UT 2. Pipelined Computers

# Tema 2.5 Multiple instruction issue

A. Doménech, J. Duato, P. López, V. Lorente,
A. Pérez, S. Petit, J.C. Ruiz, S. Sáez, J. Sahuquillo

Department of Computer Engineering
Universitat Politècnica de València


## Contents

1 Introduction

2 Superscalar processors

3 Multithreaded processors

4 VLIW processors

5 Superpipelined processors

## Bibliography

John L. Hennessy and David A. Patterson.
*Computer Architecture, Fifth Edition: A Quantitative Approach*.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5 edition, 2012.

## Contents

1 Introduction

2 Superscalar processors

3 Multithreaded processors

4 VLIW processors

5 Superpipelined processors

The execution time of a program is defined in terms of :

$$T_e = I \times CPI \times T$$

Pipelining + dynamic instruction scheduling $\rightarrow$ CPI $\approx 1$

Is it possible to further reduce $T_e$?

1. Reduce the average number of clock cycles per instruction *CPI*.
   $\rightarrow$ Increases the number of issued instructions per clock cycle.
   $\Rightarrow$ **Superscalar processors**.
2. Reduce the number of executed instructions *I*
   $\rightarrow$ Increase the work performed by each instruction.
   $\Rightarrow$ **VLIW processors**.
3. Reduce the clock cycle *T*.
   $\rightarrow$ Pipeline the instruction cycle in more stages (longer pipeline).
   $\Rightarrow$ **Superpipelined processors**.

## ILP vs. TLP

ILP: *Instruction Level Parallelism*
TLP: *Thread Level Parallelism*

The ILP exploited by the machine is limited by:

- The ILP of the program.
- The machine hardware constraints.

$\rightarrow$ This lowers the utilization of the functional units,
$\rightarrow$ and reduces the potential performance.

Alternative: to support the execution of several threads $\rightarrow$ exploit TLP

- Replicating machine resources: **multicore processors, multiprocessors**
- Sharing machine resources: **multithreaded processors**.

## Contents

1 Introduction

2 Superscalar processors

3 Multithreaded processors

4 VLIW processors

5 Superpipelined processors

## Sperscalar concept

$m$ instructions can be issued at each clock cycle $\rightarrow$ CPI=$\frac{1}{m}$
$\rightarrow$ Instructions Per Cycle (IPC)=$m$

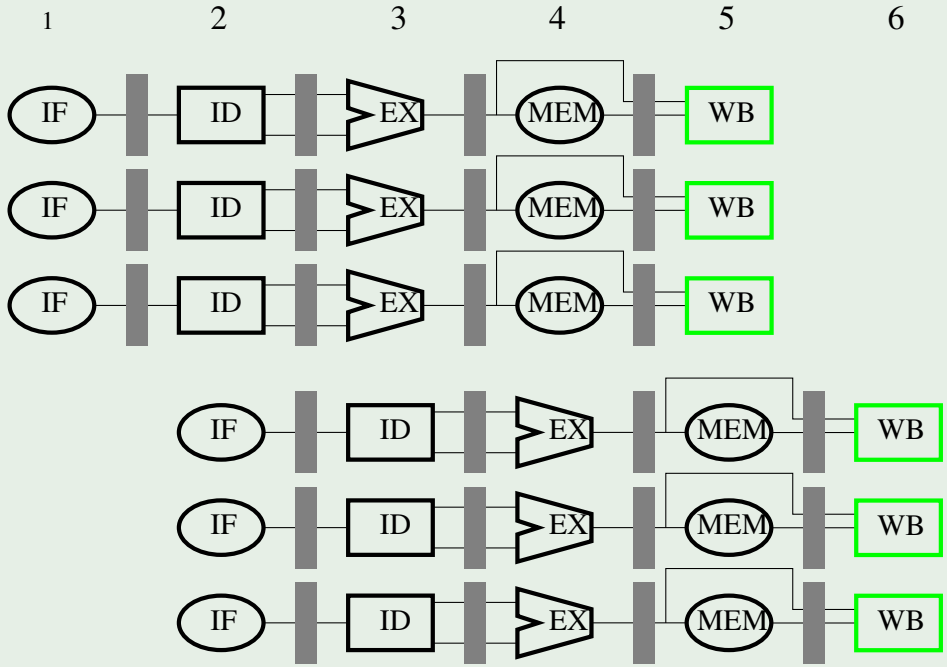
$m$: number of ways in the superscalar computer

