Computer Architecture

Introduction

Amir Mahdi Hosseini Monazzah

Room 332,
School of Computer Engineering,
Iran University of Science and Technology,
Tehran, Iran.

monazzah@iust.ac.ir

Spring 2025

*Parts of slides are adopted from prof. David Brooks lectures, Department of Electrical Engineering and Computer Science, Harvard University

Outline

- The legal stuff
  - Introducing class elements
  - Class policies
  - Course outline
- Introduction
  - Why take Computer Architecture (CA)?
  - What you will learn in CA?
  - What is computer architecture?
- Performance: The main goal!
  - How to measure performance?
  - Standard metrics
  - Benchmarks
  - Conclusion
  - Case studies
  - In-class assignment

Who we are?

- Introductions
  - Lecturer: Amir Mahdi Hosseini Monazzah
  - Email: monazzah@iust.ac.ir

  - TAs: Ali Asnaashari (ali.asnaashari2000@gmail.com)
    Mohammad javad Jalilvand (jalilvand.mj@gmail.com)
    Sina sedghi motlagh (sinasedghimotlagh@gmail.com)
    Erfan shabgir (erfanshabgir11@gmail.com)
    Sepideh Yousefi (sepideh.yousefi2016@gmail.com)
    Saleh shirvani (shirvani.saleh79@gmail.com)
    MohammadAli Gholami (gholamiali41380@gmail.com)

Who we are?

- Introductions
  - Lecturer: Amir Mahdi Hosseini Monazzah
  - Email: monazzah@iust.ac.ir

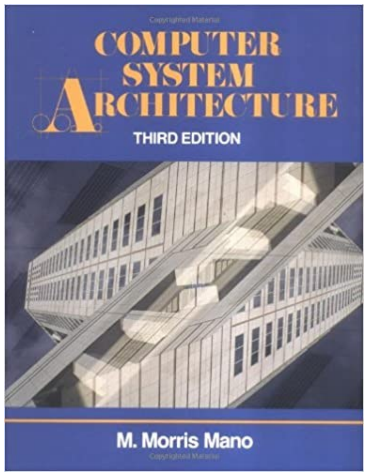
  - TAs: Mahdi Ghazavi (realmahdighazavi@gmail.com)
    Kian Yari (kianyarii1383@gmail.com)
    Ali Alizadeh (alializadeh.dev11@gmail.com)
    Saeed nourian (saeednourian82@gmail.com)
    Mehrshad Fallah (fallahmehrshad768@gmail.com)
    Roham Izadidoost (izadidoostroham@gmail.com)

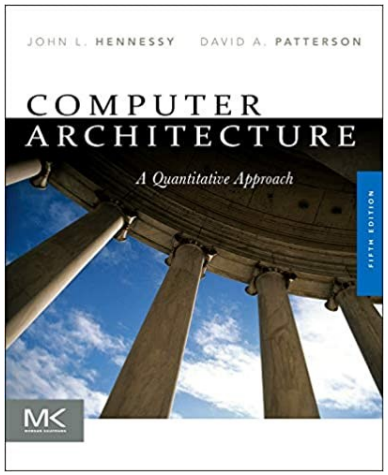
Who we are?

- Class
  - Time: 10:30 – 12:00 on Sunday & Tuesday
  - Room: -121

What do we read?

- Textbook
  - Morris Mano, Computer System Architecture, Prentice-Hall, 1993
  - John L. Hennessy and David A. Patterson, Computer Architecture: A Quantitative Approach, Fifth Edition, Morgan Kauffmann, 2012.
- Prerequisites
  - Digital Logic





What do I teach?

- Slides
  - Adopted (with modifications) from several resources includes
    - Computer Architecture by prof. David Brooks lectures, Department of Electrical Engineering and Computer Science, Harvard University, Fall 2019
    - Computer System Architecture by guiseok@yahoo.com (no further information!)
    - Profs. Onur Mutlu and Seth Copen Goldstein, Carnegie Mellon University, Fall 2013.
    - Arvind and Krste Asanovic, Computer Science & Artificiel Intelligence Lab, M.I.T, 2005.
    - And any other interesting resources (will be mentioned!)

What do I teach?

