# CS 352: Computer Systems Architecture

## Lecture 1: What is Computer Architecture?

January 16, 2007

William R. Mark
Computer Sciences Department
University of Texas at Austin
billmark@cs.utexas.edu



## Questions we'll address in this course

- How do we separate software from hardware?
  - So that new computers can run old software
- How is computer hardware organized?
  - Processor, Memory, I/O, etc.
- How is the processor organized? Why?
- How do we measure computer performance?
- How do we think about concurrent programming?
  - Doing more than one thing at once

## Logistics

| | | | |
|---|---|---|---|
| Lectures | T/Th 9:30-11:00, RAS 213 | | |
| Instructor | Prof. William R. Mark | | |
| TA | Juhyun Lee | | |
| Grading | Final Exam | 1 | 35% |
| | Midterm Exam | 1 | 25% |
| | Homework | ~7 | 15% |
| | Project | 1 | 25% |
| Text | Hennessy & Patterson, *Computer Organization and Design* (Third Edition) | | |

UTCS Lecture 1 3

## CS352 Online

URL: www.cs.utexas.edu/~billmark/teach/cs352-07-spring

email list: **cs352-mark@cs.utexas.edu**
subscribe by sending email to TA
(mandatory – see web page for details)

Computer Architecture Seminar Series:
www.cs.utexas.edu/users/cart/arch

UTCS Lecture 1 4

## Why might this course be useful?



## Some reasons you might care

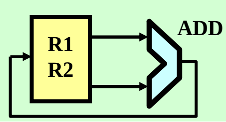
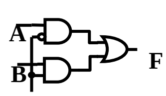
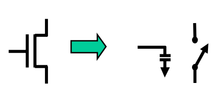
First, the obvious possibilities…

- You become a CPU architect
  - Unlikely for most of you!
- You design other computer hardware
  - The basic concepts and techniques are the same
- You design compilers, OS's, Java runtimes, etc.
  - These interact with computer hardware

## Less obvious (but more likely) reasons

- You write software
  - And care if it runs fast
  - And care if it is secure
- You design any kind of complex system
  - Design tradeoffs
  - System interfaces
  - Quantitative analysis of performance, cost, etc.
- You want to purchase a fast computer
  - And want to be an informed consumer
- You are curious about how computers work
  - Perhaps the best reason



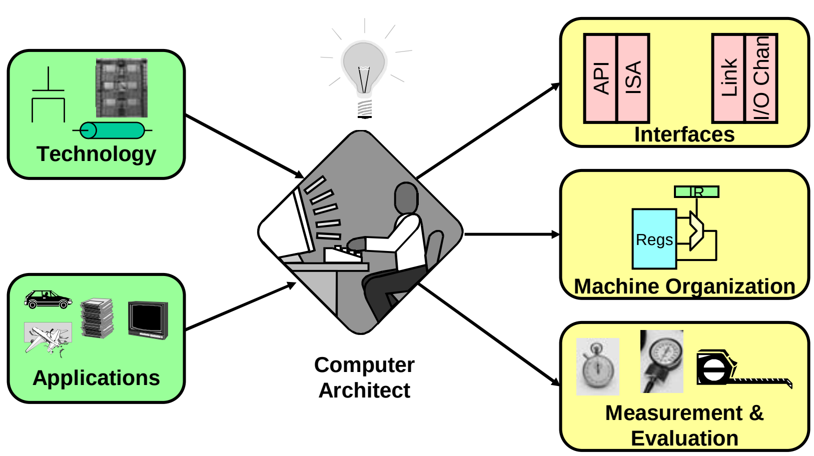
| | | |
|---|---|---|
| Specification | compute the fibonacci sequence | |
| Program | `for(i=2; i<100; i++) { a[i] = a[i-1]+a[i-2];}` | |
| ISA (Instruction Set Architecture) | `load r1, a[i];` `add r2, r2, r1;` | CS 310, CS 352 |
| microArchitecture |  | CS 352 |
| Logic |  | EE 316 |
| Transistors |  | |
| Physics/Chemistry | I = C dV/dt | |

# CS352 Topics

- Underlying technology trends
- Instruction set architectures
- Microarchitecture
  - Pipelining
  - Instruction level parallelism
- Cache memory systems
- Virtual memory
- I/O
- Multiprocessors and parallelism
- Security
- Computer system implementation



# What is Computer Architecture?



## How to design something:

- List goals
- List constraints
- Generate ideas for possible designs
- Evaluate the different designs
- Pick the best design
- Refine it

In reality, this process is iterative.
As constraints change, best design will change too.
[Use kitchen remodel as example of design process]



## Design goals for an architecture

## Design goals for an architecture

- High performance
  - Computation
  - Storage capacity
  - Communication speed
- Low cost
  - To manufacture, AND to design.
- Easy to program
- Compatibility and Longetivity
  - Run existing programs – fast today, faster tomorrow.
- Security and Reliability
- Low power consumption
  - For laptops, cell phones, etc.
  - Even for desktop CPUs!



