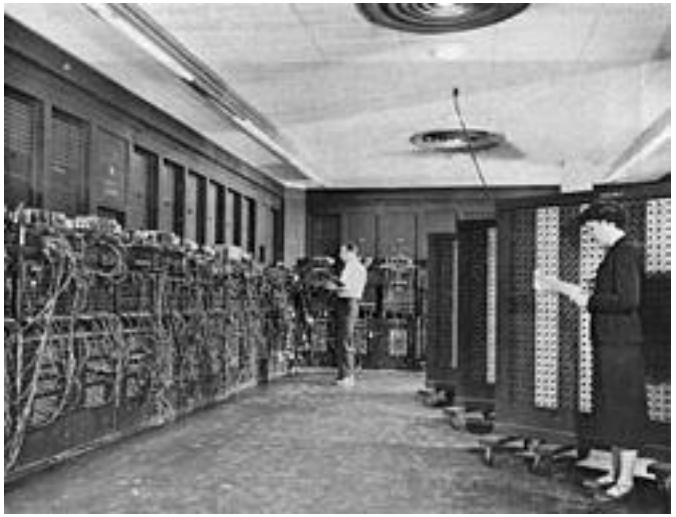
Operating Systems

CSC-4320/6320 –Summer 2014

Dustin Kempton (dkempton1@cs.gsu.edu)

Short Architecture Review

- ENIAC—Electronic Numerical Integrator and Calculator(1946)
  - Most consider this to be the first electronic general purpose computer
  - Used vacuum tubes instead of transistors
  - Programmed through punch cards and wires
  - Around 50,000+ times slower than my laptop (or possibly even your smart phone)
  - About 100ft long

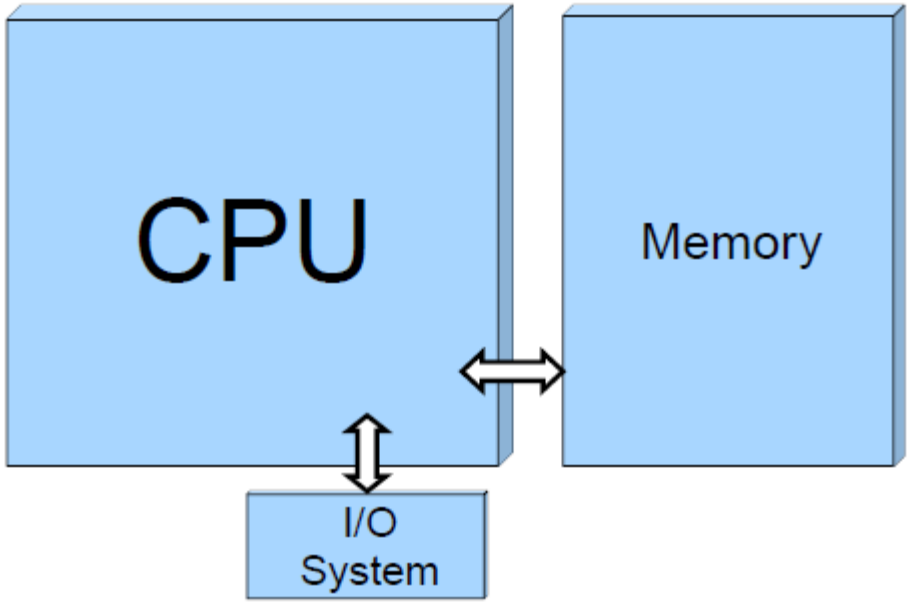


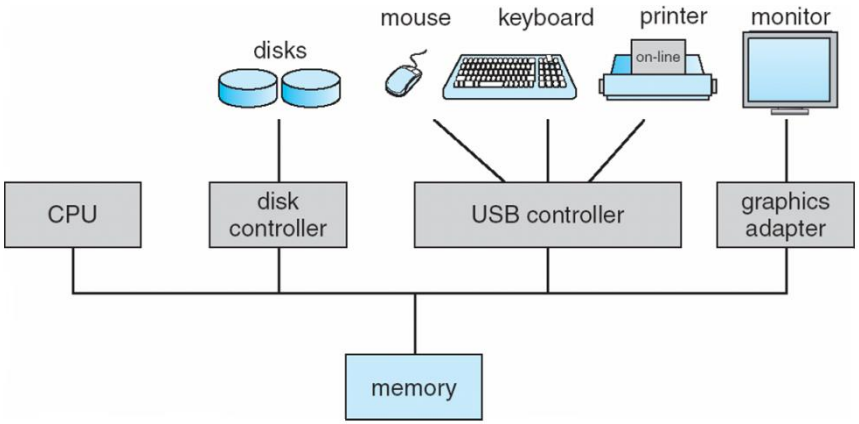
Von-Neumann

- In 1944, John von Neumann joined ENIAC
- His memo about computer architecture formalized the ides used in ENIAC
- These ides became the Von Neumann architecture model
  - A processor that performs operations and controls all that happens
  - A memory that contains code and data
  - I/O of some kind

Von-Neumann Model

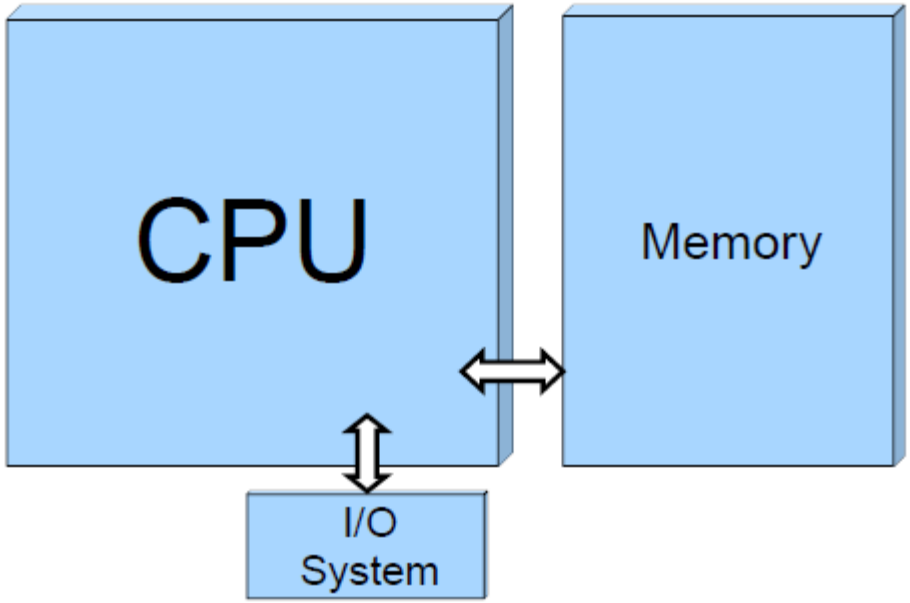
- We can still think of the computer in this fashion (a model from ~70 years ago)
- Though the computer today has a few more peripheral I/O devices

