

Lecture 1-2

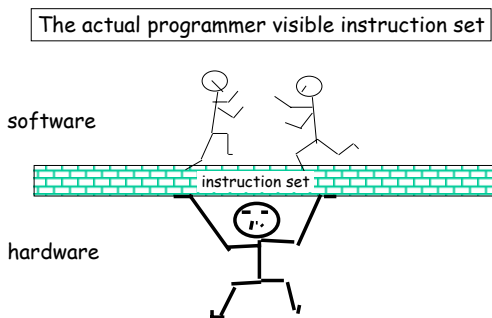
Fundamentals of Computer Architecture  
Amdahl's Law

Instructor: Jun Yang

What is \*Computer Architecture\*

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

The Instruction Set: a Critical Interface



Instruction-Set Processor Design

- **Architecture (ISA)** programmer/compiler view
  - "functional appearance to its immediate user/system programmer"
  - **Opcodes, addressing modes, architected registers, IEEE floating point**
- **Implementation ( $\mu$ architecture)** processor designer/view
  - "logical structure or organization that performs the architecture"
  - **Pipelining, functional units, caches, physical registers**
- **Realization (chip)** chip/system designer view
  - "physical structure that embodies the implementation"
  - **Gates, cells, transistors, wires**

## Pentium 4 Microarchitecture

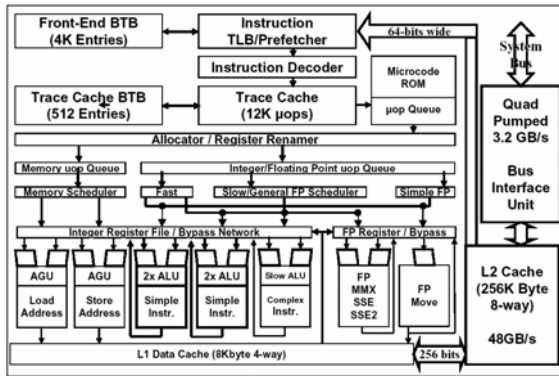
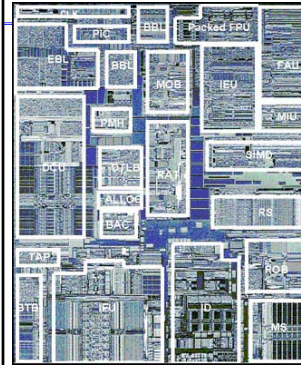


Figure 4: Pentium® 4 processor microarchitecture

## Pentium III Die Photo



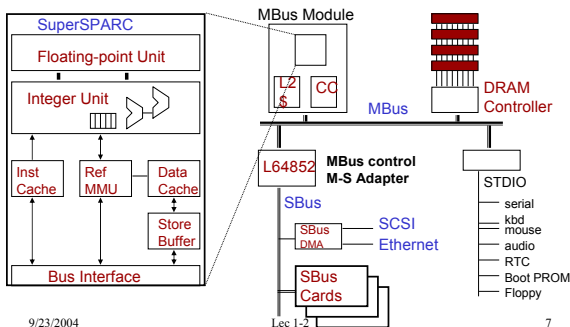
- EBL/BBU - Bus logic, Front, Back
- MOB - Memory Order Buffer
- Packed FPU - MMX Fl. Pt. (SSE)
- IEU - Integer Execution Unit
- FAU - Fl. Pt. Arithmetic Unit
- MIU - Memory Interface Unit
- DCU - Data Cache Unit
- PMH - Page Miss Handler
- DTLB - Data TLB
- BAC - Branch Address Calculator
- RAT - Register Alias Table
- SIMD - Packed Fl. Pt.
- RS - Reservation Station
- BTB - Branch Target Buffer
- TFU - Instruction Fetch Unit (+T\$)
- ID - Instruction Decode
- ROB - Reorder Buffer
- MS - Micro-instruction Sequencer

1st Pentium III, Katmai: 9.5 M transistors, 12.3 \* 10.4 mm in 0.25-μm with 5 layers of aluminum

6

## Example Organization

- TI SuperSPARC™ TMS390Z50 in Sun SPARCstation20



9/23/2004

Lec 1-2

7

## 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
  - ...

9/23/2004

Lec 1-2

8

## 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.**

9/23/2004

Lec 1-2

9

## Applications and Requirements

- Scientific/numerical: weather prediction, molecular modeling
  - Need: large memory, floating-point arithmetic
- Commercial: inventory, payroll, web serving, e-commerce
  - Need: integer arithmetic, high I/O
- Embedded: automobile engines, microwave, PDAs
  - Need: low power, low cost, interrupt driven
- Home computing: multimedia, games, entertainment
  - Need: high data bandwidth, graphics