## Possible design constraints for architecture

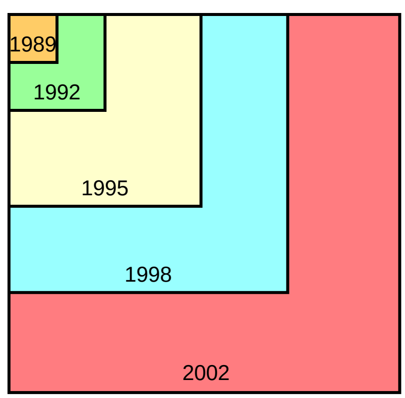
## Possible design constraints for architecture

- Maximum cost
- Maximum power consumption
- Backward compatibility
- Time to market
- Etc.

UTCS Lecture 1 15

## Technology Constraints

- Yearly improvement
  - Semiconductor technology
    - 60% more devices per chip (doubles every 18 months)
    - 15% faster devices (doubles every 5 years)
    - Slower wires
  - Magnetic Disks
    - 60% increase in density
  - Circuit boards
    - 5% increase in wire density
  - Cables
    - no change




100x more devices since 1989
8x faster devices

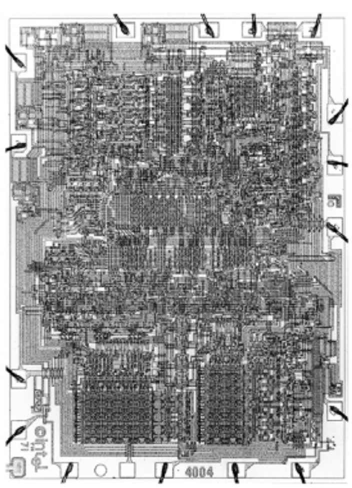
UTCS Lecture 1 16

## Changing Technology leads to Changing Architecture

- 1970s
  - multi-chip CPUs
  - semiconductor memory very expensive
  - microcoded control
  - complex instruction sets (good code density)
- 1980s
  - single-chip CPUs, on-chip RAM feasible
  - simple, hard-wired control
  - simple instruction sets
  - small on-chip caches
- 1990s
  - lots of transistors
  - complex control to exploit instruction-level parallelism
- 2000s
  - even more transistors
  - slow wires
  - multi-core chips

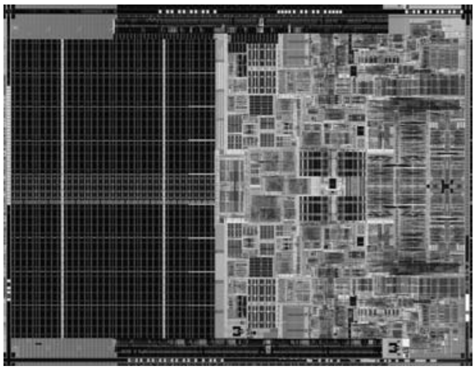


## Intel 4004 - 1971




- The first microprocessor
- 2,300 transistors
- 108 KHz
- 10μm process

## Intel Core 2 Duo - 2006

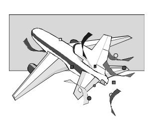


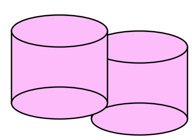
- "State of the art"
- 291 million transistors
- 3 GHz
- 0.065 $\mu$m (65 nm) process
- Could fit ~100,000 4004s on this chip!

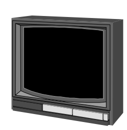


## Many kinds of systems and applications

- Personal:
  - Desktop, Laptop
  - Cell phone / PDA
  - Game machine
- Server:
  - Web servers
  - Transaction processing
- Engineering/Scientific:
  - Weather simulation
  - Drug design
- Embedded Control:
  - Anti-lock brake system
  - Microwave oven







## What is an "interface"

- *Interfaces* are visible, *Implementations* aren't
  - Same interface can have multiple implementations
  - We allow performance (time behavior) to change!
- Example interfaces:
  - Ethernet connector / protocol
  - X86 architecture
  - Java language
- Example NON-interfaces
  - Power connector for cell phone charger
- Good interfaces are simple



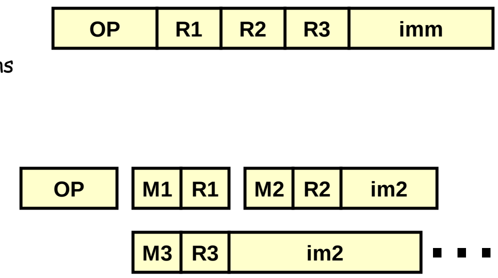
## Several kinds of interfaces

- Between system layers
  - Programming language
  - API
  - ISA
- Between modules
  - Network protocol (Ethernet)
  - I/O channel or bus (SCSI or PCI)
- Standard representations
  - ASCII
  - IEEE floating-point