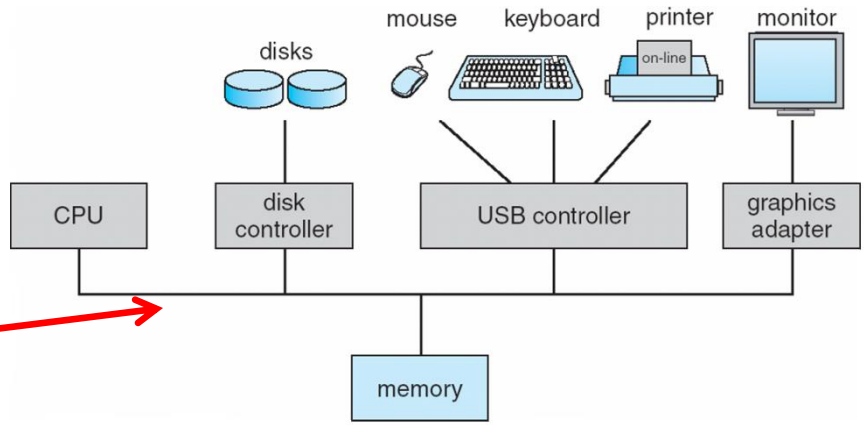




Von-Neumann Model

- We can still think of the computer in this fashion (a model from ~70 years ago)
- Though the computer today has a few more peripheral I/O devices

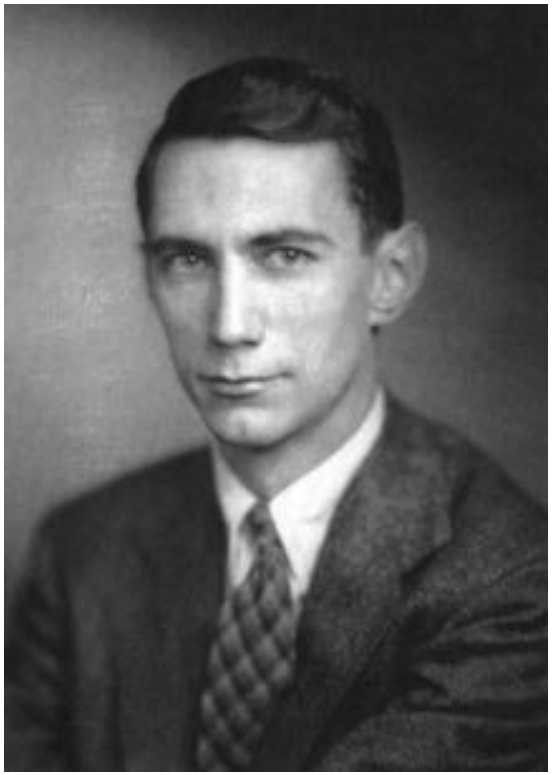




Memory buss allowing DMA

Data in Memory

- All “information” in a computer is in binary form
  - Ever since Claude Shannon’s M.S. thesis in the 30’s demonstrating that the electrical applications of Boolean algebra could construct and resolve any logical relationship
  - 0: zero voltage, 1: positive voltage (e.g., 5v historically but more often 3.3v or 1.8 in RAM or CPU now)
  - bit: the smallest unit of information (0 or 1)



Data in Memory

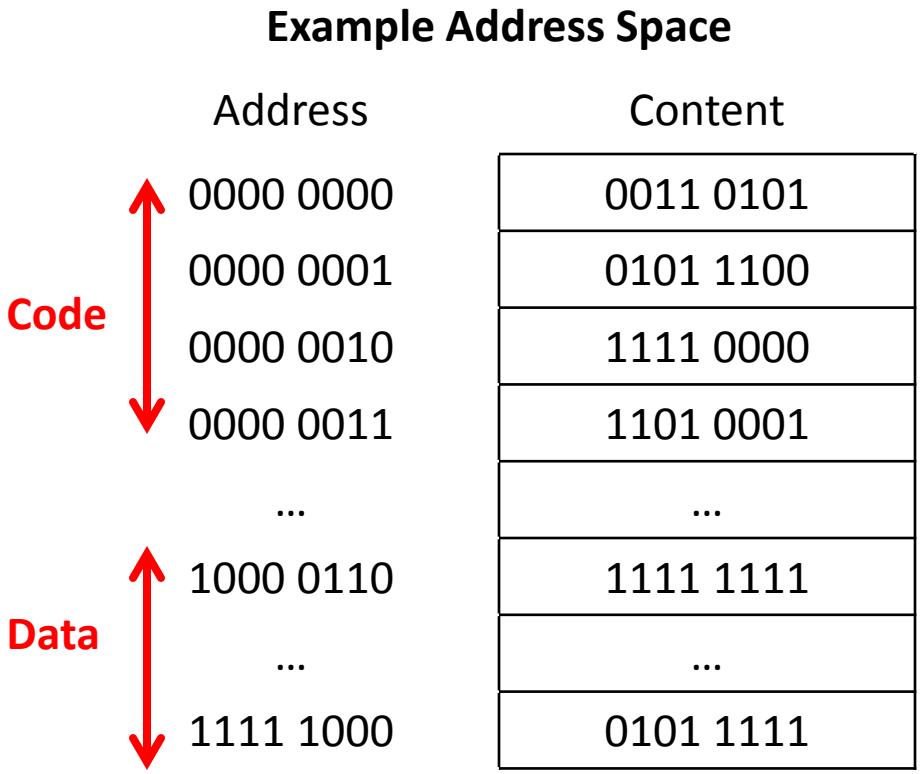
- The basic unit of memory is a byte
  - 1 Byte = 8 bits, e.g., “0101 1111”
- Each byte in memory is labeled with a unique address
- All addresses in the machine have the same number of bits
  - For example 16-bit addresses (or now more common with general purpose computers 64-bit)
- Processors have instructions that say “Read the byte at address X and give me its value” similarly “Write some value into the byte at address X”

