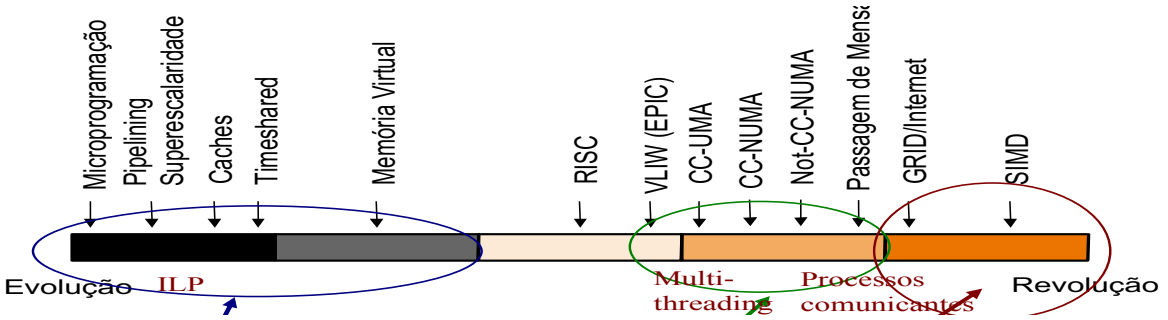
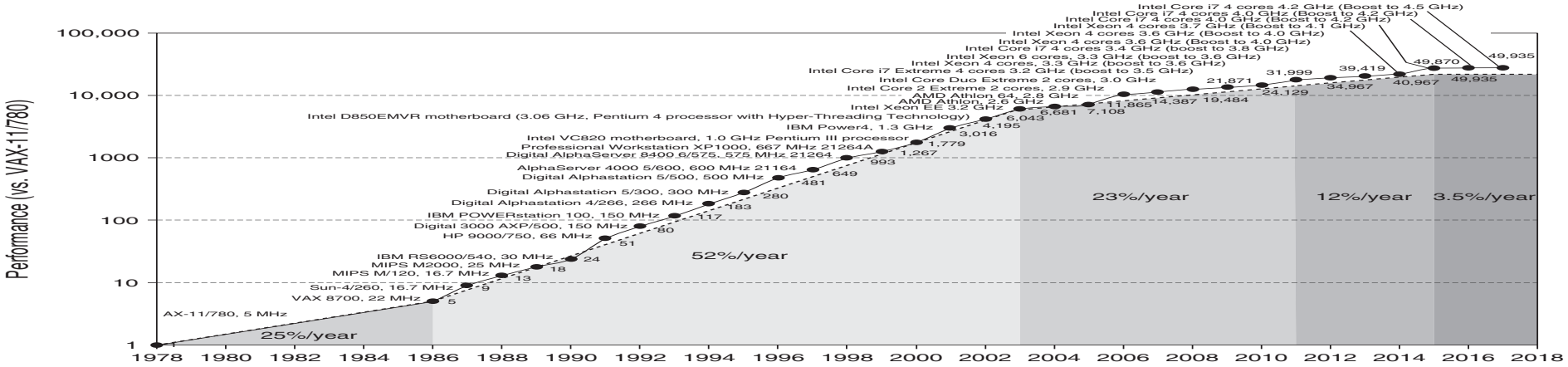
Mestrado em Engenharia Informática

