

# High-Performance Computing for Embedded Systems (HPEC)

[Lab 1 - Advanced CPU features]

[Lab 1 - Advanced CPU features]

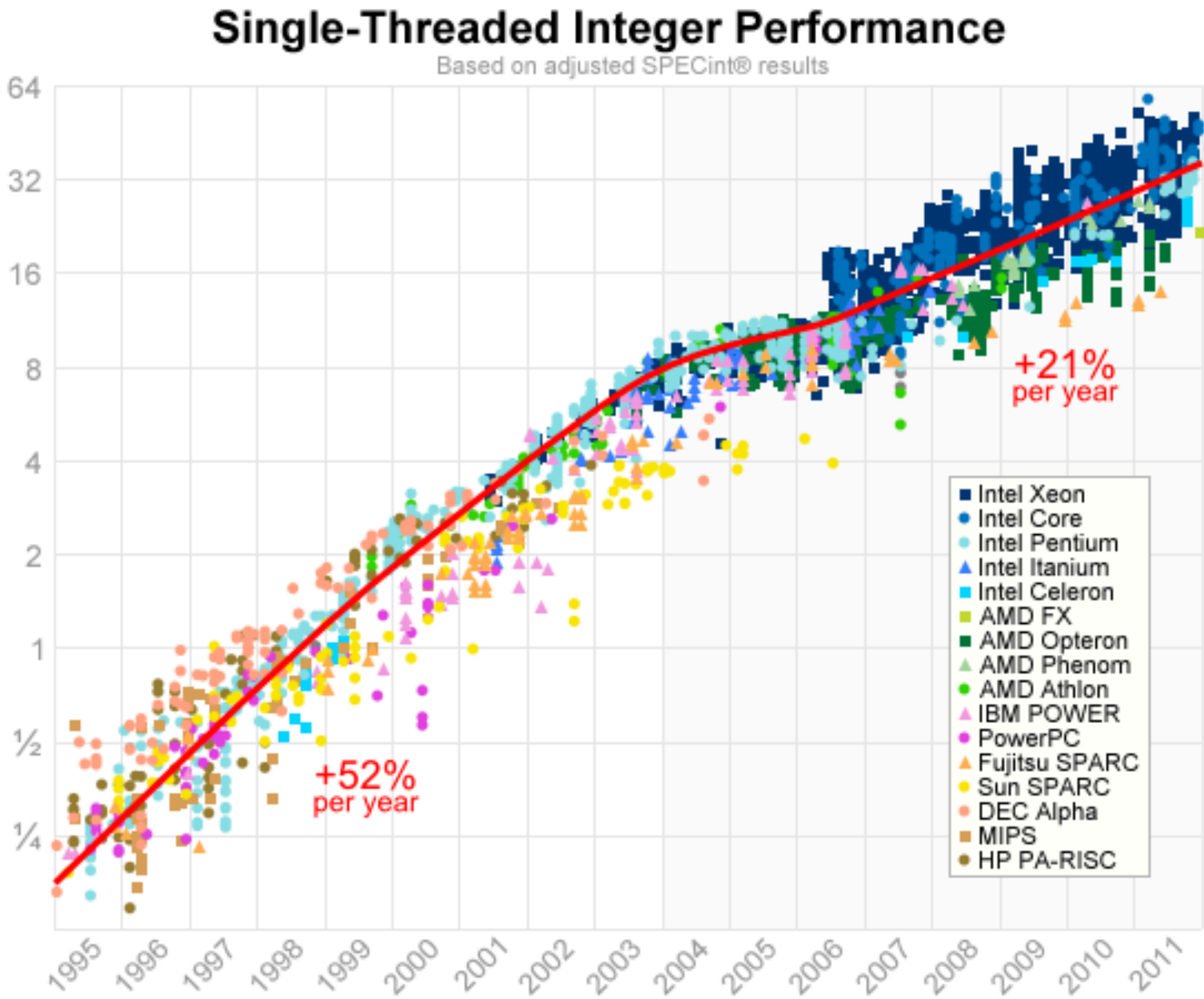
Bertrand LE GAL

IRISA/INRIA laboratories  
D3 department (Architecture), TARAN team  
ENSSAT, University of Rennes, France