Conceptual View of Memory

address	content
0000 0000 0000 0000	0011 0101
0000 0000 0000 0001	0101 1100
0000 0000 0000 0010	1111 0000
0000 0000 0000 0011	1101 0001
0000 0000 0000 0101	0000 0000
0000 0000 0000 0110	1111 1111
0000 0000 0000 0111	1100 0011
0000 0000 0000 1000	0101 1111
…	…

Code and Data in Memory

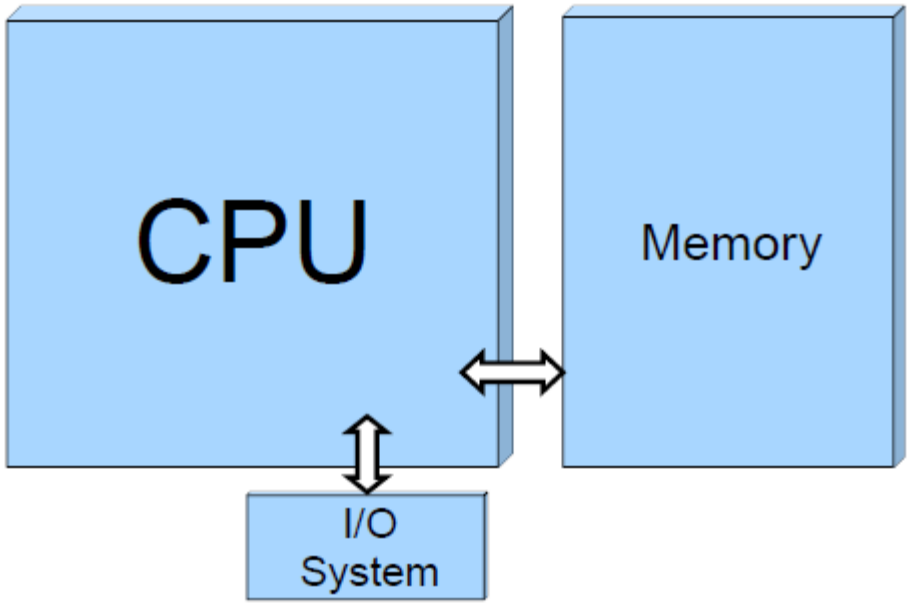
- When a program is loaded into memory, its address space contains both code and data
- To the CPU, there is no real difference, but the programmer knows which bytes are data and which are code
  - This is generally hidden from you unless you program in assembly
  - We will have to keep code/data straight for our lectures



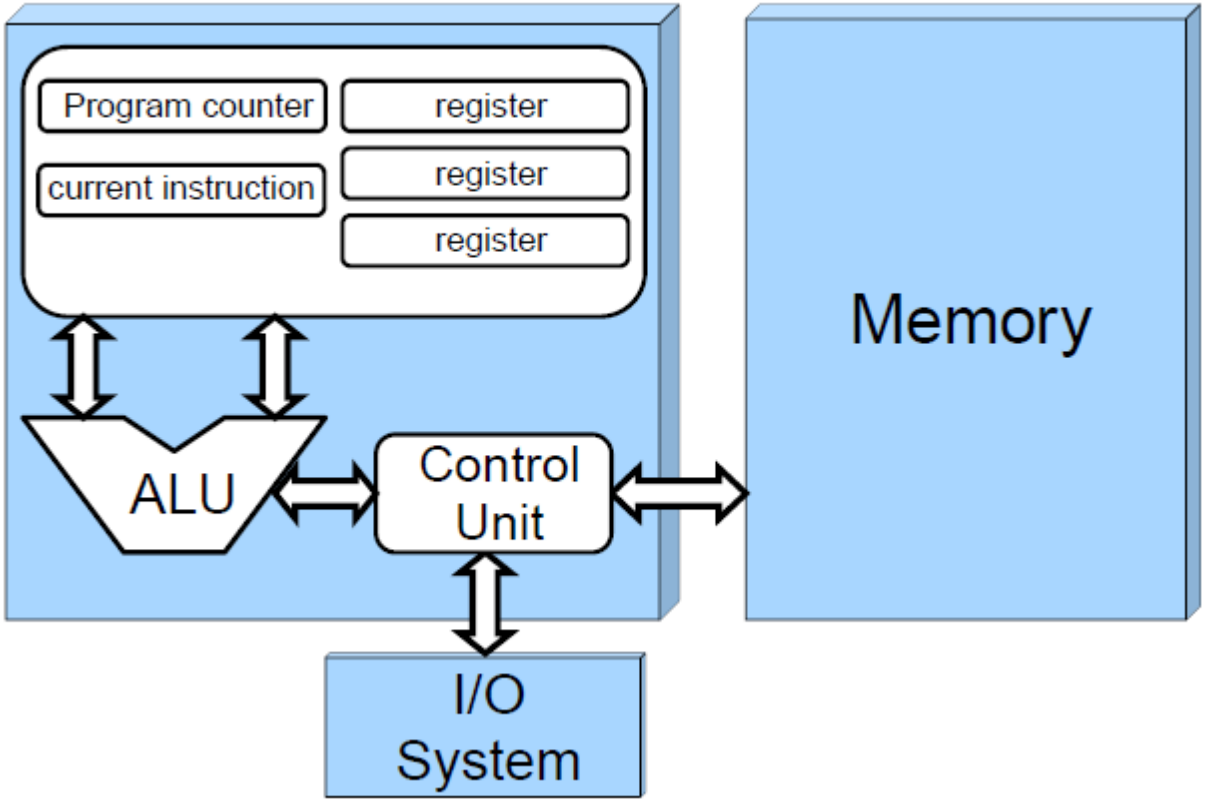
The CPU

- We have data and code in memory, now we need some way to execute a function upon this information
- Enter the CPU
  - These bytes presumably have some useful meaning
    - For instance, integers, ASCII codes of characters, floating point numbers, RGB values…
    - Possibly even, instructions that tell the computer what to do with the data; what instructions each pattern of 0010, 1100 means to the processor is set by the vendor of said processor
  - The CPU is the hardware that modifies the memory content
    - It can be thought of as the device that takes the information from one memory state and creates another memory state