2024/25

J. L. Sobral

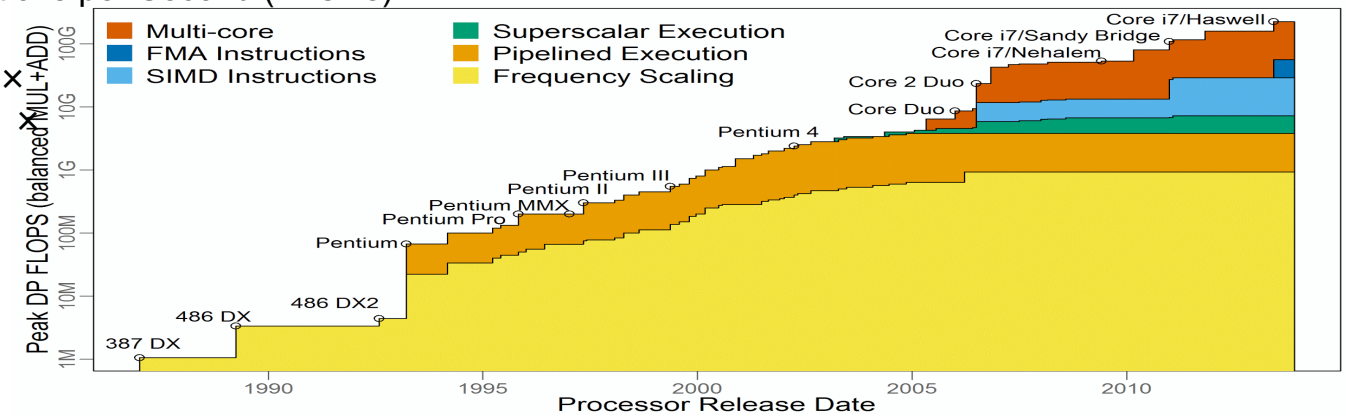
[Instruction Level] Parallelism





Peak FLoating-Point OPerations per Second (FLOPs) =

Number of cores ×
FLOP per instruction ×
Instructions per cycle ×
Cycles per second



Texe = #I x
1/#cores × (ideal case)
1/SIMDlen × (" ")
CPI × (CPI = 1/IPC)
Tcc (Tcc = 1/freq)

(Task/Thread-Level Parallelism)
(Vector: SIMD and FMA)
(Instruction Level Parallelism)
(Frequency)

– Task dependencies and program *task dependency graph*

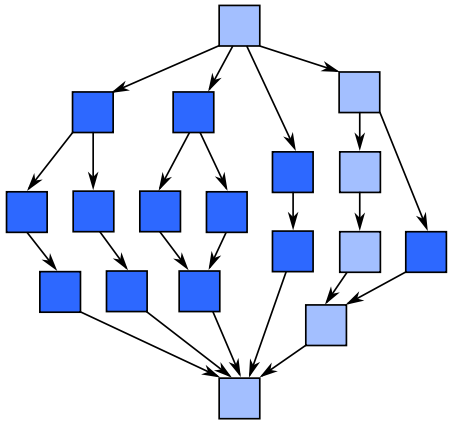


FIGURE 2.8

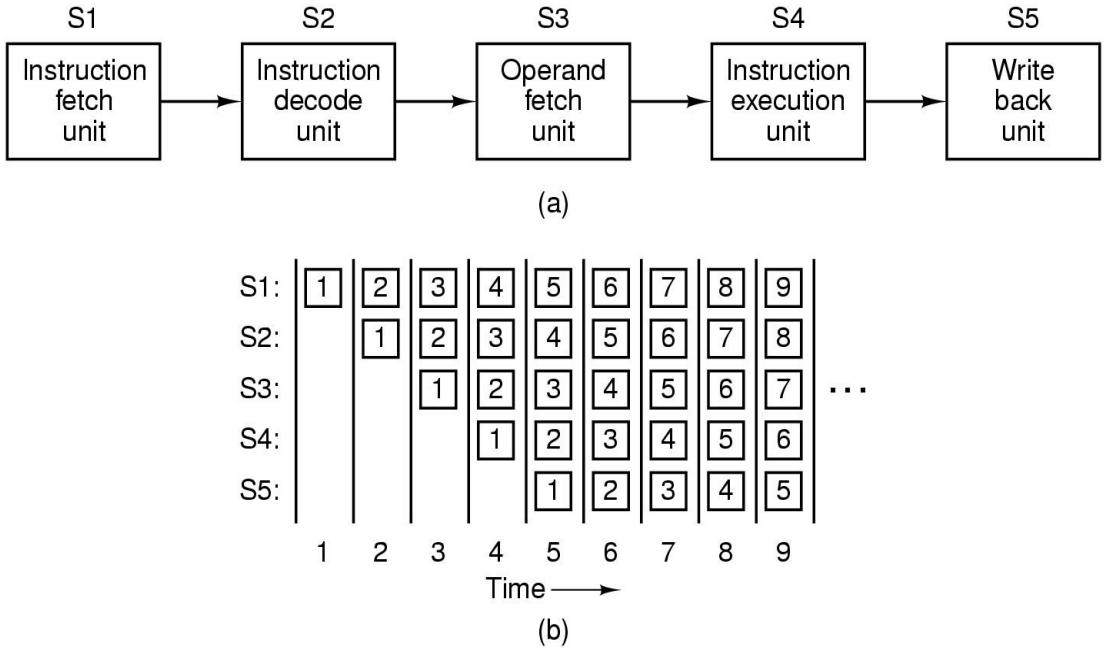
Work and span. Arrows denote dependencies between tasks. Work is the total amount of computation, while span is given by the critical path. In this example, if each task takes unit time, the work is 18 and the span is 6.

