

SYSTEM HARDWARE

Tutorial 1 - Introduction

OBJECTIVES

- ③ Know the difference between computer organization and computer architecture.
- ③ Understand units of measure common to computer systems.
- ③ A basic understanding of computer components.
- ③ A basic introduction of the RISC Architecture.
- ③ Appreciate the evolution of computers.
- ③ Understand the computer as a layered system built as a tower of interfaces.

1.1 OVERVIEW

Why study computer organization and architecture?

- ⦿ Design better programs, including system software such as compilers, operating systems, and device drivers.
- ⦿ Optimize program behavior.
- ⦿ Understand time, space, and price tradeoffs.

1.1 OVERVIEW

⊙ Computer organization

- ⊙ Encompasses all physical aspects of computer systems.
- ⊙ E.g., circuit design, control signals, memory types.
- ⊙ *How does a computer work?*

⊙ Computer architecture

- ⊙ The contract specifying the hardware/software interface.
- ⊙ Logical aspects of system implementation as seen by the programmer.
- ⊙ E.g. instruction sets, instruction formats, data types, addressing modes.
- ⊙ *How do I design a computer?*

1.2 COMPUTER COMPONENTS

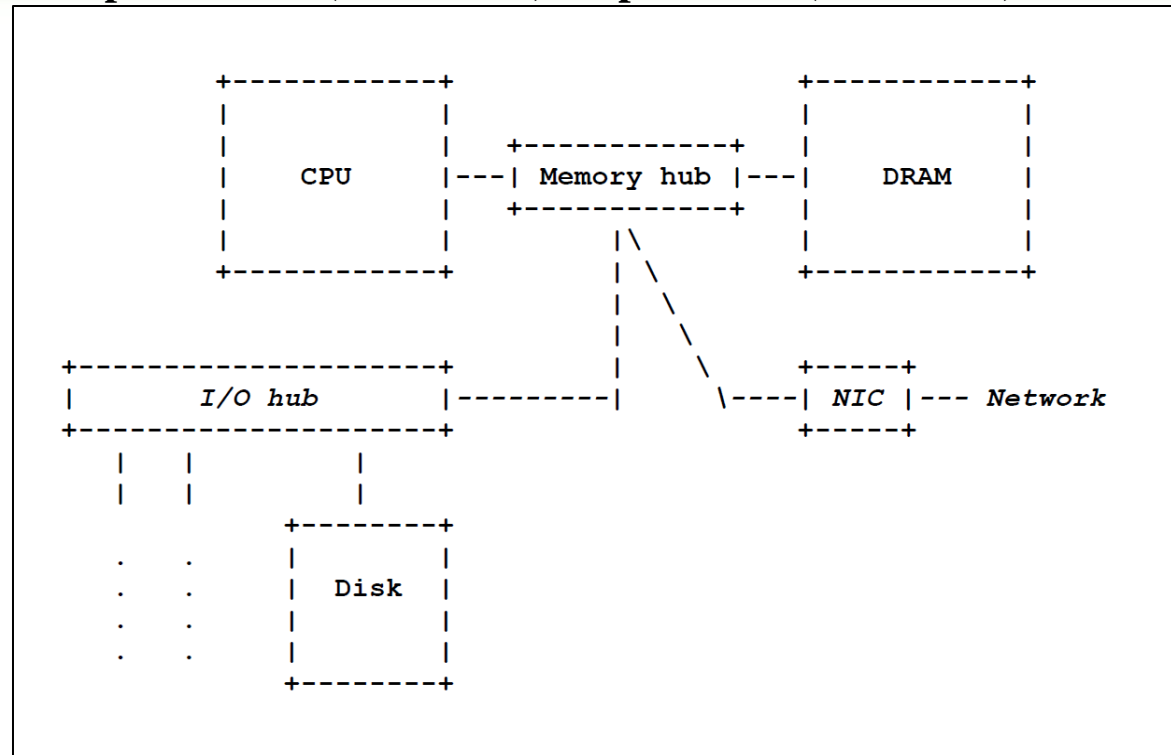
- ⊙ There is no clear distinction between matters related to computer organization and matters relevant to computer architecture.
- ⊙ Principle of Equivalence of Hardware and Software:
 - ⊙ ***Any task done by software can also be done using hardware, and any operation performed directly by hardware can be done using software.****

* Assuming speed is not a concern.

1.2 COMPUTER COMPONENTS

A VERY BASIC DIAGRAM

A picture of a (standalone) uniprocessor (circa 2003)



Source: Dr. Probst Lecture 1 Notes

1.2 COMPUTER COMPONENTS

- ◎ The 3 most basic computer components are:
 - ◎ The processor
 - ◎ The memory System (E.g. DRAM)
 - ◎ The Storage System (E.g. Disk(s))
- ◎ An interconnection network (wires) shows how signals are transmitted among the 3 components.
- ◎ A mechanism for transferring data to and from the outside world (I/O peripheral devices).
- ◎ The processor talks directly to the memory but data must be moved from storage to memory before it can be accessed by the processor.

1.2 COMPUTER COMPONENTS

THE INTERCONNECTION NETWORK

- ◎ In larger computers, the interconnect system between processors and memories is an elaborate network.
- ◎ To achieve a *high peak data-transfer rate* between processors and memories, a *high-bandwidth* communication fabric is required.
 - ◎ **Bandwidth:** The maximum data transfer rate of a **network** or Internet connection. (E.g. 100 megabytes per second)
- ◎ A special processor that is able to sustain a significant fraction of that high bandwidth (i.e. keep itself supplied with useful data) is called a *latency-tolerant processor*.
- ◎ Presently, there is essentially none of these. But the computer industry is slowly recognizing the potential value.

1.2 COMPUTER COMPONENTS

THE INTERCONNECTION NETWORK

- ③ Modern computers use ***direct point-to-point links*** (wires) between compatible devices for greater speed & bandwidth.
- ③ In contrast, a ***bus is a shared communication link*** (party line) that connects multiple subsystems.
- ③ The point-to-point interconnect between processors and memories can be designed to provide high peak bandwidth.
- ③ The bus interconnect between a processor and its peripherals is necessarily of low bandwidth because of the number and the diversity of the I/O devices that it serves.
- ③ The ***memory controller hub*** is the switch in the point-to-point network & ***I/O controller hub***, the switch in the bus network.

1.2 COMPUTER COMPONENTS

THE RISC ARCHITECTURE

- ⊙ A **Reduced Instruction Set Computer**, or **RISC** is a computer that has a small set of simple and general instructions. RISC allows a computer's microprocessor to use fewer cycles per instruction (CPI) when compared with CISC (Complex ISC)
- ⊙ ***Instruction Set Architecture (ISA)*** refers to an abstract model of a computer. ISA will be covered later in the course.
- ⊙ **RISC-V** is an open-source hardware ISA based on established RISC principles. Used largely for academic purposes.
- ⊙ Unlike other academic designs which are optimized only for simplicity of exposition, the designers state that the RISC-V instruction set is for practical computers. (source: wikipedia)

1.2 COMPUTER COMPONENTS

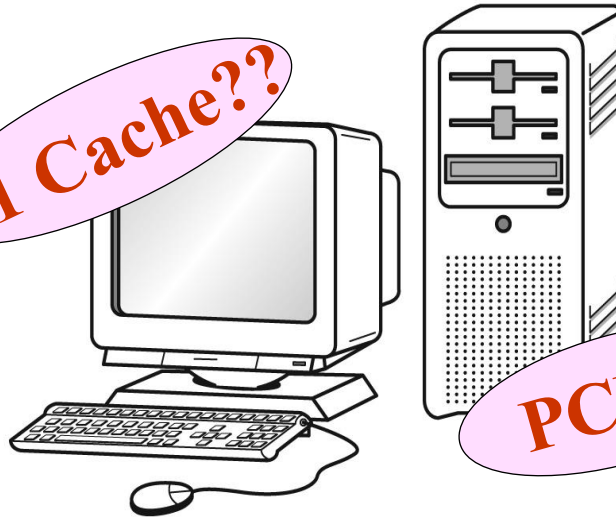

RISC ARCHITECTURE FEATURES

- ⊙ Direct hardware implementation of instruction set
- ⊙ Fixed instruction size
- ⊙ Small number of addressing modes
- ⊙ Reduced memory access — Only load and store instructions access memory.
- ⊙ Ease of pipelining — The instructions are designed to be easily divisible into parts.
- ⊙ A floating-point coprocessor
- ⊙ And more ... The course will detail many of these features.

1.3 AN EXAMPLE SYSTEM

Consider this advertisement:

GHZ??



PCI??



GB??