درس اصلی۱۱: معماری کامپیوتر

نام درس	معماری کامپیوتر
نام درس به انگلیسی	Computer Architecture
نوع واحد	اصلی | مهندسی کامپیوتر | ۳ واحد
مقطع	کارشناسی
هم‌نیازها	-
پیش‌نیازها	مدارهای منطقی
مطالب پیش‌نیاز	آشنایی با اصول طراحی مدارهای منطقی، مبانی کامپیوتر و برنامه‌سازی
کتاب(های) مرجع	[۱] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*. ۴th Edition, Morgan Kaufmann Publishers Inc., ۲۰۱۰.
اهداف درس	هدف از این درس، آشنایی دانشجویان رشته‌ی مهندسی کامپیوتر با معماری و سازمان‌دهی پردازنده‌ها است که شامل آشنایی با معماری دستورالعمل و نیز ساختار داخلی پردازنده می‌شود. در ضمن دانشجویان با محاسبات کامپیوتری مورد استفاده در پردازنده‌های عام منظوره نیز آشنا می‌شوند که شامل نمایش اعداد و عملیات اصلی شامل جمع، تفریق، ضرب و تقسیم در سیستم‌های عددی مختلف می‌شود. در ادامه سلسله مراتب حافظه در سیستم‌های پردازشی مورد بحث قرار می‌گیرد. از آنجاییکه مدل‌سازی و آزمایش معماری‌های مختلف که از اهداف درس می‌باشد با بکارگیری زبان‌های توصیف سخت‌افزار میسر می‌گردد استفاده از زبان وریلاگ و یاداوری مفاهیم پایه ان در حین تدریس کلاس پیشنهاد می‌شود.
نتایج درس	دانشجویانی که این درس را با موفقیت پشت سر بگذارند بینش مناسبی در موارد زیر خواهند داشت: ۱- معماری‌های مختلف کم‌دستور/پردستور ۲- تحلیل کارایی پردازنده‌ها ۳- طراحی و پیاده‌سازی پردازنده‌ها ۴- الگوریتم‌های حساب کامپیوتری در پردازنده‌ها ۵- طراحی اجزای جانبی و ارتباط آنها با پردازنده ۶- اشنایی با زبان وریلاگ و شبیه‌سازی ساختارهای پایه‌ای معماری کامپیوتر با آن
فهرست مباحث	۱- مقدمات - تاریخچه‌ی کامپیوتر و پردازنده‌ها - کاربرد پردازنده‌های در دنیای کنونی - دسته‌بندی انواع پردازنده ۲- معرفی مفاهیم پایه - معرفی اجزای اصلی یک پردازنده - معرفی مفهوم مجموعه دستورالعمل - مفاهیم معماری کامپیوتر و سازمان کامپیوتر - سیستم‌های عددی و عملیات پایه ۳- معرفی زبان مدل‌سازی سخت‌افزاری وریلاگ - معرفی زبان و ساختارهای پایه مدل‌سازی در آن - مثالهای مدلسازی اجزای یک پردازنده شامل بخش‌های ترکیبی و ترتیبی و خط لوله ۴- طراحی پردازنده - طراحی مسیر داده

	- طراحی واحد کنترل ○ کنترل سیم‌پندی شده ○ کنترل به صورت ریزبرنامه‌ریزی - مقایسه و تحلیل معماری RISC و CISC - ارزیابی کارایی پردازنده‌های کامپیوتری ۵- معرفی مکانیسم خط لوله - مسیر داده خط لوله - مسیر کنترل خط لوله - معرفی مخاطرات خط لوله و روش‌های حل یا کاهش این مخاطرات - ارزیابی کارایی پردازنده‌های دارای خط لوله‌ی ۶- سلسله مراتب حافظه - تحلیل علل نیاز به وجود سلسله مراتب حافظه - حافظه‌ی نهان ۷- حساب کامپیوتری - الگوریتم‌های حسابی صحیح برای عملیات جمع، تفریق، ضرب و تقسیم برای اعداد - الگوریتم‌های ممیز شناور برای عملیات جمع، تفریق، ضرب و تقسیم برای اعداد ۸- تجهیزات جانبی پردازنده - روش‌های ارتباطی پردازنده با ادوات جانبی - ارتباط برنامه‌ریزی شده (Programmed I/O) - ارتباط با استفاده از وقفه (Interrupted I/O) - دسترسی مستقیم به حافظه (DMA) - انواع گذرگاه‌ها ۹- معرفی پردازنده‌های چندهسته‌ای
نرم‌افزارهای مورد نیاز	برای بهبود توانایی مدل‌سازی و آزمایش معماری‌های مختلف بکارگیری زبان‌های توصیف سخت‌افزار VHDL یا Verilog با استفاده از نرم‌افزارهای Modelsim ISE یا Quartus پیشنهاد می‌شود.
تکالیف پیشنهادی	تعداد ۴ تکلیف حاوی مطالب تئوری بیان شده در درس
پروژه‌های پیشنهادی	تعداد ۴ تکلیف کامپیوتری در طراحی بخش‌های مختلف پردازنده
نمره‌دهی پیشنهادی	تکالیف دستی ٪۱۰ پروژه‌ها ٪۲۰ امتحان میان‌ترم ٪۳۰ امتحان پایان‌ترم ٪۴۰
سایر مراجع	[۱] M. Mano, Computer System Architecture, Prentice Hall, ۳rd Edition, ۱۹۹۳. [۲] J. P. Hayes, *Computer Architecture and Organization*. McGraw-Hill, ۱۹۸۸.

How do we evaluate ourselves?

- Exams to be done on the date given
- Class participation and attendance
  - Attendance is expected but not required
  - Participation will be encouraged
  - +10% of overall grade indirectly related to class participation
- Feedback
  - Honest open feedback is expected!

How do we evaluate ourselves?

- Grading: Projects, Assignments, Exams
  - Homework 20%
  - Attendance Up to 10% (in-class assignments)
  - Midterm 25%
  - Final 35%
  - Project 20%

How do we evaluate ourselves?

- Homework:
  - Theoretical assignments
  - Practical assignments
    - Snipersim, gem5, Quartus, Modelsim, etc.

Topics and Lectures Based on Textbook

1.	Introduction (Chap. 1)	2 Sessions
2.	Fundamentals of computer architecture (Chap. 2)	3 Sessions
3.	Hardware description language (Chap. 3)	4 Sessions
4.	Processor design (Chap. 4)	5 Sessions
5.	Pipelining (Chap. 5)	4 Sessions
6.	Memory hierarchy (Chap. 6)	4 Sessions
7.	Computer Arithmetic Unit Design (Chap. 7)	3 Sessions
8.	Multiprocessor Design (Chap. 8)	2 Sessions

Overall, we need (at least) 27 sessions!

