

Lecture 1

Fundamentals of Computer Architecture
Amdahl's Law

Instructor: Jun Yang

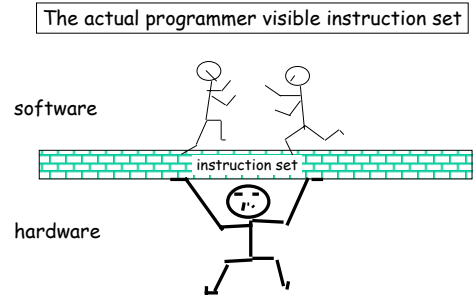
Three Different Computing Markets

- Desktop Computing
 - Still the largest market in \$ terms.
 - Driven by optimizing *price-performance*.
- Servers - computers that provide large-scale services such as reliable, long-term file storage and access, larger memory and more computing power.
 - Availability
 - Scalability
 - Throughput
- Embedded Computers
 - Price
 - Real-time performance requirement
 - Memory size and power consumption

What is *Computer Architecture*?

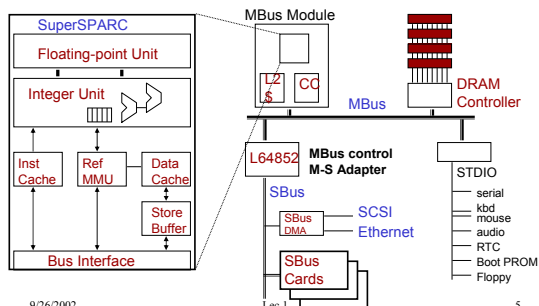
Computer Architecture =
Instruction Set Architecture +
Organization +
Hardware + ...

The Instruction Set: a Critical Interface



Example Organization

- TI SuperSPARC™ TMS390Z50 in Sun SPARCstation20



Hardware

- Machine specifics:
 - Feature size (10 microns in 1971 to 0.18 microns in 2001)
 - Minimum size of a transistor or a wire in either the x or y dimension
 - Logic designs
 - Packaging technology
 - Clock rate
 - Supply voltage
 - ...

Relationship Between the Three Aspects

- Processors having identical ISA may be very different in organization.
 - e.g. NEC VR 5432 and NEC VR 4122
- Processors with identical ISA and nearly identical organization are still not nearly identical.
 - e.g. Pentium II and Celeron are nearly identical but differ at clock rates and memory systems

➤ Architecture covers all three aspects.

9/26/2002

Lec 1

7

Measuring Performance

- CPU Execution time (not including the time waiting for I/O or running other program in a multi-programmed environment)
 - user CPU time + system CPU time (inaccurate)
- Throughput (tasks completed per unit time)
 - measuring large data processing center, etc.

9/26/2002

Lec 1

8

Choosing Programs to Evaluate Perf.

- Synthetic benchmarks
 - Attempt to match average frequencies of operations and operands in real workloads.
 - e.g., Whetstone, Dhrystone
- Toy benchmarks
 - 10 - 100 lines of code
 - e.g., quicksort, puzzle
- Kernels
 - Time critical excerpts of real programs
 - e.g., livermore loops
- Real programs
 - e.g., gcc, spice, SPEC89 - SPEC2000

9/26/2002

Lec 1

9

Performance Terminology

"X is n times faster than Y" means:

$$\frac{\text{Execution time}_y}{\text{Execution time}_x} = n$$

"X is $m\%$ faster than Y" means:

$$\frac{\text{Execution time}_y - \text{Execution time}_x}{\text{Execution time}_x} \times 100\% = m$$

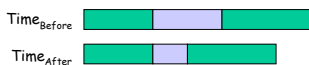
9/26/2002

Lec 1

10

Compute Speedup - Amdahl's Law

Speedup is due to enhancement(E):



$$\text{Speedup (E)} = \frac{\text{Execution time w/o E (Before)}}{\text{Execution time w E (After)}}$$

Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected, what is the *Execution time_{after}* and *Speedup(E)*?

9/26/2002

Lec 1

11

Amdahl's Law

$$\text{Execution time}_{\text{after}} = \text{ExTime}_{\text{before}} \times \left[(1-F) + \frac{F}{S} \right]$$

$$\text{Speedup(E)} = \frac{\text{ExTime}_{\text{before}}}{\text{ExTime}_{\text{after}}} = \frac{1}{\left[(1-F) + \frac{F}{S} \right]}$$

9/26/2002

Lec 1

12

Amdahl's Law - An Example

Q: Floating point instructions improved to run 2X; but only 10% of execution time are FP ops. What is the execution time and speedup after improvement?

