University of Southern California

Viterbi School of Engineering

EE352

Computer Organization and Architecture

Introduction

References:

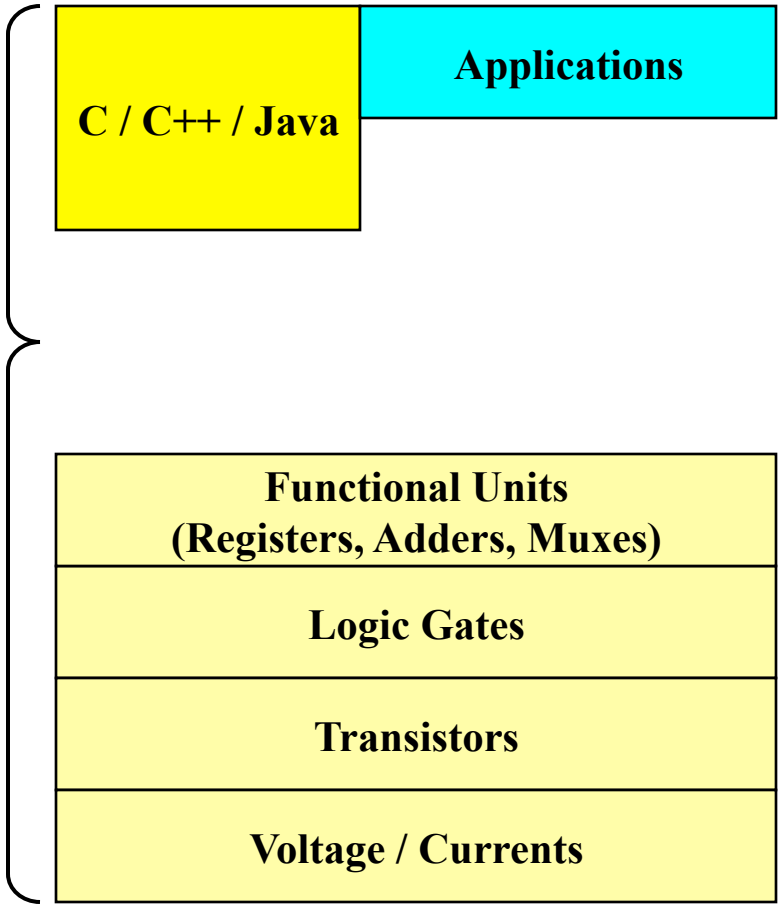
1) **Textbook**
2) **Mark Redekopp's slide series**

Shahin Nazarian

Spring 2010

Computer Systems Abstractions

- CS 101,102
  - Programming with high-level languages (HLLs) like C/C++/ Java
- EE 101,201
  - Digital hardware (registers, adders, muxes, etc.)

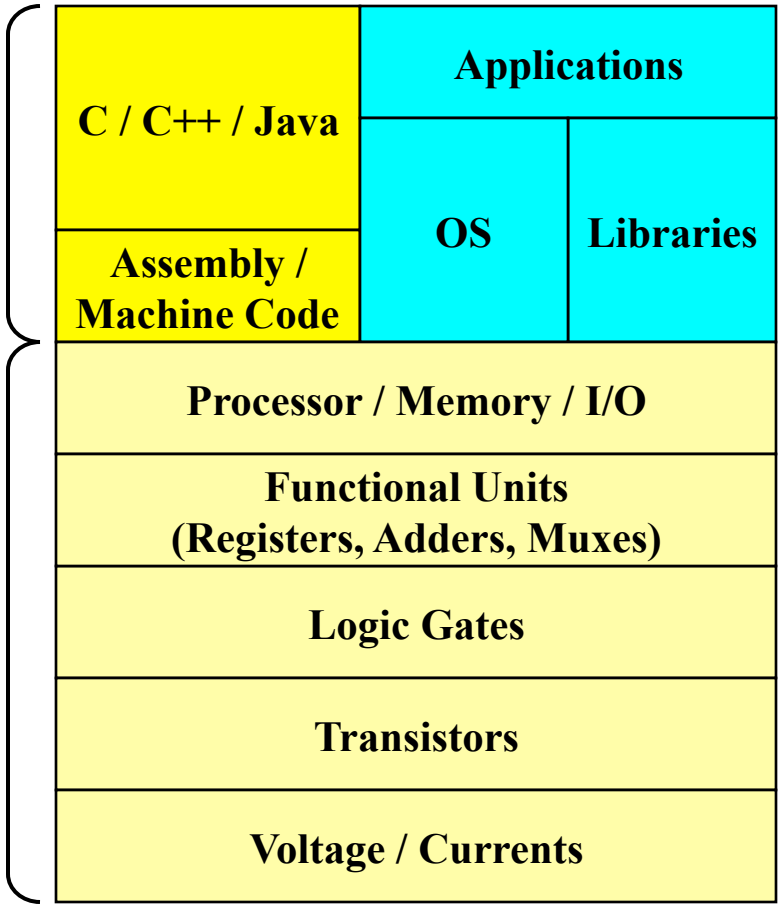


Computer Systems Abstractions

- CS101,102
  - Programming with high-level languages (HLLs) like C/C++/ Java
- EE101,201
  - Digital hardware (registers, adders, muxes, etc)
- EE352,357
  - Computer organization and architecture
    - *HW/SW System Perspective*
  - Topics
    - HW/SW interface
    - System Software
    - Assembly Language
    - Computer Architecture

SW

HW

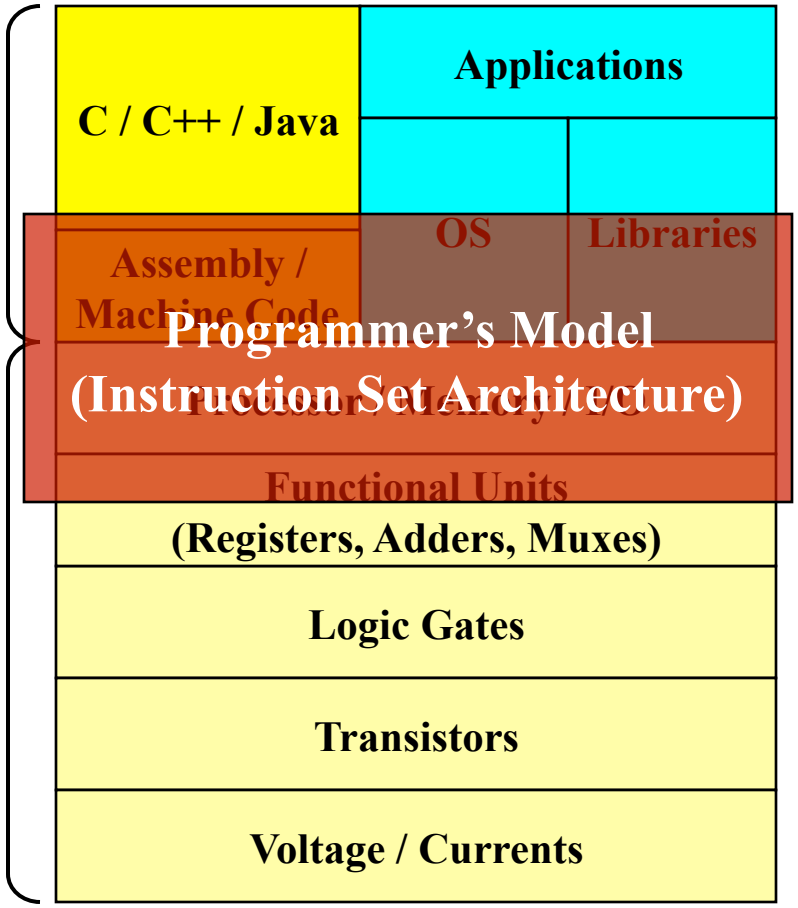


Computer Systems Abstractions

- EE 352,357
  - Computer organization and architecture
  - Topics
    - HW/SW interface
    - System Software
    - Assembly Language
    - Computer Architecture

      =

    - Instruction Set Architecture (ISA) / Programmer's Model



ISA

- ISA (may simply be referred to as the architecture) is one of the most important abstractions and is the interface between the hardware and the lowest level software
- ISA includes anything programmers need to know to make a binary machine language program work correctly, including instructions, I/O devices, etc.
- Typically OS (operating system) will encapsulate the details of doing I/O, allocating memory, and other low-level system functions so that application programmers do not need to worry about such details
- ISA allows computer designers to talk about functions independently from the hardware that performs them
- Implementation is the hardware that obeys the architecture abstraction

EE 352

- Focus on assembly language
  - What the basic software instructions are and how they are used to implement software programs
- Embedded Systems
  - Programming and low-level bit manipulations
- Computer organization/architecture
  - Organization of HW components (processor, memory, I/O) and its effect on software performance
  - Actual design of simple processors and other system components

Assembly Language

- 1000110010100000 can mean add A, B

- Communicating to computers in binary numbers was too tedious, therefore new notations were invented that were closer to the way humans think, e.g., programs to translate from symbolic notation to binary. The first of these programs was referred to as an assembler. E.g., the programmer writes Add A, B, and the assembler would translate that notation to 1000110010100000

- The symbolic language is referred to as the assembly language. In contrast the binary language that the machine understands is the machine language

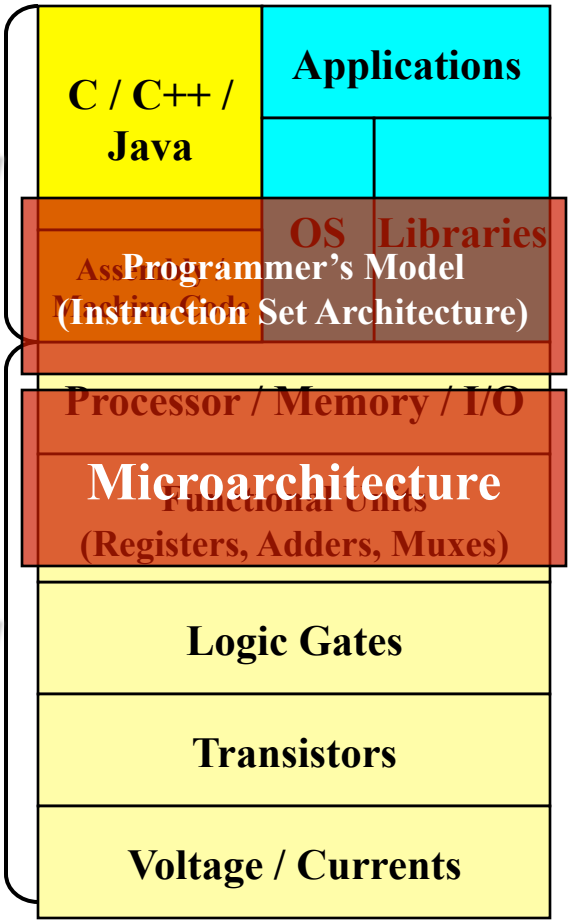
- Programmers owe their productivity and sanity to the creation of high-level programming languages and compilers that translate programs in those languages into assembly instructions

Organization & Architecture

- Computer organization refers to the components and interconnection necessary to form a computer system
- Computer architecture refers to a specific organization of components and other design choices

Levels of Architecture

- Levels of Architecture
  - System architecture
    - High-level HW org.
  - Instruction Set Architecture
    - A contract or agreement about what the HW will support and how the programmer can write SW for HW
    - Vocabulary that the HW understands and SW is composed of
  - Microarchitecture (detailed internal architecture of a processor)
    - HW implementation for executing instructions
    - Usually transparent to SW programs but not performance
    - Example: Intel and AMD have different microarchitectures but support essentially the same instruction set