### Comparison with a pipelined processor

- $T_{\text{pipelined}} = I \times CPI \times T$
- $T_{\text{superescalar}} = I' \times CPI' \times T' = I \times \frac{CPI}{m} \times T$

$\Rightarrow$ Performance increase by a factor of $m$.

## Example for $m = 2$:



## Hardware requirements:

- Simultaneous access to several instructions (IF) and simultaneous access to multiple data in memory
  - Cache memory must supply $m$ words per clock cycle $\rightarrow$ **wider** cache or a cache built out of **multiple modules**.
- Multiple instructions decoding (I)
  - $m$ instructions decoder.
  - Checks dependencies among the $m$ fetched instructions, and among these and in-flight instructions.

$\rightarrow$

  - More complex/faster decoder.
  - Multiple decoding cycles

## Hardware requirements: (cont.)

- Simultaneous read access to several operands (I)
  - Multiple read ports in the register file.
  - Multiple read ports in the ROB.
- Simultaneous execution of several instructions (EX)
  - Different functional units working in parallel are required.
- Several instructions at WB stage
  - Multiple common data buses.
  - Multiple write ports in the ROB.
- Committing several instructions at the same time (C)
  - Several instructions (from the ROB head) should be able to *Commit* simultaneously.
  - Multiple write ports in the register file.

## Consequences:

- Hazard likelihood increases $\rightarrow$ In addition to hazards among in-flight instructions, hazards may also occur among the *m* instructions issued at the same cycle:

**Example (for a simple instruction unit with 5 stages):**

```
inst R1,R2,R3    IF ID EX M  WB
inst R13,R4,R5   IF ID EX M  WB
inst R2,R1,R3       IF ID EX M  WB
inst R2,R1,R2       IF ID ID EX M  WB
```

## Consequences: (cont.)

- Increases the penalty associated to hazards $\rightarrow$ A single stall prevents the issue of *m* instructions:

**Example (for a simple instruction unit with 5 stages):**

```
LD R1,M(R2)       IF ID EX M  WB
inst R12,R3,R4    IF ID EX M  WB
inst R4,R1,R3        IF ID ID EX M  WB
inst R5,R6,R7        IF ID ID EX M  WB
```

$\rightarrow$ Higher relevance of *ILP* techniques: dynamic instruction scheduling, branch prediction, speculation, ...

## Consequences: (cont.)

- Additional problem with branches (1):
  The branch target address may be misaligned in the cache → the IF stage does not provide *m* valid instructions after the branch:

**Example (for a simple instruction unit with 5 stages):**

```
inst          IF  ID  EX  M   WB
BEQZ R1,L     IF  ID  EX  M   WB
L-4:              IF  <invalid>
L:                IF  ID  EX  M   WB
```

## Consequences: (cont.)

- Additional problem with branches (2):
  The branch instruction may not be the last one in the group of $m$ instructions $\rightarrow$ the rest of instructions must be discarded:

**Example (for a simple instruction unit with 5 stages):**

```
BEQZ R1, L   IF ID EX M  WB
inst         IF <invalid>
L-4:            IF <invalid>
L:              IF ID EX M  WB
```

- The instruction set of the original pipelined processor is not modified $\rightarrow$ They provide **binary compatibility** with the original pipelined processor.