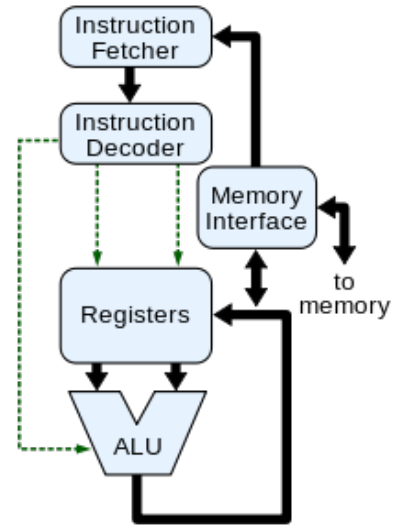
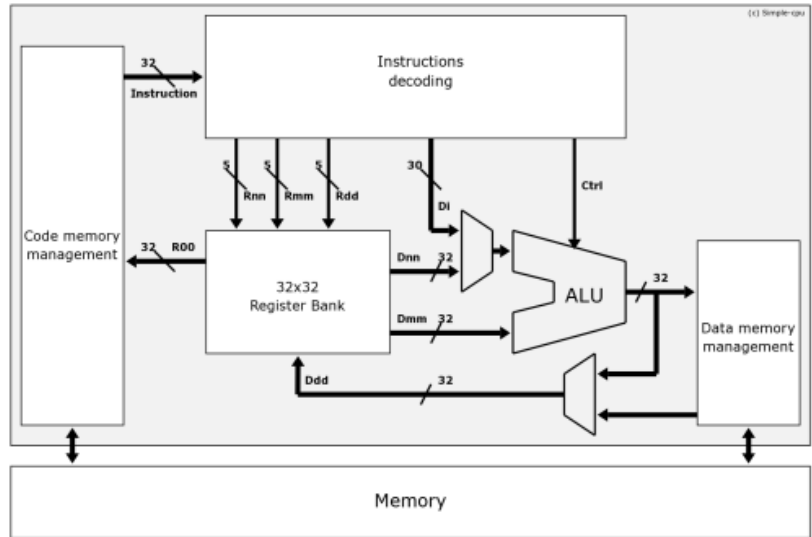
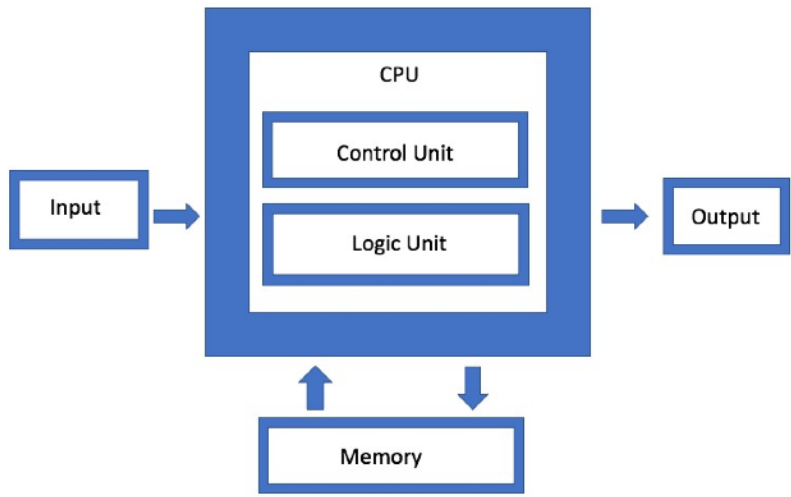
Lessons @Bordeaux INP (ENSEIRB-MATMECA) - 30/10/2023



# Evolution of the processing performance of CPUs

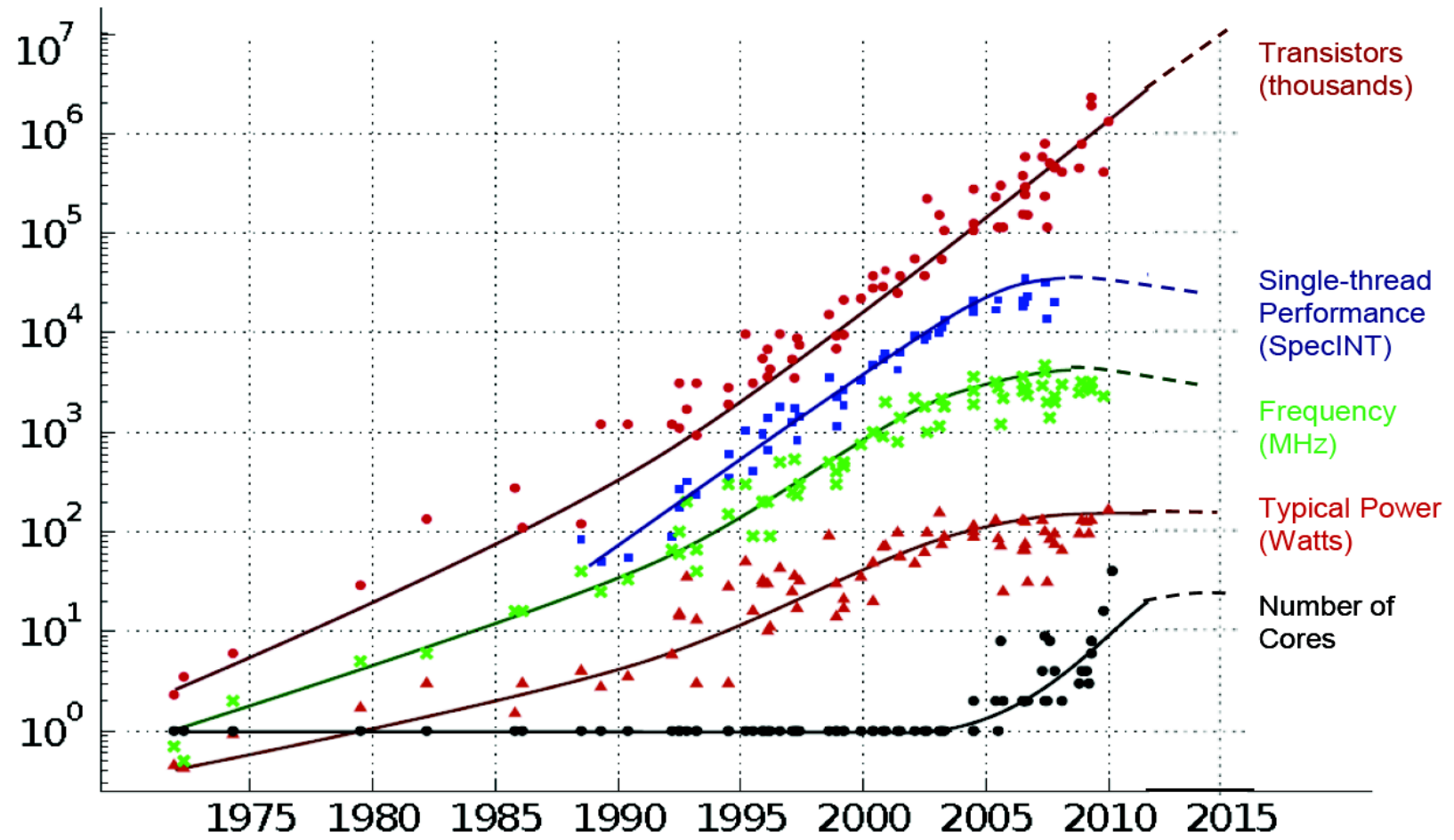


# From microcontrollers to microprocessors



# Introduction au pipelined architectures

## 35 YEARS OF MICROPROCESSOR TREND DATA



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten  
Dotted line extrapolations by C. Moore

# Instruction pipelining in processors

- To be executed a instr. needs to execute: fetch, decode, load, execute and write operations.