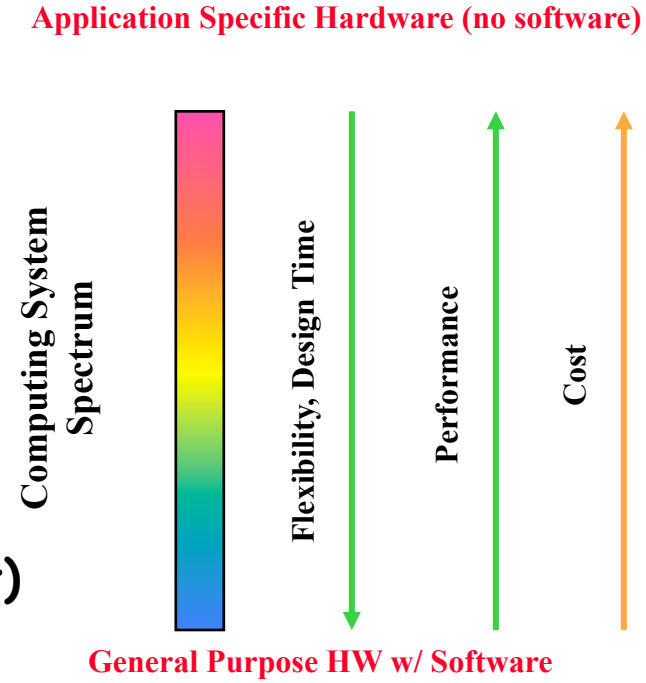


Why is Architecture Important

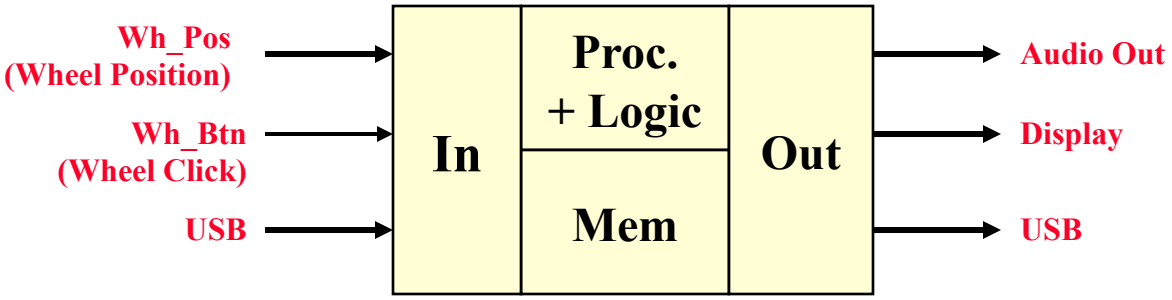
- Enabling even more powerful computers
- Different systems require different architectures
  - PCs
  - Servers
  - Embedded Systems
    - Simple control devices like ATMs, toys, appliances
    - Media systems like game consoles and MP3 players
    - Robotics

Digital System Spectrum

- Key idea: Any "algorithm" can be implemented in HW or SW or some mixture of both
- A digital system can be located anywhere in a spectrum of:
  - ALL HW: (a.k.a. Application-Specific IC's)
  - ALL SW: An embedded computer system
- Advantages of application specific HW
  - Faster, less power
- Advantages of an embedded computer system (i.e. general purpose HW for executing SW)
  - Reprogrammable (i.e. make a mistake, fix it)
  - Less expensive than a dedicated hardware system (single computer system can be used for multiple designs)
- MP3 Player: System-on-Chip (SoC) approach
  - Some dedicated HW for intensive MP3 decoding operations
  - Programmable processor for UI (User Interface) & other simple tasks



Embedded Example: iPod™

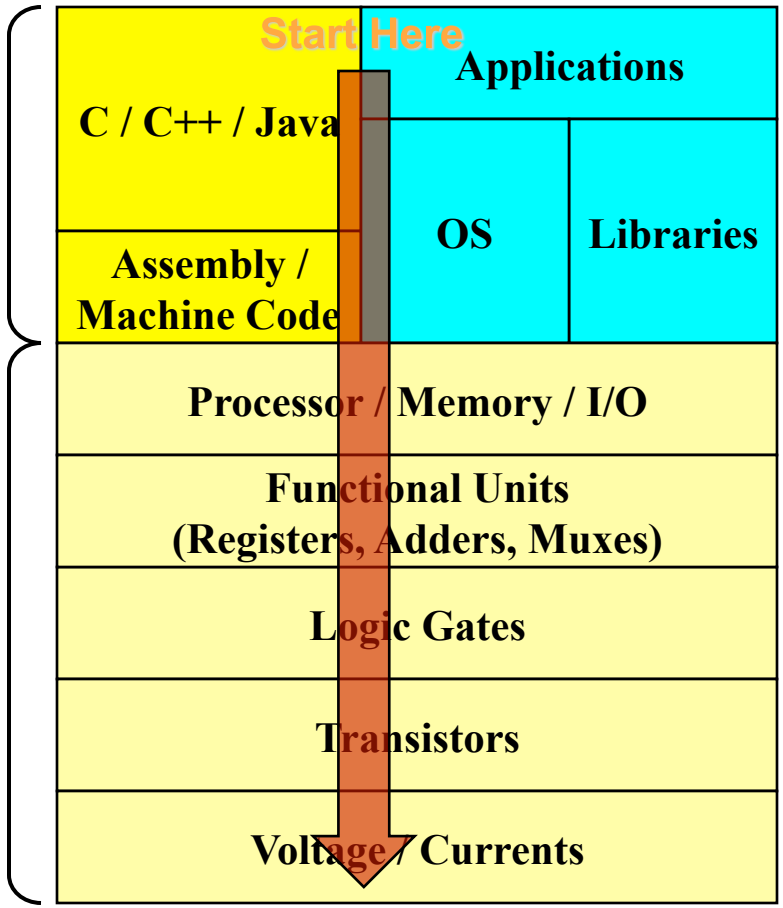


Using an embedded computer system we can write software code to control I/O rather than designing state machines, etc

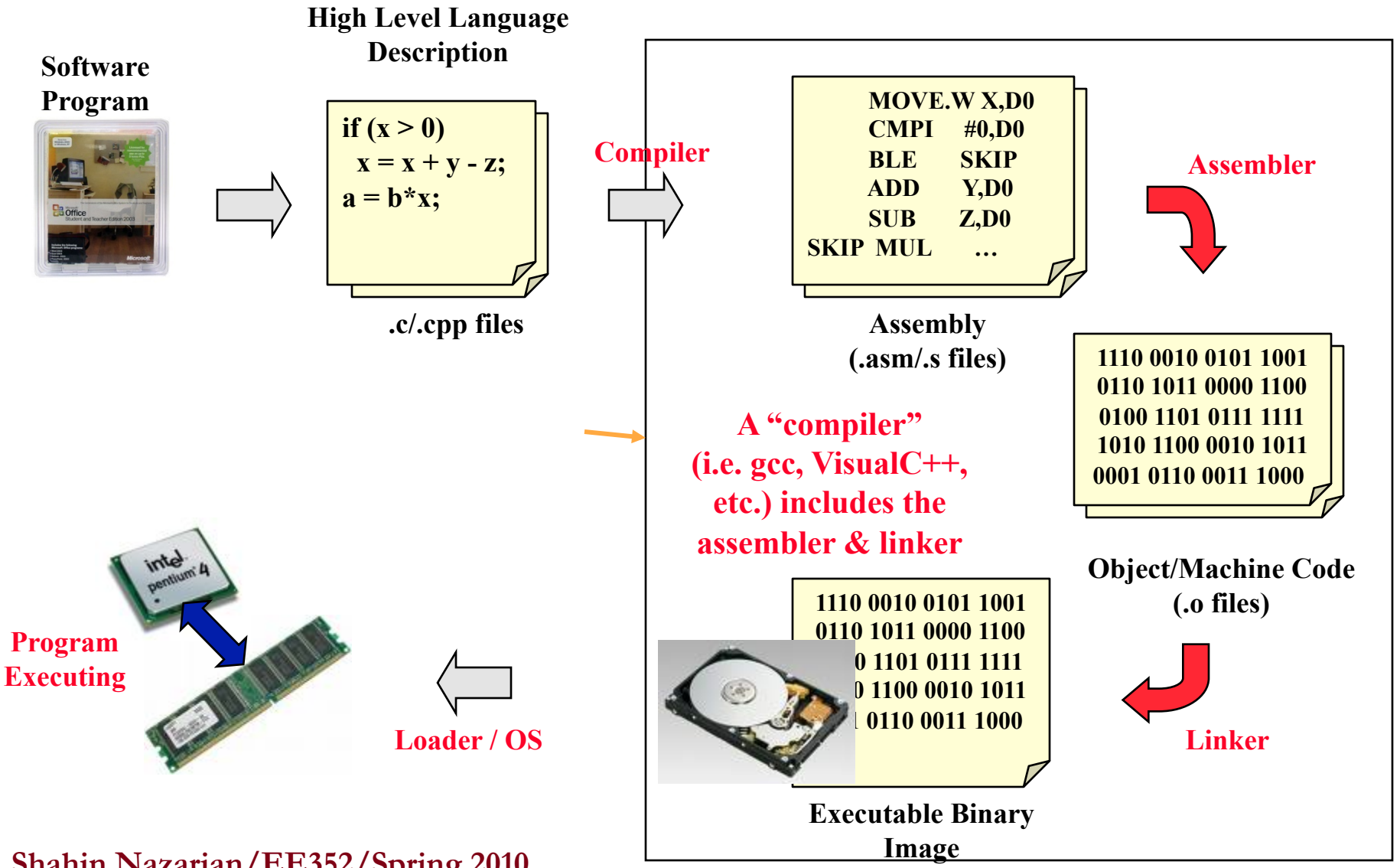
```
if (wh_btn && wh_pos==PLAY)
  {
  load_selected_file();
  play_file();
  start_time();
  }
else if (…)

void start_time(){
  time = 0;
  while (PLAYING) {
    sleep_1sec();
    time = time + 1;
    display_time(time);
  }
}
```

Computer Systems Tour

- How does a SW program get mapped and executed on a computer
- What components is a computer composed of and what are their functions
- How does the architecture affect performance



Software Process



Linker

- Based on what presented so far, a single change to one line of one procedure requires compiling and assembling the whole program. Complete retranslation is a terrible waste of computing resources, particularly for standard library routines, because programmers would be compiling and assembling routines that by definition almost never change
- An alternative is to compile and assemble each procedure independently, so that a change to one requires compiling and assembling only one procedure. This alternative requires a new system program called a link editor or linker which takes all the independently assembled machine language programs and stitches them together
- The linker produces an executable file that can be run on a computer

Loader

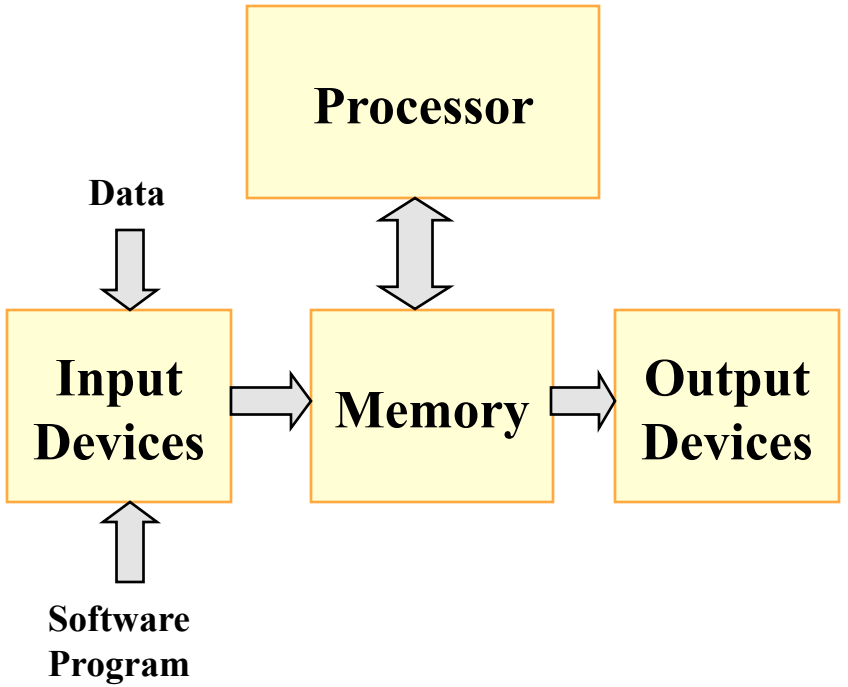
- OS reads the executable file to memory and starts it. The loader follows the following steps:
  - Reads the executable file header to determine size of the text and data
  - Copies the instructions and data from the executable file into memory
  - Copies the parameters (if any) to main program onto the stack
  - Initializes the machine registers and sets the stack pointer to the 1st free location
  - Jumps to start-up routine that copies the parameters into the argument registers and calls the main routine of the program. When the main routine starts, the start-up routine terminates the programs with an exit system call

Compiler Process

- A compiler such as 'gcc' performs 3 tasks:
  - Compiler
    - Converts HLL (high-level language) files to assembly
  - Assembler
    - Converts assembly to object (machine) code
  - Static Linker
    - Links multiple object files into a single executable resolving references between code in the separate files
  - Output of a compiler is a binary image that can be loaded into memory and then executed.
- Loader/Dynamic Linker
  - Loads the executable image into memory and resolves dynamic calls (to OS subroutines, libraries, etc.)

