

# How to Write Fast Numerical Code

Spring 2019

*Lecture:* Architecture/Microarchitecture and Intel Core

**Instructor:** Markus Püschel

**TA:** Tyler Smith, Gagandeep Singh, Alen Stojanov

**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

## Technicalities

- **Midterm:** *Mon, April 15<sup>th</sup>*
- **Research project:**
  - Let us know once you have a partner
  - If you have a project idea, talk to me (break, after class, email)
  - Deadline: *March 4th*
- **Finding partner:** [fastcode-forum@lists.inf.ethz.ch](mailto:fastcode-forum@lists.inf.ethz.ch)

# Today

- Architecture/Microarchitecture: *What is the difference?*
- In detail: Intel Haswell and Sandybridge
- Crucial microarchitectural parameters
- Peak performance
- Operational intensity

3

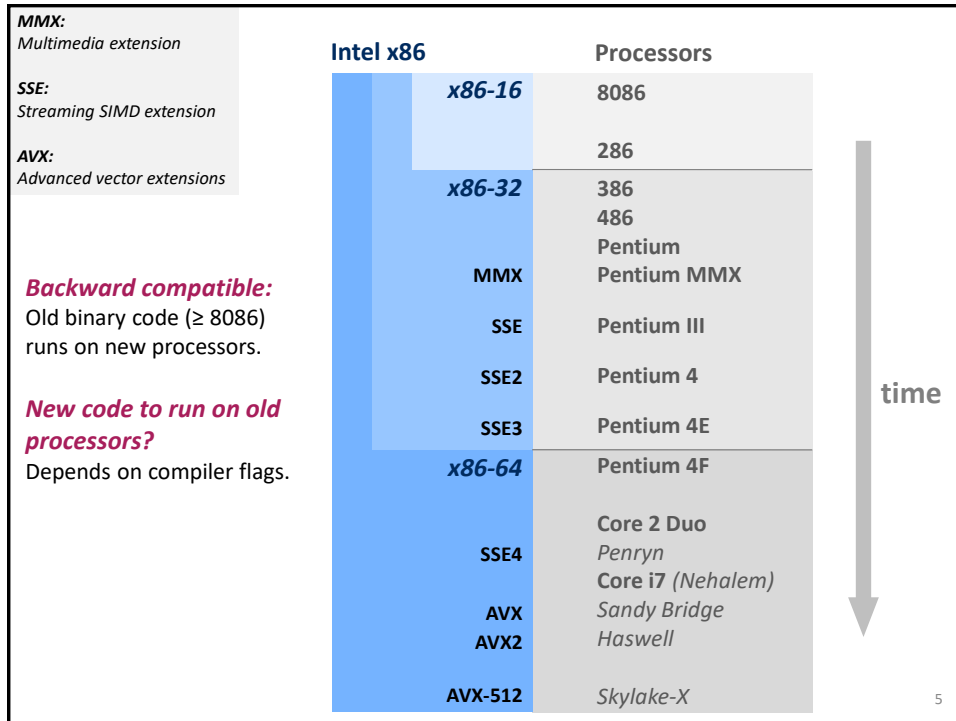
# Definitions

- **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 ISAs**
  - x86
  - ia
  - MIPS
  - POWER
  - SPARC
  - ARM

## Some assembly code

```
ipf:
  xorps  %xmm1, %xmm1
  xorl   %ecx, %ecx
  jmp    .L8
.L10:
  movslq %ecx,%rax
  incl   %ecx
  movss (%rsi,%rax,4), %xmm0
  mulss (%rdi,%rax,4), %xmm0
  addss %xmm0, %xmm1
.L8:
  cmpl  %edx, %ecx
  jl    .L10
  movaps %xmm1, %xmm0
  ret
```

4



## ISA SIMD (Single Instruction Multiple Data) Vector Extensions

### ■ 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

$$\begin{array}{|c|c|c|c|} \hline \color{yellow}{\square} & \color{green}{\square} & \color{red}{\square} & \color{blue}{\square} \\ \hline \end{array} + \begin{array}{|c|c|c|c|} \hline \color{yellow}{\square} & \color{green}{\square} & \color{red}{\square} & \color{blue}{\square} \\ \hline \end{array} \quad \color{black}{\times} \quad \begin{array}{|c|c|c|c|} \hline \color{yellow}{\square} & \color{green}{\square} & \color{red}{\square} & \color{blue}{\square} \\ \hline \end{array} \quad \color{black}{4\text{-way}}$$

- Names: MMX, SSE, SSE2, ..., AVX, ...