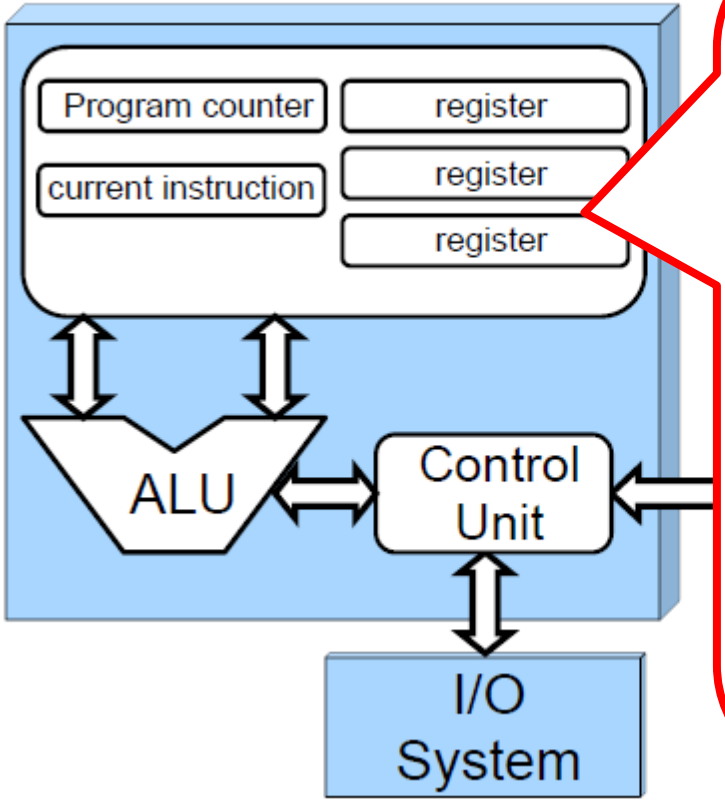
So, what is the CPU?



So, what is the CPU?



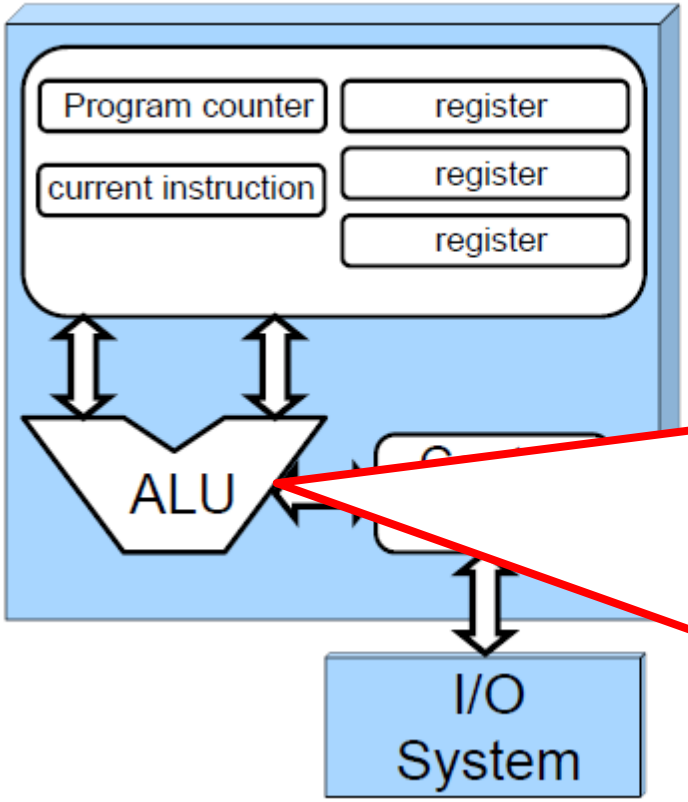
So, what is the CPU?



Registers: the "variables" that hardware instructions work with

- Data is loaded from memory into a register
- Data is also stored from a register back to memory
- The operands and results from computations are in registers
- There are a limited number of registers

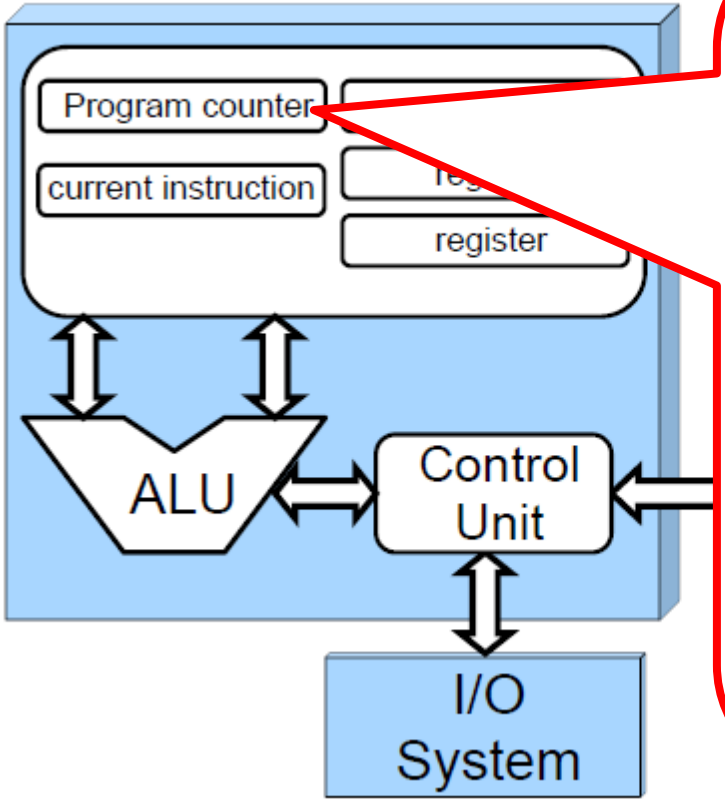
So, what is the CPU?