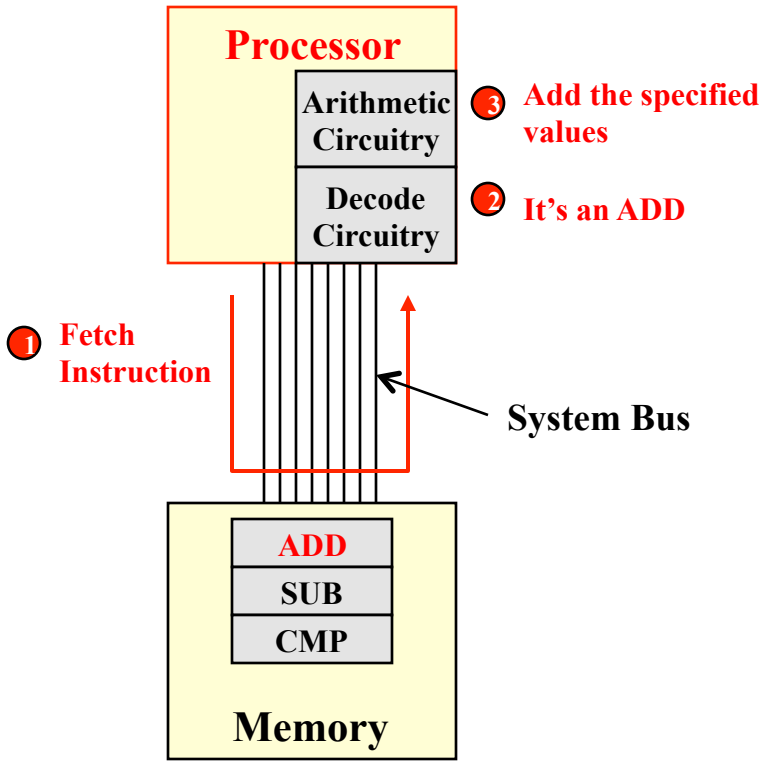
Hardware Components

- Processor
  - Executes the program and performs all the operations
  - Examples: Pentium 4, PowerPC, M68K /Coldfire
- Main Memory
  - Stores *data* and *program* (*instructions*)
  - Different forms:
    - RAM = read and write but volatile (lose values when power off)
    - ROM = read-only but non-volatile (maintains values when power off)
  - Moderately slower than the processor speeds
- Input / Output Devices
  - Generate and consume data from the system
  - Examples: Keyboard, Mouse, CD-ROM, Hard Drive, USB, Monitor display
  - MUCH, MUCH slower than the processor



Processor

- Performs the same 3-step process over and over again
  - Fetch an instruction from memory
  - Decode the instruction
    - Is it an ADD, SUB, etc.?
  - Execute the instruction
    - Perform the specified operation
- This process is known as the Instruction Cycle



Processor

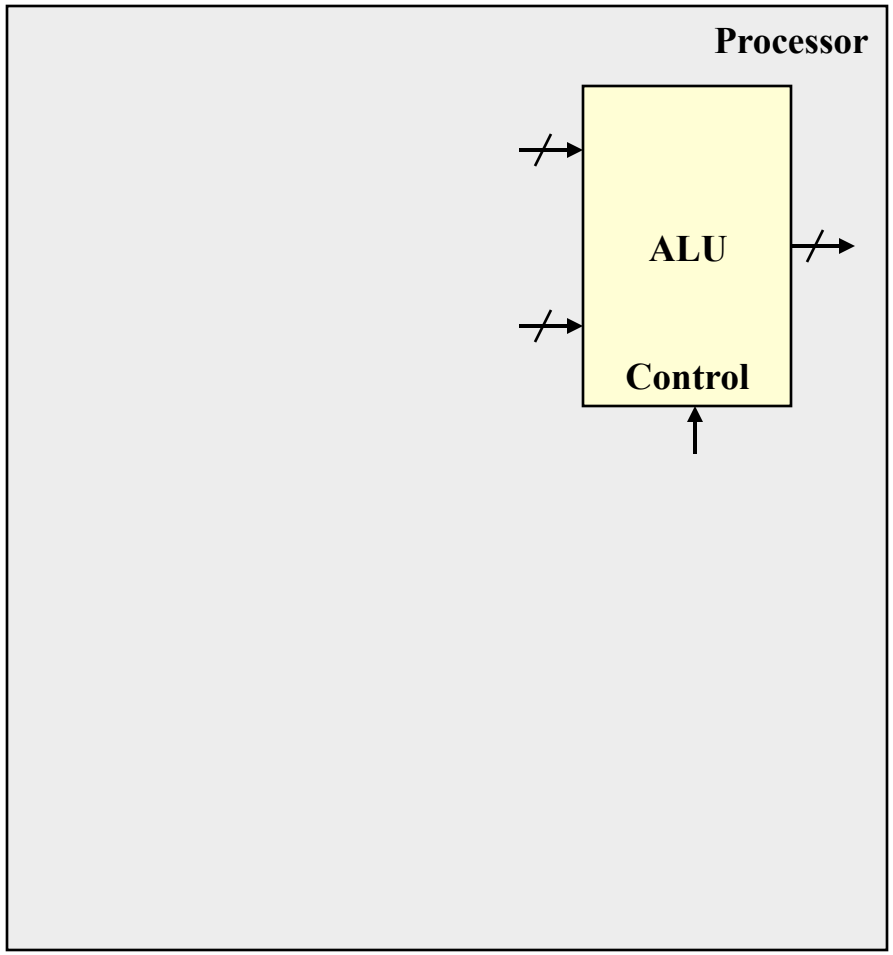
- Processor aka CPU (central processor unit) is the active part of the board, following the instructions of a program. It adds numbers, tests numbers, signals I/O devices to activate, etc.
- A processor logically comprises two main components: datapath and control. The datapath performs the arithmetic operations and control tells the datapath, memory, and I/O devices what to do according to the wishes of the instructions of the program
- Inside a processor there may be cache memory
- Cache memory consists of a small, fast memory that acts as a buffer for the DRAM memory. Cache is built using a different memory technology, SRAM (static random access memory). SRAM is faster but less dense and hence more expensive than DRAM

Processor (Cont.)

- Processors contain 4 subcomponents
  1. ALU (Arithmetic & Logical Unit)
  2. Registers
  3. Control Circuitry & System-Bus Interface
  4. Cache (Optional)

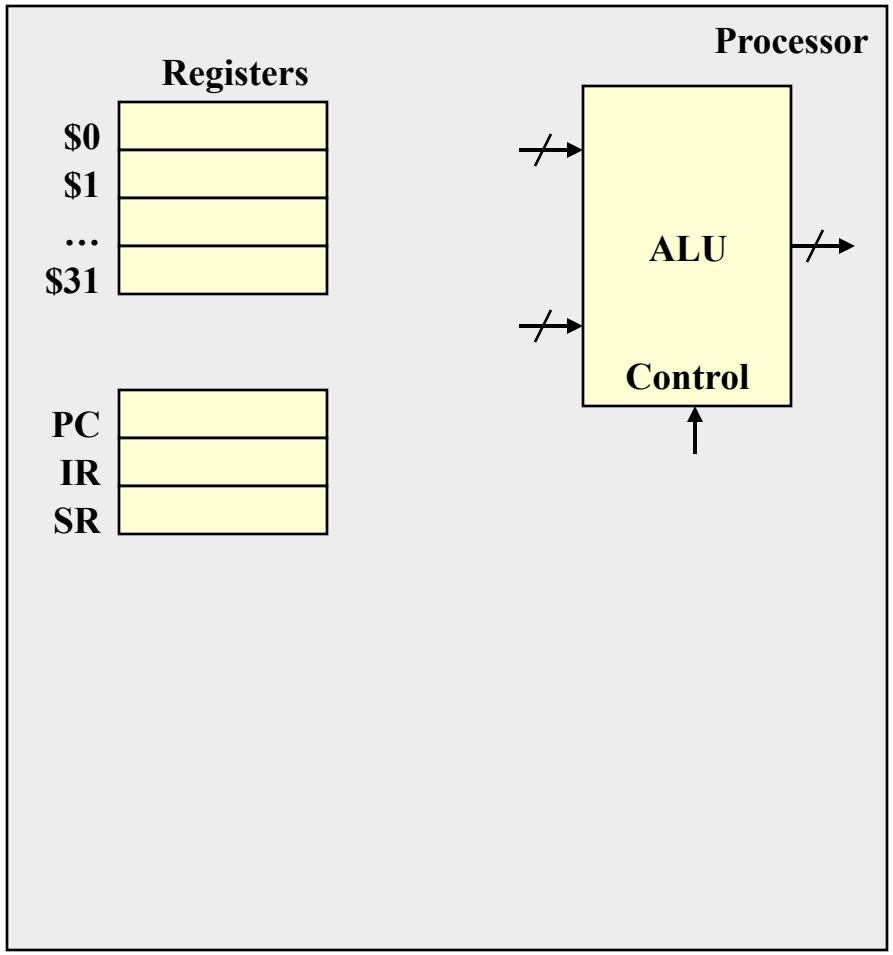
ALU

- Performs arithmetic and logical operations
- 2 inputs and 1 output value
- Control inputs to select operation (ADD, SUB, AND, OR…)