– Scheduling

- Assigns tasks to available resources considering all dependencies
- Can be static (compile-time) or dynamic (run-time)

Processor pipeline

(with 5 phases)



Ideally:

- **CPI = 1**

Problems:

- Data dependencies
  - among instructions due to shared data in registers or in memory
  - Solutions (on a pipelined processor): shortcuts / pipeline **stalls** (e.g., do nothing)
- Control dependencies
  - conditional jumps/indirect jumps or calls
  - Solutions (to reduce the impact): predict the next IP (e.g., assume the jump as taken)

The main goal is to reduce the CPI:

Texe = #I * CPI * Tcc

- Impact of dependencies on CPI (on a pipelined processor)
  - Pipeline CPI =

    Ideal pipeline CPI +

    Structural stalls + → due to resource conflicts in the simultaneous execution of overlapped instructions

    Data hazard stalls + → when an instruction depends on the results of a previous instruction

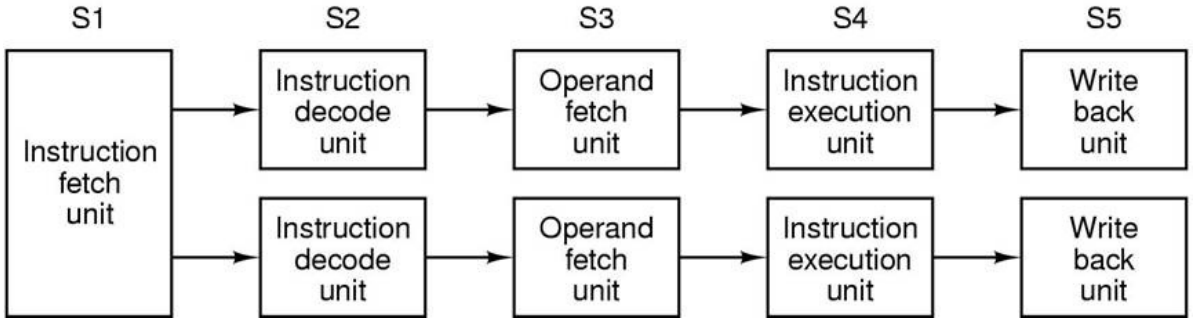
    Control stalls → due to the pipelining of branches and other instructions that modify the IP

- Parallelism within a basic block is limited
  - Typical size of basic block = 3-6 instructions
  - Must optimise across branches

Example of a 2-Way superscalar architecture

Ideally :

- **CPI = 0.5 (two-way, when combined with pipeline**



Problems:

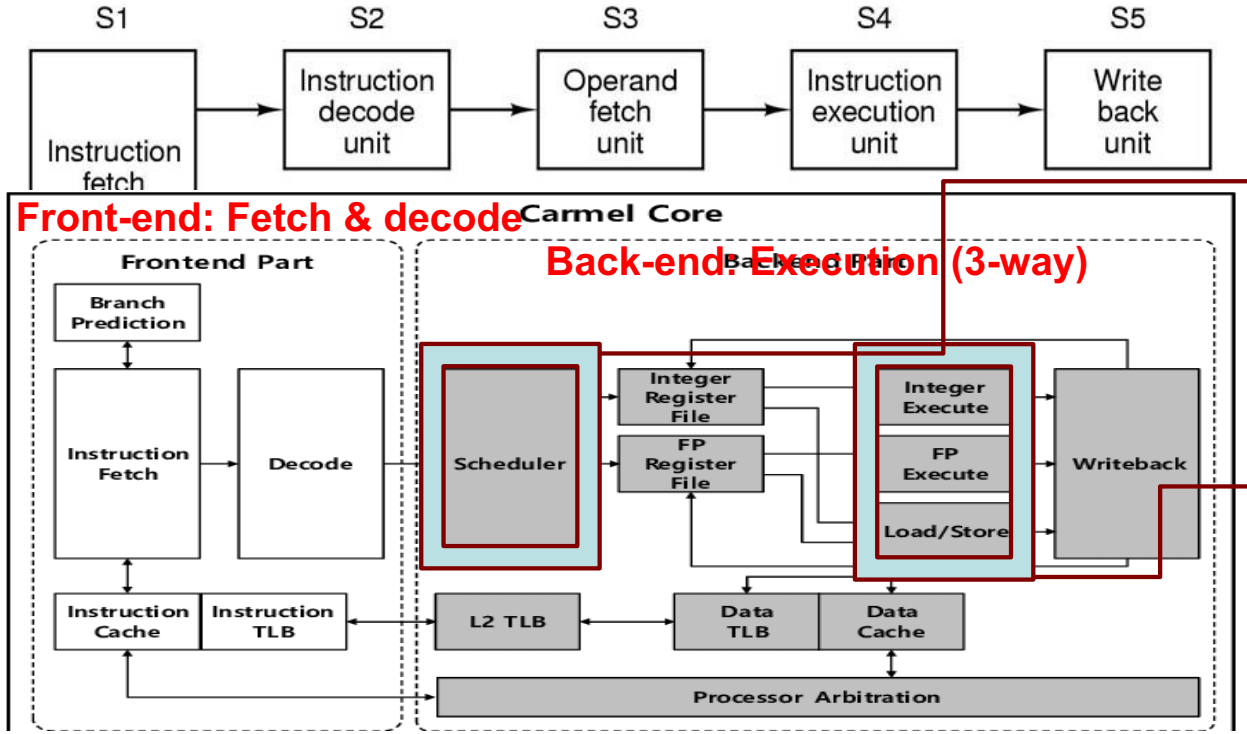
Higher penalisation with dependencies (data & control)

Requires more sophisticated workarounds: out-of-order execution (& scheduling)

Real example of a superscalar architecture – 3-way

Requires an instruction scheduling init

Ideal CPI = 0.33 (3-way)



The scheduler:
- Selects (max) 3 instructions per cycle to start execution
- Out-of-order execution

3-way execution
- 1 int (*int add*, etc)
- 1 FP (*float add*, etc)
- 1 Memory access (*Load* or *Store)*

[Instruction Level] Parallelism

Example 3 (Evolution skylake-> sunnycove ->raptor cove)

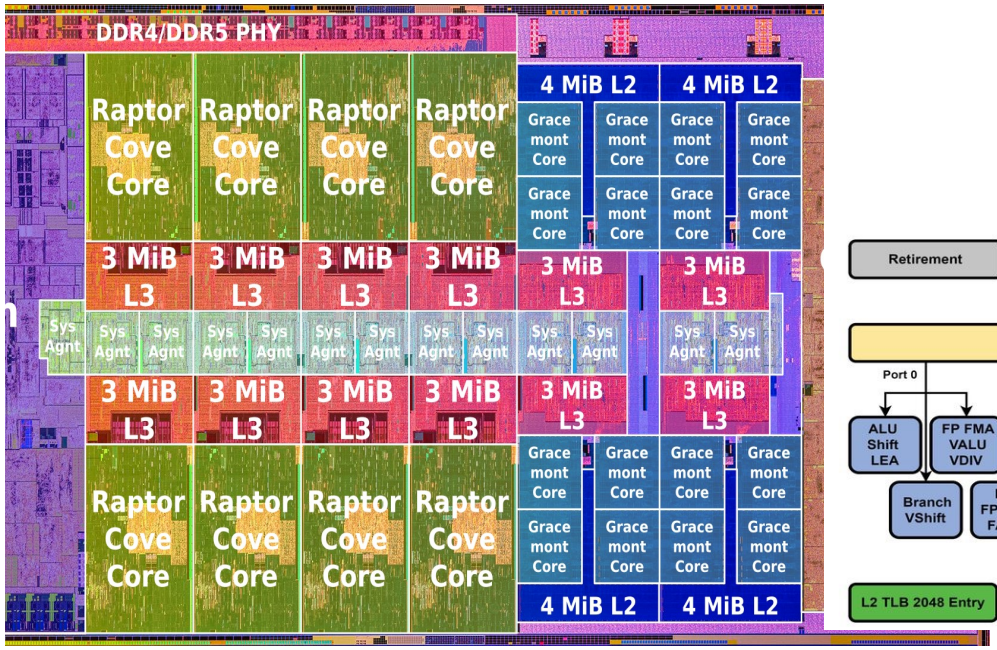
Raptor Cove (2022-Gen13)