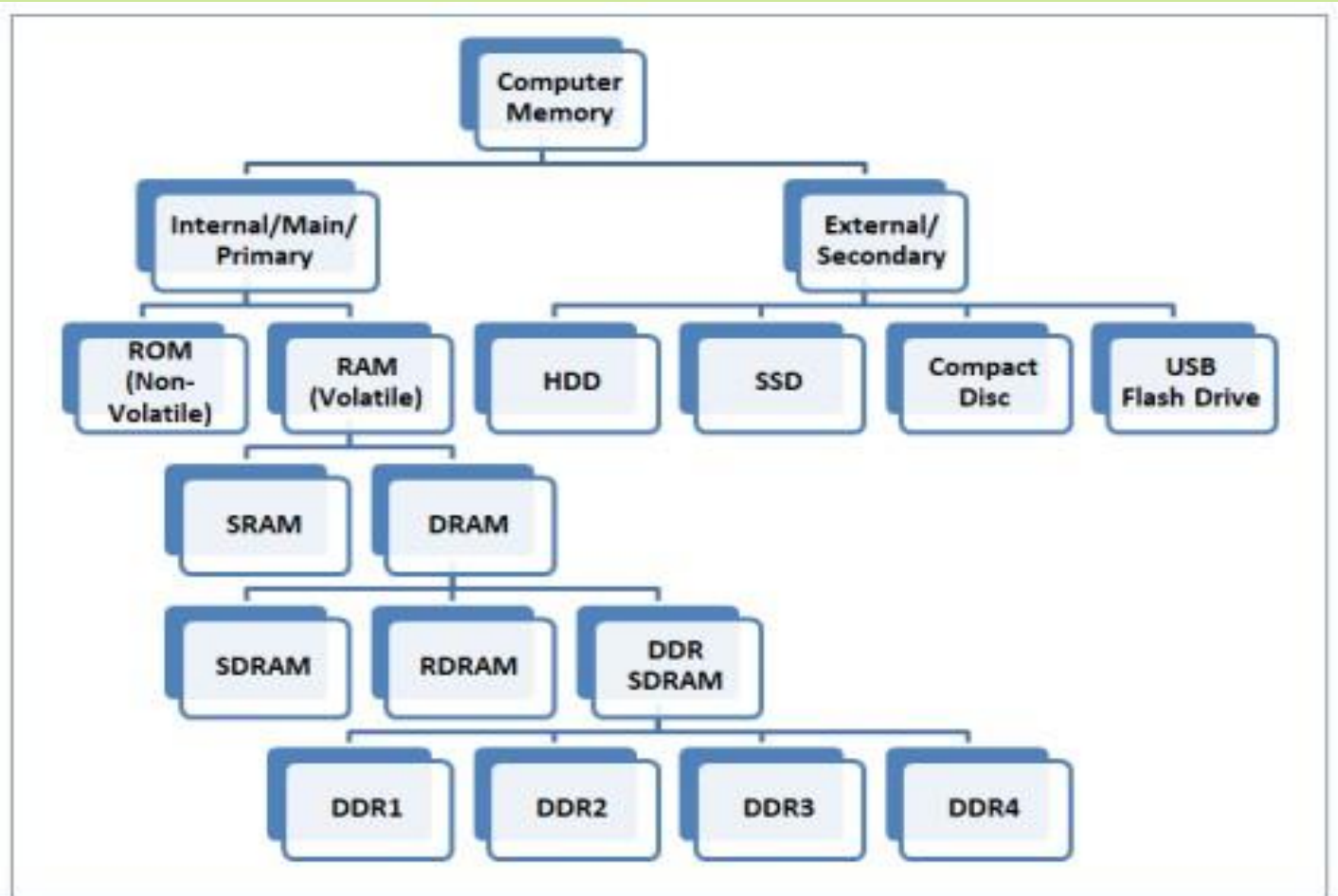
USB??

- Intel Pentium Dual Core, 3.06 GHz
- 1333MHz 4GB DDR SDRAM
- 128KB L1 cache, 2MB L2 cache
- 500GB serial ATA hard drive (7200 RPM)
- 4 USB ports, 1 serial port, 1 parallel port, 4 FireWire slots (1 PCI, 1 PCI x 16, 2 PCI x 1)
- Choice of monitor: 19", .24mm AG, 1280x1024 at 75Hz
- 20.1" 1280x1024 SXGA, 250 cd/m2, active matrix, 16.7 million colors (static), 5ms, 24-bit color (16.7 million colors), 100 Hz
- VGA/DVI input
- 16X DVD +/- RW Drive
- 1GB PCIe video card
- PCIe sound card
- Integrated 10/100/1000 Ethernet

What does it all mean??

1.3 AN EXAMPLE SYSTEM

TYPES OF MEMORY



1.3 AN EXAMPLE SYSTEM

TYPES OF MEMORY

Internal Memory (aka Primary Memory): Stores relatively smaller amounts of data that can be accessed quickly while the computer is running.

- ⦿ ROM: Read-only memory. Boot up data retained without power.
- ⦿ RAM: Random Access Memory. Stores data to be processed by CPU.
 - ⦿ SRAM: Static, keeps data as long as power is on, expensive.
 - ⦿ DRAM: Dynamic, has to be refreshed to keep data, cheaper.

External Memory (aka Secondary Memory): Refers to external, embedded or removable storage devices that are used to store larger amounts of data permanently.

1.3 AN EXAMPLE SYSTEM

Measures of capacity and speed:

- Kilo- (K) = 1 thousand = 10^3 and 2^{10}
- Mega- (M) = 1 million = 10^6 and 2^{20}
- Giga- (G) = 1 billion = 10^9 and 2^{30}
- Tera- (T) = 1 trillion = 10^{12} and 2^{40}
- Peta- (P) = 1 quadrillion = 10^{15} and 2^{50}
- Exa- (E) = 1 quintillion = 10^{18} and 2^{60}
- Zetta- (Z) = 1 sextillion = 10^{21} and 2^{70}
- Yotta- (Y) = 1 septillion = 10^{24} and 2^{80}

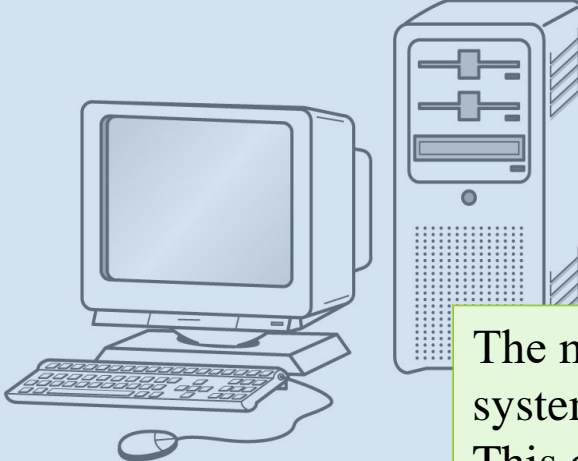
Whether a metric refers to a power of ten or a power of two typically depends upon what is being measured.

1.3 AN EXAMPLE SYSTEM

- ◎ Hertz = clock cycles per second (frequency)
 - ◎ 1MHz = 1,000,000Hz
 - ◎ Processor speeds are measured in MHz or GHz.
- ◎ Byte = a unit of storage
 - ◎ 1KB = 2^{10} = 1024 Bytes
 - ◎ 1MB = 2^{20} = 1,048,576 Bytes
 - ◎ 1GB = 2^{30} = 1,073,741,824 Bytes
 - ◎ 1TB = 2^{40} = 1,099,511,627,776 Bytes
 - ◎ Main memory (RAM) is measured in GB
 - ◎ Disk storage is measured in GB for small systems, TB (2^{40}) for large systems.

1.3 AN EXAMPLE SYSTEM

FOR SALE: OBSOLETE COMPUTER – CHEAP! CHEAP! CHEAP!



- Intel Pentium Dual Core, 3.06 GHz
- 1333MHz 4GB DDR SDRAM
- 128KB L1 cache, 2MB L2 cache
- 500GB serial ATA hard drive (7200 RPM)
- 4 USB ports, 1 serial port, 1 parallel port, 4 PCI expansion slots (1 PCI, 1 PCI x 16, 2 PCI x 1)
- Choice of monitor: 19", .24mm AG, 1280x1024 at 75Hz

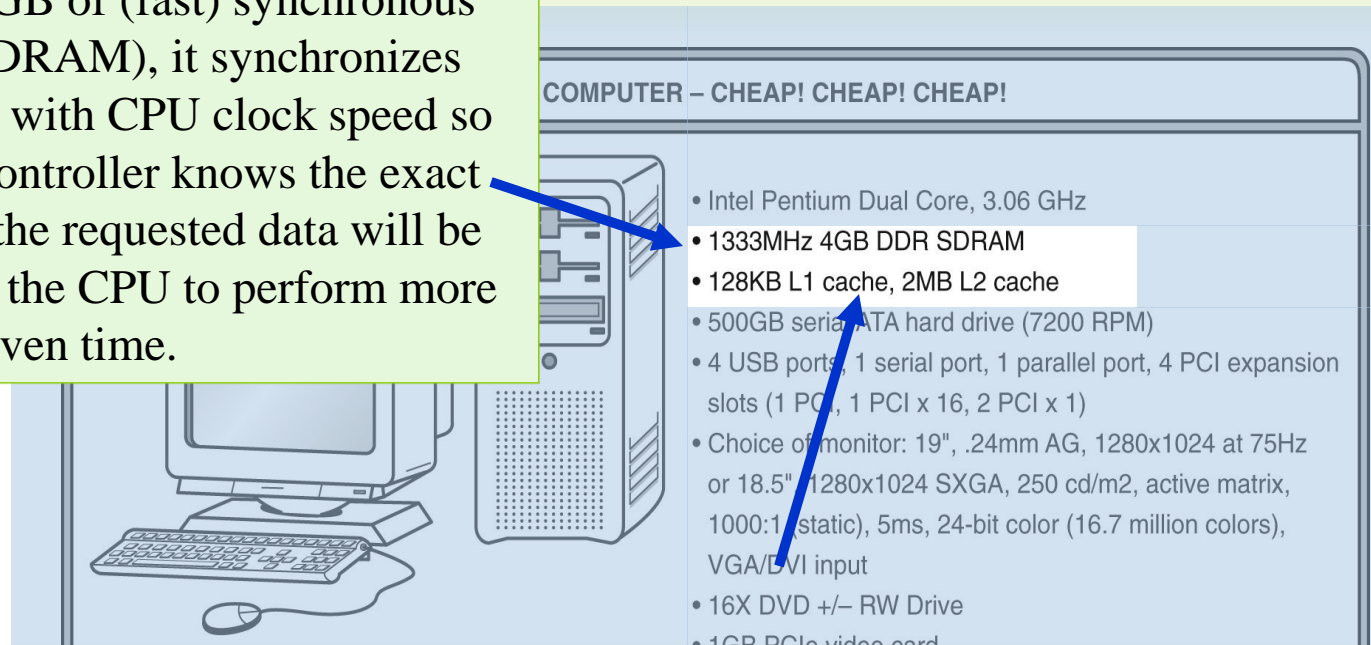
The microprocessor is the “brain” of the system. It executes program instructions. This one is a Pentium (Intel) running at 3.06GHz.