Arithmetic and Logic Unit: the component of the processor that does the actual computation

- Used to compute a value based upon the current register values and stores the result back into a register
- +,-,/,*,OR,AND,XOR,…

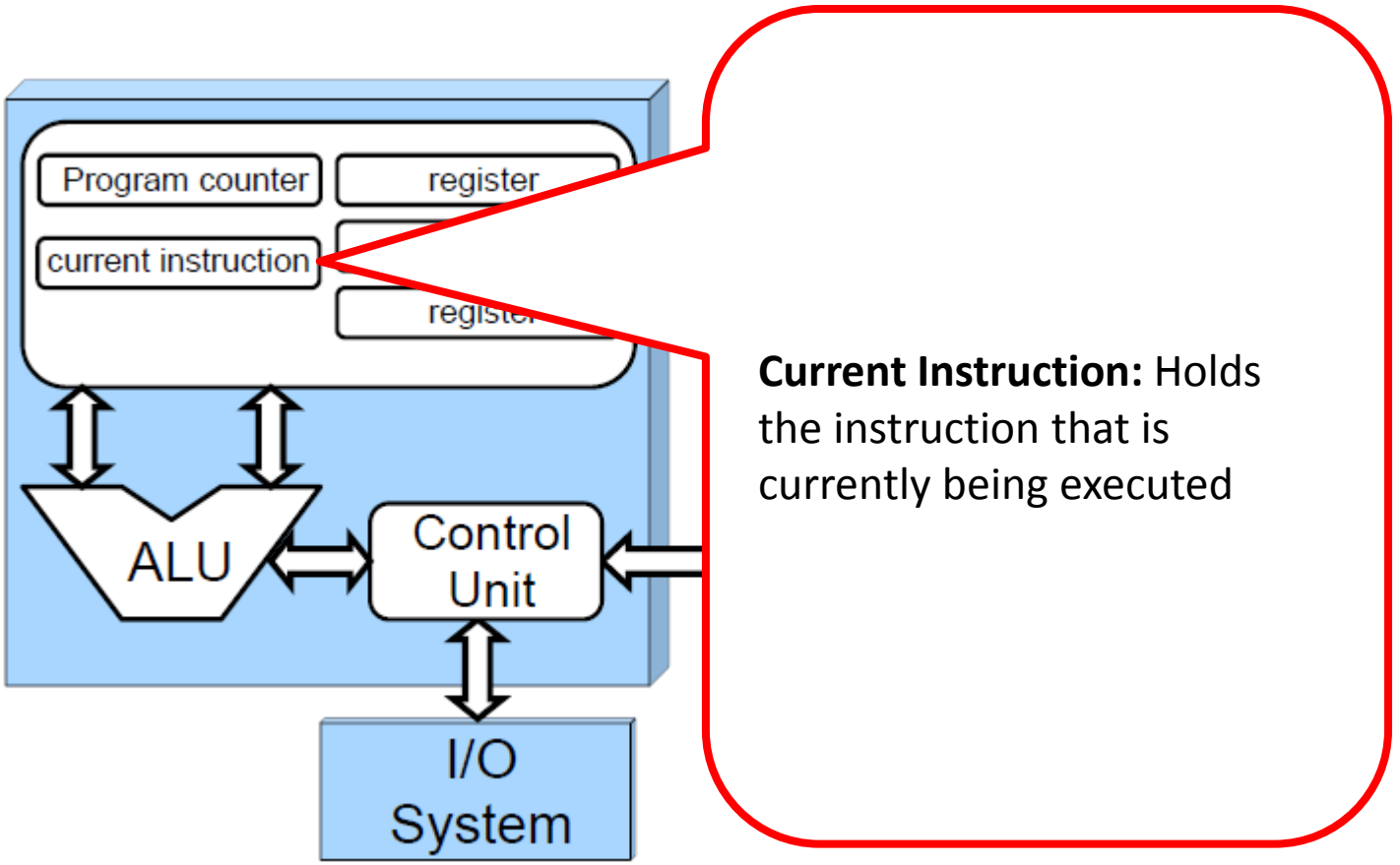
So, what is the CPU?



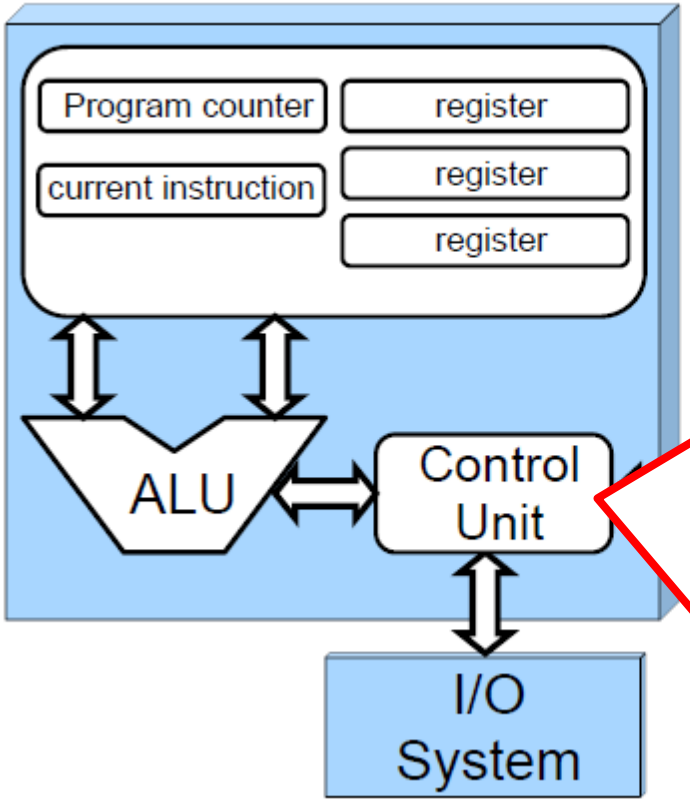
Program Counter: Points to the next instruction

- This is a special register that contains the address in memory of the next instruction that should be executed (it is incremented after each instruction, or it can be set to any valid (or invalid) memory address.

So, what is the CPU?



So, what is the CPU?



