Cpu code optimizations

Jiří Novotný & Karel Kubíček

23. October 2015

Why to write optimized code

We don't want our code to procrastinate!

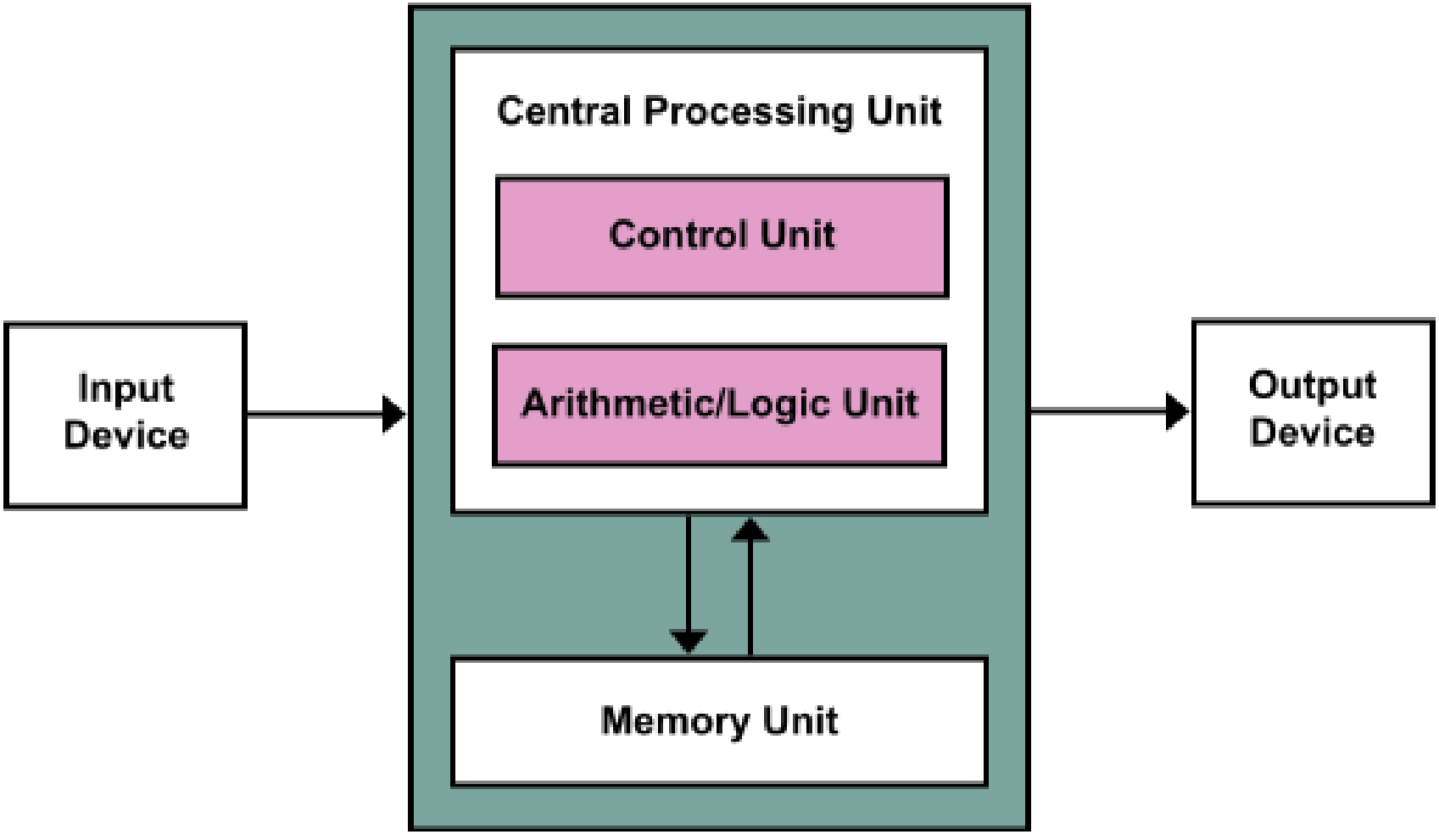
Use best algorithm for you use case

```
bool isPrime(int p) {
  for (int i = 2; i < p; i++)
    if (p % i == 0) return false;
  return true;
}

bool isPrimeBetter(int p) {
  for (int i = 2; i < sqrt(p); i++)
      if (p % i == 0) return false;
  return true;
}

bool isPrimeEvenBetter(int p) {
  for (int i = 2; i*i < p; i++) //or store sqrt once
    if (p % i == 0) return false;
  return true;
}
```