1.3 AN EXAMPLE SYSTEM

- Computers with large main memory capacity can run larger programs with greater speed than computers having small memories.
- RAM is an acronym for *Random Access Memory*. Random access means that memory contents can be accessed directly if you know its location (memory address).
- Cache is a type of temporary memory that can be accessed faster than RAM.

1.3 AN EXAMPLE SYSTEM

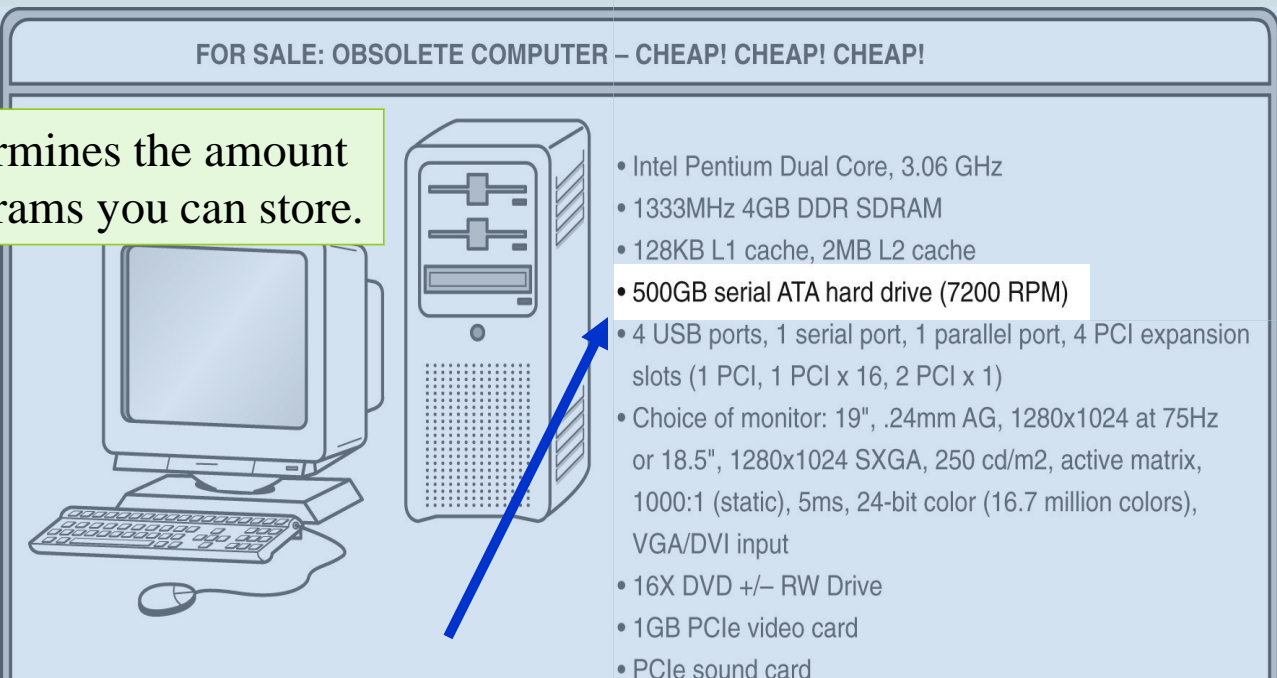
This system has 4GB of (fast) synchronous dynamic RAM (SDRAM), it synchronizes the memory speed with CPU clock speed so that the memory controller knows the exact clock cycle when the requested data will be ready. This allows the CPU to perform more instructions at a given time.



... and two levels of cache memory, the level 1 (L1) cache is smaller and (probably) faster than the L2 cache. Note that these cache sizes are measured in KB and MB.

1.3 AN EXAMPLE SYSTEM

Hard disk capacity determines the amount of data and size of programs you can store.



FOR SALE: OBSOLETE COMPUTER – CHEAP! CHEAP! CHEAP!

- Intel Pentium Dual Core, 3.06 GHz
- 1333MHz 4GB DDR SDRAM
- 128KB L1 cache, 2MB L2 cache
- **500GB serial ATA hard drive (7200 RPM)**
- 4 USB ports, 1 serial port, 1 parallel port, 4 PCI expansion slots (1 PCI, 1 PCI x 16, 2 PCI x 1)
- Choice of monitor: 19", .24mm AG, 1280x1024 at 75Hz or 18.5", 1280x1024 SXGA, 250 cd/m2, active matrix, 1000:1 (static), 5ms, 24-bit color (16.7 million colors), VGA/DVI input
- 16X DVD +/- RW Drive
- 1GB PCIe video card
- PCIe sound card

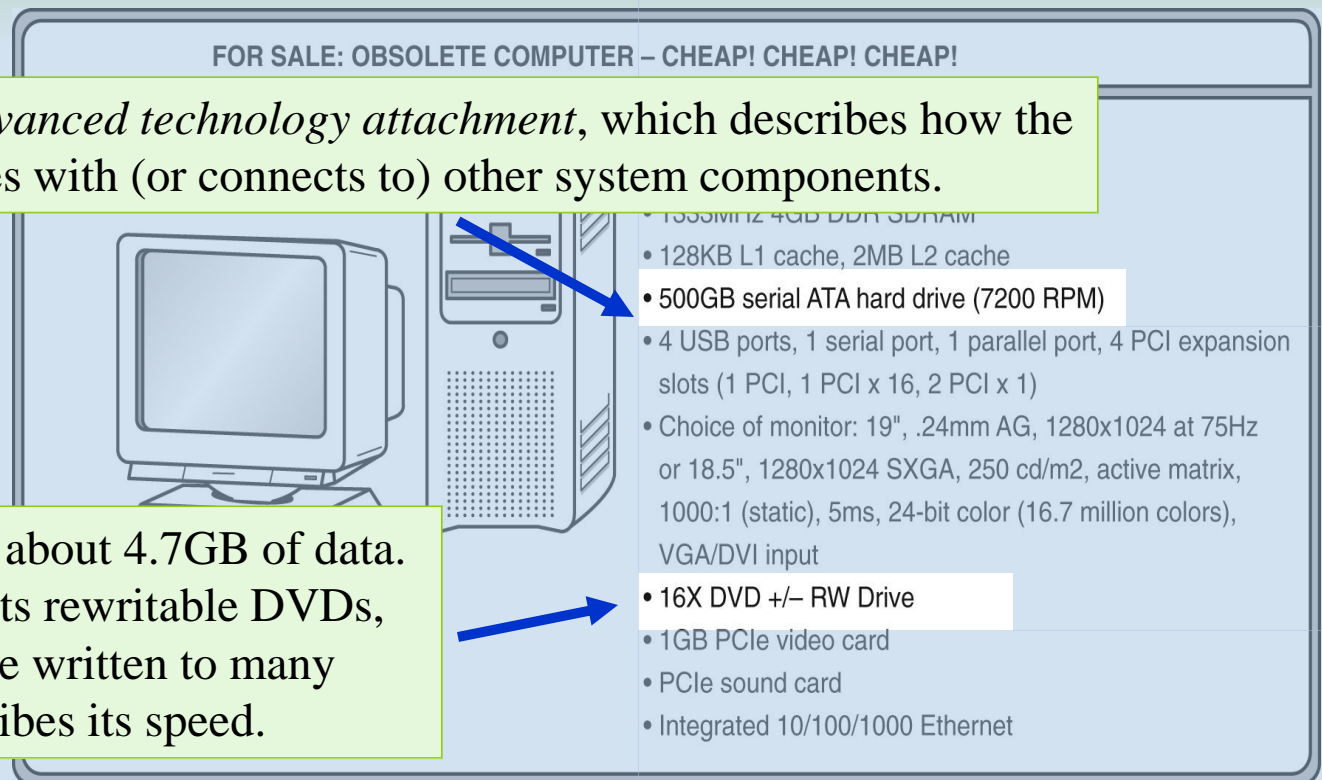
This one can store 500GB. 7200 RPM is the rotational speed of the disk. Generally, the faster a disk rotates, the faster it can deliver data to RAM. (There are many other factors involved.)

1.3 AN EXAMPLE SYSTEM

ATA stands for *advanced technology attachment*, which describes how the hard disk interfaces with (or connects to) other system components.

A DVD can store about 4.7GB of data. This drive supports rewritable DVDs, +/-RW, that can be written to many times.. 16x describes its speed.

FOR SALE: OBSOLETE COMPUTER – CHEAP! CHEAP! CHEAP!

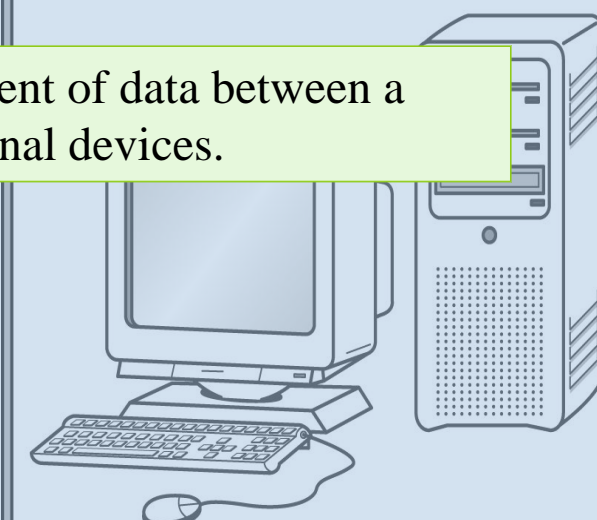


- 1333MHz 4GB DDR SDRAM
- 128KB L1 cache, 2MB L2 cache
- 500GB serial ATA hard drive (7200 RPM)
- 4 USB ports, 1 serial port, 1 parallel port, 4 PCI expansion slots (1 PCI, 1 PCI x 16, 2 PCI x 1)
- Choice of monitor: 19", .24mm AG, 1280x1024 at 75Hz or 18.5", 1280x1024 SXGA, 250 cd/m2, active matrix, 1000:1 (static), 5ms, 24-bit color (16.7 million colors), VGA/DVI input
- 16X DVD +/- RW Drive
- 1GB PCIe video card
- PCIe sound card
- Integrated 10/100/1000 Ethernet

1.3 AN EXAMPLE SYSTEM

Ports allow movement of data between a system and its external devices.

