Algorithms and Computation in Signal Processing

special topic course 18-799B
spring 2005
4^{th} Lecture Jan. 20, 2005

Instructor: Markus Pueschel
TA: Srinivas Chellappa

For Publications

- A problem has a complexity
- An algorithm has a cost (e.g., operations count, runtime, memory requirement, area requirement in hardware)
- Cost = runtime can only be analyzed asymptotically
- In a precise sense, an algorithm does not have a complexity

Problem	Complexity Runtime compl. (asympt.)
Algorithm	Cost Runtime (asymptotic)

In research/writing/publications:

If your contribution is an algorithm, you have to analyze it. As follows:

1) state your cost/complexity measure (what you count);
2) compute the cost of the algorithm as precise as possible/necessary, at least asymptotically;
3) state what you know about the complexity of the problem you address (from theory, other algorithms, ...)

Architecture and Microarchitecture: What’s Important for the Programmer

Definitions

- Architecture: (also instruction set architecture: ISA) The parts of a processor design that one needs to understand to write assembly code. Examples: instruction set specification, registers. Counterexamples: cache sizes and core frequency.

  Example (ISA): x86, ia, ipf

- Microarchitecture: Implementation of the architecture.

  Example: Pentium4 microarchitectures *link* *link*

Microarchitecture: memory hierarchy, cache structure, and processor

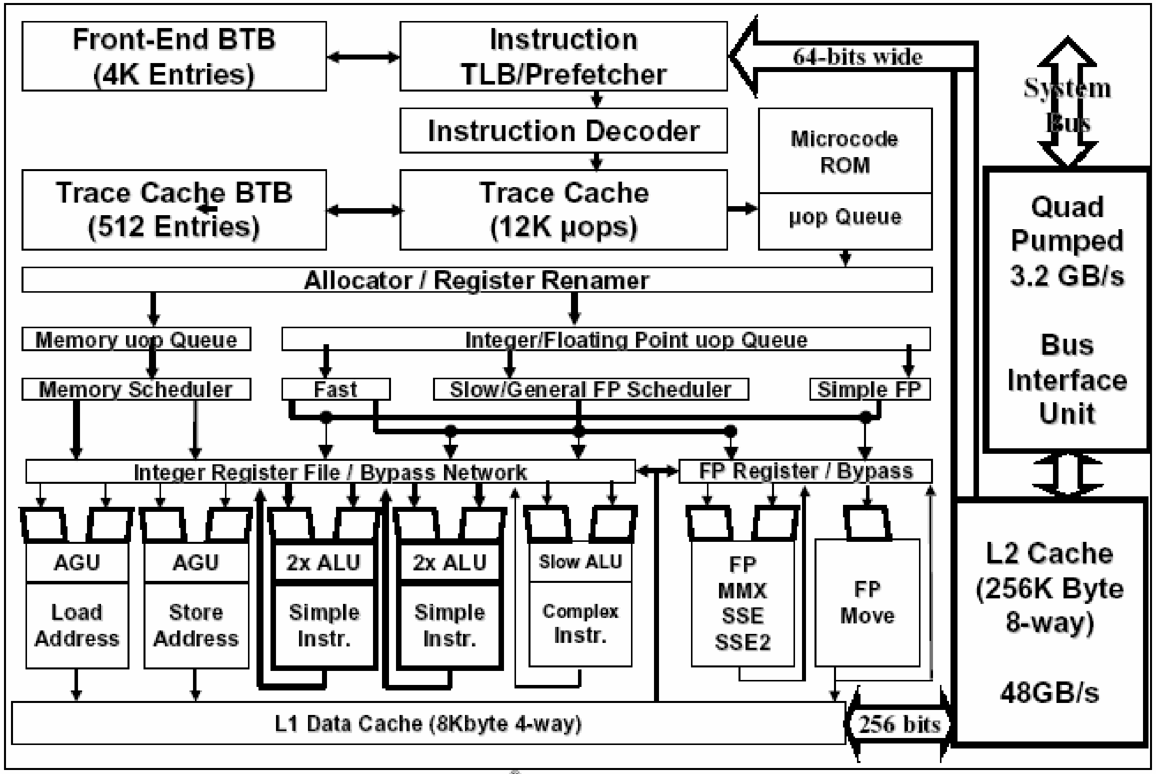

Figure 4: Pentium® 4 processor microarchitecture

we take the software developers view … (blackboard)

Source: "The Microarchitecture of the Pentium 4 Processor," Intel Technology Journal Q1 2001