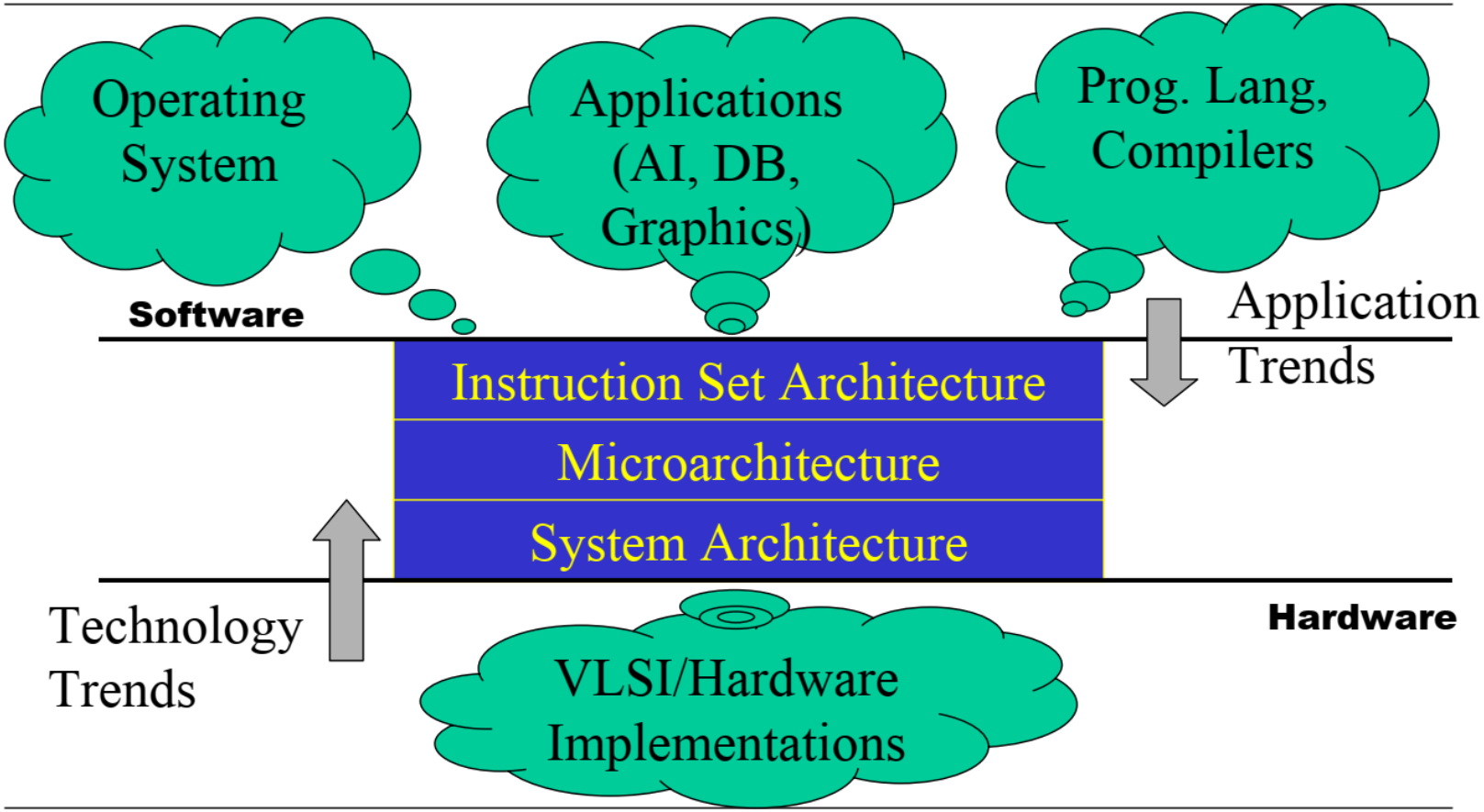
Why we need to take CA?

- How does computing hardware work?
- Where is computing hardware going in the future?
  - And learn how to contribute to this future…
- How does this impact system software and applications?
  - Essential to understand OS/compilers/PL
  - For everyone else, it can help you write better code!
- How are future technologies going to impact computing systems?

Topics of study

- Focus on what computer architects worry about (both academia and industry)
- Get through the basics of processor design
- Understand the interfaces between architecture and system software (compilers, OS)
- Understand the interfaces between architecture and I/O (disks, memory, multiprocessors)
- Look at technology trends, recent research ideas, and the future of computing hardware

Finding the suitable place to define it!