FOR SALE: OBSOLETE COMPUTER – CHEAP! CHEAP! CHEAP!



- Intel Pentium Dual Core, 3.06 GHz
- 1333MHz 4GB DDR SDRAM
- 128KB L1 cache, 2MB L2 cache
- 500GB serial ATA hard drive (7200 RPM)
- 4 USB ports, 1 serial port, 1 parallel port, 4 PCI expansion slots (1 PCI, 1 PCI x 16, 2 PCI x 1)
- Choice of monitor: 19", .24mm AG, 1280x1024 at 75Hz or 18.5", 1280x1024 SXGA, 250 cd/m2, active matrix, 1000:1 (static), 5ms, 24-bit color (16.7 million colors), VGA/DVI input
- 16X DVD +/- RW Drive
- Integrated 10/100/1000 Ethernet

This system has ten ports.

1.3 AN EXAMPLE SYSTEM

- Serial ports send data as a series of pulses along one or two data lines.
- Parallel ports send data as a single pulse along at least eight data lines.
- USB, Universal Serial Bus, is an intelligent serial interface that is self-configuring. (It supports “plug and play.”)

1.3 AN EXAMPLE SYSTEM

System buses can be augmented by dedicated I/O buses. PCI, *peripheral component interface or interconnect*, is one such bus.



This system has two PCI devices: a video card and a sound card. A PCI device can be plugged directly into a PCI slot on the computer's motherboard.

ER – CHEAP! CHEAP! CHEAP!

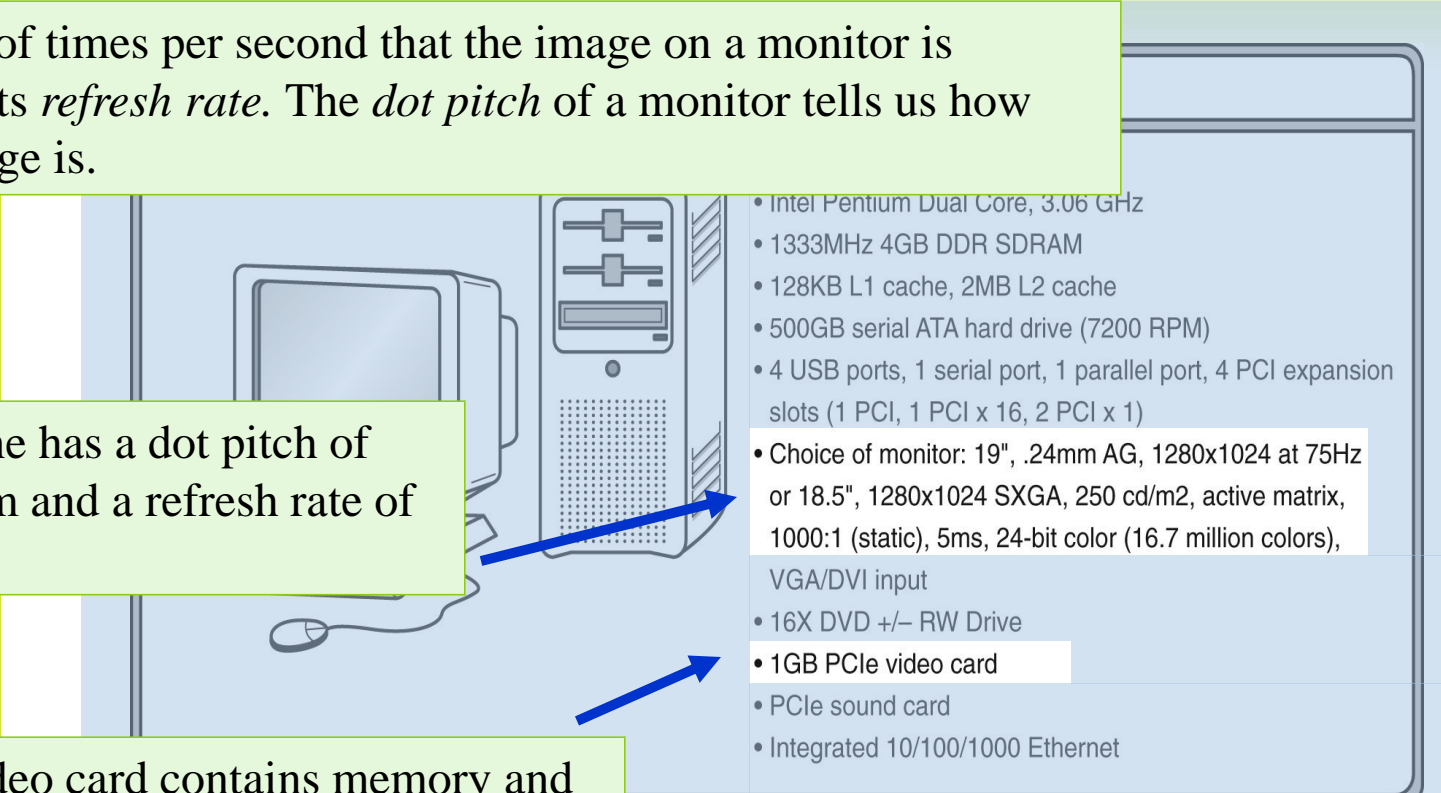
- Intel Pentium Dual Core, 3.06 GHz
- 1333MHz 4GB DDR SDRAM
- 128KB L1 cache, 2MB L2 cache
- 500GB serial ATA hard drive (7200 RPM)
- 4 USB ports, 1 serial port, 1 parallel port, 4 PCI expansion slots (1 PCI, 1 PCI x 16, 2 PCI x 1)
- Choice of monitor: 19", .24mm AG, 1280x1024 at 75Hz or 18.5", 1280x1024 SXGA, 250 cd/m2, active matrix, 1000:1 (static), 5ms, 24-bit color (16.7 million colors), VGA/DVI input
- 16X DVD +/- RW Drive
- 1GB PCIe video card
- PCIe sound card
- Integrated 10/100/1000 Ethernet

1.3 AN EXAMPLE SYSTEM

The number of times per second that the image on a monitor is repainted is its *refresh rate*. The *dot pitch* of a monitor tells us how clear the image is.

This one has a dot pitch of 0.24mm and a refresh rate of 75Hz.

The video card contains memory and programs that support the monitor.

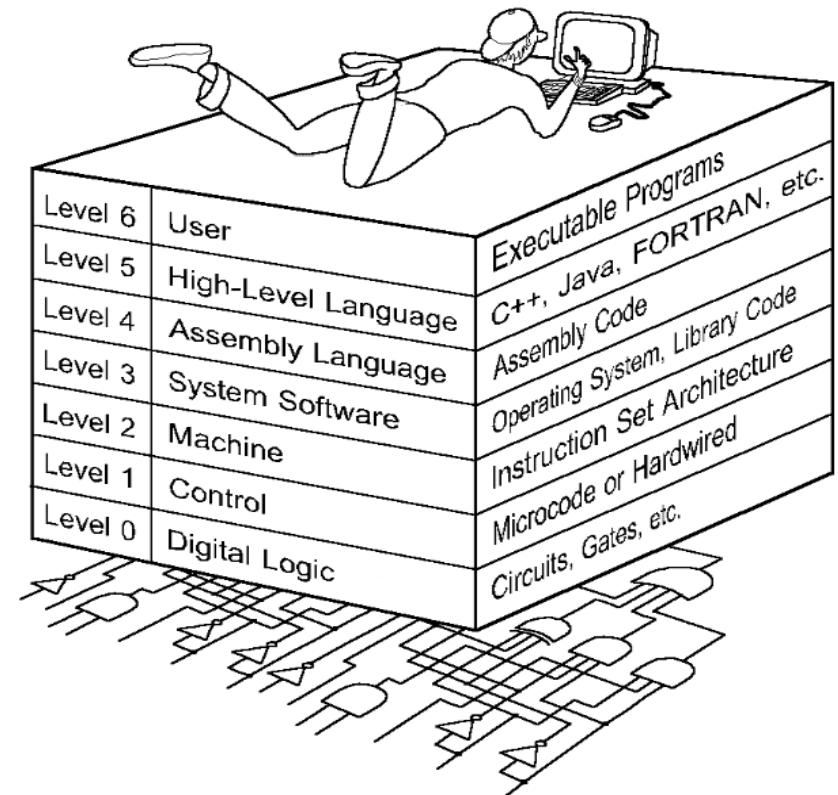
- 
- Intel Pentium Dual Core, 3.06 GHz
 - 1333MHz 4GB DDR SDRAM
 - 128KB L1 cache, 2MB L2 cache
 - 500GB serial ATA hard drive (7200 RPM)
 - 4 USB ports, 1 serial port, 1 parallel port, 4 PCI expansion slots (1 PCI, 1 PCI x 16, 2 PCI x 1)
 - Choice of monitor: 19", .24mm AG, 1280x1024 at 75Hz or 18.5", 1280x1024 SXGA, 250 cd/m2, active matrix, 1000:1 (static), 5ms, 24-bit color (16.7 million colors), VGA/DVI input
 - 16X DVD +/- RW Drive
 - 1GB PCIe video card
 - PCIe sound card
 - Integrated 10/100/1000 Ethernet

1.4 THE COMPUTER LEVEL HIERARCHY

- ⊙ Computers consist of many things besides chips.
- ⊙ Before a computer can do anything worthwhile, it must also use software.
- ⊙ Writing complex programs requires a “divide and conquer” approach, where each program module solves a smaller problem.
- ⊙ Complex computer systems employ a similar technique through a series of virtual machine layers.