# Instruction-Set Architecture

Hardware/Software Interface

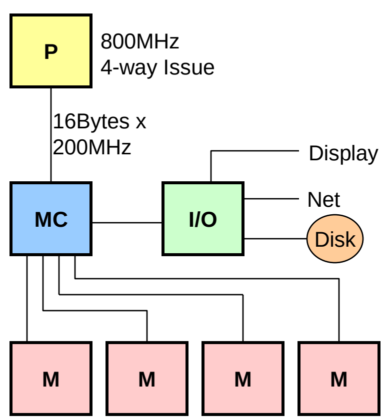
- Software impact
  - support OS functions
    - restartable instructions
    - memory relocation and protection
  - a good compiler target
    - simple
    - orthogonal
  - dense
- Hardware impact
  - admits efficient implementation
    - across generations
  - admits parallel implementation
    - no 'serial' bottlenecks
- Abstraction without interpretation





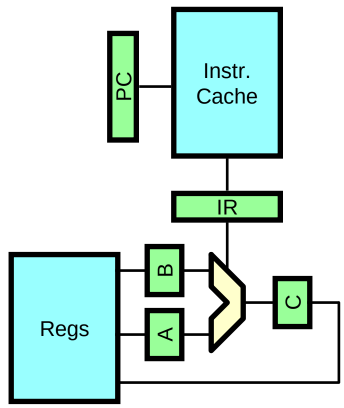

# System-Level Organization

- Design at the level of processors, memories, and interconnect.
- More important to application performance than CPU design
- Feeds and speeds
  - constrained by IC pin count, module pin count, and signaling rates
- System balance
  - for a particular application
- Driven by
  - performance/cost goals
  - available components (cost/perf)
  - technology constraints



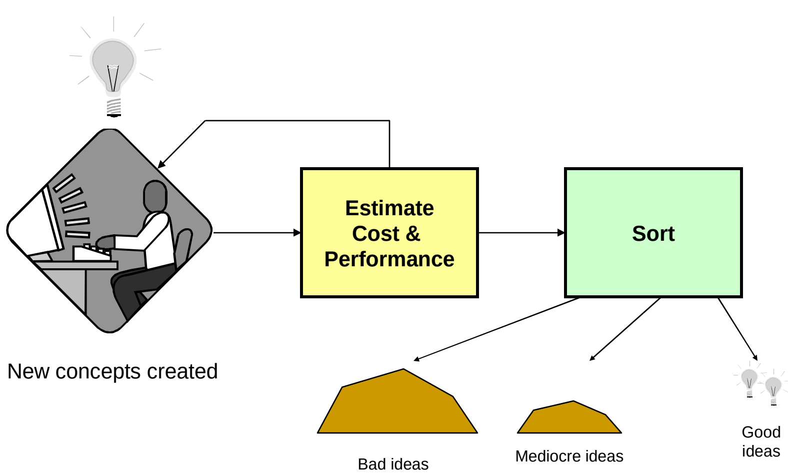
# Microarchitecture

- Register-transfer-level (RTL) design
- Implement instruction set
- Exploit capabilities of technology
  - locality and concurrency
- Iterative process
  - generate proposed architecture
  - estimate cost
  - measure performance
- Current emphasis is on overcoming sequential nature of programs
  - deep pipelining
  - multiple issue
  - dynamic scheduling
  - branch prediction/speculation





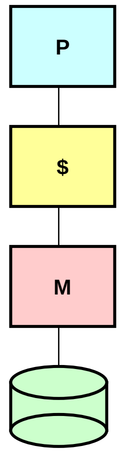

# The Architecture Process

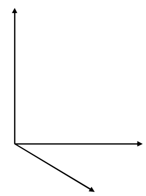


# Performance Measurement and Evaluation

**Many Dimensions to Performance**

- CPU execution time
  - by instruction or sequence
    - floating point
    - integer
    - branch performance
- Cache bandwidth
- Main memory bandwidth
- I/O performance
  - bandwidth
  - seeks
  - pixels or polygons per second
- Relative importance depends on applications



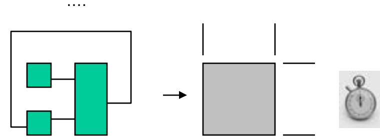


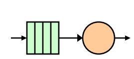
# Evaluation Tools

- Benchmarks, traces, & mixes
  - macrobenchmarks & suites
    - application execution time
  - microbenchmarks
    - measure one aspect of performance
  - traces
    - replay recorded accesses
    - cache, branch, register
- Simulation at many levels
  - ISA, cycle accurate, RTL, gate, circuit
    - trade fidelity for simulation rate
- Area and delay estimation
- Analysis
  - e.g., queuing theory

| | |
|---|---|
| MOVE | 39% |
| BR | 20% |
| LOAD | 20% |
| STORE | 10% |
| ALU | 11% |

LD 5EA3
ST 31FF
....
LD 1EA2
....





## Don't forget the simple view

All a computer does is

- Store and move data
- Communicate with the external world
- Do these two things conditionally
- According to a recipe specified by a programmer

It's complex because

- We want it to be fast
- We want it to be reliable and secure
- We want it to be simple to use
- It must obey the laws of physics



## Next Time

- Evaluation of Systems
  - Performance
    - Amdahl's Law, CPI
  - Cost
- Computer system elements
  - Transistors and wires
- Reading assignment
  - P&H Chapter 1