### ■ Why do they exist?

- Useful:** Many applications have the necessary fine-grain parallelism  
Then: speedup by a factor close to vector length
- Doable:** Chip designers have enough transistors to play with; easy to build with replication

### ■ We will have an extra lecture on vector instructions

- What are the problems?
- How to use them efficiently

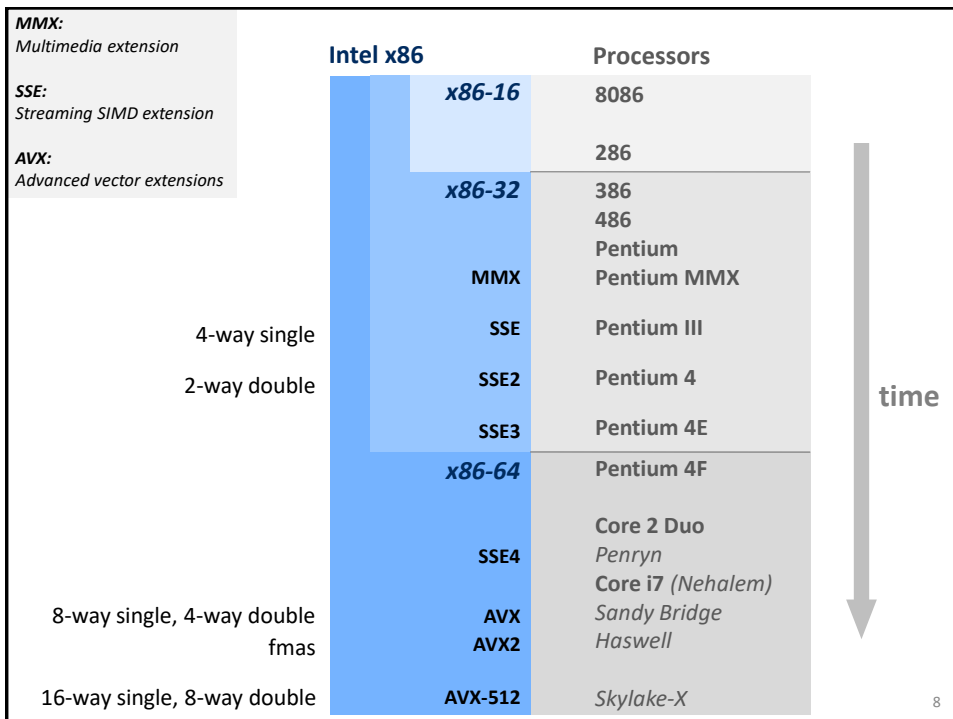
# FMA = Fused Multiply-Add

- $x = x + y * z$
- Done as one operation, i.e., involves only one rounding step
- Better accuracy than sequence of mult and add
- Natural pattern in many algorithms

```
/* matrix multiplication; A, B, C are n x n matrices of doubles */
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      C[i*n+j] += A[i*n+k]*B[k*n+j];
```

- Exists only recently in Intel processors (*Why?*)

7



8

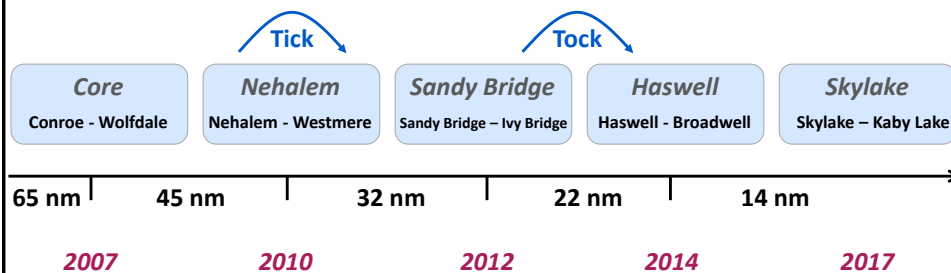
# Definitions

- **Microarchitecture:** Implementation of the architecture
- **Examples:** caches, cache structure, CPU frequency, details of the virtual memory system
- **Examples**
  - Intel processors ([Wikipedia](#))
  - AMD processors ([Wikipedia](#))