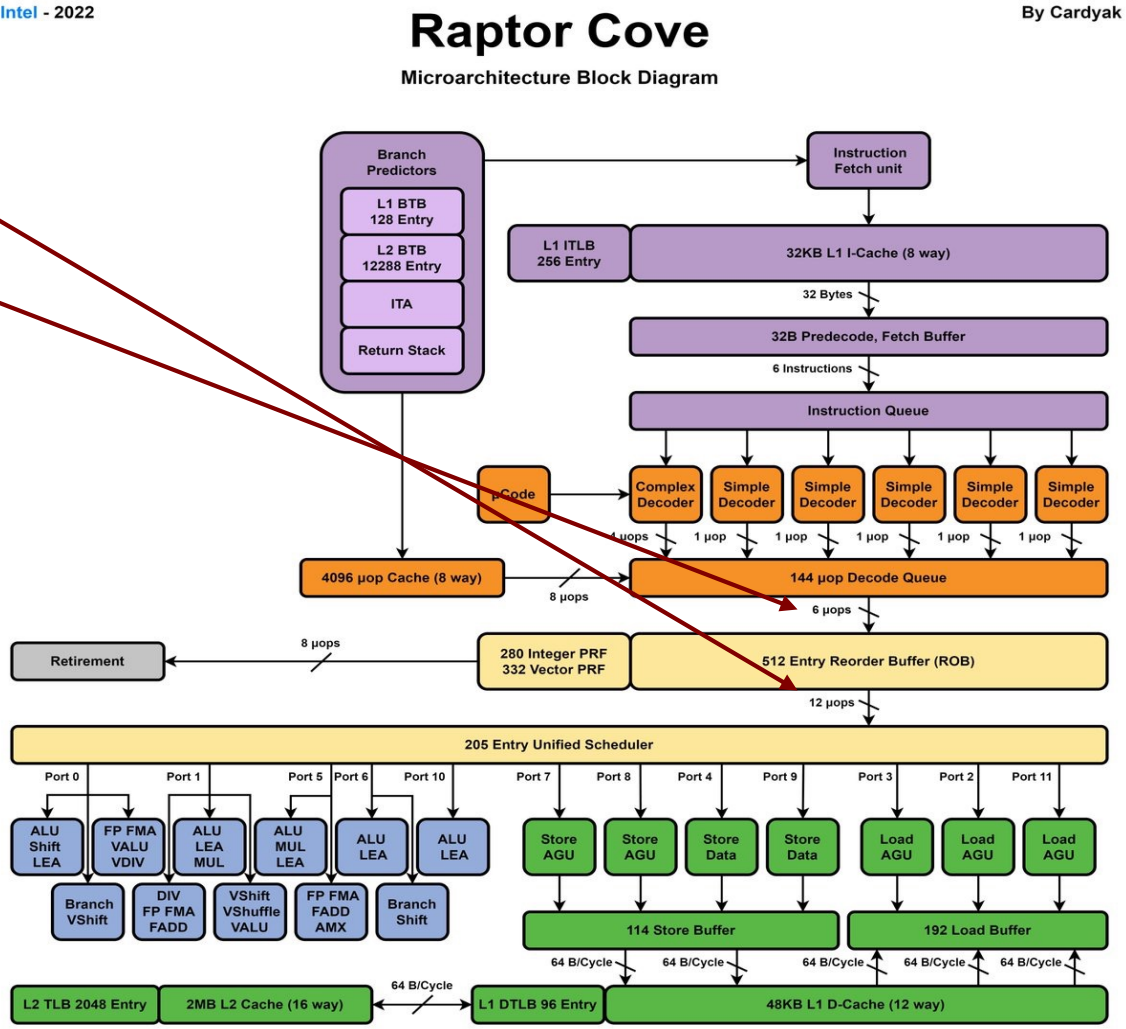
Front-end: 6-way / Backend 12-way

(Intel: ~20% less CPI)

Intel i9 13900K

- 8 Raptor Cove cores (3.0 a 5.8 GHz)
- 16 Gracemount cores (2.2 a 4.3 GHz)





Dependencies: a closer look

- Control dependencies
  - The execution of task B is conditioned by task A (e.g., *if A them B*)
  - ILP solutions: prediction + speculation, predication
- Data dependencies
  - **RAW** (Read After Write) – task j reads data produced by previous task i
    - Example:

```
i: mov 0(%r9), %r10
j: add %r10, %r11
```

