What is Computer Architecture?

Architecture

- the **interface** between the hardware & lowest levels of the software
  - **model** of the hardware
  - **contract** between the hardware & software
- 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 arguments: in registers? on the stack?

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

  - MIPS R2000, R3000, R12000
  - Intel x86, Pentium series
  - IBM 360/85, 360/91, 370s

  $\Longrightarrow$ different points in the cost/performance curve

  $\Longrightarrow$ share software development costs

  $\Longrightarrow$ binary compatible: same software could run on all machines

  $\Longrightarrow$ open architecture: third party software

Different Architectures

Why have different architectures?

- different architecture philosophies & therefore different styles
  - support high level language operations: CISC
  - support basic primitive operations: RISC
- different application areas

  example: multimedia instructions
- “ours is better” within the same style

Basic Architectural Design Principles

Design for the common case & make it fast

- common cases in hardware, uncommon cases in software
- make the common case hardware fast, even if it slows down the uncommon cases
- if a feature is not the common case, must have a good reason for adding it
- *examples*:
  - basic floating point operations in hardware
    software library function for the cosine routine
  - memory access in hardware
    trap to software for a page fault

Why does this principle work?

- executing in hardware is faster than emulating in software
- **Amdahl's Law**: performance gain due to a hardware feature is limited by the fraction of time the feature is used

Basic Architectural Design Principles

Smaller is faster

- *examples*:
  - memory hierarchy: registers, caches, main memory
  - distributed rather than centralized designs

Keep it simple

- simplicity leads to smaller, regular designs and shorter design & debug time
- *example*: 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 (or cache) 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
  - easier for humans to understand than patterns of 1's & 0's

- where assembly language is used in practice
  - things that aren't expressible in a high-level language
    example: subroutine linkage
  - privileged tasks
    example: programs that need access to protected registers (I/O)
  - size-critical applications
    example: programs for embedded processors
  - time-critical applications
    example: real-time applications, OpenGL library

- why assembly language is not widely used
  - lower programmer productivity
    example: longer coding time, more debugging
  - optimizing 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