Control Unit: Decodes instructions and make them happen

- Logic hardware that decodes instructions (meaning the bits in memory) and sends the appropriate signals to hardware components in the CPU

Direct Memory Access

- DMA is used in all modern computers
- It allows memory-I/O (data transfer) to occur independent of communications with the CPU
  - Suppose you are writing 1GiB from memory to an external device, such as a disk, network card, graphics card, or something similar
    - Without DMA, the CPU would be busy during this slow transfer (loading from memory to registers, then from registers to disk)
  - With DMA, data transfer operations occur independent of the CPU through the DMA controller
    - The CPU simply “tells” the DMA controller to initiate the transfer (by writing to some registers in the DMA controller)
  - When the transfer is complete, the DMA controller generates an interrupt to tell the CPU it has completed
  - This leaves the CPU free to do useful work in the meantime

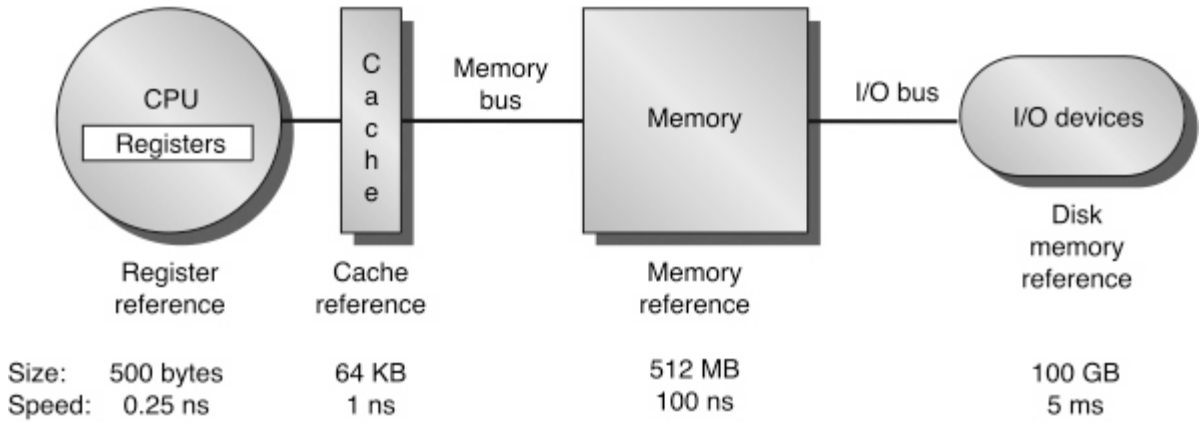
DMA costs

- DMA data transfers utilize the memory bus
  - Code executing on the CPU will most likely wish to use the memory bus as well
  - This leads to interference
  - Interference can be managed by several different modes
    - DMA has priority
    - CPU has priority
  - In general, DMA leads to much greater performance

The Slow RAM problem

- It would be great to have a computer with very large and very fast memory
- However, technology (cost) allows for either slow and large or fast and small
- We need large main memory for large programs and data
- Thus, from the CPU's perspective, main memory is horribly slow
- We use a little trick to provide the illusion of a fast memory
- This trick is called the memory hierarchy

The Memory Hierarchy



- The real world has multiple levels of caches(L1, L2, L3)
- Data is brought in from far away memory in chunks and copied to keep around in nearby memory
  - The same data exists in multiple levels of memory at once (this can be an issue as we will see later on)
  - Miss: when data item is not found in a level
  - Hit: when data item is found in a level

Caching

- What happens when your program accesses a byte of memory?
  - The value stored in the byte is brought from the slow main memory to the faster cache
  - But that's not all, the memory around the byte you accessed is also preemptively brought into cache from the slow memory
- A short analogy:
  - You need a book from the library
  - You go find the book on the shelves
  - You bring home all the books on the shelf an put them on your own bookshelf
  - Next time you need that book, or one of the books "around it," you have them readily available and do not need to go to the library to get them
    - This assumes that all the books on the shelf at the library are about the same topic, so you'll need the books around the book you wanted in the first place

Why is this helpful?

- Temporal Locality: a program tends to reference addresses it has recently referenced
  - On the first access, you have the high cost of going far away to slow memory to fetch the counter's content
  - Following accesses are fast
- Spatial Locality: a program tends to reference addresses next to addresses it has recently referenced
  - Accessing the first element ***i*** in an array may be costly
  - Subsequent accesses to element ***i+1*** are fast because they were fetched with the first element

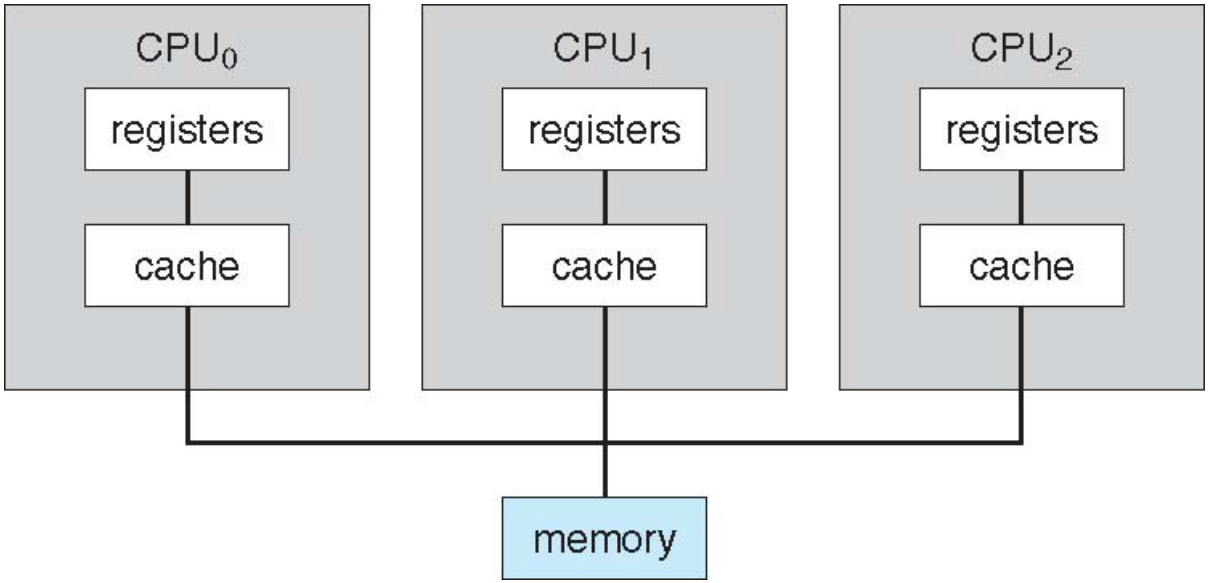
Memory Management

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

- Main memory and disks are managed by the OS
- When dealing with a “slow” level, it is beneficial to try and be clever (i.e., spending more time trying to make good decisions)
  - Part of why OSes are doing complicated things, as opposed to hardware which try to do simple things fast