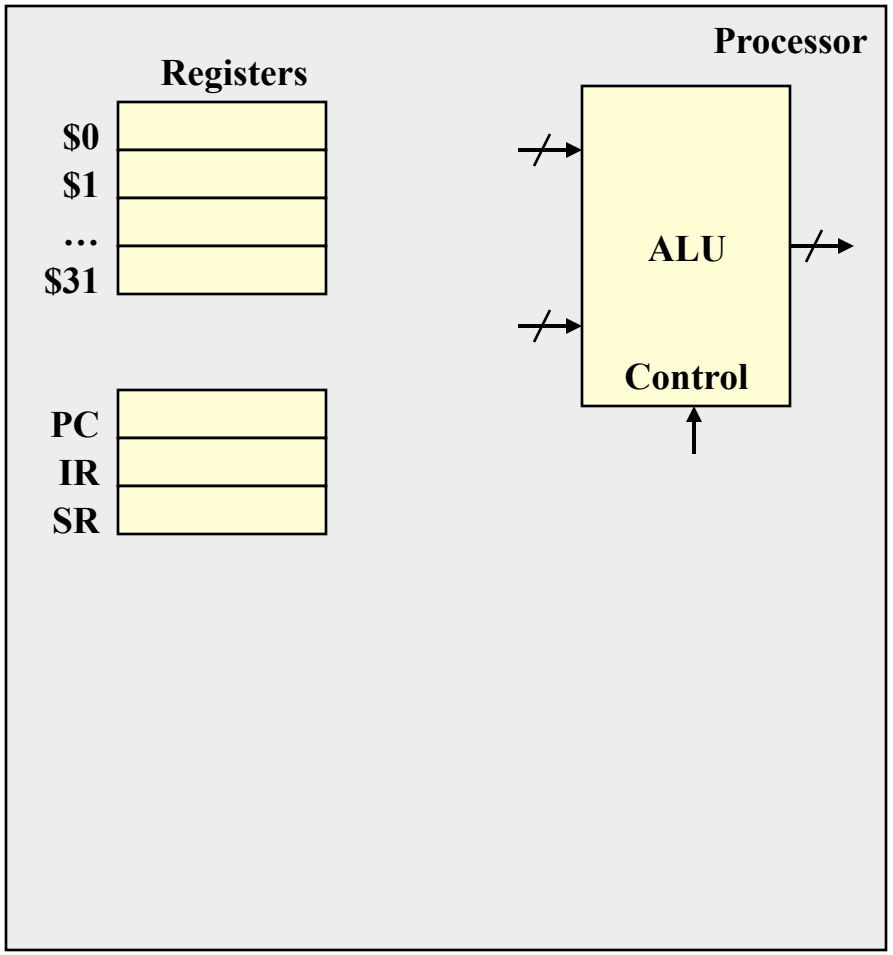
Registers

- Provide temporary storage for data
- 2 categories of registers
  - General Purpose Registers (GPR's)
    - For program data
    - Can be used by programmer as desired
    - Given names (e.g. $0 -$31)
  - Special Purpose Registers
    - For internal processor operation (not for program data)



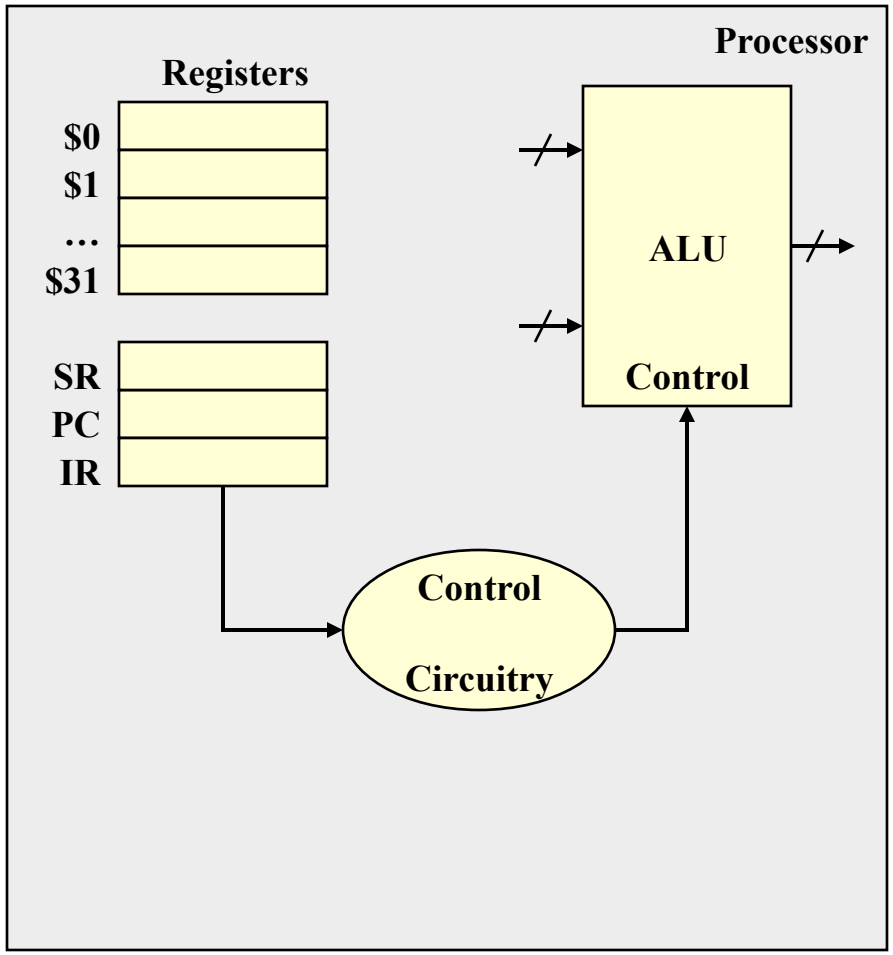
Registers

- GPR's
  - Faster to access than main memory
  - Keep data you are working with in registers to speed up execution
- Special Purpose Registers
  - Hold specific information that the processor needs to operate correctly
  - PC (Program Counter)
    - Pointer to (address of) instruction in memory that will be executed next
  - IR (Instruction Register)
    - Stores the instruction while it is being executed
  - SR (Status Register)
    - Stores status/control info
  - ….



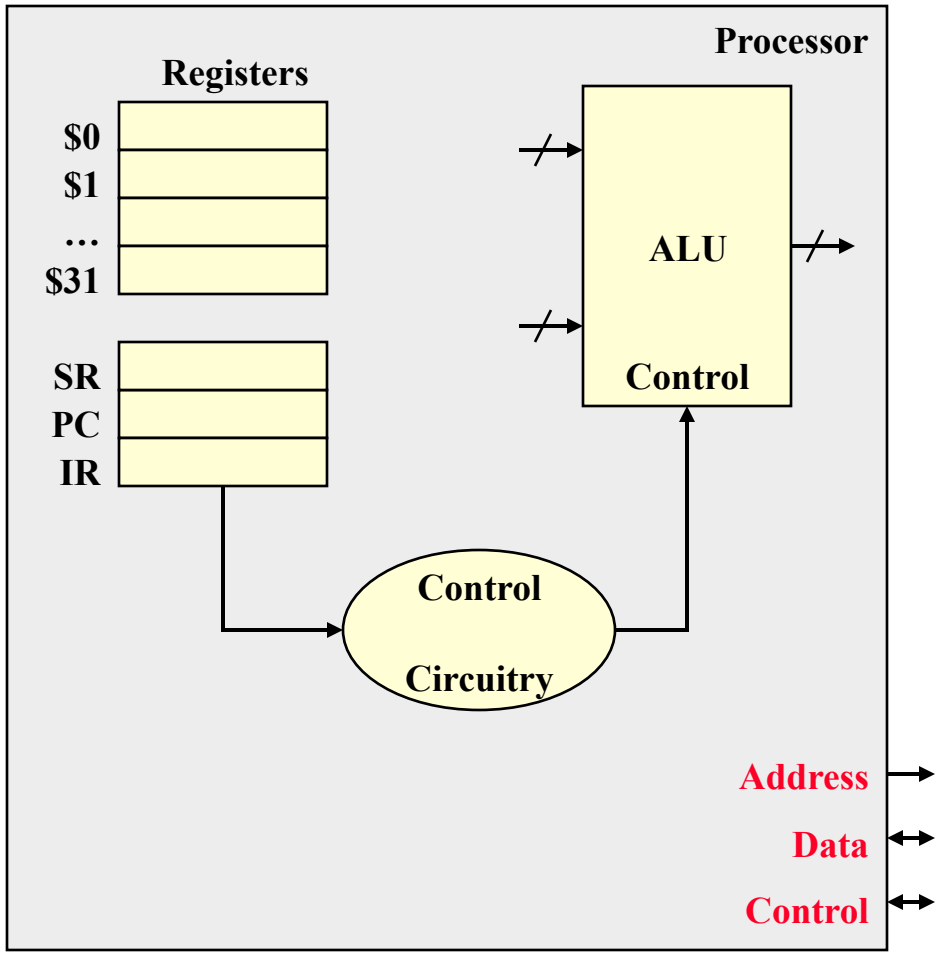
Control Circuitry

- Decodes each instruction
- Selects appropriate registers to use
- Selects ALU operation
- And more…



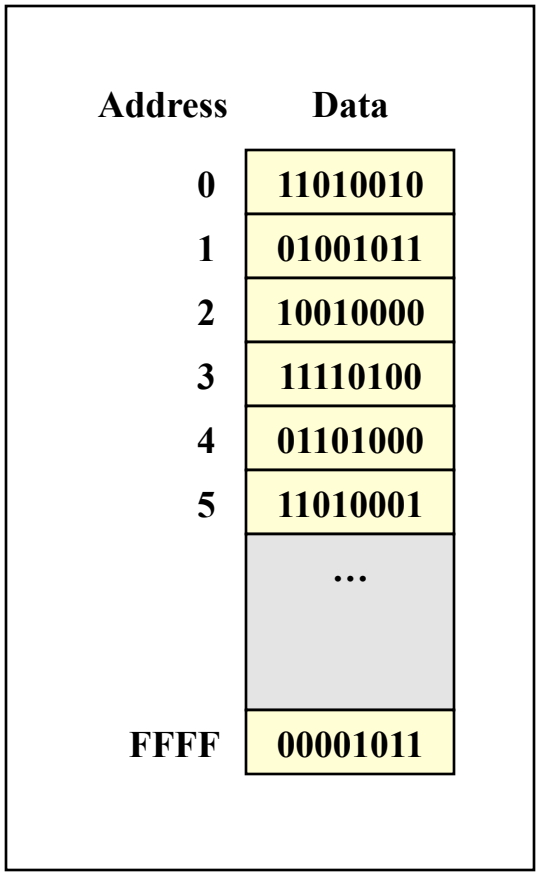
System Bus Interface

- System bus is the means of communication between the processor and other devices
  - Address
    - Specifies location of instruction or data
  - Data
  - Control



Memory

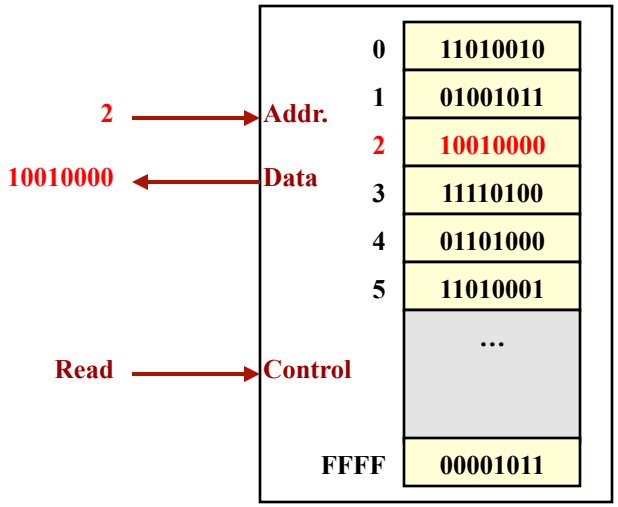
- Set of cells that each store a group of bits (usually, 1 byte = 8 bits)
- Unique address assigned to each cell
  - Used to reference the value in that location
- Numbers and instructions are all represented as a string of 1's and 0's