Execution Units: Pentium 4

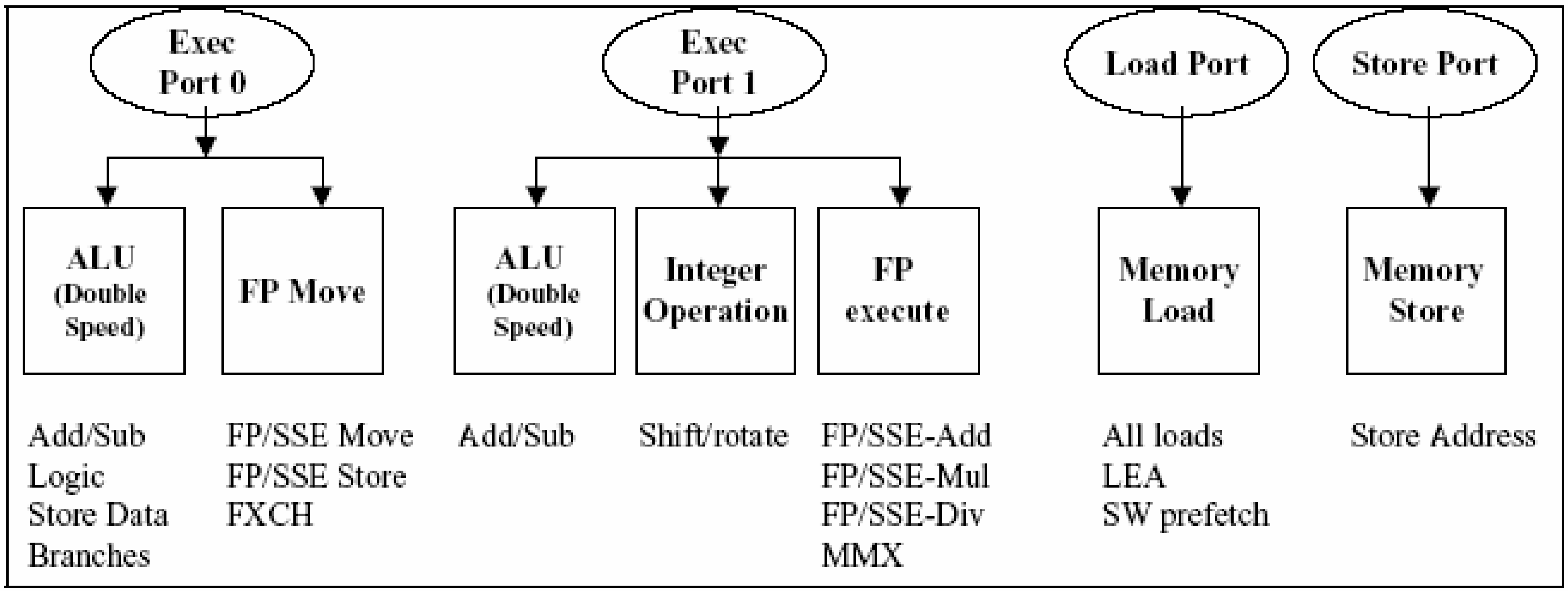

Figure 6: Dispatch ports in the Pentium® 4 processor

Source: "The Microarchitecture of the Pentium 4 Processor," Intel Technology Journal Q1 2001

Remarks

- HW optimizations
  - partially frees programmer from optimization
  - targets most common code patterns and most important benchmarks
- Many HW optimizations/features are not (or not well) documented
- Performance is hard to understand. Two major unknowns: compiler and actual execution
- No very clear guidelines how to optimize code
  - some provided in vendor's SW optimization manuals

Remarks (cont'd)

- Often vendor compilers are best
  - but, e.g., icc cannot distinguish different processor cores (switches p2, p3, p4)

- Not always clear which compiler flags are best (in particular gcc)

- Most benchmarks/software is not floating point based (think Word); thus, HW optimizations target first integers ops

Optimization of Numerical Software: First Thoughts

- It's all about keeping the floating point units busy
- Need to optimize for memory hierarchy
  - for several levels
  - often requires algorithm modifications or proper algorithm choice
  - divide-and-conquer algorithms are in principal good
    (recursive is better than iterative)
- Need for fine-grain instruction parallelism
- Rule: don't code in assembly if you can avoid it
- Use a good compiler and make sure you understand flags

Microarchitectural Parameters Most Important for Programmers

- Memory hierarchy:
  - How many caches
  - Cache sizes and structure
  - Number of registers
- Processor
  - Frequency
  - Execution units
  - Latency and throughput of fadd, fmult, etc.
  - Floating point peak performance
- How to get it?
  - Digging through manuals, vendor websites.
  - Measuring. E.g., cpuid (Windows only), X-Ray

ISA: SIMD (Signal Instruction Multiple Data) Vector Instructions

- What is it?
  - Extension of the ISA. Data types and instructions for the parallel computation on short (length 2-8) vectors of integers or floats.

4-way

- Why do they exist?
  - **Useful:** Many application (e.g., multimedia) have the necessary fine-grain parallelism. Then, large potential speedup (by a factor close to vector length).
  - **Doable:** Chip designers have enough transistors to play with.

- We will have an extra lecture on vector instructions
  - What are the problems?
  - How to use them efficiently.