanners
  - Up to 320 MB/sec
- IEEE 1394
  - Sometimes called FireWire (used by Apple)
  - Transfer speeds of up to 100 MB/sec
    - Camcorders and similar multimedia devices
  - No need for a central controller (unlike USB)

Typical PC architecture

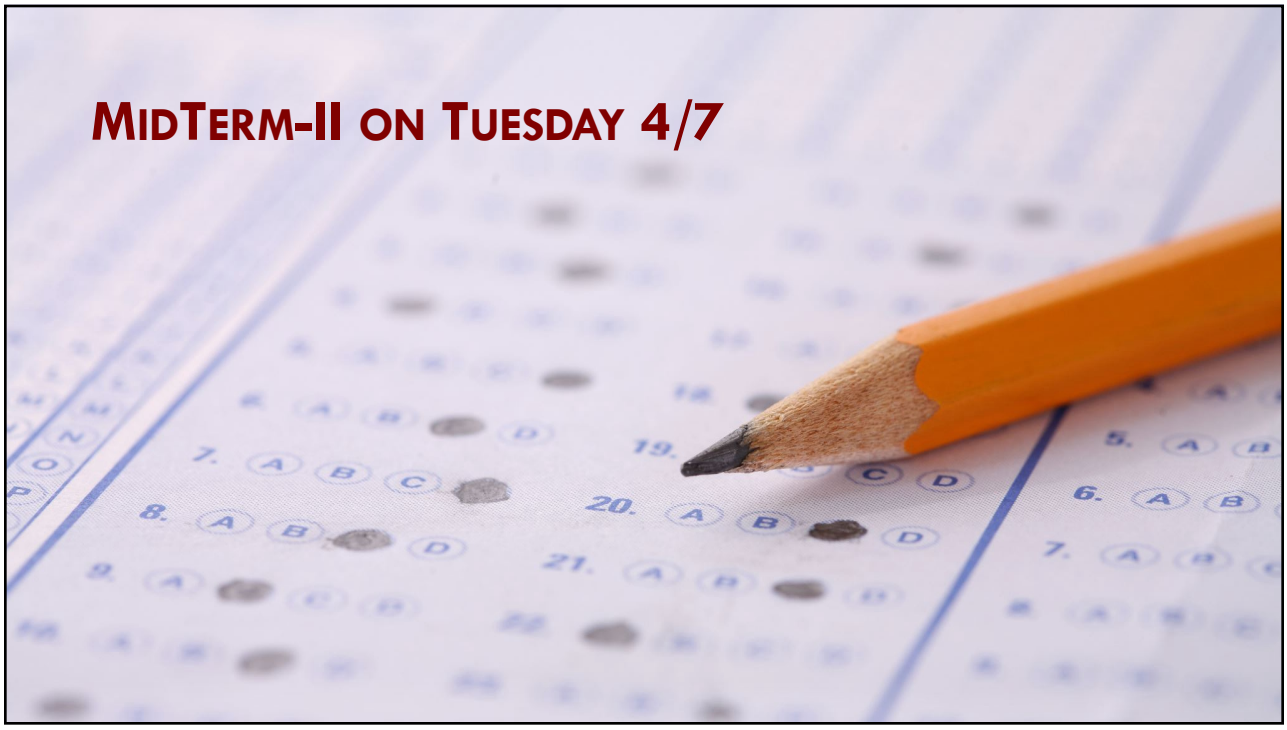


Typical desktop CPU/GPU layout

- All GPU devices are connected to the processor via the PCI-E bus
  - To get data from the processor, we need to go through the Northbridge device over the FSB (front-side bus)
- The FSB can run anything up to 1600 MHz clock rate, although in many designs it is much slower

The Northbridge/Southbridge chipsets

- The **Northbridge** chipset deals with all the high-speed components
  - Memory, CPU, PCI-E bus connections, etc.
- The **Southbridge** chip deals with the slower devices such as hard disks, USB, keyboard, network connections, etc.
- Since the 2010s, functions performed by northbridges are now often incorporated into other components
  - Die shrink and improved transistor density allow for chipset integration
  - Notably in 2019, both Intel and AMD released chipsets where all northbridge functions were integrated into the CPU



49

Will account for 15% of your course grade

- The exam itself is for 75 points
- Everything we covered in Networking
  - 50 points
- Everything besides networking including the lecture on Thursday
  - 25 points
- If you have SDC accommodations, please ensure that they are aware that you will take the test at the University Testing Center

50

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 5]
- Jonathan E. Steinhart. *The Secret Life of Programs: Understand Computers -- Craft Better Code*. ISBN-10/ ISBN-13 : 1593279701/ 978-1593279707. No Starch Press. [Chapter 5]
- Randall Hyde. Write Great Code, Volume 1, 2nd Edition: Understanding the Machine 2nd Edition. ASIN: B07VSC1K8Z. No Starch Press. 2020. [Chapter 11]