9

# Intel's Tick-Tock Model

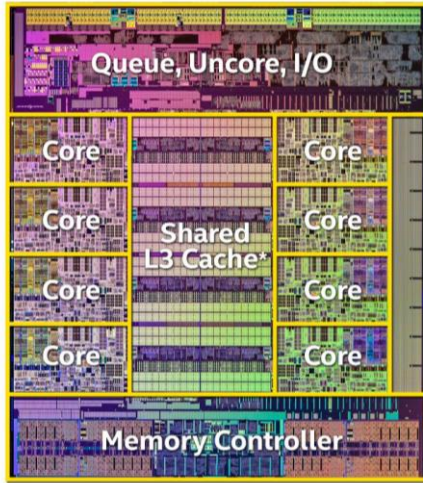
- **Tick:** Shrink of process technology
- **Tock:** New microarchitecture
- **Example: Core and successors**  
Shown: Intel's microarchitecture code names (server/mobile may be different)



*In 2016 the Tick-tock model got discontinued  
Now: process – architecture – optimization (since Tick becomes harder)*

10

# Intel Processors: Example Haswell



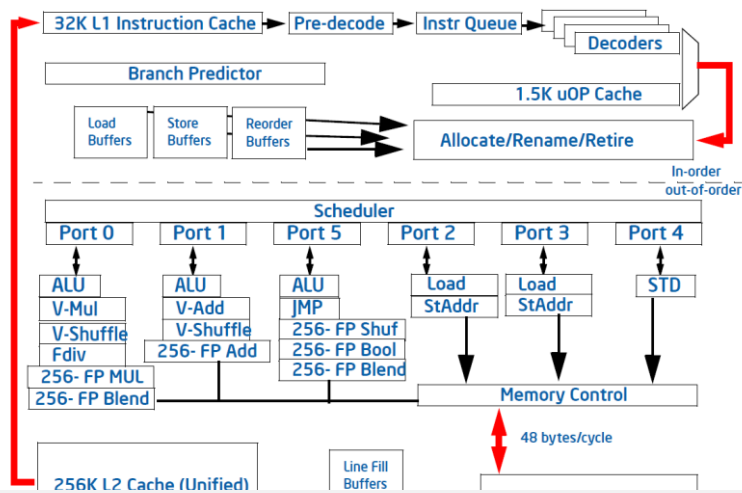
<http://www.anandtech.com>

Intel® Core™ i7-5960X Processor Extreme Edition  
Transistor count: 2.6 Billion  
Die size: 17.6mm x 20.2mm



[Detailed information about Intel processors](#)

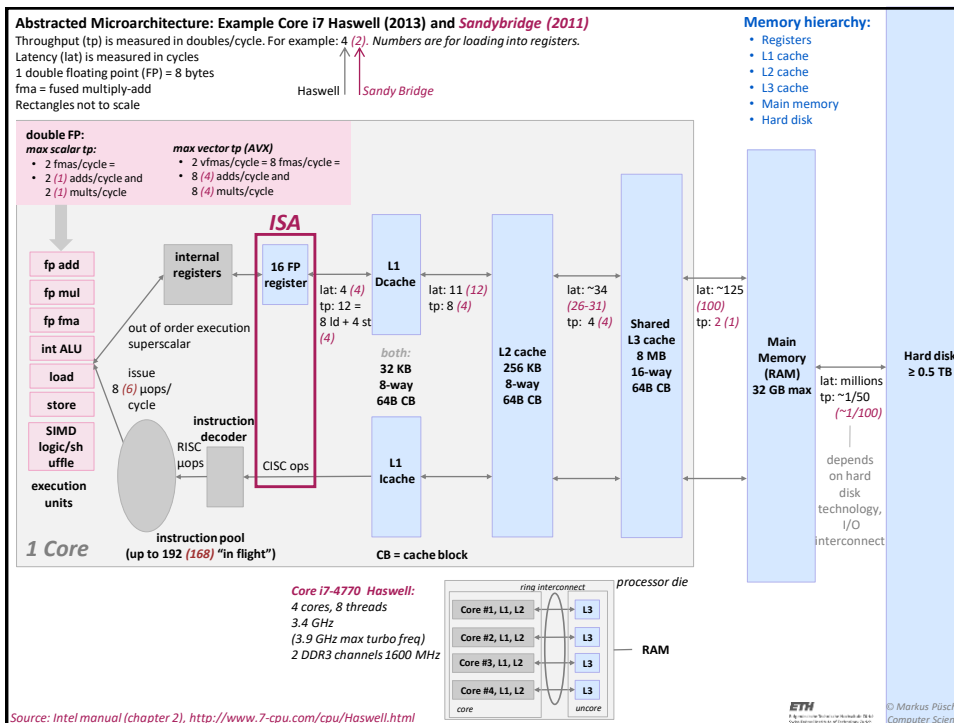
## Microarchitecture: The View of the Computer Architect



*we take the software developer's view ...*

Source: [Intel Architectures Optimization Reference Manual](#)