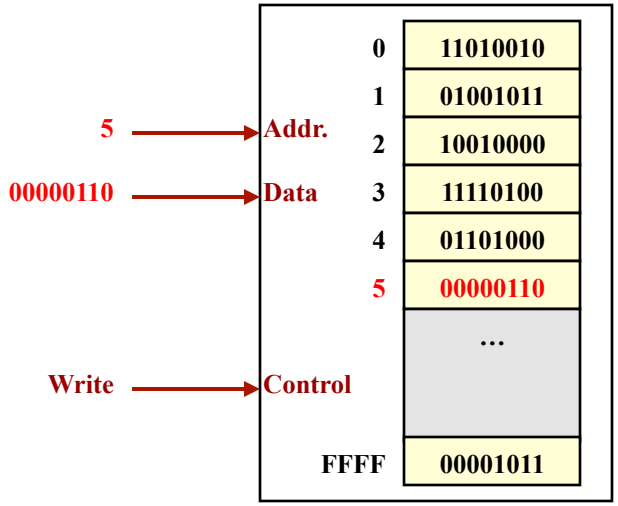
Memory Device

Memory Operations

- Memories perform 2 operations
  - Read: retrieves data value in a particular location (specified using the address)
  - Write: changes data in a location to a new value
- To perform these operations a set of address, data, and control inputs/outputs are used
  - Note: A group of wires/signals is referred to as a 'bus'
  - Thus, we say that memories have an address, data, and control bus



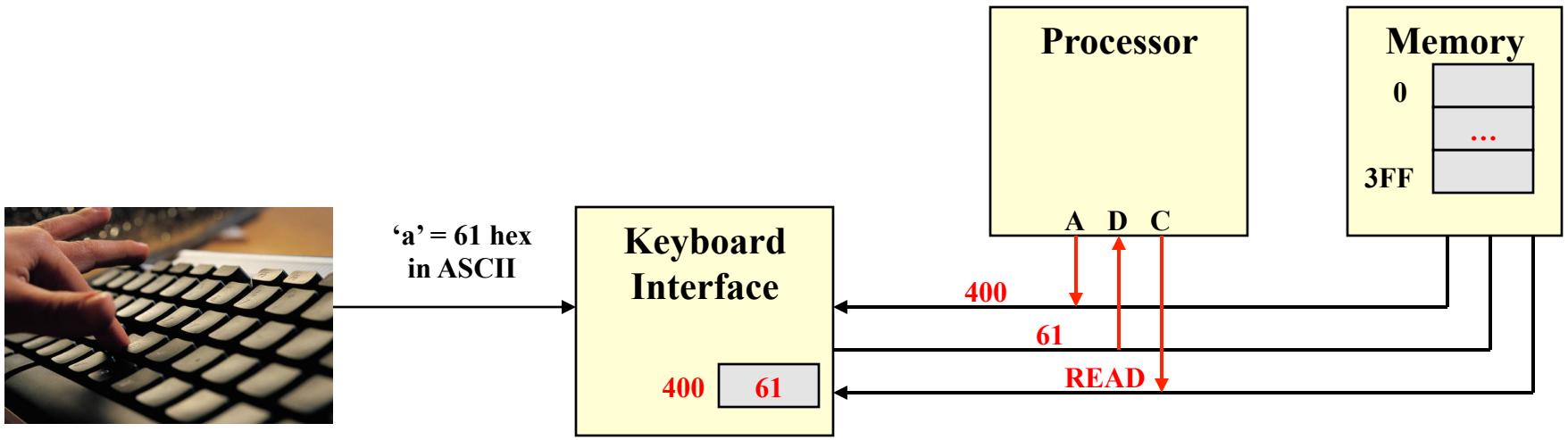
A Read Operation



A Write Operation

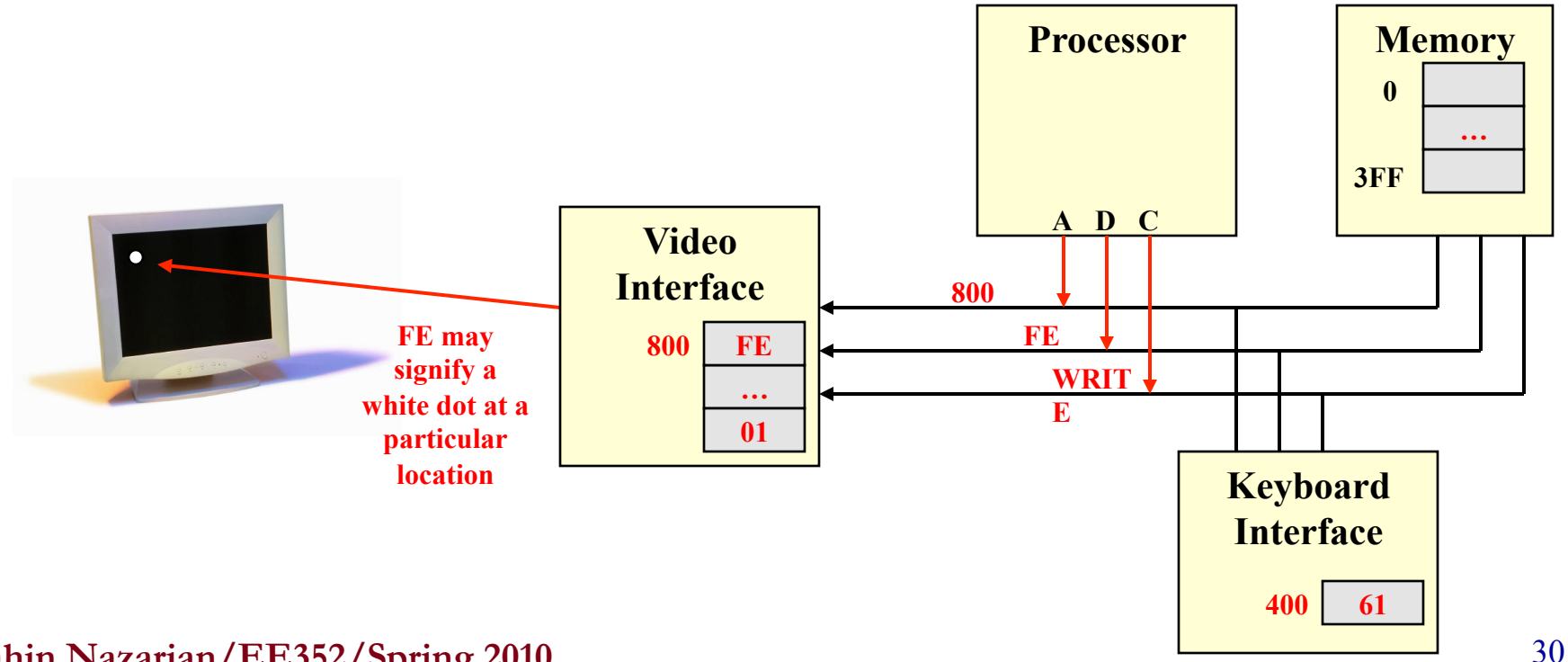
Input / Output

- Keyboard, Mouse, Display, USB devices, Hard Drive, Printer, etc.
- Processor can perform reads and writes on I/O devices just as it does on memory
  - I/O devices have locations that contain data that the processor can access
  - These locations are assigned unique addresses just like memory



Input / Output

- Writing a value to the video adapter can set a pixel on the screen

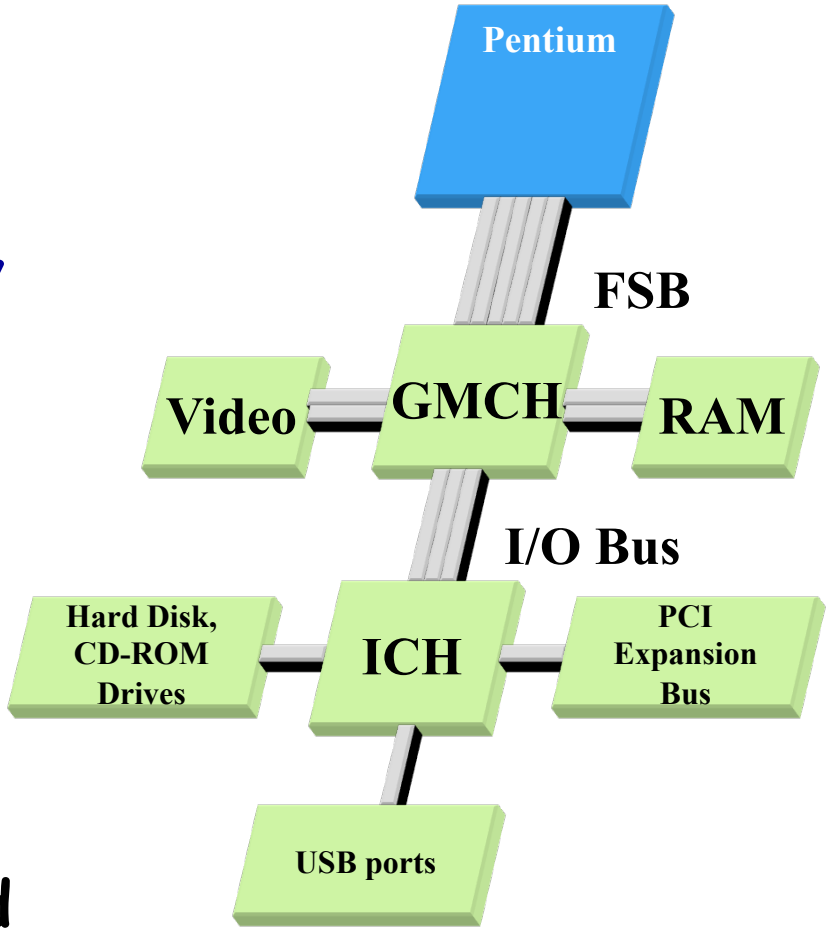


Computer Organization Issues

- Components run at different speeds
  - Processor can perform operations very quickly (~ 1 ns)
  - Memory is much slower (~ 50 ns) due to how it is constructed & its shear size [i.e. it must select/look-up 1 location from millions]
    - Speed is usually inversely proportional to size (i.e. larger memory => slower)
  - I/O devices are much slower
    - Hard Drive (~ 1 ms)
  - Intra-chip signals (signals w/in the same chip) run much faster than inter-chip signals
- Design HW and allocate HW resources to accommodate these inherent speed differences

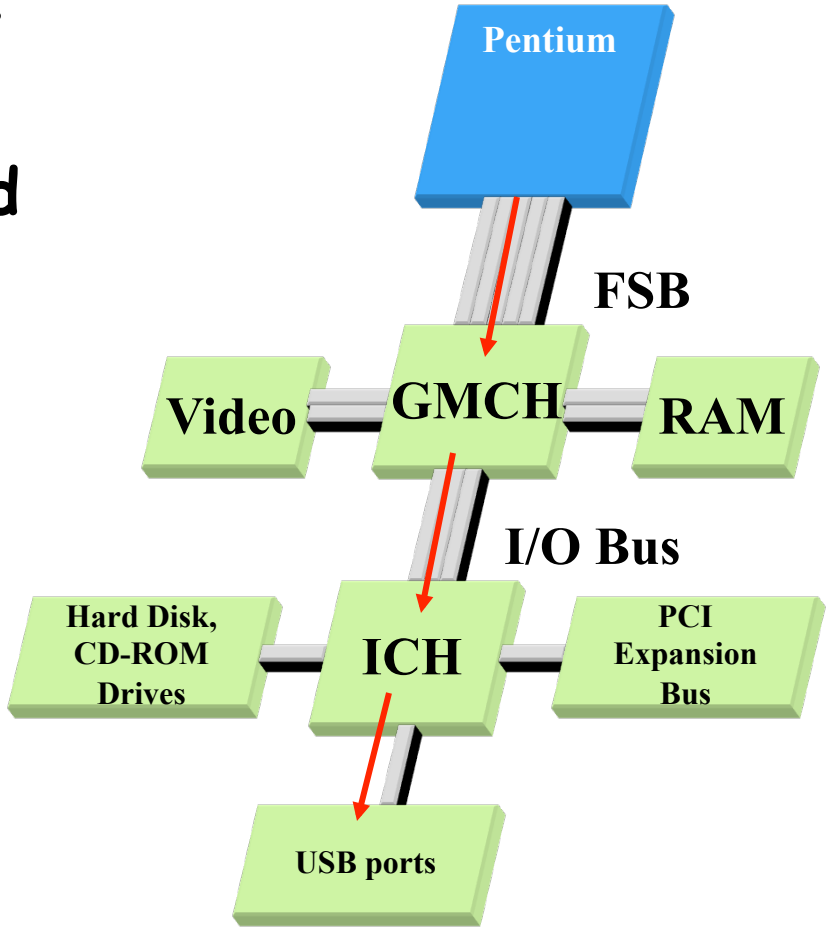
Implementation Example – Pentium

- Arrange bus in segments based on needed speed of access
- Front Side Bus (FSB) connects processor and GMCH (Graphics and Memory Controller Hub)
  - Wide (64-bits)
  - Faster
- I/O Bus connects GMCH and ICH (I/O Controller Hub)
  - Narrow (>12-bits)
  - Slower
- PCI expansion bus allows for new I/O devices to be added



