What is Computer Architecture?

**Architecture**
- abstraction of the hardware for the programmer
- instruction set architecture
  - instructions:
    - operations
    - operands, addressing the operands
    - how instructions are encoded
- storage locations for data
  - registers: how many & what they are used for
  - memory: its size & how it is accessed
- I/O devices & how to access them
- software conventions:
  - subroutine calls: who saves the registers, which ones are saved
  - passing parameters: in registers? on the stack?
- the interface between the software & hardware

What is Computer Organization?

**Organization or Microarchitecture**
- basic components of a computer
  - on the CPU (ALU, registers, PC, etc.)
  - memory (levels of the cache hierarchy)
  - how they operate
  - how they are connected together

Organization is mostly invisible to the programmer
- today some components are considered *part of the architecture*
- why? because a programmer can get better performance if he/she knows the structure
- for example: the caches, the pipeline structure
Separate Architecture & its Organization

Why separate architecture & organization?
- many implementations for 1 architecture
  family of implementations: sequences of machines that have the same ISA
  - IBM 360/85, 360/91, 370s
  - MIPS R2000, R3000, R10000
  - Intel x86, Pentium, Pentium-Pro
  - DEC Alpha 21064, 21164, 21264

⇒ different points in the cost/performance curve
⇒ binary compatible: same software could run on all machines
⇒ open architecture: third party software

Different Architectures

So why have different architectures?
- different architecture philosophies & therefore different styles
  - support high level language operations: CISC
  - support basic primitive operations: RISC
- different application areas
  for example, multimedia instructions
- “ours is better” within the same style
Basic Architectural Design Principles

Design for the common case
common cases in hardware, uncommon cases in software
• basic floating point operations in hardware
  software function for the cosine routine
• memory access in hardware
  trap to software for a page fault

Smaller is faster
must have a good reason for adding an instruction, register, etc.
• memory hierarchy: registers, caches, main memory

Keep it simple, stupid: the KISS principle
simplicity favors regularity, regularity leads to smaller designs and shorter design time
• RISC instructions are all 32 bits

Good design demands compromise
• trade-off in instruction format between
  - the size of the register file (how many bits are needed to specify a register) &
  - the number of operations (how many bits are needed to specify an instruction)
• trade-off between register size & cycle time

Assembly Language
Symbolic form of computer machine language
• advantages for us
  • learn at the machine level what a computer does
  • thorough understanding through a hands-on experience
• where assembly language is used in practice
  • things that aren’t expressible in a high-level language
    for example, subroutine linkage
  • privileged tasks
    for example, programs that need access to protected registers (I/O)
  • size-critical applications
    for example, programs for embedded processors
  • time-critical applications
    for example, real-time applications, OpenGL library
• why assembly language is not widely used
  • lower programmer productivity
    for example, longer coding time, more debugging
  • compilers can produce almost the same quality code
  • not portable across architectures
Still Lower

Implementation

• design of organizational components or microarchitecture

Technology

• semiconductor material
  for example, silicon
• circuit technology (how build gates from transistors)
  for example, CMOS
• packaging
  for example, pin-grid array
• generation
  for example, vacuum tubes, VLSI

A Simplified Machine Model