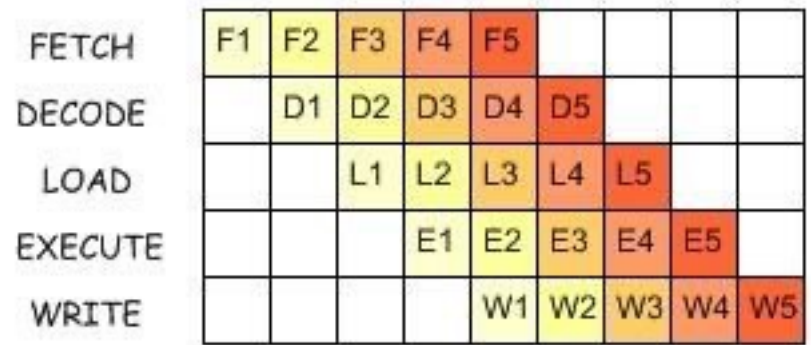
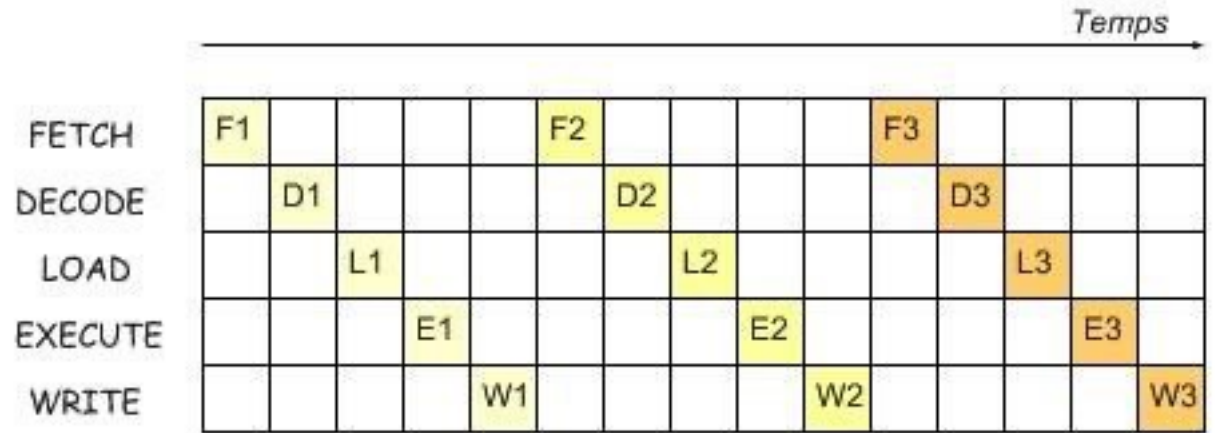
- 1 intr. needs 5 clock cycles

- This sequential task can be split into parallel sub-tasks

- The same operation are executed over diff. instructions

- Performances

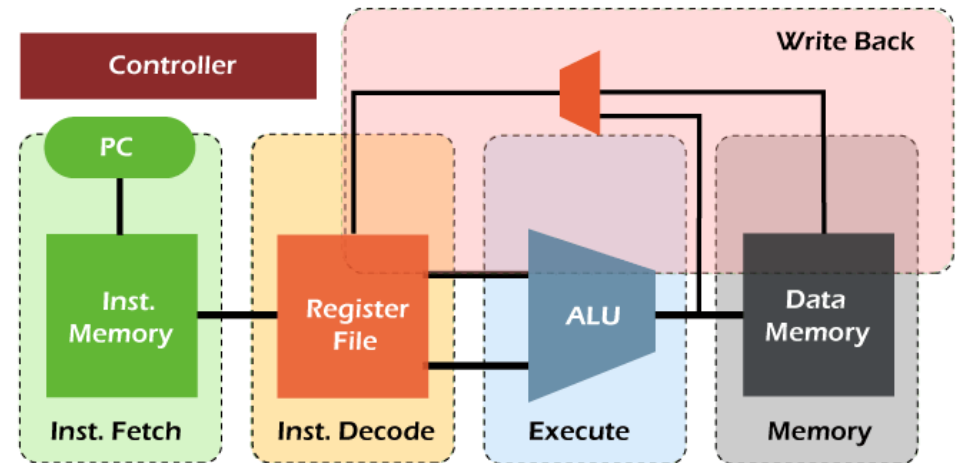
- Reduces silicon critical paths,
- Increase execution parallelism,
- Need hardware to manage pipeline.