## Non-uniform/Constrained superscalar processors

$\Rightarrow$ Problem: High hardware cost
Solution: constraints are imposed to the type of instructions that can be simultaneously executed $\rightarrow$ **Non-uniform superscalar processor**

Consequences:

- Appearance of structural hazards
- The compiler can help by grouping structural hazard-free instructions
- Binary compatibility may not be so efficient.

## Non-uniform/Constrained superscalar processors (cont.)

### Example

2-wide superscalar processor, which can combine:

- 2 integer arithmetic instructions.
- 1 integer arithmetic instruction + 1 floating point instruction.
- 1 load/store instruction + 1 integer/floating-point arithmetic instruction.
- 1 branch instruction + 1 load/store instruction or 1 integer/floating-point instruction

$\rightarrow$ only one instruction accesses to memory,
$\rightarrow$ only the integer register file needs to provide dual port access.
$\rightarrow$ it is not necessary to replicate all the functional units

## Limits of ILP

The ILP techniques improve performance with the advantage of being transparent to the programmer.
How much ILP can be exploited? Important for properly dimension hardware and compiler design.
Three case studies:

- Ideal processor.
- Real processor.
- Commercial processor: Pentium 4.

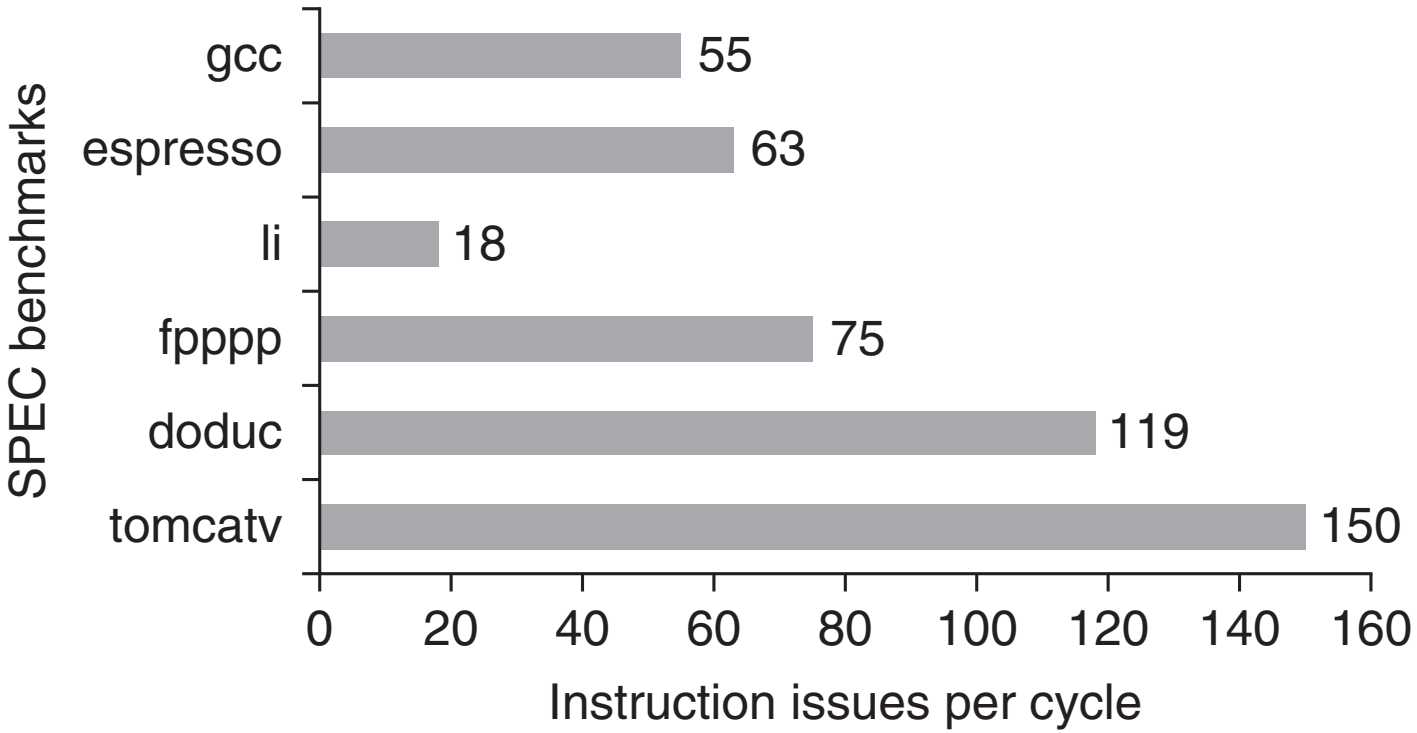
## Limits of ILP: ideal processor

- Unbounded number of instruction issues per cycle.
- Unbounded register renaming $\rightarrow$ no WAR nor WAW hazards.
- Perfect branch predictor.
- Perfect memory disambiguation.
- Perfect cache (100% hit rate) $\rightarrow$ 1-cycle memory access time.
- 1-cycle latency functional units.

$\rightarrow$ the limit is imposed by true data dependencies.

## Limits of ILP: ideal processor (cont.)

Average number of instruction issues per cycle:




IPC:

- Integer (gcc, espresso, li): 18 to 63
- Floating Point (fpppp, doduc, tomcatv): 75 to 150

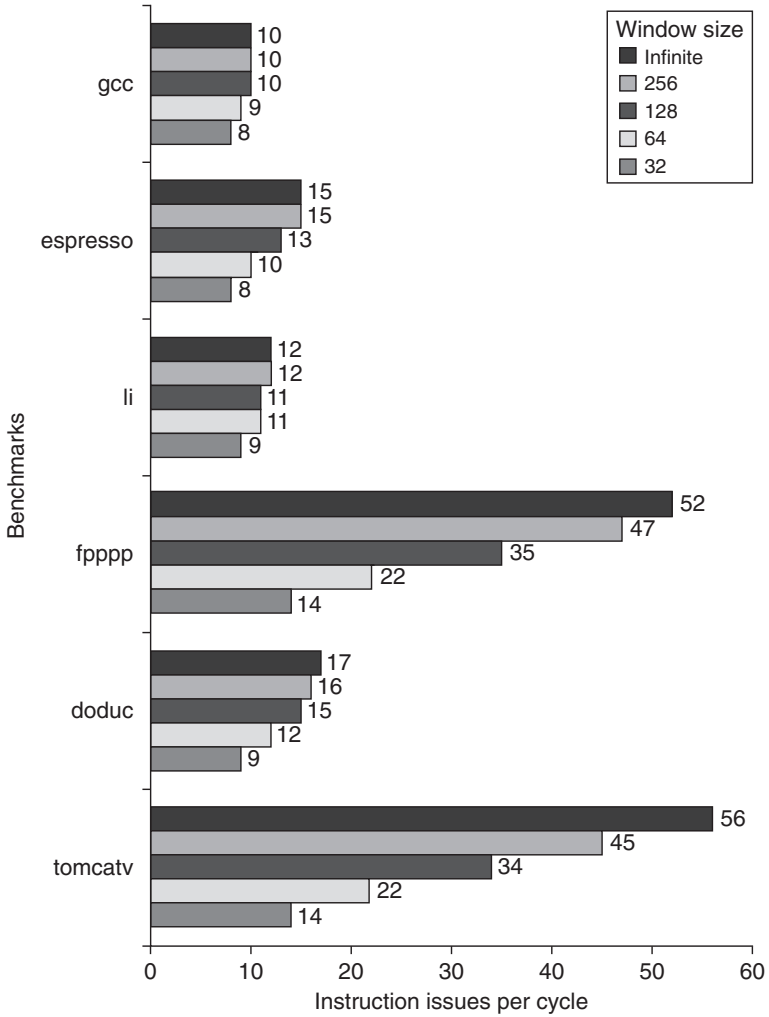
## Limits of ILP: real processor

Assume a processor better than what is available in 2011:

- Issue width: 64 instructions per cycle (10X the *Issue* bandwidth of a real processor).
- Register renaming with 64 additional integer registers + 64 additional floating-point registers.
- Branch predictor *tournament predictor* with 1K-entry and a 16-entry return address stack.
- Perfect memory desambiguation.
- Cache hit ratio 100%.
- Operators with 1-cycle latency.

## Limits of ILP: real processor (cont.)

Average number of instruction issues per cycle (window-size: max. number of in flight instructions):



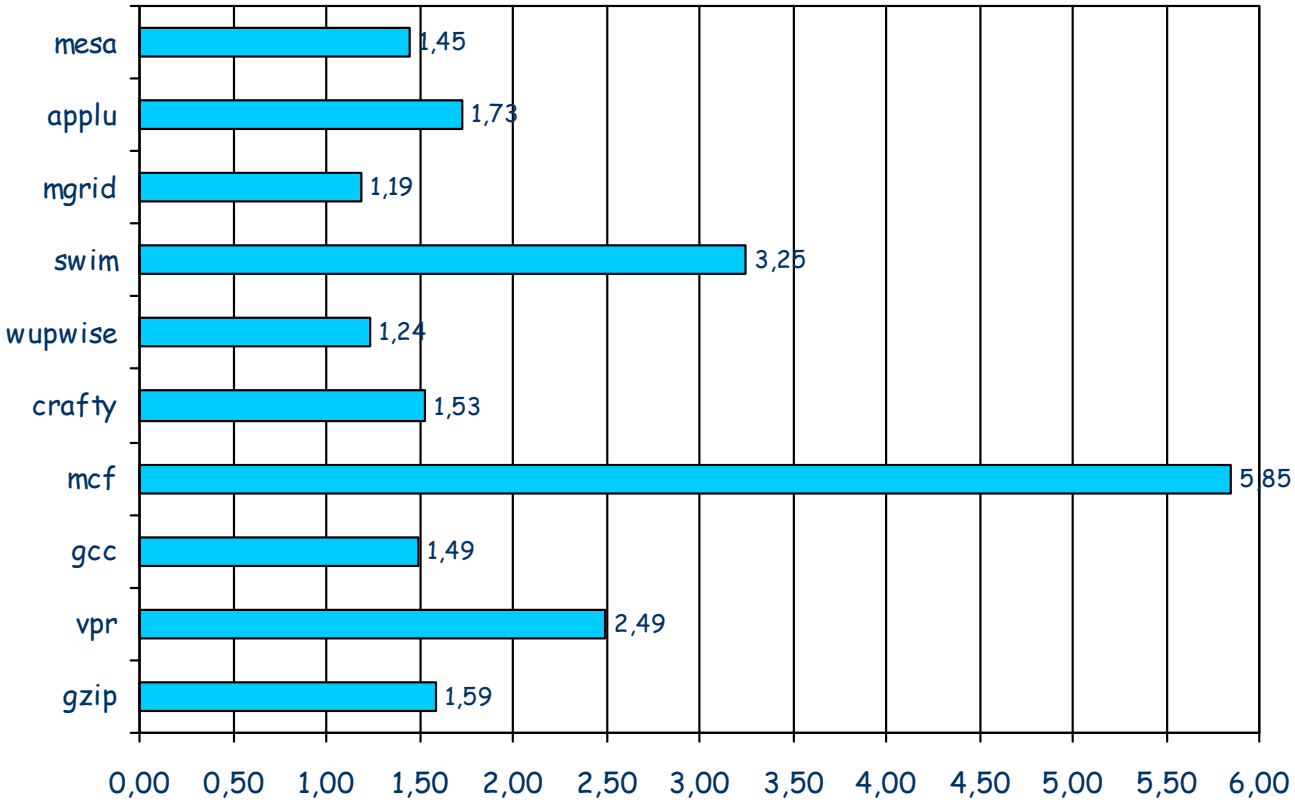

- The effect of the window size for integer benchmarks is not as severe as for FP benchmarks.
- The availability of loop level parallelism in two FP benchmarks results in high ILP.
- Other factors such as branch prediction, register renaming and less parallelism limit ILP in integer benchmarks.