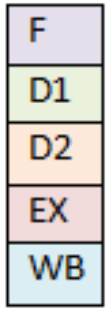
CPU – you know it as a blackbox



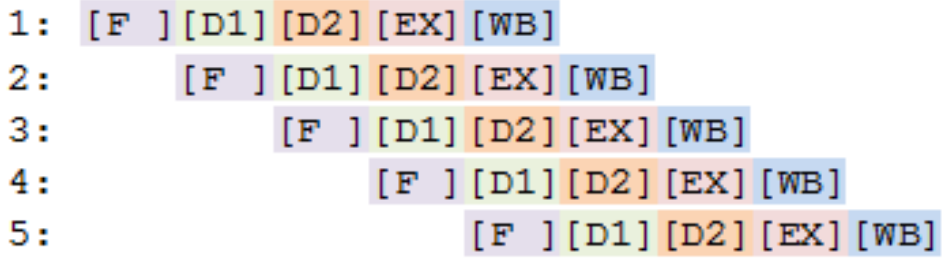
CPU

Little of history :)

- Direct load of instruction from memory
  - Solution: cache
- Load, decode, translate needed memory address, execute, retire
  - Pipeline



The i486 Pipeline

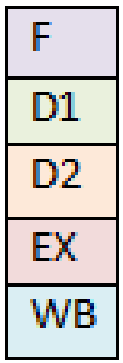


Five instructions going through a pipeline at the same time.

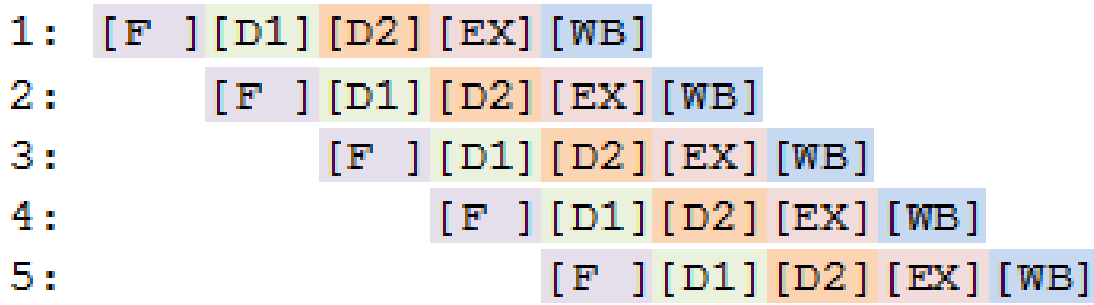
CPU – Little of history :)

- Will this work?

```
void swap(int &a, int &b) {
  int tmp = a;
  a = b;
  b = tmp;
}
```