# Some characteristics of INTEL processors

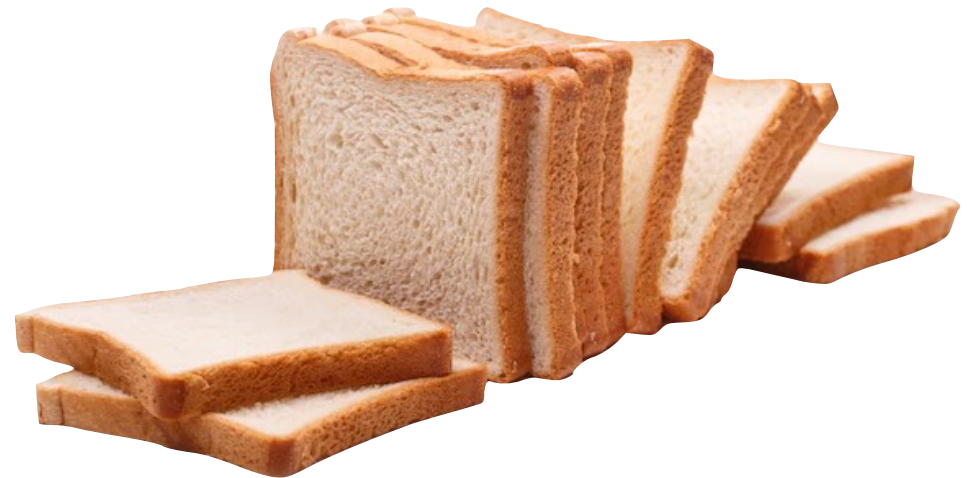
## Number of pipeline stages

- 5 stages - (Pentium, 1993)
- 14 stages - (Pentium III, 1995)
- 31 stages - (Pentium IV, 2006)
- 14~19 stages - (Core-i7, 2016)



## Execution time of x86 instr. (EX)

- Integer addition (1 cycle)
- Float addition (5 cycles)
- Integer multiplication (3 cycles)
- Float multiplication (6~7 cycles)
- Integer division (13~44 cycles)
- Float division (15 cycles)



[https://www.agner.org/optimize/instruction\\_tables.pdf](https://www.agner.org/optimize/instruction_tables.pdf)

[https://en.wikipedia.org/wiki/Comparison\\_of\\_CPU\\_microarchitectures](https://en.wikipedia.org/wiki/Comparison_of_CPU_microarchitectures)

# Execution hazards & data dependencies

● Pipeline is (usually) efficient when it is full of (independent) instructions,

- Not so easy from an algorithmic point of view !

FETCH	F1	F2	F3	F4	F5				
DECODE		D1	D2	D3	D4	D5			
LOAD			L1	L2	L3	L4	L5		
EXECUTE				E1	E2	E3	E4	E5	
WRITE					W1	W2	W3	W4	W5

● Pipeline stalls happened frequently:

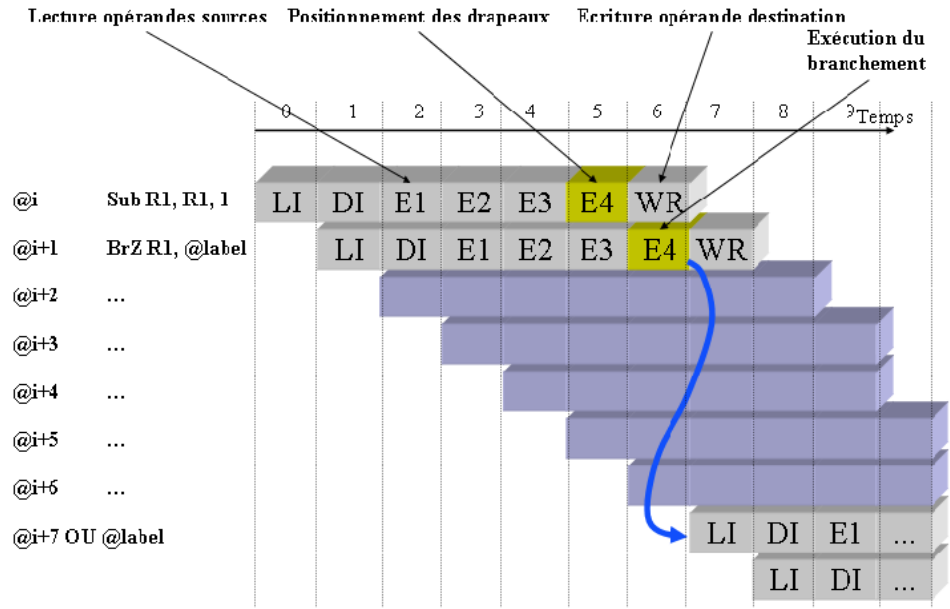
- Data dependency
- Control dependency
- High latency operation
- Memory access (see after)

FETCH	F1	F2	F3	F4	F4	F4	F5				
DECODE		D1	D2	D3	D3	D3	D4	D5			
LOAD			L1	L2	L2	L2	L3	L4	L5		
EXECUTE				E1	⊗	⊗	E2	E3	E4	E5	
WRITE					W1	⊗	⊗	W2	W3	W4	W5

[https://fr.wikipedia.org/wiki/Pipeline\\_\(architecture\\_des\\_processeurs\)](https://fr.wikipedia.org/wiki/Pipeline_(architecture_des_processeurs))

# Pipeline execution and control statements

- Another pipeline usage issue comes from control statements
  - Control statement involve compare and jump instructions,
  - Which instr. should be loaded ???
  - NOP vs branch prediction.