9/23/2004

Lec 1-2

10

## Classes of Computers

- High performance (supercomputers)
  - Supercomputers - Cray T-90
  - Massively parallel computers - Cray T3E
- Balanced cost/performance
  - Workstations - SPARCstations
  - Servers - SGI Origin, UltraSPARC
  - High-end PCs - Pentium quads
- Low cost/power
  - Low-end PCs, laptops, PDAs - mobile Pentiums

9/23/2004

Lec 1-2

11

## Why Study Computer Architecture

- Aren't they fast enough already?
  - Are they?
  - Fast enough to do everything we will EVER want?
    - AI, protein sequencing, graphics
  - Is speed the only goal?
    - Power: heat dissipation + battery life
    - Cost
    - Reliability
    - Etc.

**Answer #1: requirements are always changing**

9/23/2004

Lec 1-2

12

## Why Study Computer Architecture

### Answer #2: technology playing field is always changing

- Annual technology improvements (approx.)
  - Logic: density + 25%, speed +20%
  - DRAM (memory): density +60%, speed: +4%
  - Disk: density +25%, disk speed: +4%
- Designs change even if requirements are fixed. But the requirements are not fixed.

9/23/2004

Lec 1-2

13

## Example of Changing Designs

- Having, or not having caches
  - 1970: 10K transistors on a single chip, DRAM faster than logic → having a cache is bad
  - 1990: 1M transistors, logic is faster than DRAM → having a cache is good
  - Will caches ever be a bad idea again?

9/23/2004

Lec 1-2

14

## Performance Growth in Perspective

- Same absolute increase in computing power
  - Big Bang - 2001
  - 2001 - 2003
- 1971 - 2001: performance improved 35,000X!!!
  - What if cars improved at this rate?

9/23/2004

Lec 1-2

15

## Measuring Performance

- Latency (response time, execution time)
  - Minimize time to wait for a computation
- Throughput (tasks completed per unit time, bandwidth)
  - Maximize work done in a given interval
  - =  $1/\text{latency}$  when there is no overlap among tasks
  - $> 1/\text{latency}$  when there is
    - In real processors there is always overlap (pipelining)
- Both are important

9/23/2004

Lec 1-2

16

## 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:

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

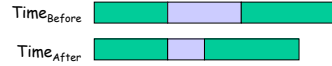
9/23/2004

Lec 1-2

17

## 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<sub>after</sub>* and *Speedup(E)*?

9/23/2004

Lec 1-2

18

## Amdahl's Law

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

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

9/23/2004

Lec 1-2

19

## 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!

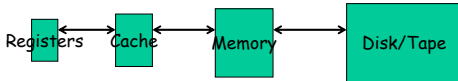
9/23/2004

Lec 1-2

20

## 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.



9/23/2004

Lec 1-2

21

## 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

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

9/23/2004

Lec 1-2

22

## 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

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

9/23/2004

Lec 1-2

23

## 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?

9/23/2004

Lec 1-2

24

## Solution

Instr.	$F_i$	$CPI_i$	$CPI_i \times F_i$	$I_i$	$CPI_i$	$CPI_i \times I_i$
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		$CPI=1.5$	1-X		$(1.7-X)/(1-X)$

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

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

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

$X \geq 0.2$

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

9/23/2004

Lec 1-2

25

## Benchmarks

- "program" as unit of work
  - There are millions of programs
  - Not all are the same, most are very different
  - Which ones to use?
- Benchmarks
  - Standard programs for measuring or comparing performance
  - Representative of programs people care about repeatable!!

9/23/2004

Lec 1-2

26

## Choosing Programs to Evaluate Perf.

- Toy benchmarks
  - e.g., quicksort, puzzle
  - No one really runs. Scary fact: used to prove the value of RISC in early 80's
- Synthetic benchmarks
  - Attempt to match average frequencies of operations and operands in real workloads.
  - e.g., Whetstone, Dhrystone
  - Often slightly more complex than kernels; But do not represent real programs
- Kernels
  - Most frequently executed pieces of real programs
  - e.g., livermore loops
  - Good for focusing on individual features not big picture
  - Tend to over-emphasize target feature
- Real programs
  - e.g., gcc, spice, SPEC89, 92, 95, SPEC2000 (standard performance evaluation corporation)

27

## MIPS and MFLOPS

- **MIPS**: millions of instructions per second:
  - $\text{MIPS} = \text{Inst. count} / (\text{CPU time} * 10^{**6}) = \text{Clock rate} / (\text{CPI} * 10^6)$
  - 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.

9/23/2004

Lec 1-2

28