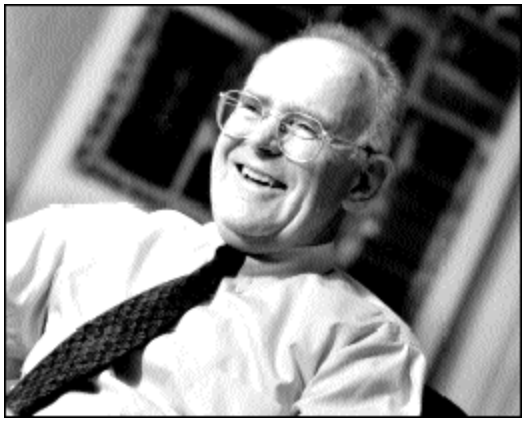
Issues With Caching

- SMP(Symmetric multi processor) Systems
  - Cache coherency (see text)

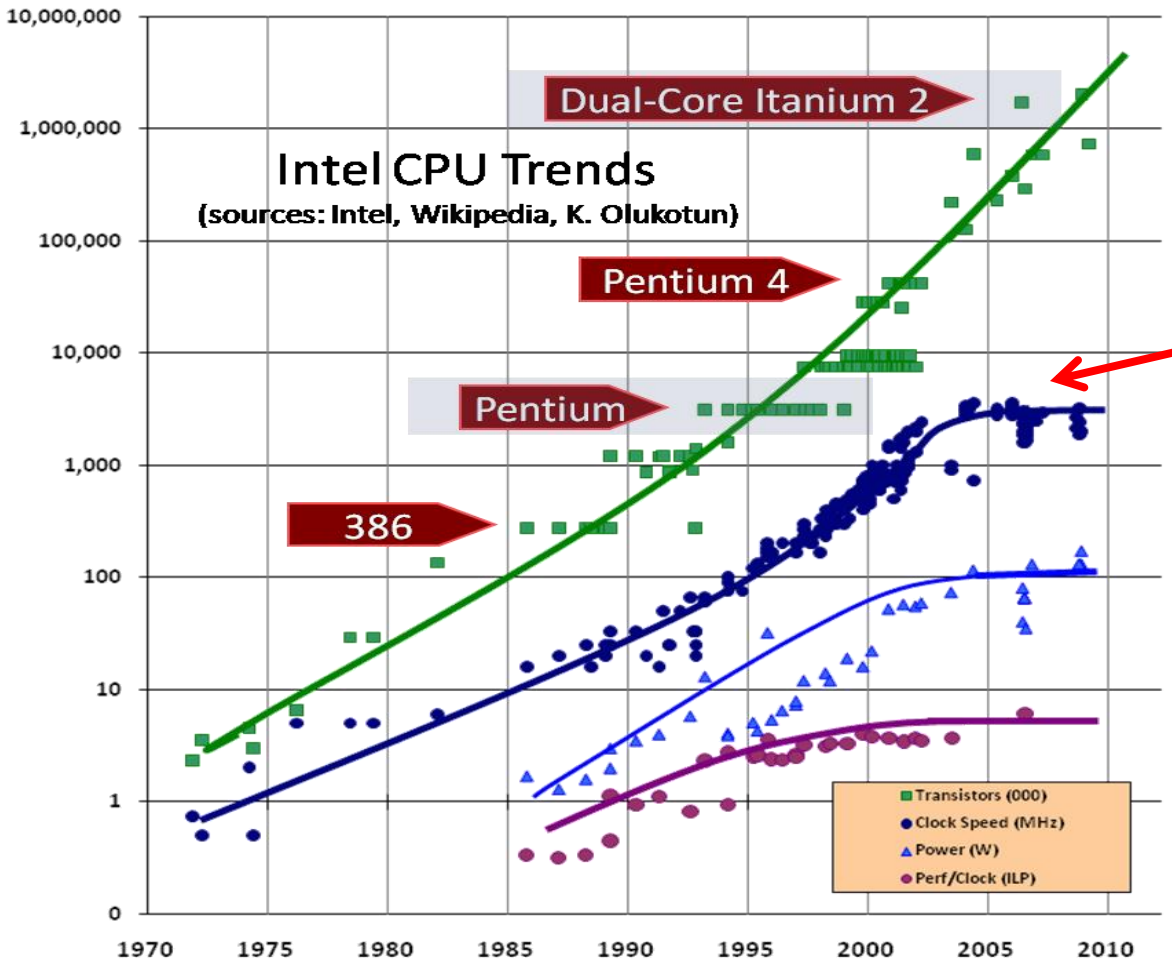


Moore's Law

- Gordon Moore (co-founder of a little company called Intel) made the prediction, in 1965, that the transistor density of semiconductor chips would double roughly every 24 months (often "misquoted" as 18 months)
- Has been right to date
- However, this law is often wrongly interpreted as: "Computers get twice as fast every 2 years"
- This incorrect interpretation was true for a while, however no longer



Moore's Law



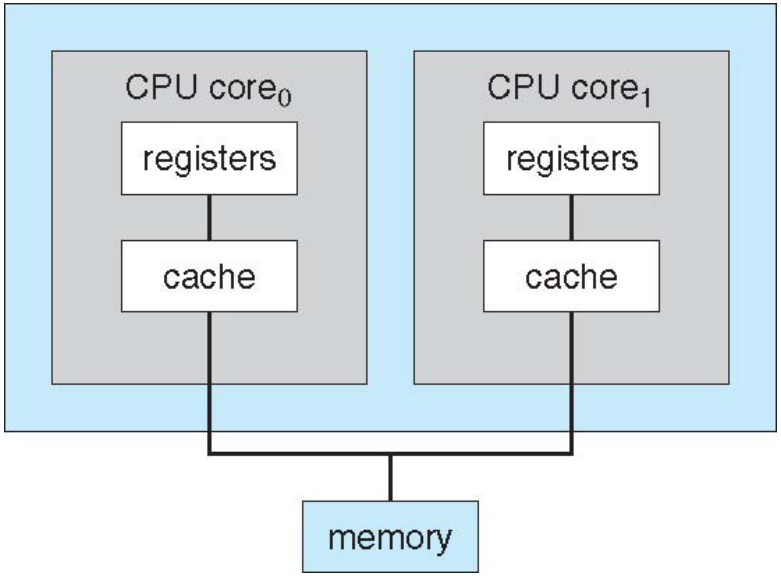
Note the plateau!