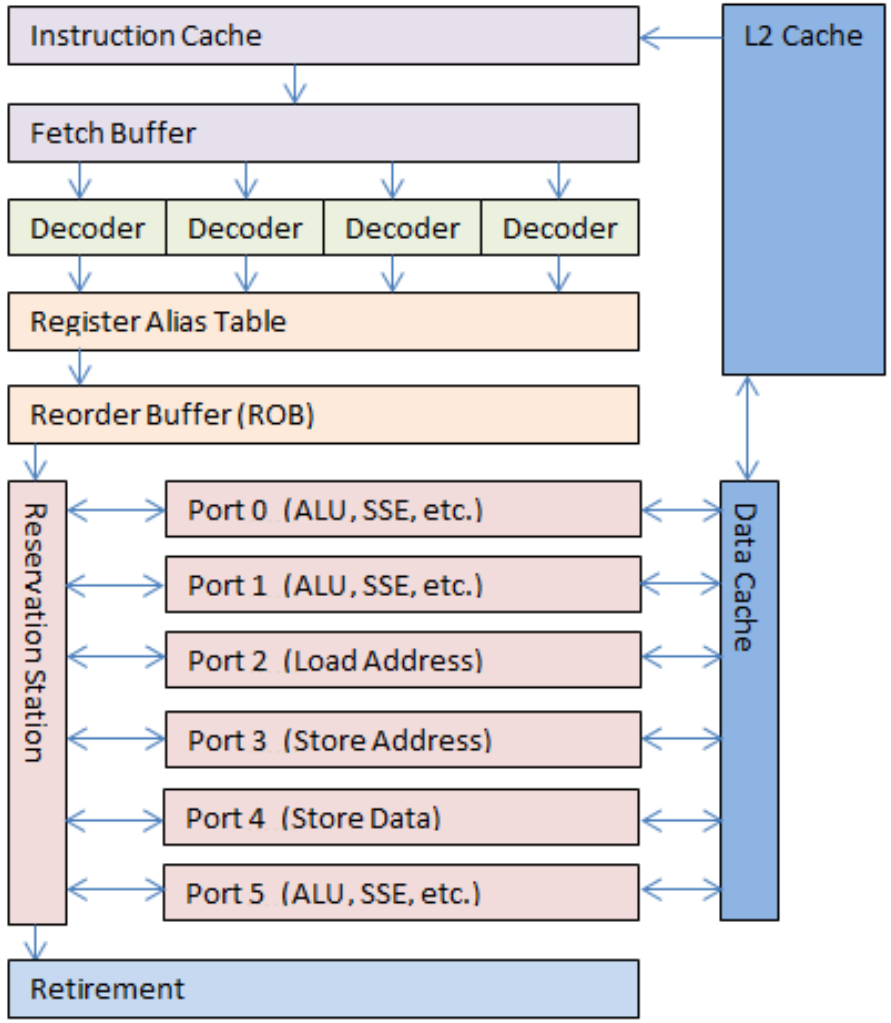
The i486 Pipeline



Five instructions going through a pipeline at the same time.

CPU – Little of history :)