Ans:

$$F = 0.1, S = 2$$

$$\text{ExTime}_{\text{after}} = \text{ExTime}_{\text{before}} \times [(1-0.1) + 0.1/2] = 0.95 \text{ ExTime}_{\text{before}}$$

$$\text{Speedup} = \frac{\text{ExTime}_{\text{before}}}{\text{ExTime}_{\text{after}}} = \frac{1}{0.95} = 1.053$$

Read examples in the book!

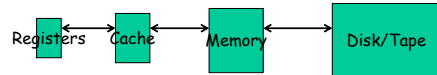
9/26/2002

Lec 1

13

Corollary: Make the common case fast

- All instructions require an instruction fetch, only a fraction require a data fetch/store.
 - Optimize instruction access over data access
- Programs exhibit locality
 - Spatial Locality
 - Temporal Locality
- Access to small memories is faster
 - Provide a storage hierarchy such that the most frequent accesses are to the smallest (closest) memories.



9/26/2002

Lec 1

14

CPU Performance

The Fundamental Law

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

Three components of CPU performance:

- Instruction count
- CPI
- Clock cycle time

Program	Inst. Count	CPI	Clock
Program	X		
Compiler	X	X	
Inst. Set Architecture	X	X	X
μArch		X	X
Physical Design			X

9/26/2002

Lec 1

15

CPI - Cycles per Instruction

Average CPI:

$$\text{CPI} = \frac{\text{Total Cycle}}{\text{Total Instruction Count}} = \sum_{i=1}^n \text{CPI}_i \times F_i \quad \text{where } F_i = \frac{\text{IC}_i}{\text{Instruction Count}}$$

$$\text{CPU time} = \text{Cycle time} \times \sum_{i=1}^n (\text{CPI}_i \times \text{IC}_i)$$

Example:

Instruction type	ALU	Load	Store	Branch
Frequency	43%	21%	12%	24%
Clock cycles	1	2	2	2

$$\text{average CPI} = 0.43 + 0.42 + 0.24 + 0.48 = 1.57 \text{ cycles/instruction}$$

9/26/2002

Lec 1

16

Example

Instruction mix of a RISC architecture.

Inst.	ALU	Load	Store	Branch
Freq.	50%	20%	10%	20%
C.C.	1	2	2	2

- Add a register-memory ALU instruction format?
- One op. in register, one op. in memory
- The new instruction will take 2 cc but will also increase the Branches to 3 cc.

Q: What fraction of loads must be eliminated for this to pay off?

9/26/2002

Lec 1

17

Solution

Instr.	F_i	CPI_i	$\text{CPI}_i \times F_i$	I_i	CPI_i	$\text{CPI}_i \times I_i$
ALU	.5	1	.5	.5-X	1	.5-X
Load	.2	2	.4	.2-X	2	.4-2X
Store	.1	2	.2	.1	2	.2
Branch	.2	2	.4	.2	3	.6
Reg/Mem				X	2	2X
	1.0		$\text{CPI}=1.5$	1-X		$(1.7-X)/(1-X)$

$$\text{Exec Time} = \text{Instr. Cnt.} \times \text{CPI} \times \text{Cycle time}$$

$$\text{Instr. Cnt.}_{\text{old}} \times \text{CPI}_{\text{old}} \times \text{Cycle time}_{\text{old}} \geq \text{Instr. Cnt.}_{\text{new}} \times \text{CPI}_{\text{new}} \times \text{Cycle time}_{\text{new}}$$

$$1.0 \times 1.5 \geq (1-X) \times (1.7-X) / (1-X)$$

$$X \geq 0.2$$

ALL loads must be eliminated for this to be a win!

9/26/2002

Lec 1

18

MIPS and MFLOPS

- **MIPS**: millions of instructions per second:
 - $MIPS = \text{Inst. count} / (\text{CPU time} * 10^{**6}) = \text{Clock rate} / (\text{CPI} * 10^6)$
 - easy to understand and to market
 - inst. set dependent, cannot be used across machines.
 - program dependent
 - can vary inversely to performance! (why? read the book)
- **MFLOPS**: million of FP ops per second.
 - less compiler dependent than MIPS.
 - not all FP ops are implemented in h/w on all machines.
 - not all FP ops have same latencies.
 - normalized MFLOPS: uses an equivalence table to even out the various latencies of FP ops.

9/26/2002

Lec 1

19

Readings

- Chapter 1
1.7 - 1.9

9/26/2002

Lec 1

20