1.4 THE COMPUTER LEVEL HIERARCHY TOWER OF INTERFACES

- ⊙ Each virtual machine layer is an abstraction of the level below it.
- ⊙ The machines at each level execute their own particular instructions, calling upon machines at lower levels to perform tasks as required.
- ⊙ Computer circuits ultimately carry out the work.
- ⊙ The operating system and the runtime system are actors in this tower of interfaces but will not be discussed in this course.



1.4 THE COMPUTER LEVEL HIERARCHY

Example: You write a program in a high-level programming language. A compiler translates it into machine instructions (object code). The computer executes the object code.

- ⦿ Level 6: The User Level

- ⦿ Program execution and user interface level.
- ⦿ The level with which we are most familiar.

- ⦿ Level 5: High-Level Language Level

- ⦿ We interact with this level when we write programs in languages such as C and Java.

- ⦿ Level 4: Assembly Language Level

- ⦿ Acts upon assembly language produced from Level 5, as well as instructions programmed directly at this level.

1.4 THE COMPUTER LEVEL HIERARCHY

- ⊙ Level 3: System Software Level
 - ⊙ Controls executing processes on the system.
 - ⊙ Protects system resources.
 - ⊙ Assembly language instructions often pass through Level 3 without modification.
- ⊙ Level 2: Machine Level
 - ⊙ Also known as the Instruction Set Architecture (ISA) Level.
 - ⊙ Consists of instructions that are particular to the architecture of the machine.
 - ⊙ Programs written in machine language need no compilers, interpreters, or assemblers.

1.4 THE COMPUTER LEVEL HIERARCHY

- ⊙ Level 1: Control Level (aka microarchitecture)
 - ⊙ A *control unit* decodes and executes instructions and moves data through the system.
 - ⊙ Control units can be *microprogrammed* or *hardwired*.
 - ⊙ A microprogram is a program written in a low-level language that is implemented by the hardware.
 - ⊙ Hardwired control units consist of hardware that directly executes machine instructions.

1.4 THE COMPUTER LEVEL HIERARCHY

- ⊙ Level 0: Digital Logic Level
 - ⊙ This level is where we find digital circuits (the chips).
 - ⊙ Digital circuits consist of gates and wires.
 - ⊙ These components implement the mathematical logic of all other levels.

RISC MICROARCHITECTURE

KILLERS MICROS

- ◎ A **microprocessor is a CPU** on a single integrated circuit (IC) chip containing millions of very small components that work together.
- ◎ A **killer micro** is a **microprocessor-based** machine that competes with mainframes and super computers. *From 1979 to 2003, killer micros* drove competing processor designs out of business.
- ◎ Almost all computers today are powered by one or more killer micros: A cellphone has a cooling killer micro, probably an ARM (Advanced RISC Machine). A server has one or more killer micros, probably Intel Xeons.
- ◎ Killer micros steadily got better in performance up until 2003.
- ◎ Note that computers with the same processor could still differ in the quality of their memory system or the quality of their interconnect.

DESIGN CONCERNS

- ⊙ Users write programs (based on algorithms) to solve problems by computer.
- ⊙ Before designing computers, hardware vendors ask: What kind of programs do my best customers wish to run? This affects design and optimization.
- ⊙ Computer design, like all engineering design, is a series of trade-offs: performance, reliability, speed, etc.
- ⊙ Programs differ in their memory use, computational requirements, storage requirements, etc.

PROGRAMS ARE DIFFERENT MEMORY USAGE

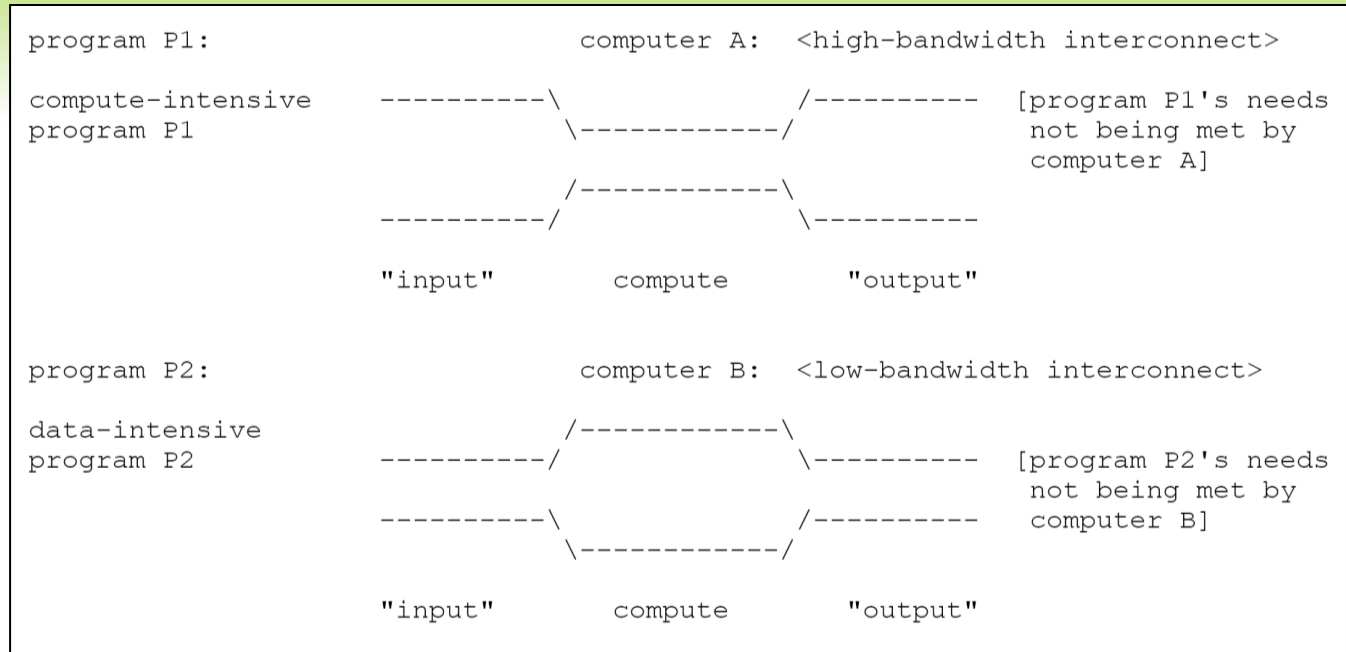
- ⊙ All computations must engage in communications, which is the major source of time spent and energy consumed by the computer. However, Memory usage is not the same among all programs:
- ⊙ **Memory-access.** One program never touches memory while another does.
- ⊙ Programs may use *short-range* or *long-range* channels depending on the type of communication (Temporal & Spatial Locality)
- ⊙ Programs can differ in their **Arithmetic Intensity** (measure of the number of arithmetic operations relative to the amount of memory accesses that are required to support these operations).
- ⊙ They can differ in their degree of **data reuse**.
- ⊙ They differ in the **predictability** of their **memory-accessing patterns**.
- ⊙ Is the program task-parallel or/and data-parallel?

PROGRAMS ARE DIFFERENT

COMPUTE-INTENSIVE OR DATA-INTENSIVE

- ◎ A **compute-intensive** program does more processing than data movement.
- ◎ In contrast, a **data-intensive** program does more data movement than processing.
- ◎ A ***compute-intensive*** program will suffer from a ***bottleneck*** if the computer has inadequate processing resources relative to its ***bandwidth capabilities*** (aka **compute-bound**)
- ◎ A ***data-intensive*** program will suffer from a ***bottleneck*** if the computer has inadequate data-movement resources relative to its ***compute power*** (aka **bandwidth-bound**)
- ◎ An imbalance between processing power and data-movement capabilities can lead to a performance bottleneck.

COMPUTE-INTENSIVE OR DATA-INTENSIVE EXAMPLE



- ⊙ Program P1 is ***compute-intensive***, running on computer A with low computing capacity and high-bandwidth interconnect.
- ⊙ Program P2 is ***data-intensive***, running on computer B with high computing capacity) and low-bandwidth interconnect.

PROGRAMS ARE DIFFERENT

SUMMARY

- ⊙ Today, most computers are much better at computation than they are at communication
- ⊙ Hence, low-bandwidth communication can be the principle performance bottleneck.
- ⊙ More programs than you might imagine are data intensive in the sense that their ***arithmetic intensity*** is low.
- ⊙ Only highly compute-intensive programs are well matched to today's computers.
- ⊙ To fully understand this last statement, we need the concept of ***program's working set***, which will be introduced later in the course.

ARCHITECTURAL CREDO

BY DR. PROBST

- ⊙ Computation today is limited by communication, not arithmetic.
- ⊙ Floating-point computation (one of the most expensive types of computation) is essentially free, in time and energy.
- ⊙ In contrast, off-chip bandwidth is limited to a few GWs/s and each word transferred consumes enormous energy.
- ⊙ Running the FPUs is not expensive but Feeding them with data is.
- ⊙ Since communication capability is so limited, we should try to:
 - ⊙ Exploit **locality** whenever possible to reduce our need for communication bandwidth.
 - ⊙ Tolerate sufficient network/memory latency through some form of parallelism to keep the limited bandwidth resources in our high-latency network/memory systems **usefully busy**.
 - ⊙ **To summarize**, we want to extract the highest possible **sustained operand bandwidth** from our limited bandwidth resources.