Pentium – Reads and Writes

- Routed to correct device based on address
  - GMCH will route the read or write to the Video controller or RAM or on to the ICH
  - ICH will route the read or write to the appropriate device or onto the PCI (Peripheral Component Interconnect) Expansion bus



A read or write to a USB device is examined by the GMCH, found to not be in the range of the Video or RAM addresses, and forwarded on

[Optional] PCI (Peripheral Component Interconnect)

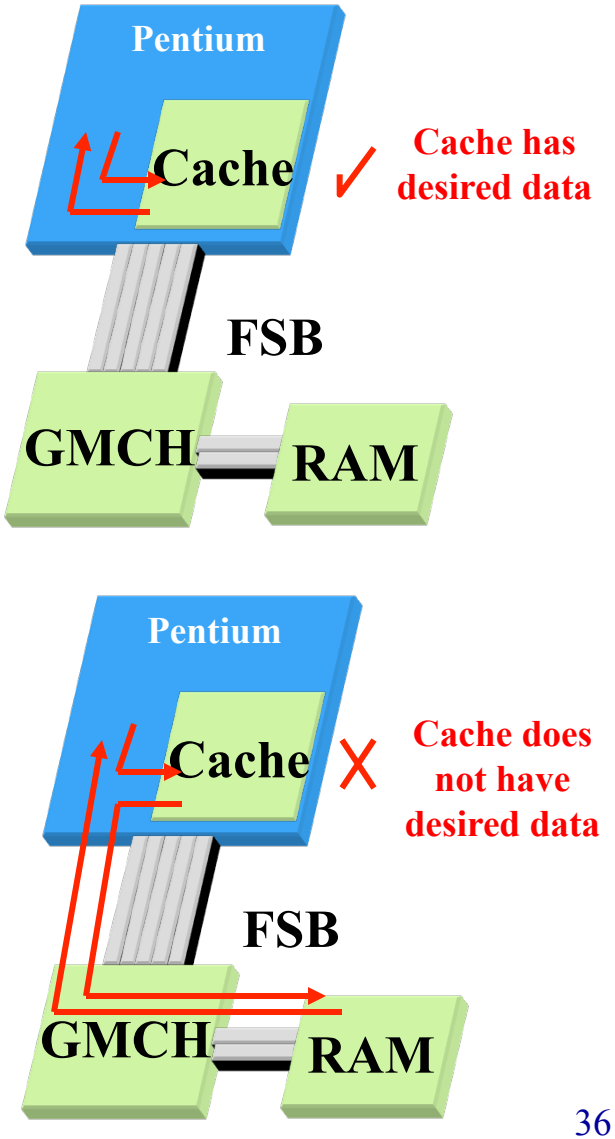
- Conventional PCI (part of the PCI Local Bus standard and often shortened to PCI) is a computer bus for attaching hardware devices in a computer. These devices can take either the form of an integrated circuit fitted onto the motherboard itself, called a *planar device* in the PCI specification, or an expansion card that fits into a slot
- The PCI Local Bus is common in modern PCs, where it has displaced ISA (Industry Standard Architecture) bus and VESA (Video Electronics Standards Association) Local Bus as the standard expansion bus, and it also appears in many other computer types
- Despite the availability of faster interfaces such as PCI-X and PCI Express, conventional PCI remains a very common interface

[Optional] PCI (Cont.)

- The PCI specification covers the physical size of the bus (including wire spacing), electrical characteristics, bus timing, and protocols. The specification can be purchased from the PCI Special Interest Group (PCI-SIG)
- Typical PCI cards used in PCs include: network cards, sound cards, modems, extra ports such as USB or serial, TV tuner cards and disk controllers. Historically video cards were typically PCI devices, but growing bandwidth requirements soon outgrew the capabilities of PCI. PCI video cards remain available for supporting extra monitors and upgrading PCs that do not have any AGP (Accelerated Graphics Port) or PCI Express slots
- Many devices traditionally provided on expansion cards are now commonly integrated onto the motherboard itself, meaning that modern PCs often have no cards fitted. However, PCI is still used for certain specialized cards, although many tasks traditionally performed by expansion cards may now be performed equally well by USB devices

Cache Example

- Small, fast, on-chip memory to store copies of recently-used data
- When processor attempts to access data it will check the cache first
  - If the cache has the desired data, it can supply it quickly
  - If the cache does not have the data, it must go to the main memory (RAM) to access it



ARCHITECTURE DRIVERS & IMPACT ON SW TRENDS

Architecture Issues

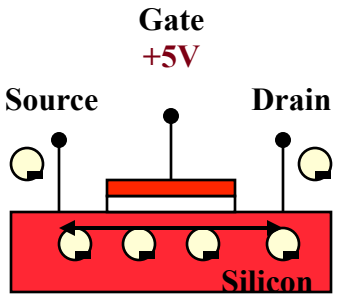
- What do we do with the ever-increasing number of transistors available to us
  - Moore's Law
- How do we mitigate the issues that affect performance
  - Memory wall
  - Sequential programming paradigm
  - Power consumption

Moore's Law & Transistors

- Moore's Law = Number of transistors able to be fabricated on a chip will double every 1.5 – 2 years
- Transistors are the fundamental building block of computer HW
  - Switching devices: Can conduct [on = 1] or not-conduct [off = 0] based on an input voltage

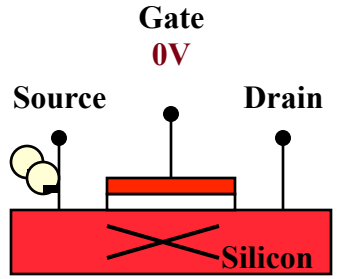
Transistor

- 3-terminal device
  - Gate input: the control input; it's voltage determines whether current can flow
  - Source & Drain: terminals that current flows from/to
- Many transistors can be fabricated on one piece of silicon (i.e. an integrated chip, IC)



Transistor is 'on'

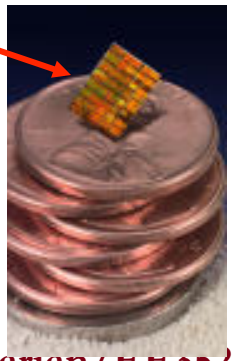
High voltage at gate allows current to flow from source to drain



Transistor is 'off'

Low voltage at gate prevents current from flowing from source to drain

Integrated Circuit

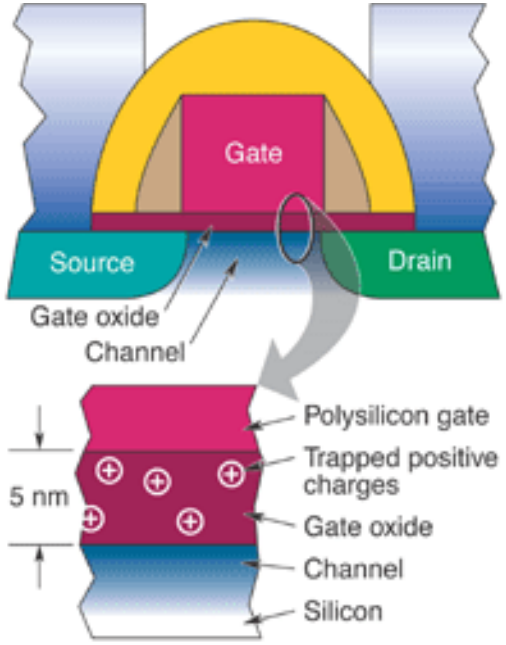


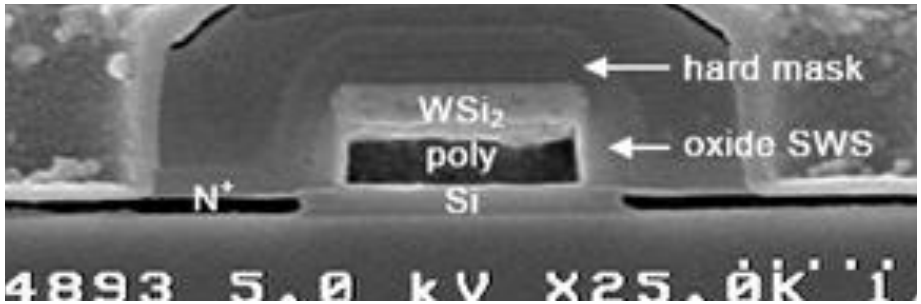
Actual silicon wafer is quite small but can contain ~300 million transistors

Silicon wafer is then packaged to form the chips we are familiar with

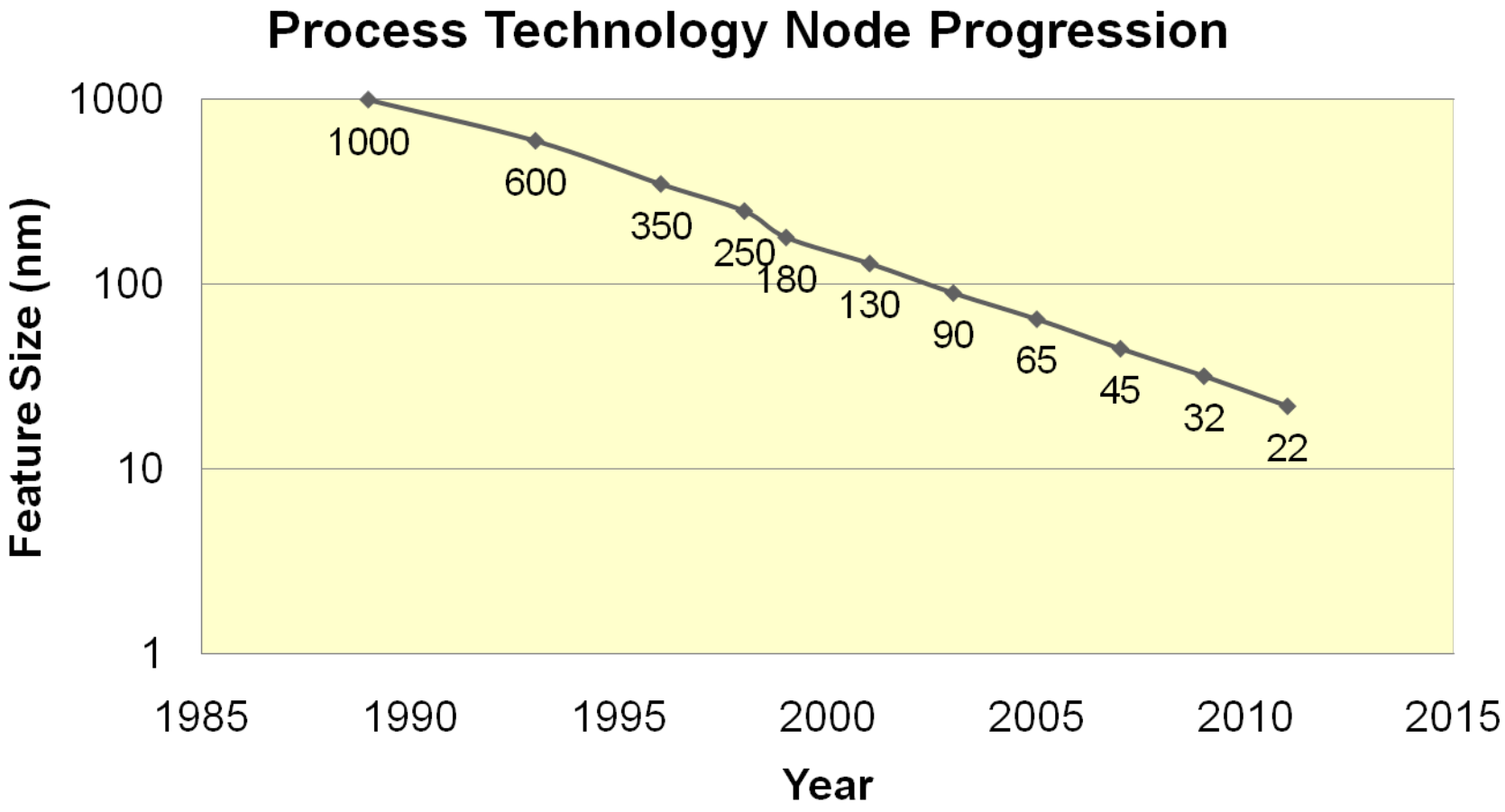
Transistor Physics

- Cross-section of transistors on an IC
- Moore's Law is founded on our ability to keep shrinking transistor sizes
  - Gate/channel width shrinks
  - Gate oxide shrinks
- Transistor feature size is referred to as the implementation "technology node"