Application areas

- General-purpose laptop/desktop
  - Productivity, interactive graphics, video, audio
  - Optimize price-performance
  - Examples: Intel Core i9, AMD Athlon XP
- Embedded computers
  - PDAs, cell-phones, sensors => Price, energy efficiency
  - Examples: Intel XScale, StrongARM (SA-110), ESP32
  - Game machines, network uPs => Price-performance
  - Examples: Sony emotion engine, IBM 750FX

Application areas (Up to here session 1)

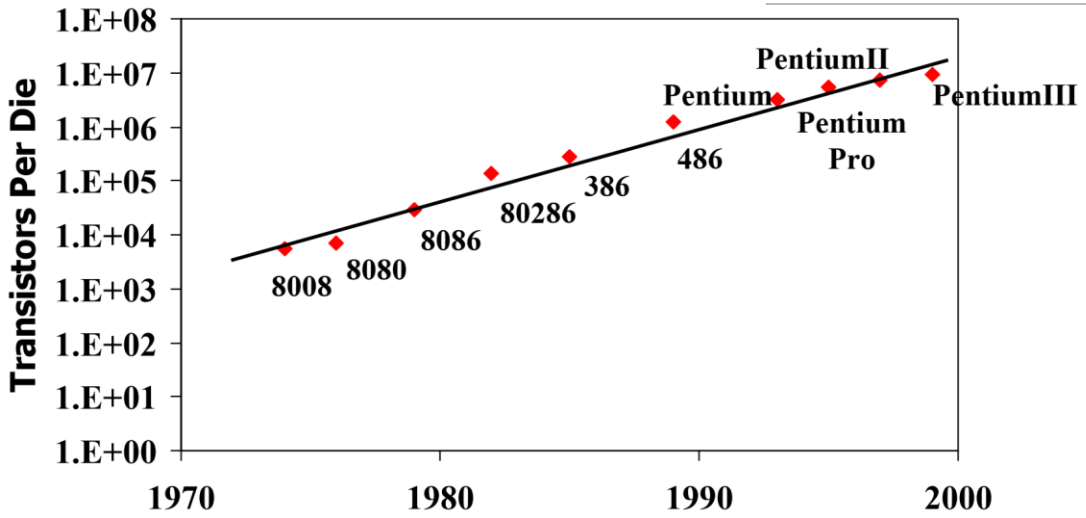
- Commercial servers
  - Database, transaction processing, search engines
  - Performance, availability, scalability
  - Server downtime could cost a brokerage company more than $6M/hour
  - Examples: Sun Fire 15K, IBM p690, Google cluster
- Scientific applications
  - Protein folding, weather modeling, CompBio, Defense
  - Floating-point arithmetic, huge memories
  - Examples: IBM DeepBlue, IBM Blue Gene, Cray T3E, etc.

Moore's law (From here session 2)

- Every 18-24 months
  - Feature sizes shrink by 0.7x
  - Number of transistors per die increases by 2x
  - Speed of transistors increases by 1.4x
- But we are starting to hit some roadblocks…
- Also, what to do with all of these transistors???

Moore’s law (density)

Year	Node	Transistor Density (per mm²)	Key Innovations
1971	10µm	~2,300 (Intel 4004)	Planar CMOS
2000s	130nm	~50 million	Strained silicon, FinFETs
2020	7nm	~100 million	EUV lithography
2025	2nm	~200 million (estimated)	GAA nanosheets, NanoFlex
2030+	1nm/3D	~500 million–1 billion	3D stacking, CFETs, quantum



How have we used these transistors?

- More functionality on one chip
  - Early 1980s – 32-bit microprocessors
  - Late 1980s – On Chip Level 1 Caches
  - Early/Mid 1990s – 64-bit microprocessors, superscalar (ILP)
  - Late 1990s – On Chip Level 2 Caches
  - Early 2000s – Chip Multiprocessors, On Chip Level 3 Caches
- What is next?
  - How much more cache can we put on a chip?
  - How many more cores can we put on a chip?
  - What else can we put on chips?

Metrics: Performance

- Reduce response/execution time
  - Time between start and completion of an event
- Increase throughput
  - Total amount of work in a given amount of time
- Execution time is reciprocal of performance
- "X is N times faster than Y"
- $N = \frac{Execution\ Time_Y}{Execution\ Time_X}$
  - Wall-clock time, CPU time (no I/O)

Performance metrics

- Execution Time is often what we target
- Throughput (tasks/sec) vs. latency (sec/task)
- How do we decide the tasks? Benchmarks
  - What the customer cares about, real applications
  - Representative programs (SPEC, SYSMARK, etc)
  - Kernels: Code fragments from real programs (Linpack)
  - Toy Programs: Sieve, Quicksort
  - Synthetic Programs: Just a representative instruction mix (Whetsone, Dhrystone)

Measuring performance

- Total Execution Time

$$\frac{1}{n}\sum_{i=1}^{n} Time_i$$

- $Time_i$ is the execution time of the i[th] program of n in workload
- This is *arithmetic* mean
  - This should be used when measuring performance in execution times (CPI)
  - The average execution time of several programs can be measured using the arithmetic mean

Measuring performance

- Weighted Execution Time

$$\sum_{i=1}^{n} Weight_i \times Time_i$$

  - Use for situations in which the programs in the workload are NOT run an equal number of times.
  - Simply assign a weight, $weight_i$, to each program to capture its relative frequency in the workload.

Amdahl's law (law of diminishing returns)

- Very intuitive – make the common case fast

$$Speedup = \frac{Execution\ Time\ for\ task\ without\ enhancement}{Execution\ Time\ for\ task\ using\ enhancement}$$

$$Execution\ time_{new} = Execution\ time_{old} \times ((1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})$$

Amdahl's law corollary

$$Speedup_{Overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

$$As\ Speedup_{enhanced} \gg 0\ Then,$$

$$Speedup_{Overall} = \frac{1}{(1 - Fraction_{enhanced})}$$

Amdahl's law example

- $Fraction_{enhanced} = 95\%, Speedup_{enhanced} = 1.1x$
- $Speedup_{Overal} = \frac{1}{((1-0.95)+\left(\frac{0.95}{1.1}\right))} = 1.094$
- $Fraction_{enhanced} = 5\%, Speedup_{enhanced} = 10x$
- $Speedup_{Overal} = \frac{1}{((1-0.05)+\left(\frac{0.05}{10}\right))} = 1.047$
- $Fraction_{enhanced} = 5\%, Speedup_{enhanced} = \text{Infinity}$
- $Speedup_{Overal} = \frac{1}{(1-0.05)} = 1.052$

Make the common case fast!