PARALLEL PROGRAMS

- ◎ **Parallel Computing:** Multiple tasks running simultaneously.
- ◎ Large computers are ***aggregates*** of uniprocessors, and hence are called ***multiprocessors***.
- ◎ On a multiprocessor or multi-core system, multiple ***threads*** can execute in ***parallel***, with every processor or core executing a separate ***thread*** simultaneously.
- ◎ **An Important Issue:** To what extent are the threads (coming from program decomposition) independent? In other words, to what extent do they communicate and synchronize?
- ◎ **An Embarrassingly Parallel** workload or problem (aka **perfectly parallel**) can be easily separated into a number of parallel tasks.
- ◎ CPUs (task-parallel) while Optimized GPUs (data-parallel)

MEMORY LATENCY

MEMORY BANDWIDTH

- ⊙ A processor is able to sustain a high ***memory-request bandwidth*** of **B** memory requests per processor cycle.
- ⊙ This processor would benefit from a DRAM that can sustain a ***high memory-reply bandwidth*** of **B** memory replies per processor cycle.
- ⊙ In contrast, imagine a processor that issues a single memory request, and must wait for a reply before it can issue its next memory request. This processor would benefit from a DRAM that is able to reply quickly to a single memory request.
- ⊙ **Memory Latency** is the time between initiating a request for data (byte or word) in memory until the data is retrieved by the processor.
- ⊙ CPUs have large, deep memory hierarchies to try to keep their processors busy but they are not very successful at running programs with irregular, unpredictable memory access patterns.

A LATENCY-TOLERANT PROCESSOR

- ⊙ A program's data is stored in the memory but can only be processed in the CPU.
- ⊙ A program might make a memory reference to retrieve an operand and then be forced to wait for the operand to arrive.
- ⊙ If the whole processor sits idle while waiting, this would be tragic.
- ⊙ Not utilizing an ALU is less tragic, because it is cheaper.
- ⊙ Some processors are very good at maintaining processor activity, including asking memory for more data, even if some of the programs or threads they are running are waiting for data to arrive.
- ⊙ This rare (and good) form of processor is called a ***latency-tolerant processor*** because its utilization does not degrade in the face of memory latency.

LATENCY-AVOIDANCE

- ◎ Most processors are not *latency-tolerant*.
- ◎ So they depend on having *latency-avoidance mechanisms*
- ◎ Such as trying to keep operands that will be used in the near future close to the processor (E.g. *processor caches*).
- ◎ CPUs are latency intolerant because of their monothreading while GPUs are partly latency tolerant because of their limited multithreading.
- ◎ Killer micros only perform well when the program's memory-accessing pattern can be exploited by the memory hierarchy, of which caches, at all levels, are the most important part.
- ◎ Lower-level caches are relevant to the storage-accessing pattern.

TWO BROAD COMPUTER FAMILIES

- ⊙ **Latency Sensitive (LS):**
- ⊙ Spend lots of time stalled waiting for memory references. Require a latency-reduction mechanism (i.e. cache)
- ⊙ **Bandwidth Sensitive (BS)**
- ⊙ Throughput-oriented processors (e.g. vector processors, GPUs)
- ⊙ Use parallelism to keep long-latency operations outstanding at all times.
- ⊙ Invest in band-width

PARALLEL PROGRAMS

MULTICORE ARCHITECTURE

- ⊙ Computers on offer from major hardware vendors are remarkably architecturally similar and do not deserve to be called ***general-purpose computer***.
- ⊙ Each computer has been optimized to provide ***good performance*** for some ***restricted*** class of applications only.
- ⊙ Chips using ***high-performance*** single processor were getting too hot. Vendors adopted a new organization: Using many low-performance processors on a single "processor" chip (aka ***multicore***).
- ⊙ There are still many big questions associated with multicore architecture.

MULTICORE ARCHITECTURE

MANY BIG QUESTIONS

- ◎ Is there a memory system that will allow us to ramp up the **number** of cores that we can profitably put and use on a single multiprocessor chip?
- ◎ Inadequate memory bandwidth may cancel the benefit of the increased arithmetic capability that a multicore provides.
- ◎ Are the pins (required for off-chip I/O) and caches (including cache coherence) a stranglehold on the future effectiveness of multicore?
- ◎ Do we require new programming models (and languages) and new system software to be able to exploit multicore efficiently?
- ◎ Will all future programmers **only** write parallel programs?

THE VON NEUMANN MODEL

- ◎ In early computers, programming was done at the digital logic level and involved moving plugs/wires.
- ◎ A different hardware configuration was needed to solve every unique problem type and required many days-labor by skilled technicians.
- ◎ The von Neumann computational model started from the idea of a computer that could store program instructions in memory. Such computers became known as von Neumann Architecture systems or models.

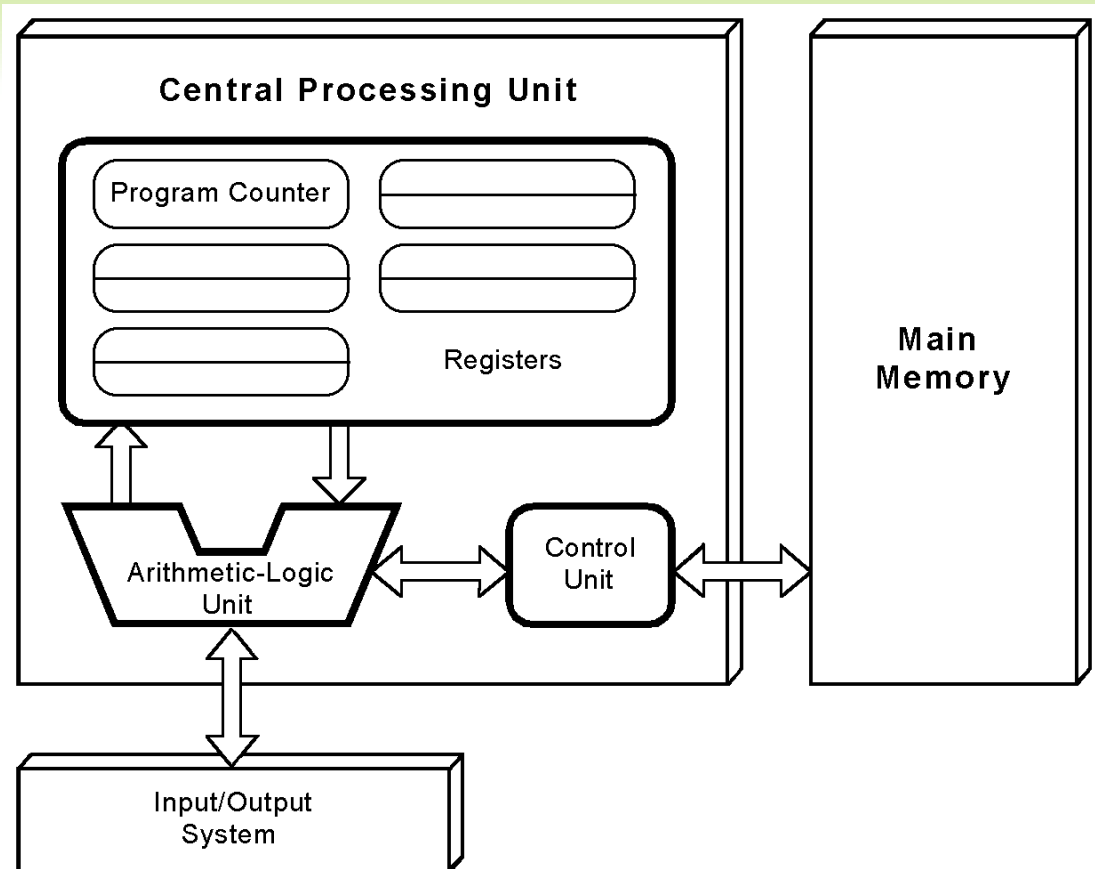
THE VON NEUMANN MODEL

Today's stored-program computers have the following features:

- ⦿ A Central processing Unit (CPU) (aka The Processor)
- ⦿ A main memory System (E.g. DRAM)
- ⦿ A storage System (E.g. Disk(s))
- ⦿ An I/O System
- ⦿ An interconnection network (wires) to show how signals are transmitted among the above components.
- ⦿ A CPU talks directly to main memory over a single data path.
- ⦿ Program instructions and data are moved from storage to memory where they can be accessed by the CPU using their memory addresses.