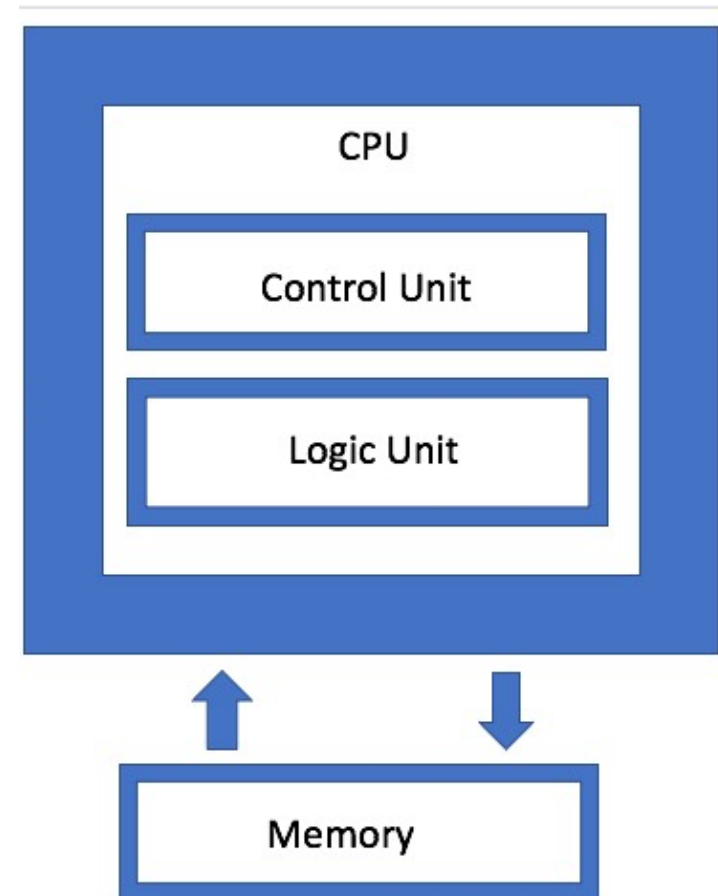
- Branch prediction unit
  - Different approaches,
  - Impacts on performance,
  - Impacts on hardware cost.
- It is possible to avoid parts of them at software level.





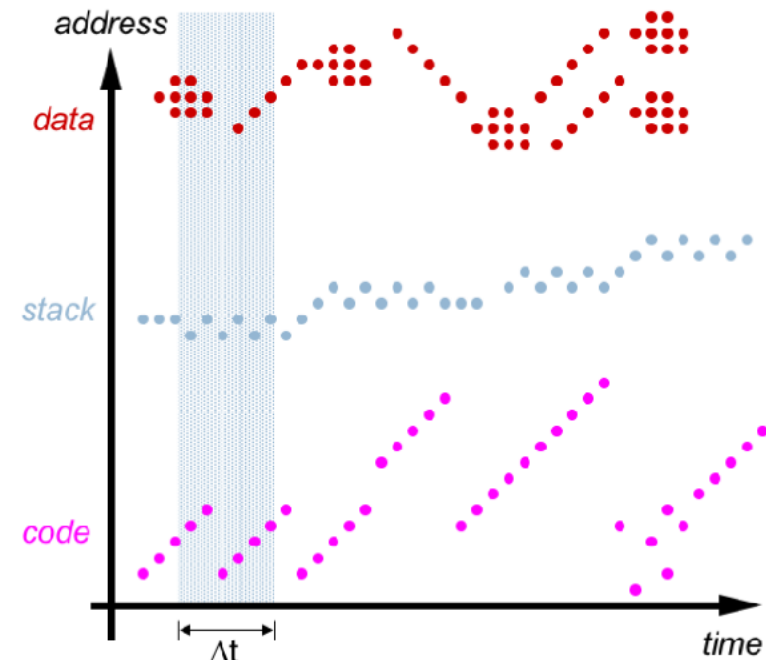
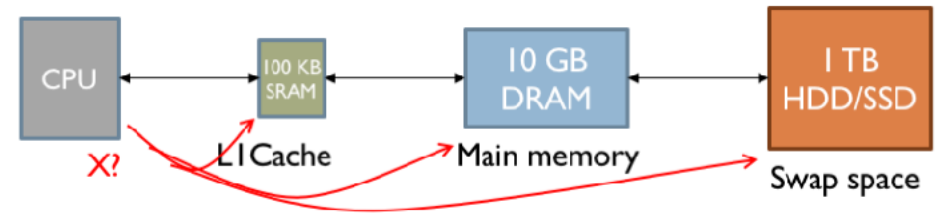
# Introduction to memory hierarchy

- A software program
  - Execute arithmetic computations
  - Load/store values in memory
- Hardware memory characteristics
  - Some GB on data,
  - Works @GHz frequencies,
  - High energy consumption (refresh),
  - Are very slow compared to CPU.
- Between CPU and DDR memory
  - Smaller & faster memories are inserted,
  - They manage data locality efficiently.