MIPS

- MIPS = instruction count/(execution time x 10^6)
  = clock rate/(CPI x 10^6)

- Problems
  - ISAs are not equivalent, e.g. RISC vs. CISC
    - 1 CISC instruction may equal many RISC!
  - Programs use different instruction mixes
  - May be ok when comparing same benchmarks, same ISA, same compiler, same OS

MFLOPS

- Same as MIPS, just FP ops
- Not useful either
  - FP-intensive apps needed
  - Traditionally, FP ops were slow, INT can be ignored
  - BUT, now memory ops can be the slowest!

GHz

- One number, no benchmarks, what can be better?
- The data underscores two key principles in processor design:
  - Clock speed is not synonymous with performance (e.g., Itanium-2 vs. Pentium 4).
  - Architectural innovations (e.g., parallelism, instruction sets, specialized units) are critical for maximizing efficiency, especially in floating-point tasks.
- Maybe as good as the others…

Processor	Clock Rate	SPEC FP2000
IBM POWER3	450 MHz	434
Intel PIII	1.4 GHz	456
Intel Pentium 4	2.4 GHz	833
Itanium-2	1.0 GHz	1356

Benchmark suites

- SPEC CPU (int and float) (desktop, server)
- EEMBC (“embassy”), SPECjvm, MiBench (embedded)
- TPC-C, TPC-H, SPECjbb, ECperf (server)
- PARSEC (Multithreaded program)

SPEC CPU2000: Integer benchmarks

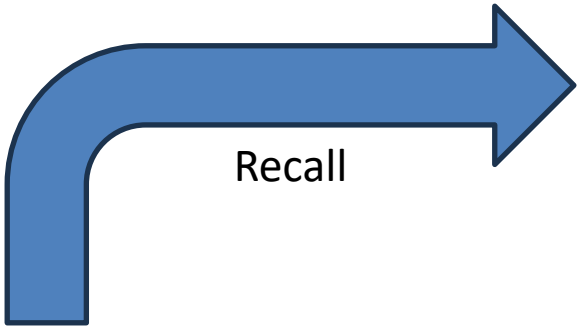
164.gzip	C	Compression
175.vpr	C	FPGA Circuit Placement and Routing
176.gcc	C	C Programming Language Compiler
181.mcf	C	Combinatorial Optimization
186.crafty	C	Game Playing: Chess
197.parser	C	Word Processing
252.eon	C++	Computer Visualization
253.perlbmk	C	PERL Programming Language
254.gap	C	Group Theory, Interpreter
255.vortex	C	Object-oriented Database
256.bzip2	C	Compression
300.twolf	C	Place and Route Simulator

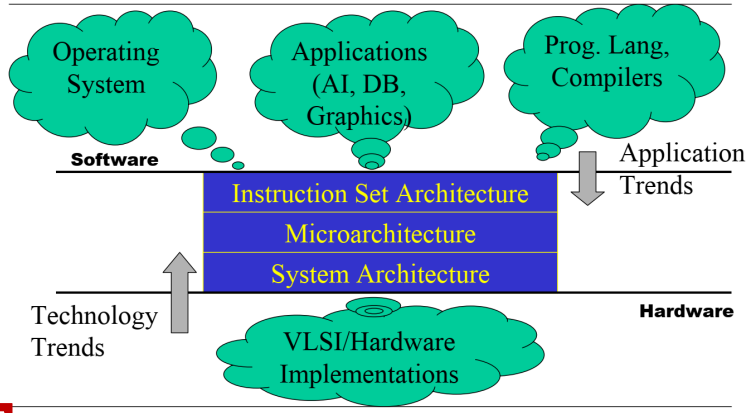
SPEC CPU2000: Floating point benchmarks

168.wupwise	Fortran 77	Physics / Quantum Chromodynamics
171.swim	Fortran 77	Shallow Water Modeling
172.mgrid	Fortran 77	Multi-grid Solver: 3D Potential Field
173.applu	Fortran 77	Parabolic / Elliptic Partial Differential Equations
177.mesa	C	3-D Graphics Library
178.galgel	Fortran 90	Computational Fluid Dynamics
179.art	C	Image Recognition / Neural Networks
183.equake	C	Seismic Wave Propagation Simulation
187.facerec	Fortran 90	Image Processing: Face Recognition
188.ammp	C	Computational Chemistry
189.lucas	Fortran 90	Number Theory / Primality Testing
191.fma3d	Fortran 90	Finite-element Crash Simulation
200.sixtrack	Fortran 77	High Energy Nuclear Physics Accelerator Design
301.apsi	Fortran 77	Meteorology: Pollutant Distribution

CPU performance equation

- Execution Time = seconds/program





$$\frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

Program	Compiler (Scheduling)	Technology
Architecture (ISA)	Organization (uArch)	Physical Design
Compiler	Microarchitects	Circuit Designers

Common architecture trick (Up to here session 3)

- Instructions/program (Path-length) is constant
  - Same benchmark, same compiler
  - Ok usually, but for some ideas compiler may change
- Seconds/Cycle (Cycle-time) is constant
  - “My tweak won’t impact cycle-time”
  - Often a bad assumption
  - Current designs are ~15-20FO4 Inverter Delays per cycle
- Just focus on Cycles/Instruction (CPI or IPC)
  - Most academic architecture studies do just this!