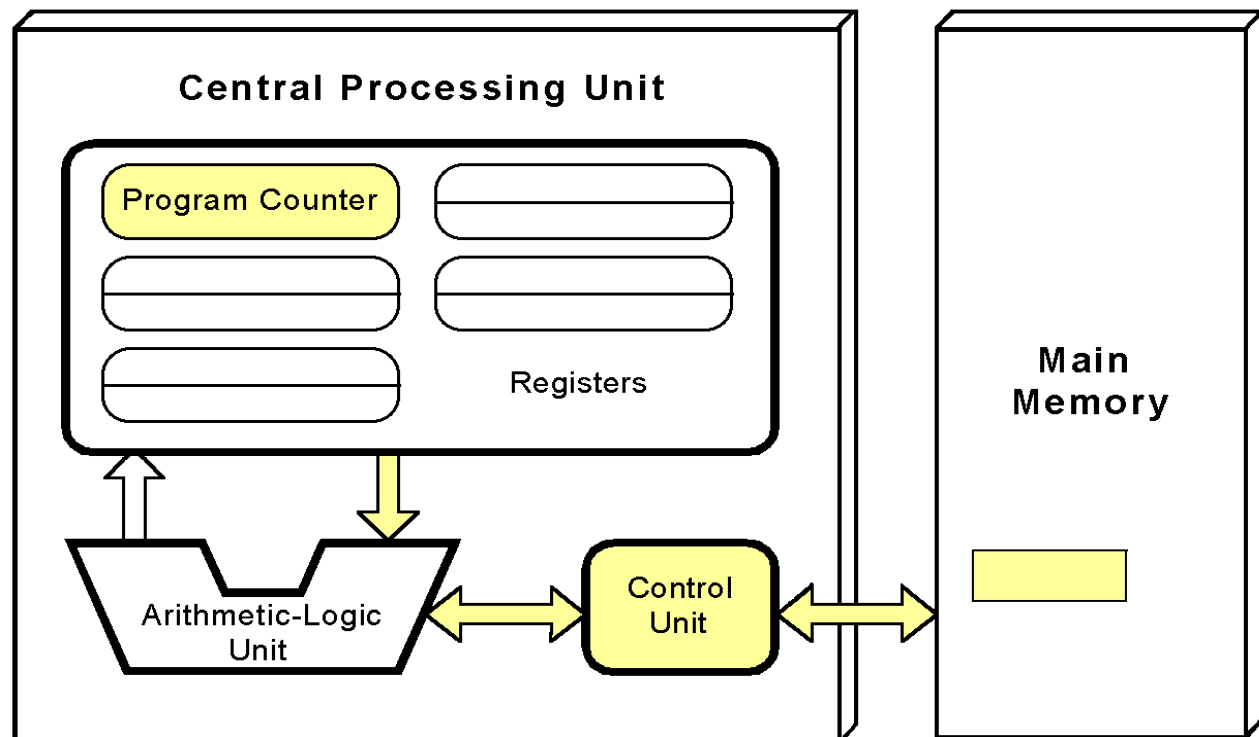
THE VON NEUMANN MODEL

- ⊙ The CPU (processor) contains:
- ⊙ An arithmetic-logic unit (ALU) that performs arithmetic and logical operations
- ⊙ A file of registers to serve as high-speed storage of operands
- ⊙ A control unit/datapath (the two are inseparable) that interprets instructions and causes them to be executed
- ⊙ A program counter (PC) that lives inside the datapath.
- ⊙ **This computer employ a fetch-decode-execute (or fetch-execute) cycle to run programs as follows . . .**



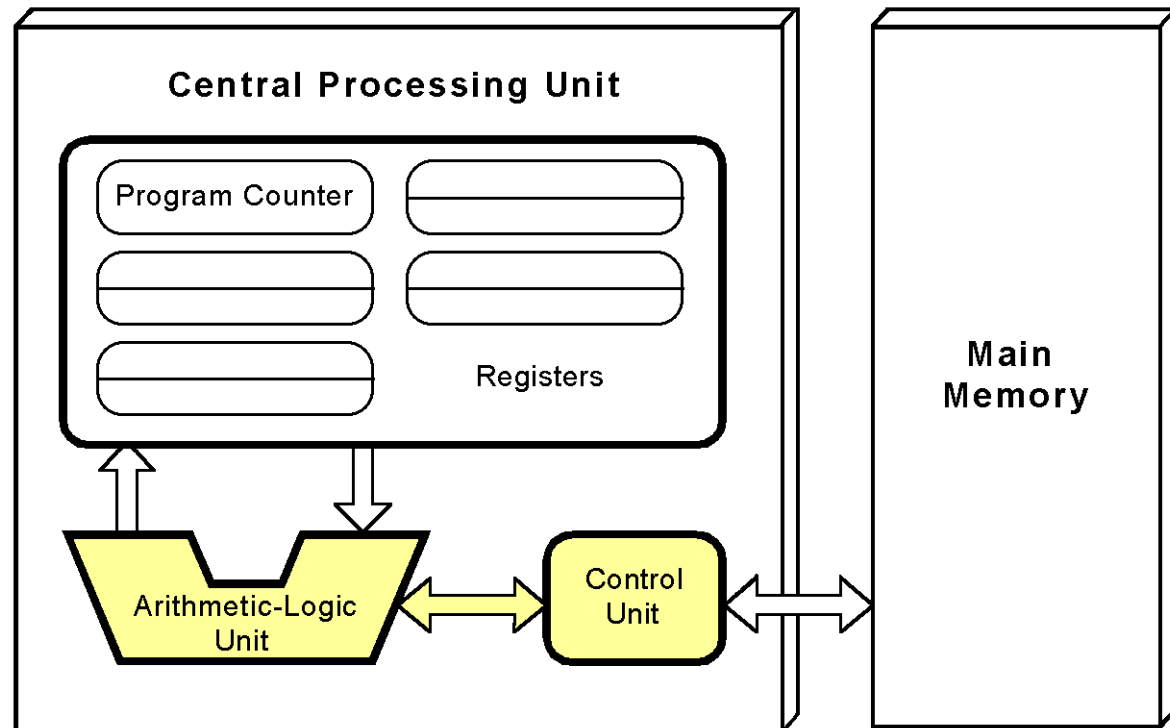
1.7 THE VON NEUMANN MODEL

- ⊙ Each machine has one **program counter (PC)**, an externally invisible register that holds the **memory address** of the next instruction to be fetched. The PC gets incremented automatically after a fetch.
- ⊙ The control unit fetches the next instruction from memory.



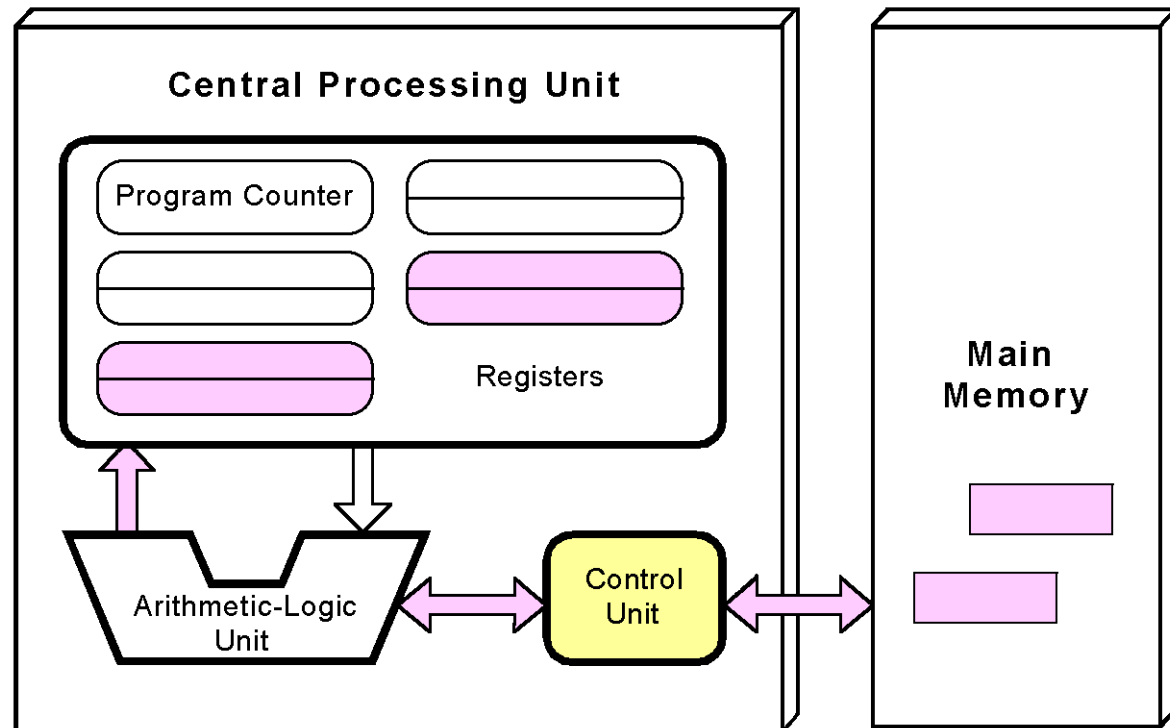
THE VON NEUMANN MODEL

- © The instruction is decoded into a language that the ALU can understand. The **datapath** (commonly called “**pipeline**”) executes instructions, while the **Control Unit** specifies how this work should be carried out.