  - **WAW** (Write After Write) – task j writes to the same data as task i

```
i: mov 0(%r9), %r10
j: add %r11, %r12, %r10
```

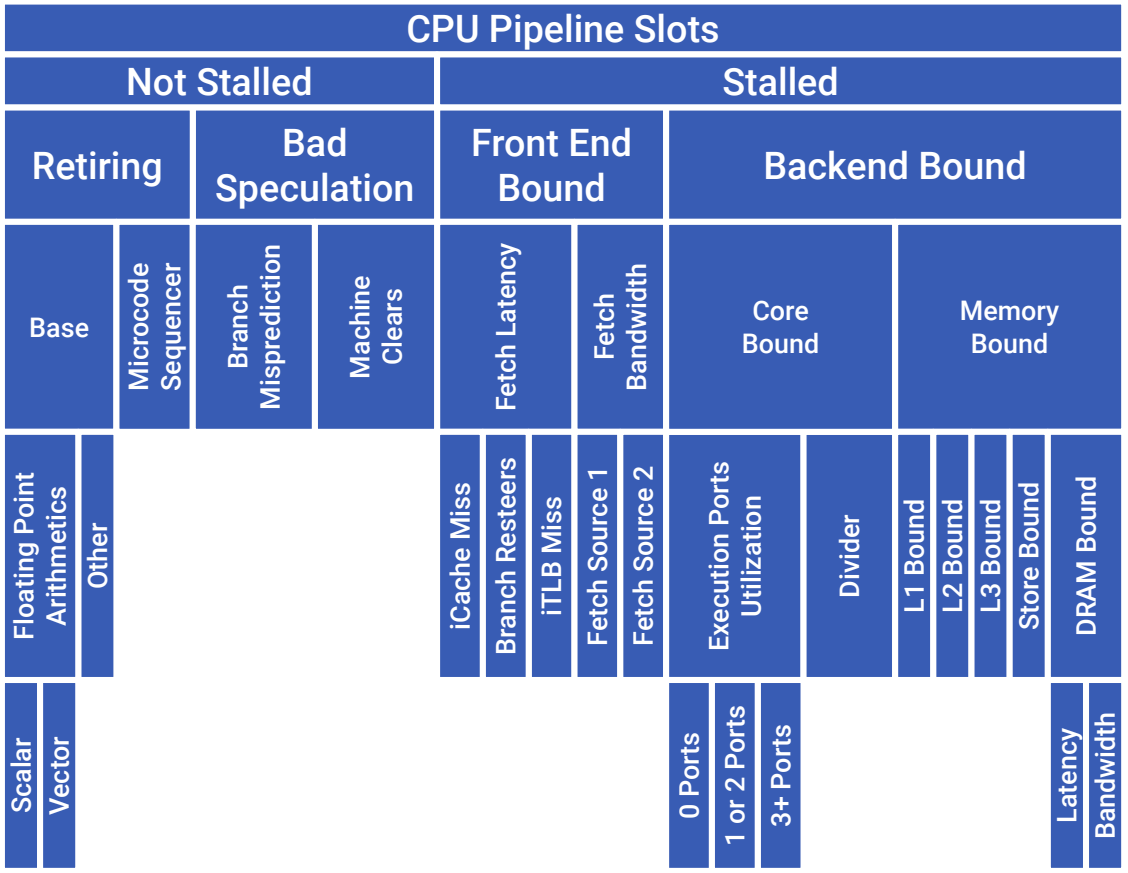
  - **WAR** (Write After Read) – task j writes to data read by previous i

```
i: mov 0(%r9), %r10
j: add %r11, %r12, %r9
```

False dependencies / can be removed [with renaming]

Top-Down Microarchitecture Analysis

- Front End Bound
  - Code Duplication
  - Code Layout (Locality)
  - Frequent Branching
  - Unnecessary Work
- Back End Bound
  - Core Bound
    - Data Dependencies
    - Divisions and Special Functions
  - Memory Bound
    - False Sharing
    - Remote Memory Accesses
    - Scattered Memory Accesses
    - Excessive Memory Accesses



Ahmad Yasin, "A Top-Down method for performance analysis and counters architecture," *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Monterey, CA, 2014, pp. 35-44, doi: 10.1109/ISPASS.2014.6844459.

Intel CPU Metrics Reference