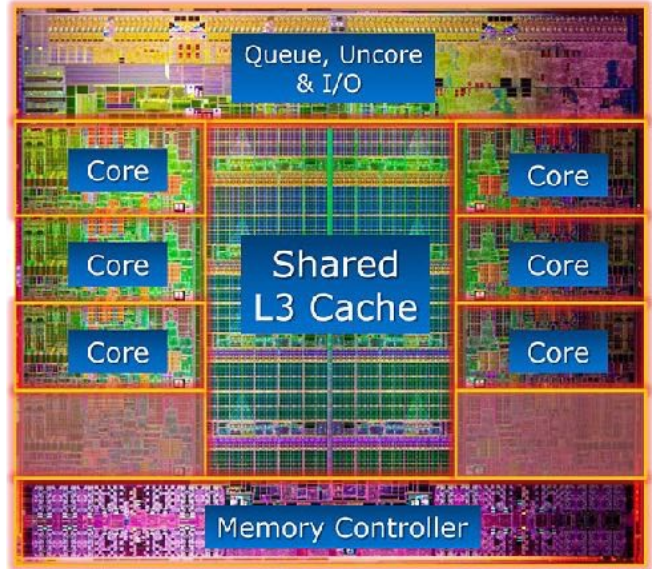
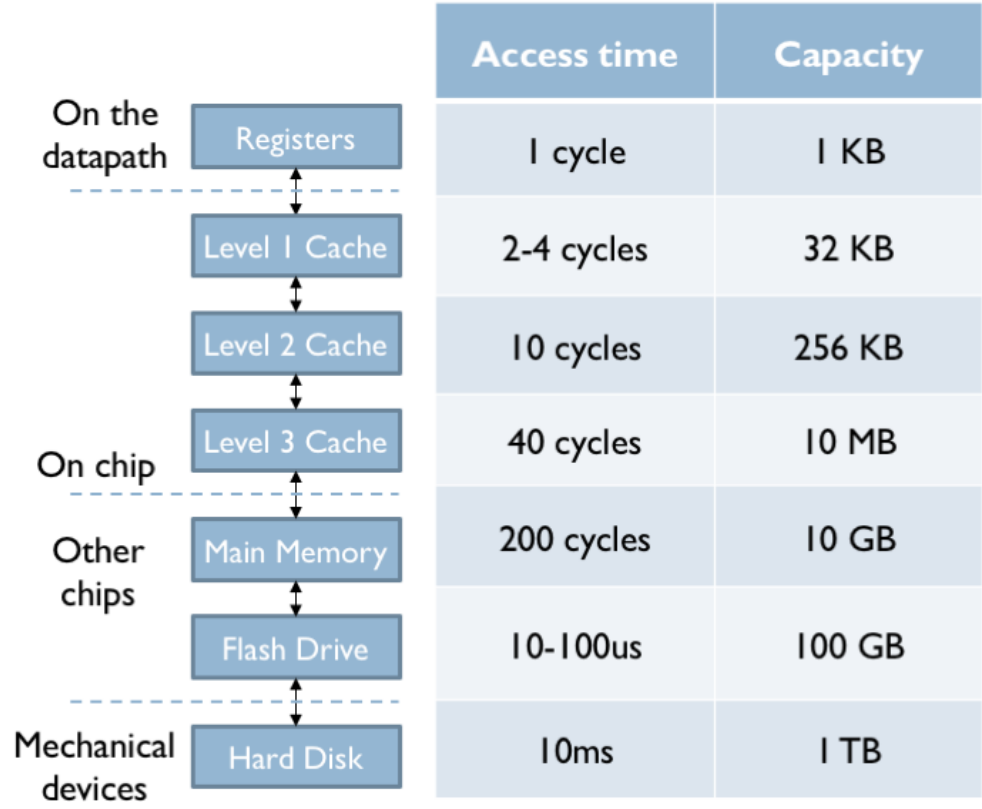
# Memory cache behavior

- Data move (automatically) from memory to registers,
  - Dedicated hardware resources make the job for you,
- Data are stored in cache for future usage (reduce time),
  - Interesting when data locality is high,
- Cache have limited sizes
  - Replace values when caches are full,
- Cache ctrl predicts memory access to anticipate moves.



# Properties of memory levels

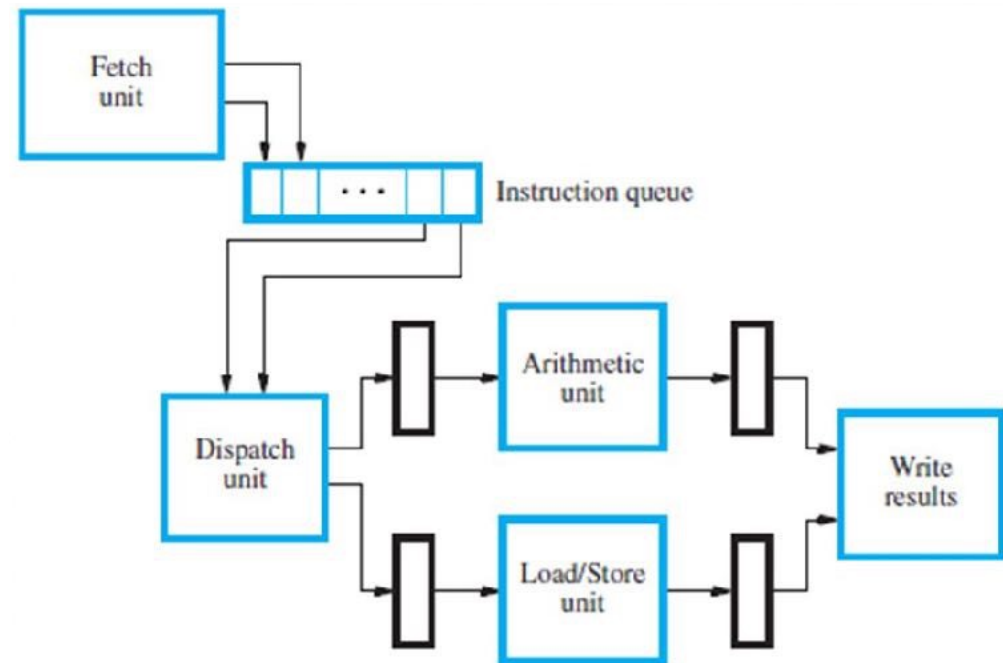
- The memory hierarchy depends on processor targets
  - Large caches increases performances
  - Cache memory are costly,
  - Tradeoff (energy, cost, performance),
- Software designer should take to cache characteristics.