■ Distribute microarchitecture abstraction



## Runtime Bounds (Cycles) on Haswell

```
/* x, y are vectors of doubles of length n, alpha is a double */
for (i = 0; i < n; i++)
    x[i] = x[i] + alpha*y[i];
```

*maximal achievable percentage  
of (vector) peak performance*

|                                      |       |      |
|--------------------------------------|-------|------|
| ■ Number flops?                      | $2n$  |      |
| ■ Runtime bound no vector ops:       | $n/2$ |      |
| ■ Runtime bound vector ops:          | $n/8$ |      |
| ■ Runtime bound data in L1:          | $n/4$ | 50   |
| ■ Runtime bound data in L2:          | $n/4$ | 50   |
| ■ Runtime bound data in L3:          | $n/2$ | 25   |
| ■ Runtime bound data in main memory: | $n$   | 12.5 |

*Runtime dominated by data movement:  
Memory-bound*

## Runtime Bounds (Cycles) on Core 2

```
/* matrix multiplication; A, B, C are n x n matrices of doubles */
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = 0; k < n; k++)
            C[i*n+j] += A[i*n+k]*B[k*n+j];
```

|                                      |           |
|--------------------------------------|-----------|
| ■ Number flops?                      | $2n^3$    |
| ■ Runtime bound no vector ops:       | $n^3/2$   |
| ■ Runtime bound vector ops:          | $n^3/8$   |
| ■ Runtime bound data in L1:          | $3/8 n^2$ |
| ■ ...                                |           |
| ■ Runtime bound data in main memory: | $3/2 n^2$ |

*Runtime dominated by data operations (except very small n):  
Compute-bound*



## Operational Intensity

- Definition: Given a program P, assume cold (empty) cache

$$\text{Operational intensity: } I(n) = \frac{W(n)}{Q(n)}$$

#flops (input size n) ←  $W(n)$   
#bytes transferred cache ↔ memory (for input size n) ←  $Q(n)$

17

## Operational Intensity (Cold Cache)

```
/* x, y are vectors of doubles of length n, alpha is a double */  
for (i = 0; i < n; i++)  
  x[i] = x[i] + alpha*y[i];
```

- Operational intensity:
  - Flops:  $W(n) = 2n$
  - Memory transfers (doubles):  $\geq 2n$  (just from the reads)
  - Reads (bytes):  $Q(n) \geq 16n$
  - Operational intensity:  $I(n) = W(n)/Q(n) \leq 1/8$

18

## Operational Intensity (Cold Cache)

```
/* matrix multiplication; A, B, C are n x n matrices of doubles */
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    for (k = 0; k < n; k++)
      C[i*n+j] += A[i*n+k]*B[k*n+j];
```

- **Operational intensity:**
  - Flops:  $W(n) = 2n^3$
  - Memory transfers (doubles):  $\geq 3n^2$  (just from the reads)
  - Reads (bytes):  $Q(n) \geq 24n^2$
  - Operational intensity:  $I(n) = W(n)/Q(n) \leq 1/12 n$

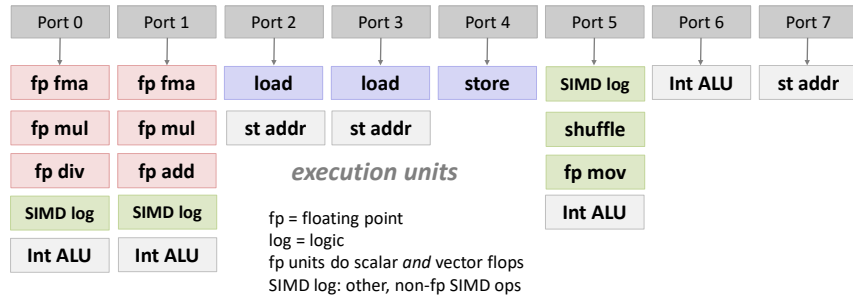
19

## Compute/Memory Bound

- A function/piece of code is:
  - **Compute bound** if it has high operational intensity
  - **Memory bound** if it has low operational intensity
- A more exact definition depends on the given platform
- More details later: Roofline model