Free Lunch is Over:
http://www.gotw.ca/publications/concurrency-ddj.htm

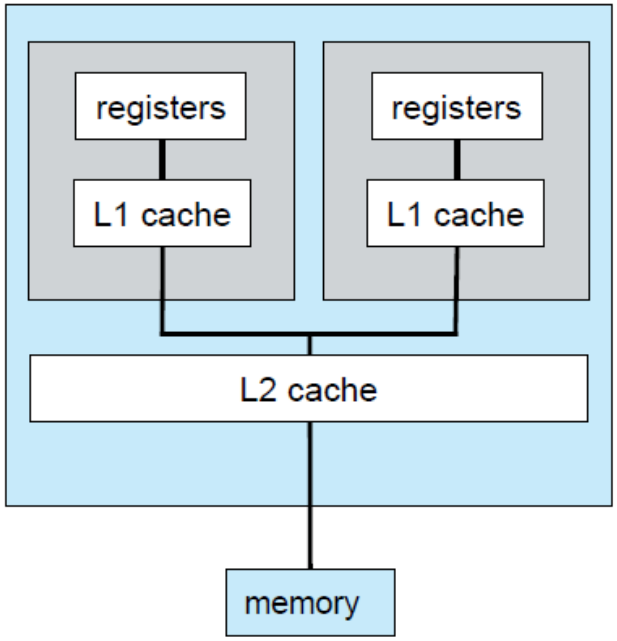
Multi-core Chips

- Chip producers can no longer continue increasing clock rate
  - Due to physical Power/heat dissipation constraints
- In comes the multi-core processor
  - Multiple “low” clock rate processors on one die
- It’s a solution to a problem as opposed to a new advancement
  - Though there are many interesting things about multi core processors
- Most user/programmers would rather have a 100GHz core than 50 2GHz cores

Multi-core Systems

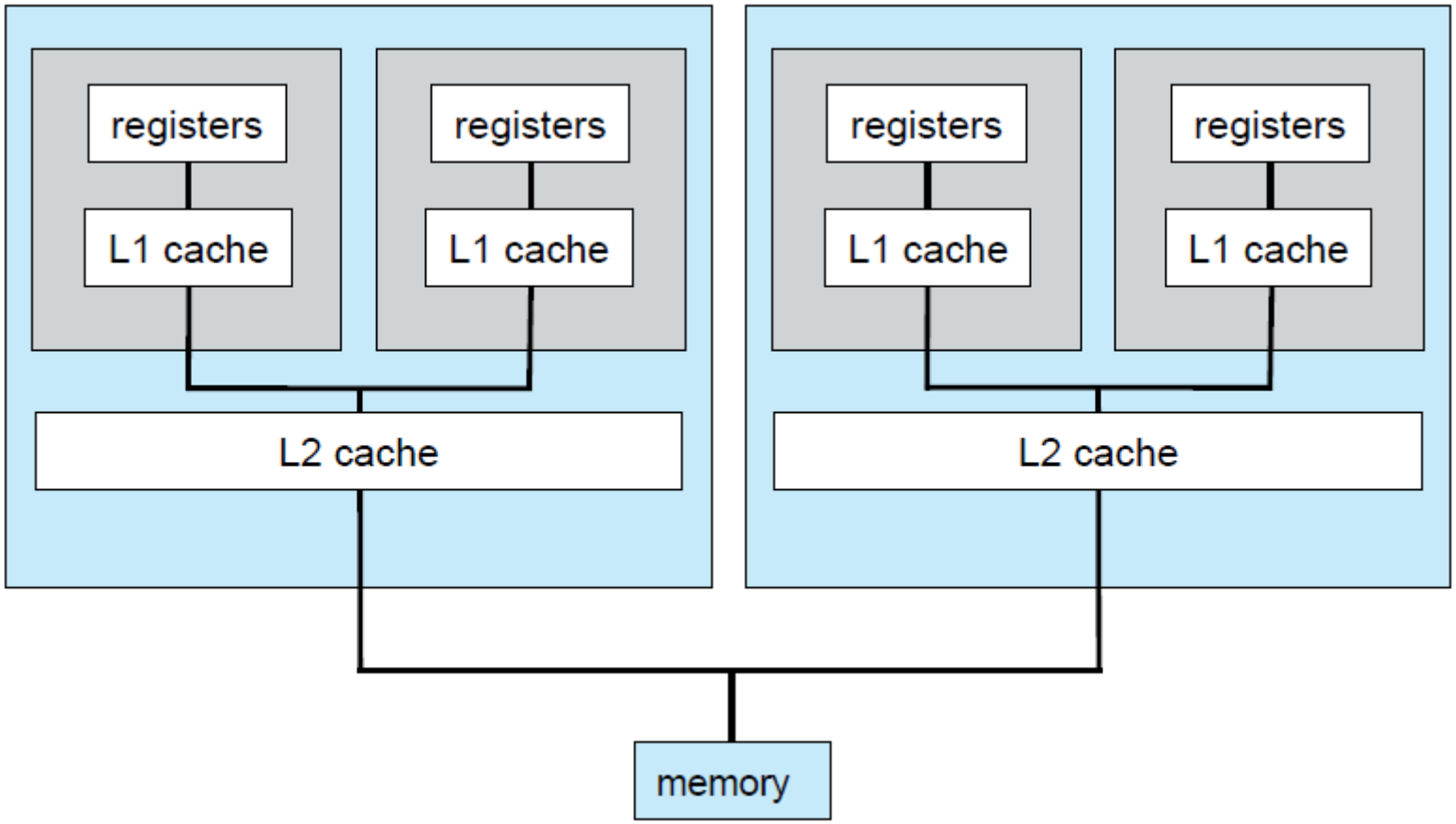


1.7 from book



Closer to reality

Multi-Proc Multi-Core Systems



Conclusion

- Interested in more
  - Take a computer architecture course (CSC-4210) or even an advanced computer architecture course (CSC-8210)
    - Classic Textbooks by Patterson and Hennessy (I've read them, you should too)
- Reading assignment: Chapter 1