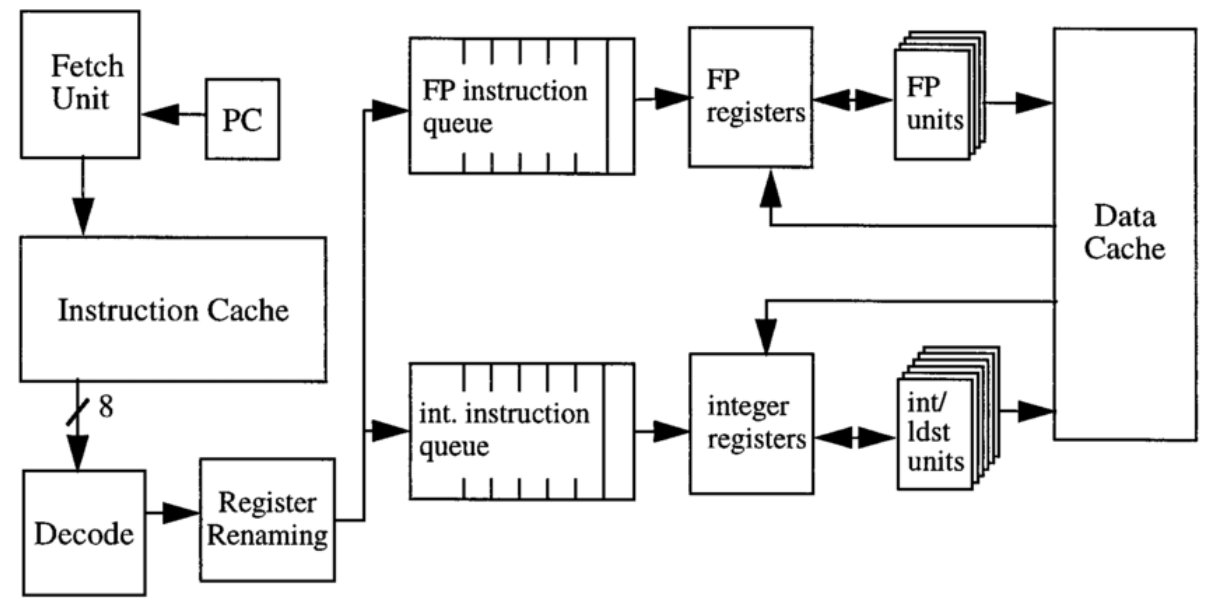
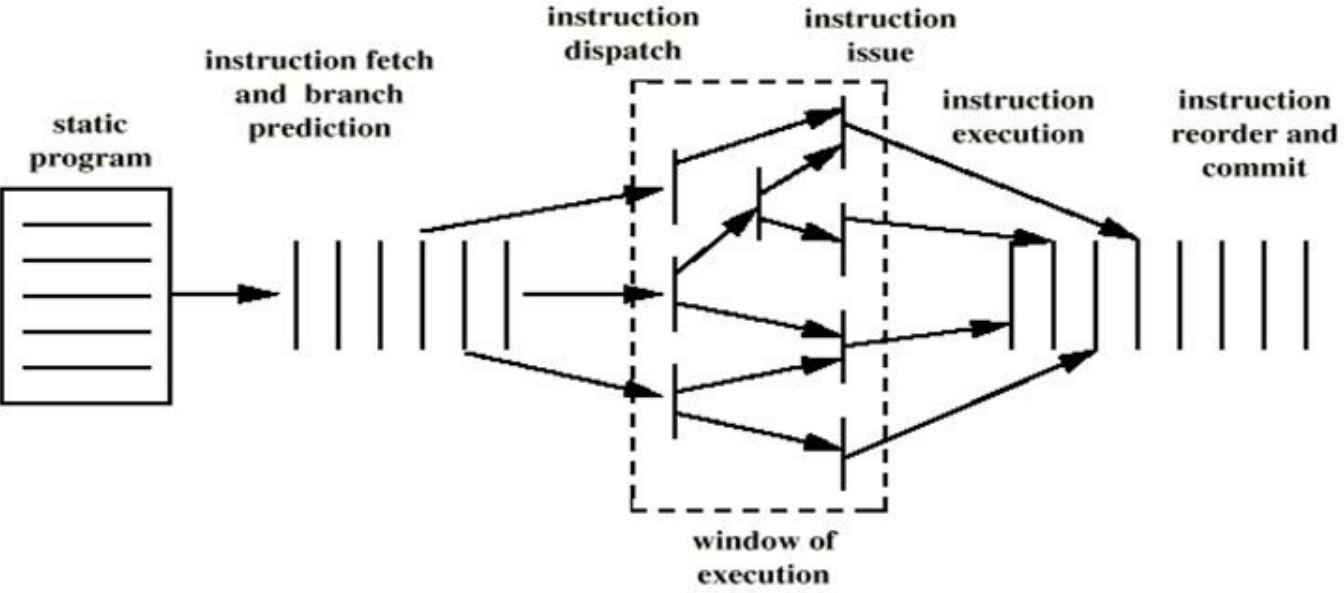
<https://computationstructures.org/lectures/caches/caches.html>

# Introduction to superscalar feature

- Executed instruction
  - Targets an hardware resource,
  - Has a variable execution time.
- Architecture offers
  - A large set of hardware resources,
  - ALU & others resources underuse.
- Superscalar architecture
  - Execute more than one instruction at a time depending on resource availability.
- Architectures can support in-order or out-of-order execution.