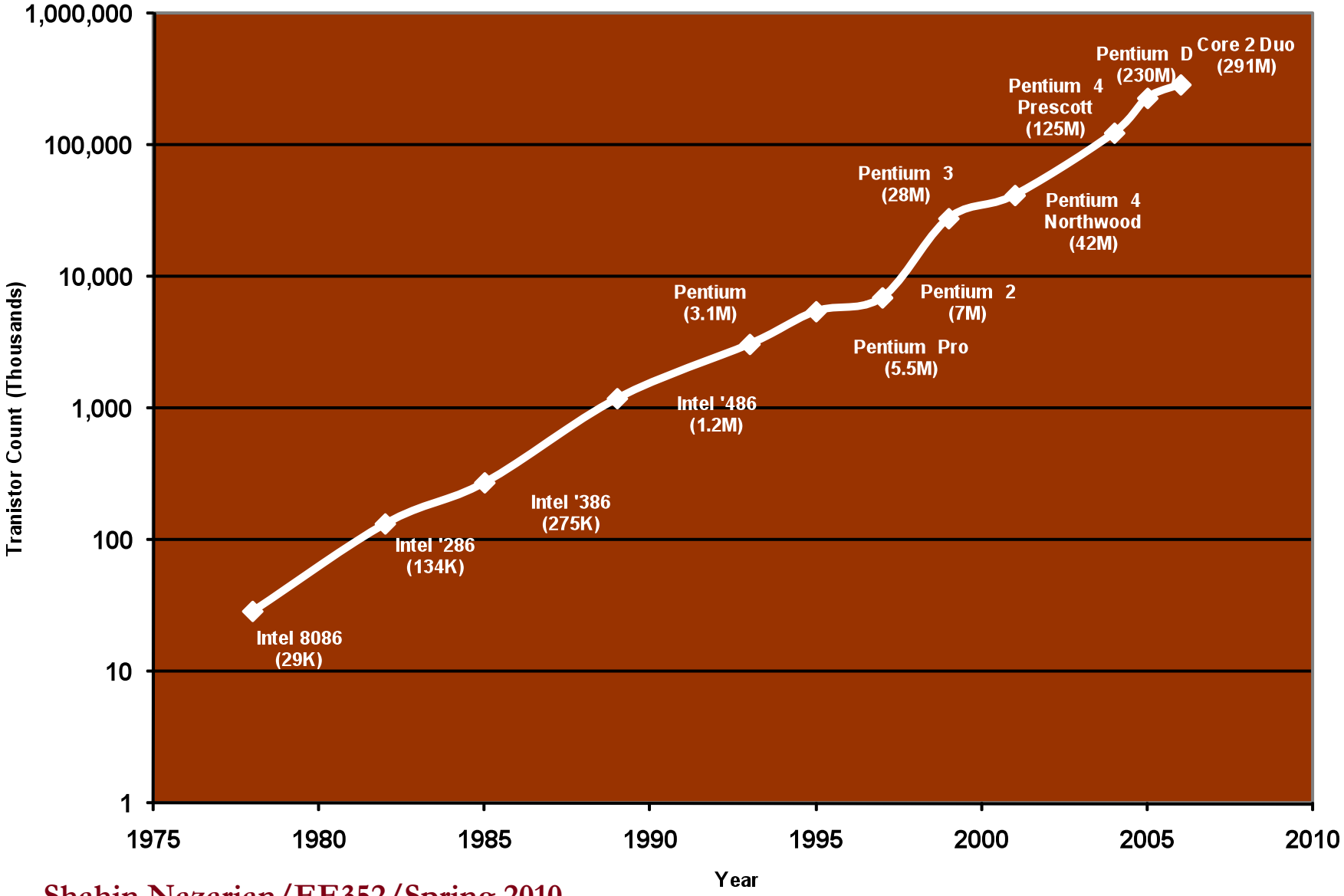




Technology Nodes



Growth of Transistors on Chip



Implications of Moore's Law

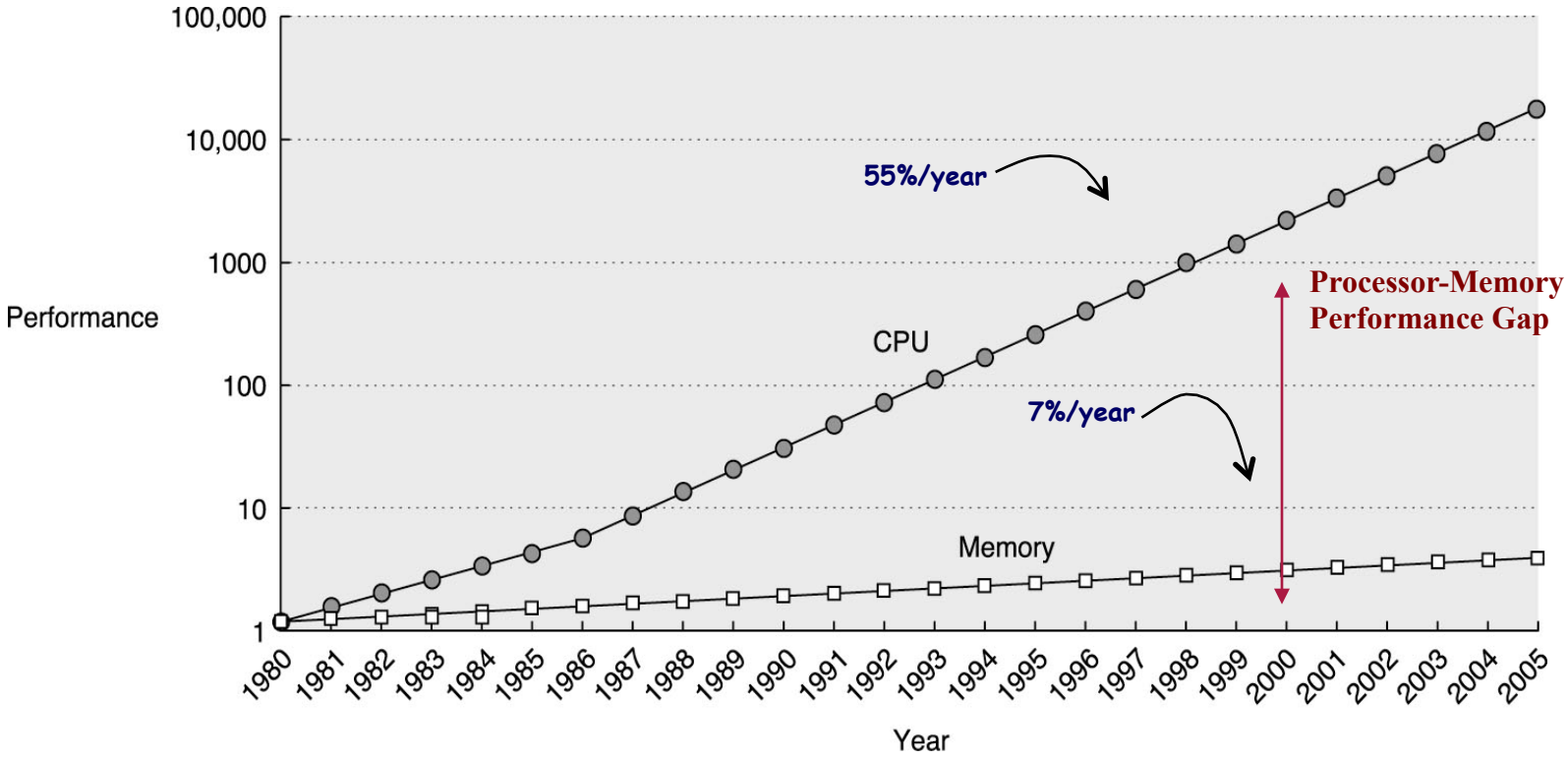
- What should we do with all these transistors
  - Put additional simple cores on a chip
  - Use transistors to make cores execute instructions faster
  - Use transistors for more on-chip cache memory
    - Cache is an on-chip memory used to store data the processor is likely to need
    - Cache is Faster than main-memory which is on a separate chip and much larger (thus slower)

Architecture Issues

- What do we do with the transistors available to us
  - Moore's Law
- How do we mitigate the issues that affect performance
  - Memory wall
  - Sequential programming paradigm
  - Power consumption

Memory Wall Problem

- Processor performance is increasing much faster than memory performance



Hennessy and Patterson,
Computer Architecture –
A Quantitative Approach (2003)

Memory Wall

- Problem
  - Memory performance measured via:
    - Latency = time to access a value from memory
    - Bandwidth/Throughput = total transfers/accesses per second
  - Latency is hard to improve, but bandwidth can be improved
- Solutions
  - Registers, Cache Memory & Local stores
    - Addresses bandwidth and latency
  - Memory organization (interleaving, banks, DDR [Double Data Rate], etc.)
    - Addresses bandwidth

Sequential Programming Paradigm

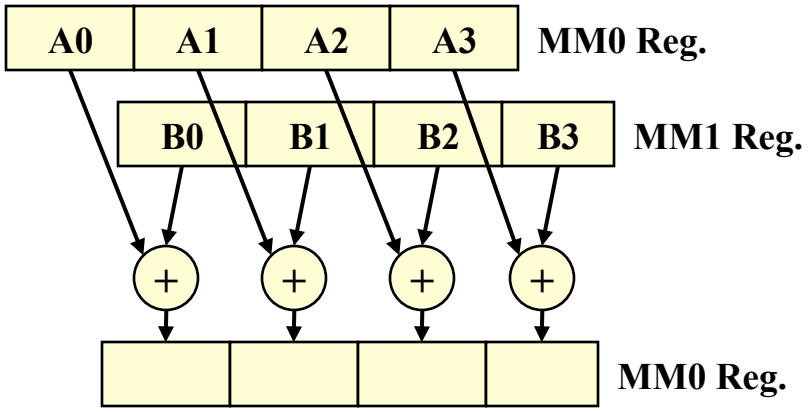
- Problem
  - Traditional, human programming paradigm has been a single thread of execution (easy for programmer)
  - Now HW is able to do MANY things in parallel (at the same time) with multicore and other architectures
- Solution = Extract Parallelism
  - Implicitly: Let hardware or compiler extract parallelism
    - Find instructions in the original sequential thread that can be executed at the same time
  - Explicitly: Change the programming paradigm and make programmer define parallel tasks (slowly moving toward this)