Measuring instruction counts and CPI (Up to here session 2)

- Hardware counters on processors
- Instrumented execution (ATOM)
- Instruction-level interpretation (instruction counts)
- Execution-driven simulation
  - Detailed simulation of execution core and memory hierarchy

History (Student self-read: From here up to end of this chapter!)

- Gem5 is the fusion of two projects
  - GEMS
    - Detailed and flexible memory system model
    - Includes support for multiple cache coherence protocols and interconnect models
    - Developed @ The University of Wisconsin Madison
  - M5
    - Highly configurable simulation framework to support multiple ISAs, and diverse CPU models
    - Developed @ The University of Michigan

Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

What is gem5?

- System simulator
  - Good support of complex components interactions (OS / CPU / Caches / Devices / …)
    - Accuracy depends on the model completeness
  - Lot of components available out-of-the-box (CPUs, memories, I/Os, …)

Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

What is gem5 useful for?

- Architectural exploration
  - gem5 provides a fast and easy framework to interconnect hardware components and evaluate them!
- Hardware/software performance evaluation?
  - gem5 has a good support of various ISA and allows for realistic HW/SW performance evaluation

Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

What is gem5 not useful for?

- Hardware/software verification?
  - RTL functional verification is much more accurate!
- Software development and verification?
  - Faster technologies are available through binary-translation (e.g. QEMU, OVP)

Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

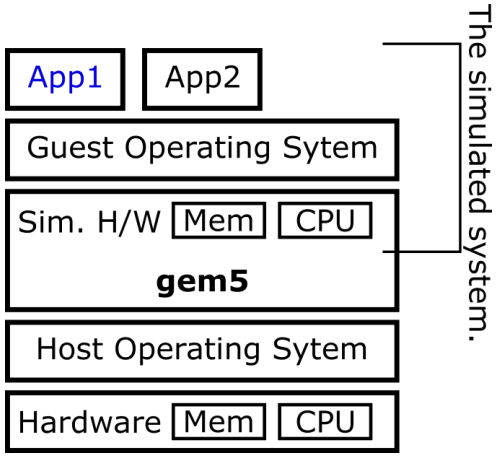
Simulation modes

- Full-system (FS)
  - Models bare-metal hardware
  - Includes the various specified devices, caches, …
  - Boots an entire OS from scratch
  - Gem5 can boot Linux (several variants) or Android out of-the-box

Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

Simulation modes

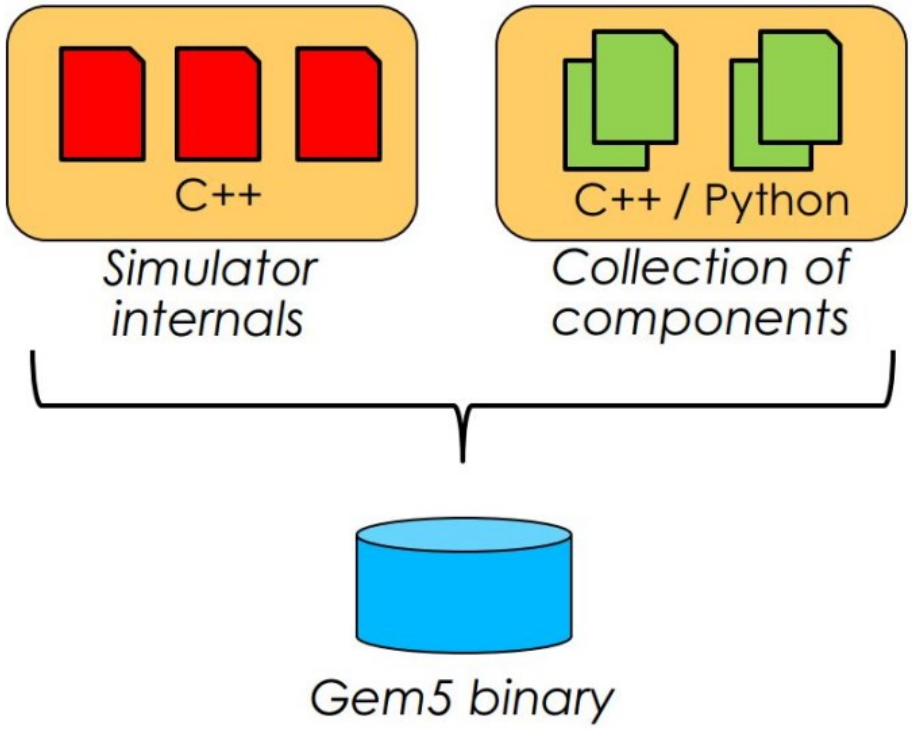
- Syscall Emulation (SE)
  - Runs a single static application
  - System calls are emulated or forwarded to the host OS
  - Lot of simplifications (address translation, scheduling, no pthread …)



Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

Behind the scene of a simulation!