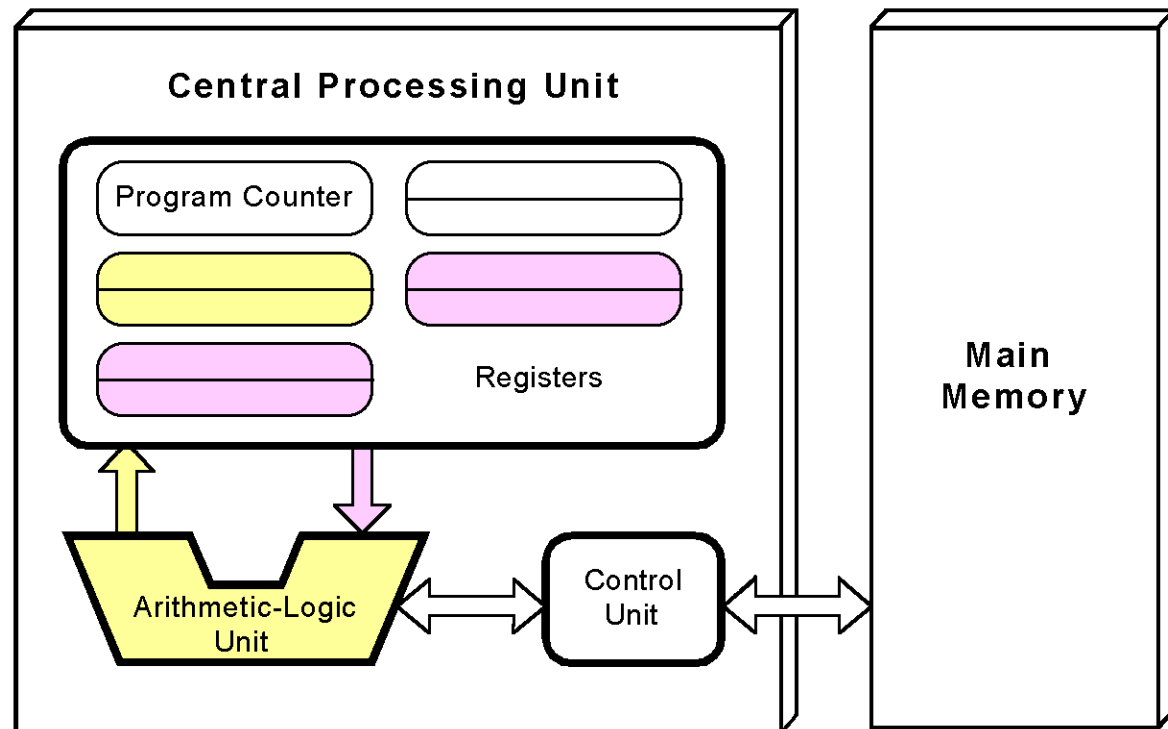
THE VON NEUMANN MODEL

- Any data operands required to execute the instruction are fetched from memory and placed into registers within the CPU.



THE VON NEUMANN MODEL

- ◎ The ALU executes the instruction and places the results in registers or memory (based on the instruction).



EXECUTION OF A STRAIGHT-LINE CODE-SEGMENT

- ⊙ Earlier slides showed the execution of one instruction.
- ⊙ Consider a straight-line (no branches) object code segment.
- ⊙ In straight-line code, the fetch-execute cycle steps through the sequence of machine instructions in program order.
- ⊙ This is the simplest form of control-flow of instructions.
- ⊙ The general form includes branches and is slightly more interesting because of the presence of ***loops*** and ***if statements***. An example can be seen on the next slide.

EXECUTION OF A CODE-SEGMENT

Add a constant value from **register f2** to every element of the array. **Register r1** has the address of the last element of the array. Register r2 points to the float address before the array. Each array element is 8 bytes.

MIPS code

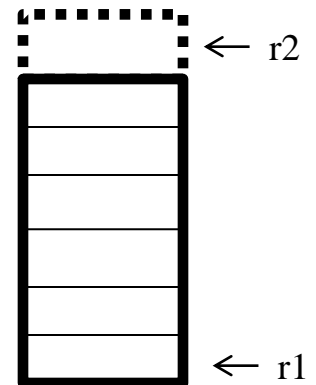
```

loop:      l.d  f0,0(r1)           ; f0 := a[j]
           add.d f4,f0,f2          ; f4 := a[j] + c
           s.d  f4,0(r1)           ; a[j] := f4
           daddiu r1,r1,#-8        ; j := j - 1
           bne  r1,r2,loop         ; if r1 != r2 then repeat
  
```

mini-MIPS (In the course, we will mostly look at MIPS code)

```

loop:      l.d  f0,Mem[r1:8]       ; f0 := a[j]
           add.d f4,f0,f2          ; f4 := a[j] + c
           s.d  f4,Mem[r1:8]       ; a[j] := f4
           subi r1,r1,8           ; j := j - 1
           bne  r1,r2,loop         ; if r1 != r2 then repeat
  
```



Floating-point array in memory

THE FETCH-EXECUTE CYCLE

1. The fetch engine ("f-box") in the pipeline fetches an instruction from memory as specified by the PC.
2. Assuming that memory is byte addressed, PC is unconditionally incremented by the length of an instruction in bytes.
3. Unless otherwise specified, we will consider all instructions to be encoded in 32 bits.
4. If the fetched instruction is a branch, further modification of the PC may take place, by some other box, at a later time.
5. The decode engine ("d-box") in the pipeline decodes the instruction, in conjunction with the control unit. This is the first time we know what kind of instruction we have fetched.
6. For a register-register instruction, the execution engine ("x-box") in the pipeline executes the instruction's **operation**. We will say the details for all types of operations later.

CONTEXT SWITCHING

- ⦿ Consider a floating-point multiply somewhere in the sequence.
- ⦿ Presumably, loads appear earlier in the program to bring the two operands of the multiply from memory.
- ⦿ Suppose the operands haven't arrived yet and the multiply cannot start execution.
- ⦿ Since we are moving through the program in program order, the whole program blocks. Since the processor is running precisely one program, the whole processor blocks (***stalls***).
- ⦿ Can the processor simply switch to another program?
- ⦿ That depends on the context. If the program will be blocked for a long time, then the cost of ***context switching*** will be worth it. (Example: a program that blocks for disk I/O).
- ⦿ However, if the program will be blocked for a much shorter time, then it makes sense just to stall the processor.
- ⦿ Modern CPUs spin wait.

NON-VON NEUMANN MODELS

- ⊙ Conventional stored-program computers have undergone many incremental improvements over the years.
- ⊙ These improvements include adding specialized buses, floating-point units (FPUs), and cache memories, to name only a few.
- ⊙ But enormous improvements in computational power require departure from the classic von Neumann architecture.
- ⊙ Adding processors is one approach.

NON-VON NEUMANN MODELS

- ◎ In the late 1960s, high-performance computer systems were equipped with dual processors to increase computational throughput.
- ◎ In the 1970s supercomputer systems were introduced with 32 processors.
- ◎ Supercomputers with 1,000 processors were built in the 1980s.
- ◎ In 1999, IBM announced its Blue Gene system containing over 1 million processors.

NON-VON NEUMANN MODELS

- ◎ Multicore architectures have multiple CPUs on a single chip.
- ◎ Dual-core and quad-core chips are commonplace in desktop systems.
- ◎ Multi-core systems provide the ability to multitask
 - ◎ E.g., browse the Web while printing a file.
- ◎ Multithreaded applications distribute mini-processes, *threads*, across one or more processors for increased throughput.

AMDAHL'S LAW

Amdahls Law deals with the Max expected improvement (in terms of speed) to an overall system when only part of the system is improved.

- ⊙ Used in parallel computing
- ⊙ Predict theoretical max speedup when using an infinite # of processors.
- ⊙ Limitation on improvement: Time required for sequential (serial) portion.

Simple Example:

- ⊙ Program P needs 20hrs to run on a single processor
- ⊙ Portion P1 requires 1h and cannot be parallelized. This is the serial portion
- ⊙ Minimum execution time is 1h no matter how many processors are used.
- ⊙ Max Speedup is 20x (old-time/improved-time = 20/1)

AMDAHL'S LAW

Another Example (Program P)

- ⊙ Portion P1 is 12% and can be parallelized → Portion P2 is 88% and cannot be (serial part of the program)
- ⊙ Max speedup of P = old-time/improved-time = $100/(100-12)$
- ⊙ $1/(1-0.12) = 1.136$ times

AMDAHL'S LAW

Another Example

A sequential task is split into four consecutive parts with following percentages of runtime: S is the speedup factor for each portion

P1=11% , cannot be improved S1=1x

P2=18%, S2=5x

P3=23% , S3=20x

P4=48%, S4=1.6x

Improved (new) running time = $P1/S1 + P2/S2 + P3/S3 + P4/S4$
 $= 0.11/1 + 0.18/5 + 0.23/20 + 0.48/1.6 = 0.4575$

Speedup = old-time/Improved running time = $1/0.4575 = 2.186$

Note: When dealing with percentages, use 1 or 100 as old-time

AMDAHL'S LAW

Another Example

Program P has portion A that uses 10% of the sequential time, and gets no parallel speed up, and portion B that uses 90% of the sequential time, and gets a parallel speed up equal to the number of cores. What is the run time of P on this parallel computer with a 100 identical cores?

$$\text{Old-time} = 10 + 90 = 100$$

$$\text{New-time} = 10 + 90/100 = 10.9$$

$$\text{Speedup (SU)} = \text{old-time/new-time} = 100/10.9 = 9.2$$

AMDAHL'S LAW

One More Example

On a uniprocessor, perfectly serial portion A of program P consumes 355 s, while perfectly parallel portion B consumes 645 s.

- a) To achieve no more than 51% of P's serial runtime requires what integral number of processors? What is then the parallel runtime?
- b) To achieve no more than 46% of P's serial runtime requires what integral number of processors? What is then the parallel runtime?

AMDAHL'S LAW

One More Example – Solution Part a

The initial serial time is $355 + 645 = 1000$ s. We want to achieve no more than 51% of this. i.e. max 510 s.

Serial portion cannot be changed so it consumes 355 s

which leaves us with no more than $510 - 355 = 155$ s for parallel part.

Processors required = old P2 time / new P2 time

$= 645 / 155 = 4.16$ so we need 5 processors.

AMDAHL'S LAW

One Last Example

On a uniprocessor, perfectly serial portion A of program P consumes 25 s, while perfectly parallel portion B consumes 75 s, for a total uniprocessor run time of 100 s. How many processors are required to achieve at least 75% of the 1,000-Processor speedup?

FORMALIZATION OF AMDAHL'S LAW

N = # processors, P is program portion that can be parallelized.

Max Possible or theoretical Speedup $S(N)$ is achieved when using an infinitely large N (an infinite number of processors)

As $N \rightarrow \text{infinity}$, $P/N \rightarrow 0$ and max speedup $S(N) = 1/(1-P)$

SpeedUp = old-time(single processor)/new-time(n processors)

Conclusion:

Best speedup or improvement results can be achieved as the value of P gets higher. We need to attempt to reduce the component $(1 - P)$ to the smallest possible value.