20

## Mapping of execution units to ports

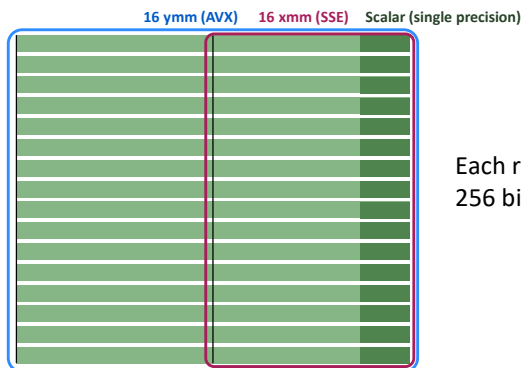


| Execution Unit (fp) | Latency [cycles] | Throughput [ops/cycle] | Gap [cycles/issue] |
|---------------------|------------------|------------------------|--------------------|
| fma                 | 5                | 2                      | 0.5                |
| mul                 | 5                | 2                      | 0.5                |
| add                 | 3                | 1                      | 1                  |
| div (scalar)        | 14-20            | 1/13                   | 13                 |
| div (4-way)         | 25-35            | 1/27                   | 27                 |

- Every port can issue one instruction/cycle
- Gap = 1/throughput
- **Intel calls gap the throughput!**
- Same units for scalar and vector flops
- Same latency/throughput for scalar (one double) and AVX vector (four doubles) flops, except for div

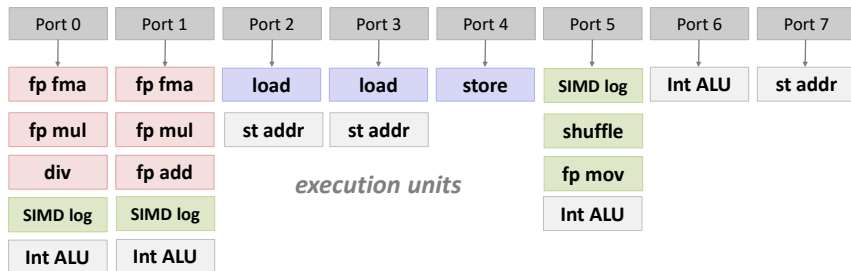
Source: Intel manual (Table C-8. 256-bit AVX Instructions, Table 2-6. Dispatch Port and Execution Stacks of the Haswell Microarchitecture, Figure 2-2. CPU Core Pipeline Functionality of the Haswell Microarchitecture).

## Floating Point Registers



- Same 16 registers for scalar FP, SSE and AVX
- Scalar (non-vector) single precision FP code uses the bottom eighth
- Explains why throughput and latency is usually the same for vector and scalar operations

22



**How many cycles are at least required (no vector ops)?**

- A function with  $n$  adds and  $n$  mults in the C code  $n/2$
- A function with  $n$  add and  $n$  mult instructions in the assembly code  $n$
- A function with  $n$  adds in the C code  $n/2$
- A function with  $n$  add instructions in the assembly code  $n$
- A function with  $n$  adds and  $n/2$  mults in the C code  $n/2$

23

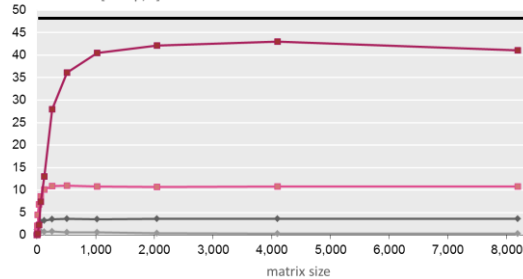
## Comments on Intel Haswell $\mu$ arch

- **Peak performance 16 DP flops/cycle (only reached if SIMD FMA)**
  - Peak performance mults: 2 mults/cycle (scalar 2 flops/cycle, SIMD AVX 8 flops/cycle)
  - Peak performance adds: 1 add/cycle (scalar 1 flop/cycle, SIMD AVX 4 flops/cycle). FMA in port 0 can be used for add, but longer latency
- **L1 bandwidth: two 32-byte loads and one 32-byte store per cycle** (*Sandy Bridge, either one 16-byte load and one 16-byte store, or one 32-byte load*)
- **Shared L3 cache organized as multiple cache slices for better scalability with number of cores, thus access time is non-uniform**
- **Shared L3 cache in a different clock domain (uncore)**

## Example: Peak Performance

Matrix-Matrix Multiplication (MMM) on 2 x Core 2 Duo 3 GHz (Sandy Bridge)

Performance [Gflop/s]



**Peak performance of this computer:**  
4 cores x  
2-way SSE x  
1 add and 1 mult/cycle  
= 16 flops/cycle  
= 48 Gflop/s

25

## Summary

- Architecture vs. microarchitecture
- To optimize code one needs to understand a suitable abstraction of the microarchitecture and its key quantitative characteristics
  - Memory hierarchy with throughput and latency info
  - Execution units with port, throughput, and latency info
- Operational intensity:
  - High = compute bound = runtime dominated by data operations
  - Low = memory bound = runtime dominated by data movement

26