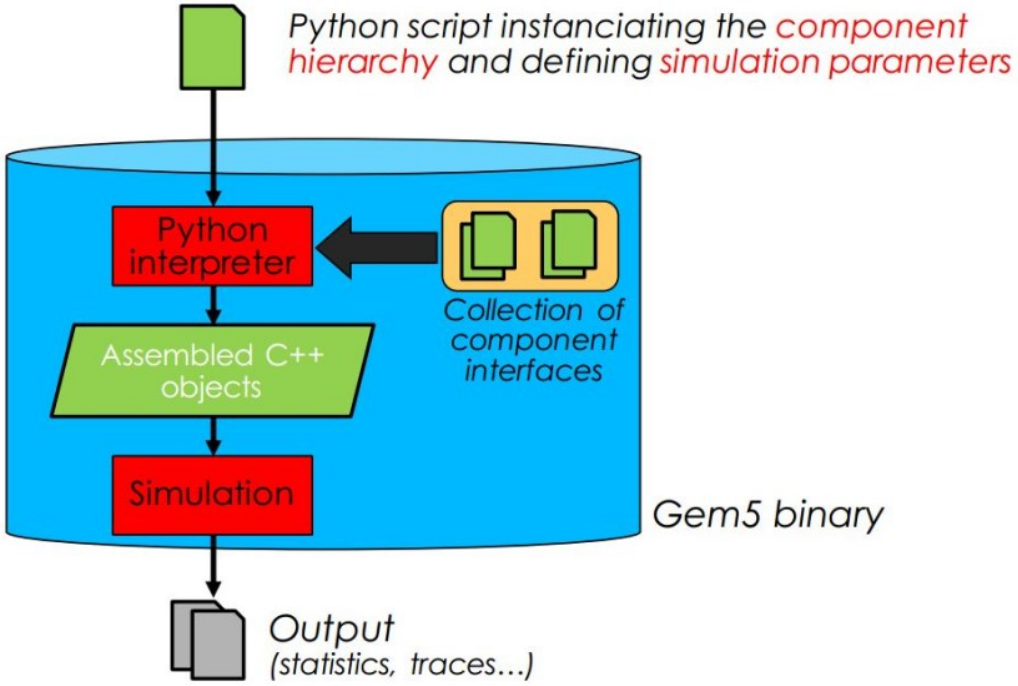
- Compilation of the simulator



Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

Behind the scene of a simulation!

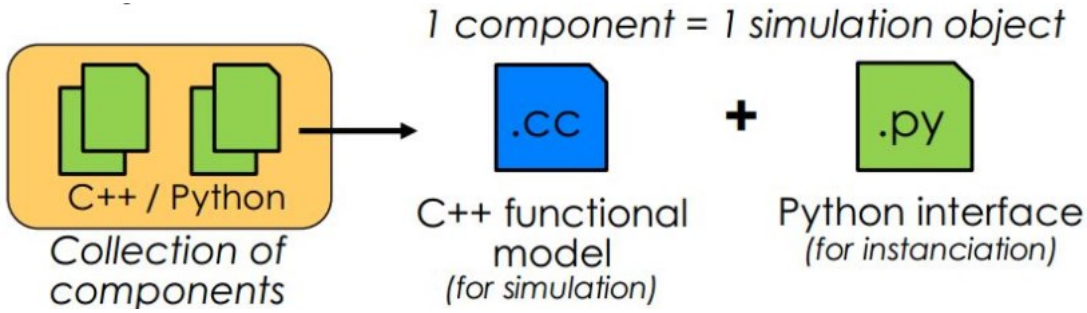
- Compilation of the simulator



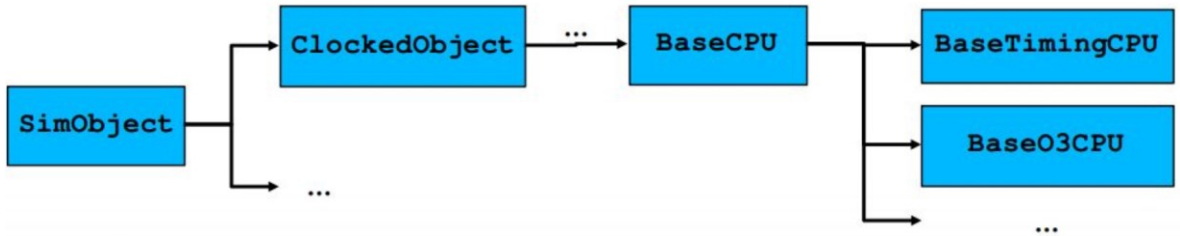
Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

What's under the hood ?

- Simulation objects



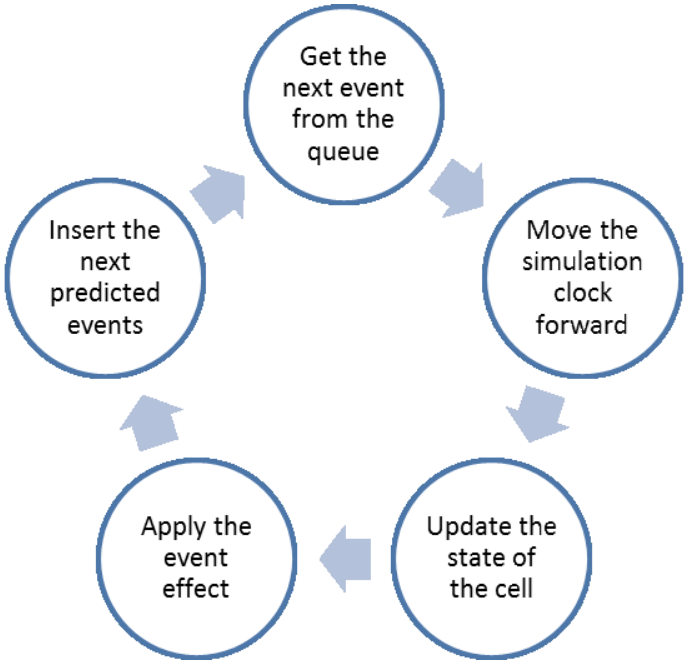
- SimObjects follow a strict C++ class hierarchy for easier extension with code reuse



Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

What's under the hood ?

- Events
  - gem5 is event-driven
    - Discrete event timing model



  - Simulation objects schedule events for the next cycle after a specific time elapsed

Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

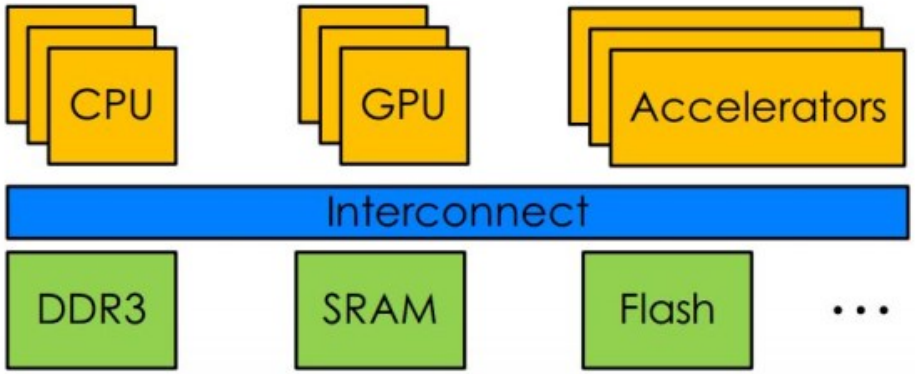
What's under the hood ?

- CPU Models
  - Supports: Alpha, ARM, MIPS, Power, SPARC, and x86
  - Configurable CPU models : Supports 3 CPU models
    - Simple Atomic/Timing
      - Fast CPU model
    - InOrder
      - Detailed pipelined in-order CPU model
    - O3
      - Detailed pipelined out-of-order CPU model
  - Supports a domain specific language to represent ISA details

Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

What's under the hood ?

- Memory system
  - Models a system running heterogeneous applications
    - Running on heterogeneous processing tiles
    - Using heterogeneous memories and interconnect



Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

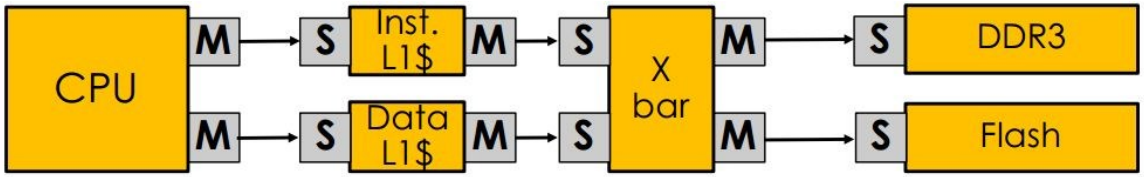
What's under the hood ?

- Memory system
  - Two memory systems in Gem5
    - Classic
      - All components instantiated in a hierarchy along with CPUs, etc.
      - Only MOESI coherence protocol
      - Fast, but less detailed than Ruby
    - Ruby
      - Detailed simulation model of various cache hierarchies
      - Various cache coherence protocols (MESI, MOESI, …)
      - Interconnection networks

Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

What's under the hood ?

- Memory ports
  - Memory ports are present on every MemObject
    - They model physical memory connections
    - You interconnect them during the hierarchy instantiation
      - E.g. a CPU data bus to a L1 cache
      - 1 master port always connects to 1 slave port
  - Data is exchanged atomicly as packets



Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

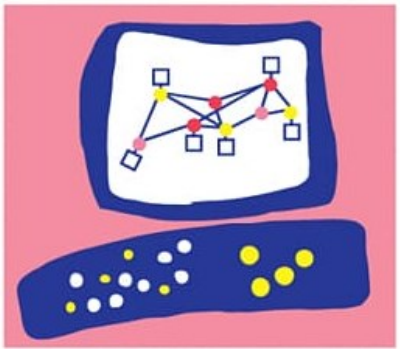
What's under the hood ?

- Memory ports
  - 3 types of transport interfaces for packets
    - Functional
      - Instantaneous in a single function call
      - Caches and memories are updated at once
    - Atomic
      - Instantaneous
      - Approximate latency, but no contention nor delay
    - Timing
      - Transaction split into multiple phases
      - Models all timing in the memory system

Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.

Nice features

- Checkpointing
  - Snapshot the relevant system state and restore it later



- Fast-forward
  - Idea is to start the simulation in atomic mode and switch over to detailed mode for important simulation period

Parts of this slide are adapted from Mr. Mohammad Tahghighi's presentation from Parallel Architecture Research Eindhoven (PARsE) research group at Eindhoven University of Technology, Eindhoven, Netherlands.


The End