- Pipeline stall (bubble)
  - Out-of-order execution

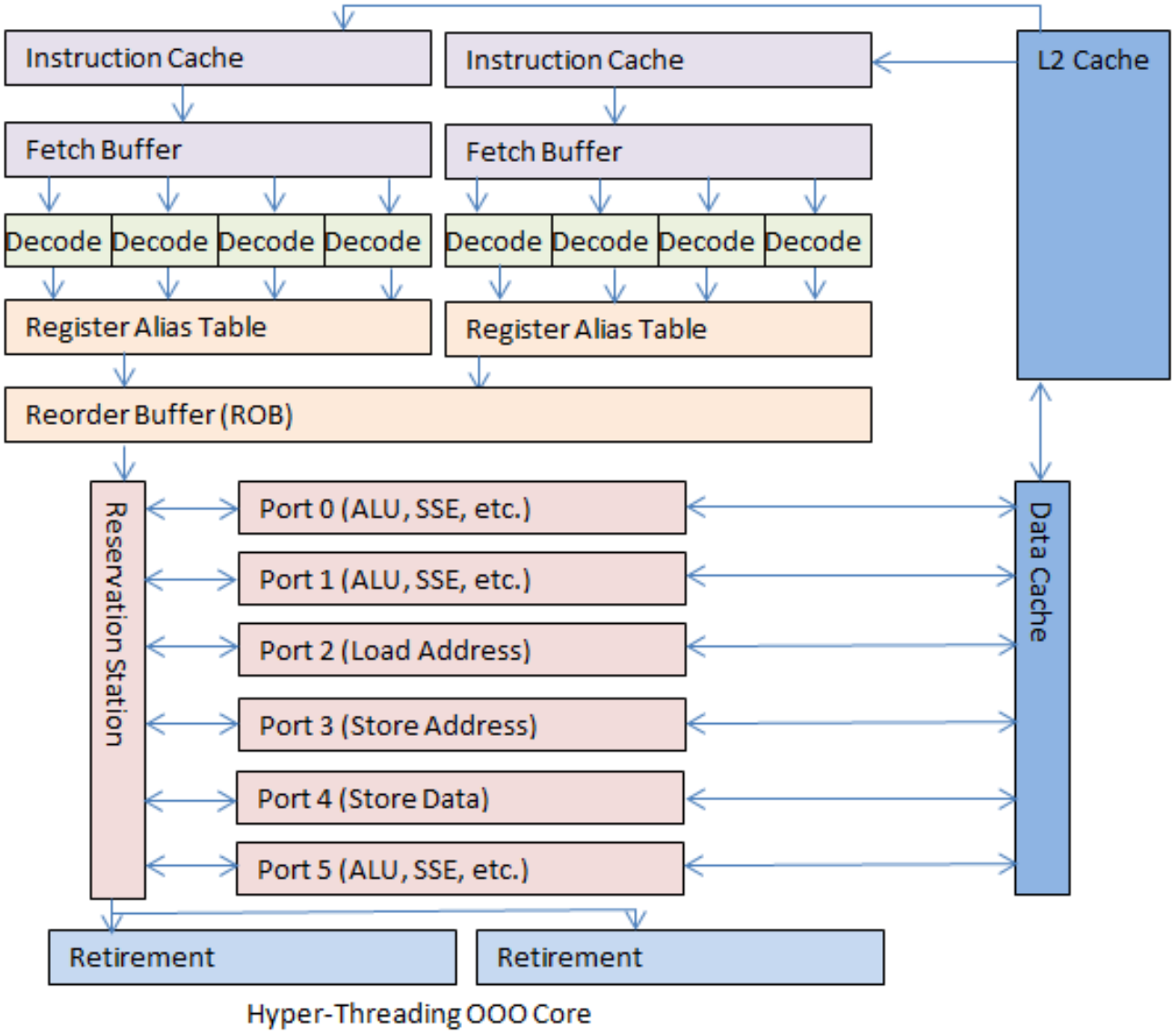


Out Of Order Core as used from 1995 to present. The color coding follows the same five stages used in previous processors. Some stages and buffers are not shown since they vary from processor to processor

CPU – Little of history :)

- OOO needs to load a lot of instructions (128), what to do with conditional jumps?
- Branch prediction
  - Branch misprediction on bublesort
- OK, finally, our CPU is really fast, so fast, that memory is slow and does not provide input data
  - Simultaneous multithreading (HT)

Simultaneous multithreading (HT)



CPU – Modern instructions (x86 is not strict RISC)

- SIMD
  - SSE, SSE2-4.2 (128b)
  - AVX, AVX2 ($\sim$ FMA in AMD's world) (256b)
  - AVX-512 (512b)
- interesting (but not SIMD) new instructions:
  - TSX – Transactional Synchronization Extensions
  - SGX (in development) – Software Guard Extensions

The C/C++ programmer view

- know your compiler, programming language and target architecture
- don't try to be smarter then your compiler – compilers are pretty smart :P
- different levels of compiler optimizations `-O0` `-O1` `-O2` `-O3`

Vectorization

- today compilers can vectorize you code automaticaly

```
for (int i = 0; i < 32; i++)
    c[i] = a[i] + b[i];
```

- . . . but several constraints must be met to make it fast
  - memory alignemnt to 16 bytes (later)
  - process data in order to make the full use of processor's cache

Default allocator

- `malloc/new`
- for most cases they will do fine but when performance is needed they are slow

Custom allocators

- preallocate the memory by the default allocator
- within the preallocated block you can implement you own allocator when you know something about your data
- different types for different purpouse
  - stack allocator (elements of variable size; stack ordering)
  - pool allocator (elements of fixed size; custom ordering)

Chaching

- cacheline – 64 bytes long
- try to fit related variables into one cacheline
- c++11 keyword `alignas`

```
alignas(64) char cacheline[64];
alignas(16) float vec4[4];
```

Profiling

Tools

- `perf` (linux)
- `Microsoft Visual Studio` (windows)

Two possible techniques

- sampling vs. instrumentation

Paralelization

C++11 high level approach (using std::async)

```
#include <future>
std::future<int> result = std::async(&work, args...);
// some work in this thread
result.get() // fetch the result
```

C++11 low level approach (using std::thread)

```
#include <thread>
std::thread t(&work, args...); // launch other thread
// some work
t.join() // wait till the other thread finisches
// ...but we can't use std::future to fetch the result
```

Useful hints

1. write functonal correct code
2. analyze the code and determine parts that are slow and critical
3. optimize them
4. GOTO 2. till satisfied

Great tools

- Online C/C++ to assembly (using many compilers)
- Intel intrinsics reference

Try it yourself!

- Go to crcs.cz -> OpenLab -> CPU optimizations -> task.zip

Sources

- Intro images & idea: A Journey Through the CPU Pipeline
- (Almost) current architecture: Haswell is here. Architecture
- Basic C++ optimizations