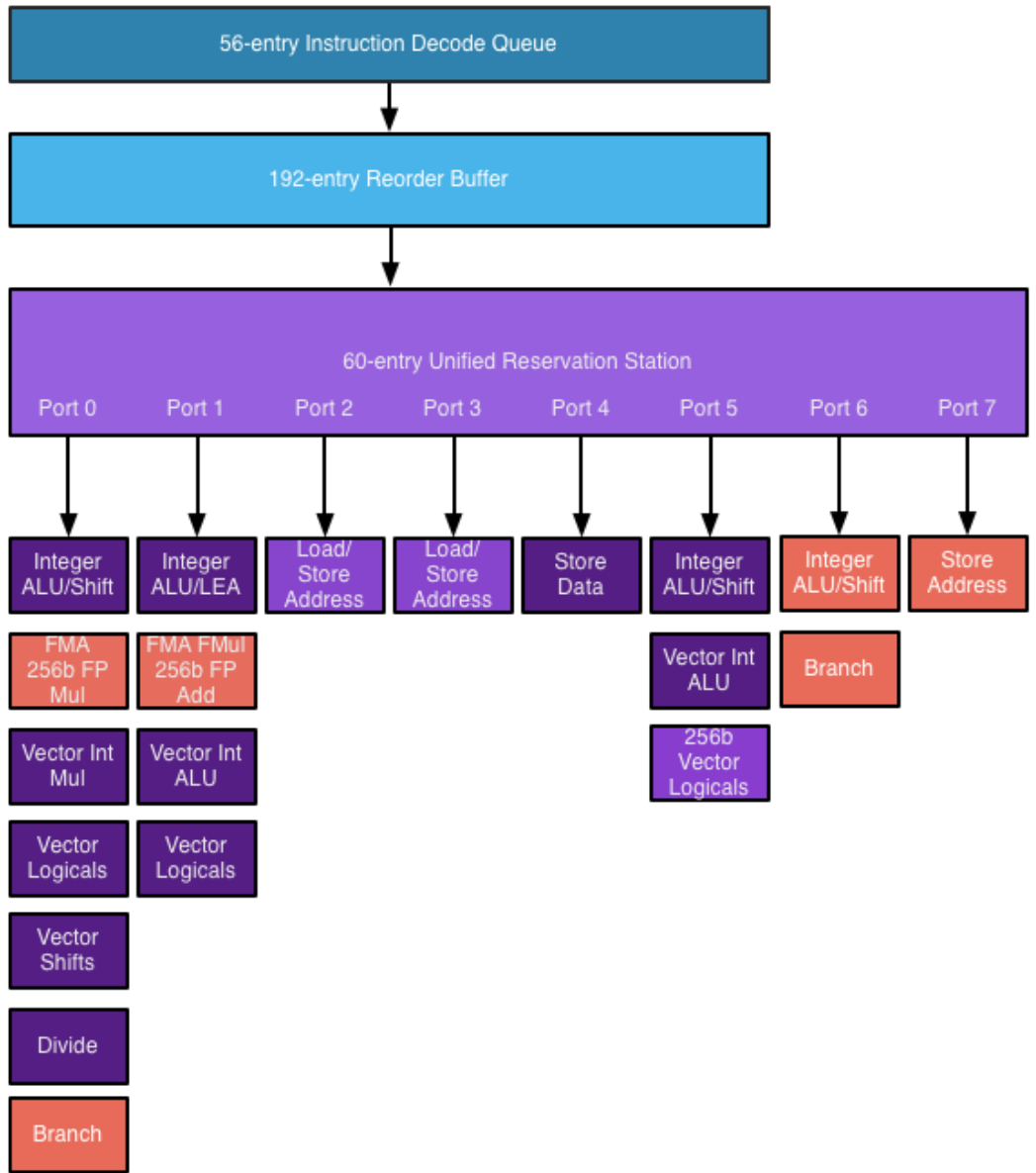


# Behavior of superscalar architectures



# Intel Haswell Execution Engine

- On INTEL architectures
  - Up to 8 instr. can be executed per cc.
  - IPC level grows from 1 to 8 theoretically.
- Units are grouped into ports
  - Asymmetric port capabilities,
  - ex. division and multiplication can't be processed in parallel.
- A processor can execute
  - 8 x clock frequency instructions per second,
  - Not so easy to achieve...
- Software descriptions can help...



# Source code and compiler impacts on perf.

## Off-the-shelf processors

- General Purpose Processor can execute word processing or games,
- (very) complex architectures to be as efficient as possible,

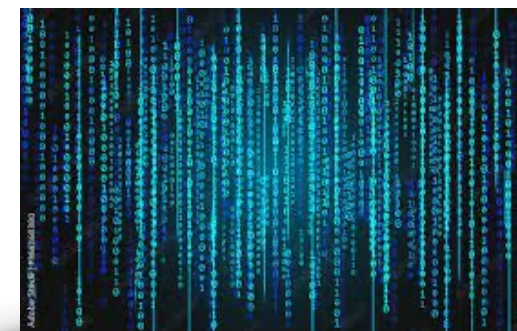
## Processor cannot do everything

- Software designer can help (code writing style, algo choice),
- Software compiler can help too !

## Between two software programs executing the same behavior,

- Speed up from 2 to 100 !

```
10  $(("word-list-out").html(" "));
11  var b = count_array_gen();
12  parseInt(licenia(),unique);
13  var c = array_bez_povt(), a = " ", d = parseInt($(".limit_val").
14  function("LIMIT_total:" + d);
15  function("rand:" + f);
16  d < f && (f = d, function("check rand\u00f3\u00f3rand: " + f + "
17  var h = [], d = d - f, e;
18  if (0 < c.length) {
19    for (var g = 0;g < c.length;g++) {
20      a = indexOf_array(b, c[g]), -1 < e && b.splice(e, 1);
21    }
22    for (g = 0;g < c.length;g++) {
23      b.unshift((use_class:"parameter", word:c[g]));
24    }
25  }
26  e = indexOf_keyword(b, " ");
27  -1 < e && b.splice(e, 1);
28  e = indexOf_keyword(h, void 0);
29  -1 < e && b.splice(e, 1);
30  e = indexOf_keyword(b, "");
31  -1 < e && b.splice(e, 1);
32  for (c = 0;c < d && c < b.length;c++) {
33    a += b[c].word + " ", h.push(b[c].word), "parameter" = b[c].
```



# Conclusion

- ◎ CPU architectures are complex !
  - Features depend on CPU vendors,
  - Pro/Con trade-offs
- ◎ Algorithms impacts on performance level,
  - Source code writing too !
  - Don't forget the compiler...
- ◎ To obtain an efficient application, YOU should take care about:
  - The processor features,
  - Your code writing style,
  - The compiler optimizations.