Sequential Programming Paradigm

- Problem
  - Other data access patterns may not match traditional threaded (fetch/decode /execute) architectures
    - Multimedia data where we do the same operation on huge sets of data
- Solution = Data Parallelism
  - Streaming / Data parallel / SIMD operations (MMX /SSE)
  - Dedicated GPU's

```
for(i=0; i < 1000; i++){
   C[i] = A[i] + B[i];
}
```

Threaded architecture performs one operation per loop iteration

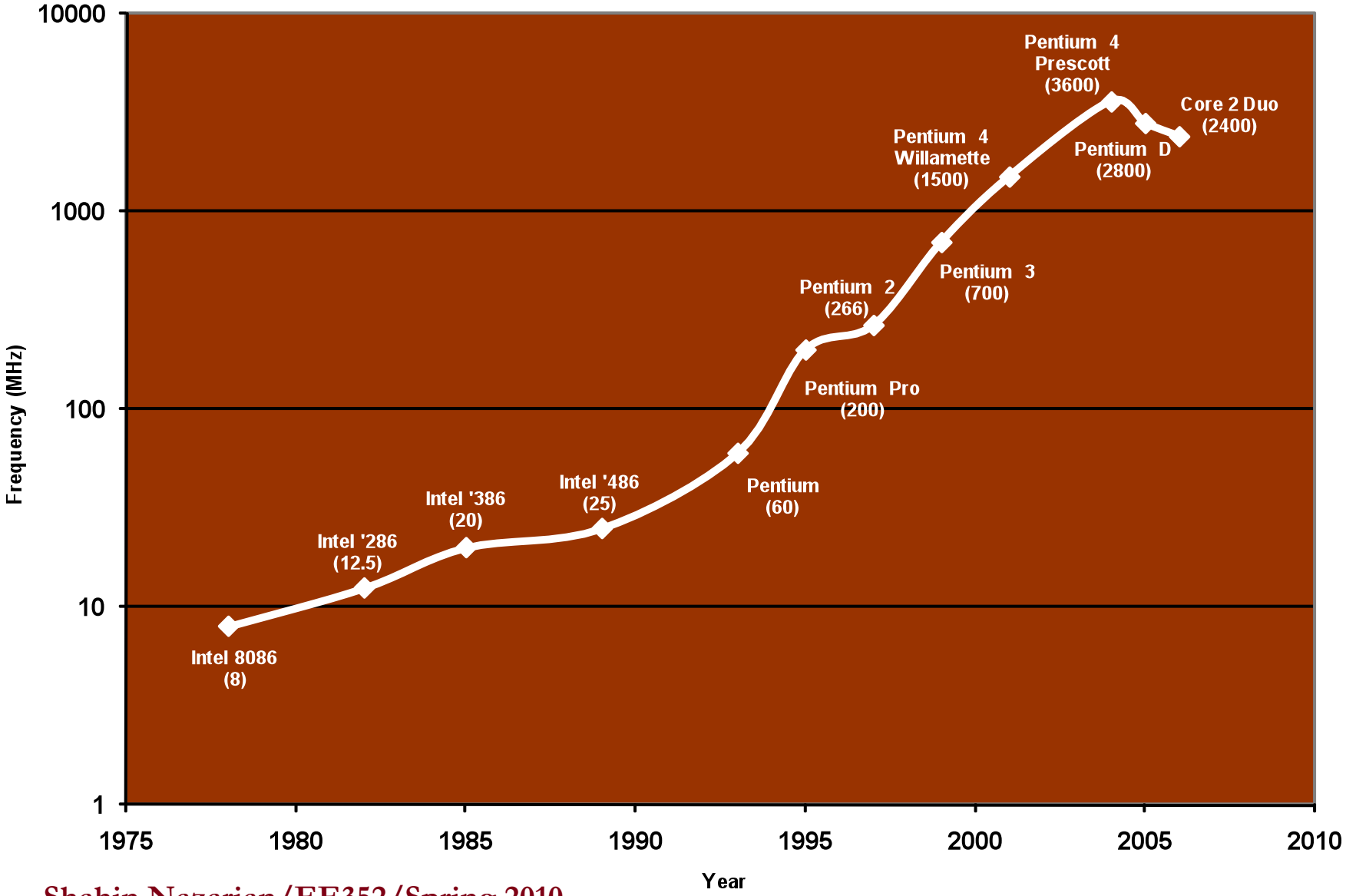


SIMD implementation of: A[i] + B[i]
PADD MM0,MM1

Power Consumption

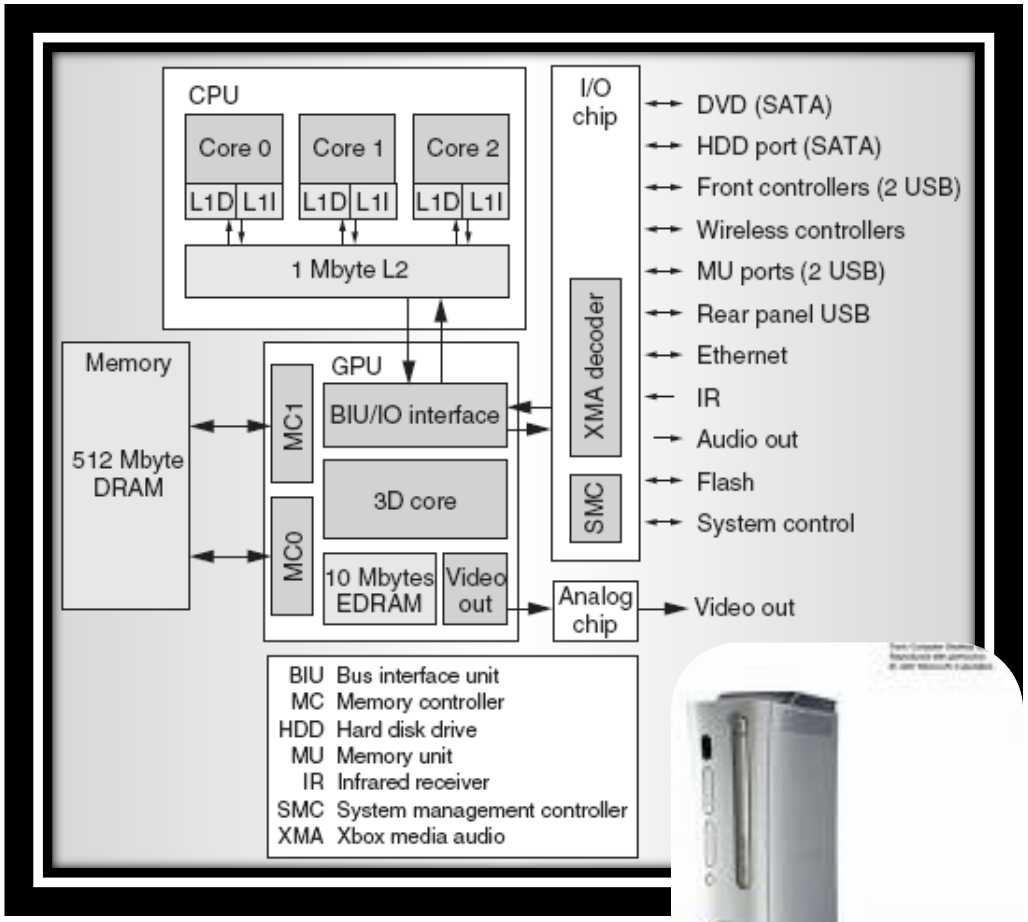
- Problem
  - Inability to dissipate heat that results from power consumption can destroy a chip
  - Dynamic power equation: $P = \frac{1}{2} CV^2f$
    - C = Capacitance = function of transistor size
    - V = Supply voltage = Higher voltage needed for faster circuit operation
    - f = Frequency = speed of operation of the processor
- Solution
  - Put unused parts of the processor to "sleep"
  - Reduction in frequency also allows reduction in supply voltage (cubic reduction in power…)
    - Implies more simple cores rather than fewer complex cores

Increase in Clock Frequency



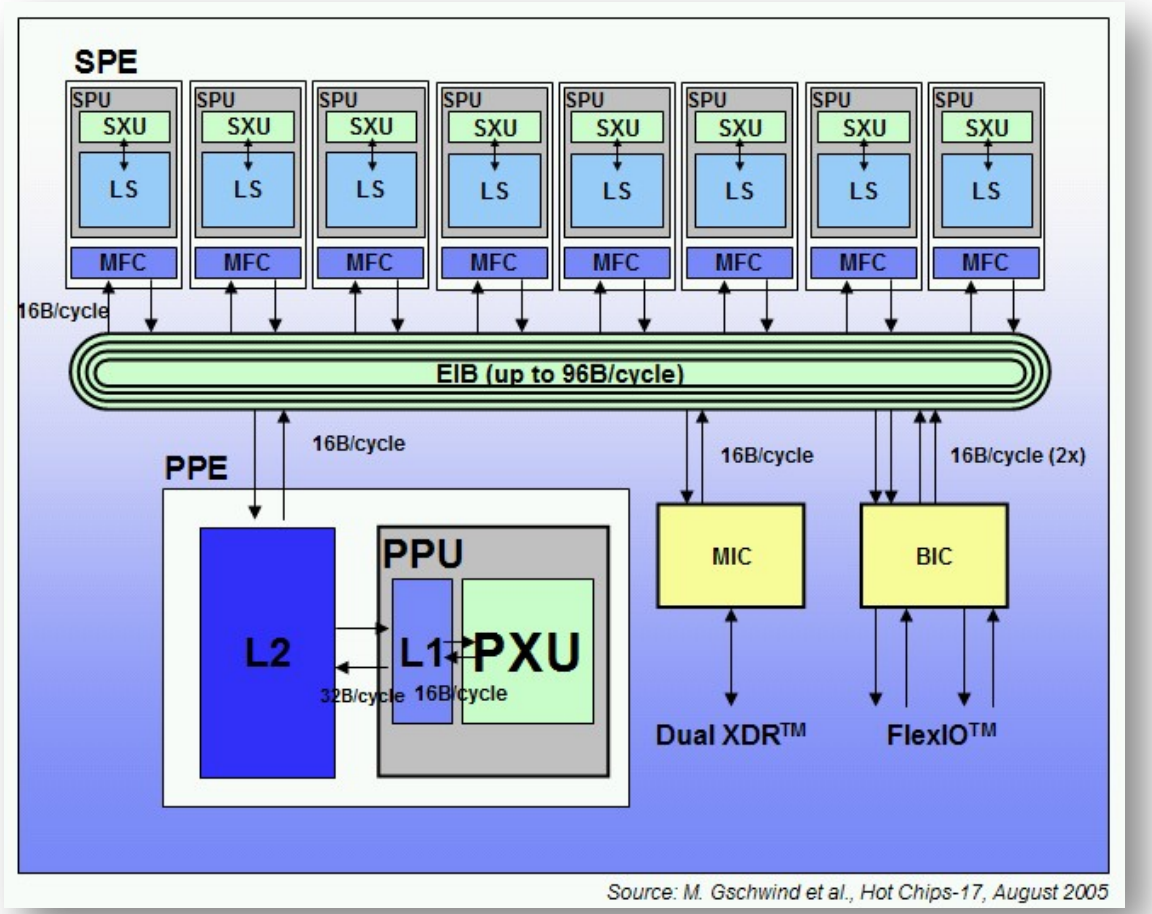
Example – Xbox 360

- 3 PowerPC (PPC) cores
- Dedicated GPU and memory controller
- I/O chip



Example – PS3 Architecture

- Cell processor
  - 1 high-perf. complex core
  - 8 simple cores (SPE's)
- GPU and I/O chip



Source: M. Gschwind et al., Hot Chips-17, August 2005

Concepts & Skills

- Concepts
  - Understand of abstraction levels
  - Note that Architecture affects performance
- Skills
  - Be familiar with the terms and topics described here
  - Enjoy doing assembly programming and relating HW to SW using ISA!