## Limits of ILP: Pentium 4 (Netburst architecture)

- Issue width: 3 inst/cycle.
- Register file: 128 integer registers.
- ROB: 128 entries.
- BTB branch predictor: 4K entries.
- L1-D cache: 16 KB, 8-way, 64B (4-12 cycles); L2: 2MB, 8-way, 128B (18 cycles).
- 7 functional units with multiple cycles latency.
- A simple instruction requires 31 cycles from IF to Commit.

## Limits of ILP: Pentium 4 (Netburst architecture) (cont.)

Measured CPI:




$\Rightarrow$ compared to other processors, CPI is relatively high. It is necessary to look for other kinds of parallelism ...

## Contents

1 Introduction

2 Superscalar processors

3 Multithreaded processors

4 VLIW processors

5 Superpipelined processors

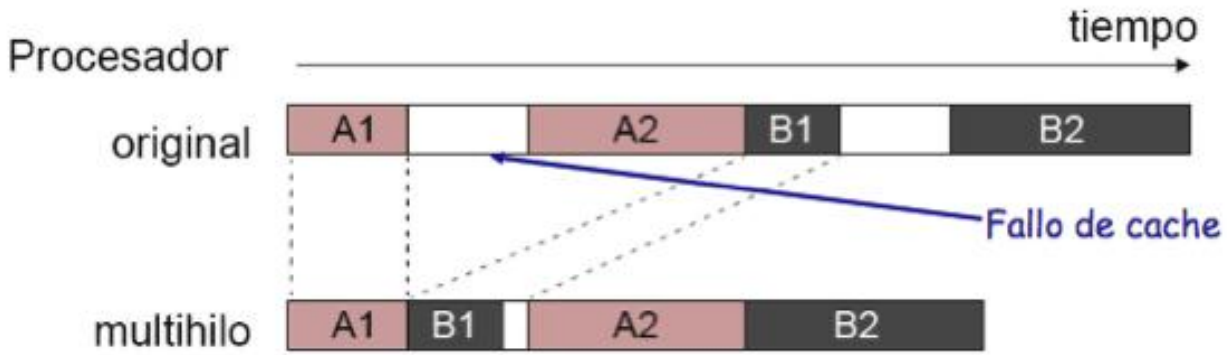
## TLP: Thread-Level Parallelism

Thread: process with its own code and data.

- ILP exploits parallelism within a loop or linear segment of code.
- TLP exploits parallelism among multiple independent threads.
- **Multithreading** provides support for multiple flows of instructions aiming at:
  - Increasing the throughput of processors running lots of independent programs $\rightarrow$ each thread is a different application.
  - Reducing the execution time of programs having multiple independent threads $\rightarrow$ each thread is a fragment of a parallel program.
- TLP can be much more cost efficient than ILP.
- Two types of TLP architectures: multiprocessors and multithreaded processors.

## Original idea: Multithreading to tolerate memory latency

- In superscalar processors, L2 or L3 cache misses cause *stalls* → functional units are underused.
- Multithreading allows avoiding most stalls by switching to another thread.




→ The instruction throughput increases.
→ If both threads belong to the same application, the execution time is reduced.

## Hardware requirements for multithreaded processors

- Multithreading: most processor components are shared by a set of threads.
- Per-thread state is replicated (private components):
  - Register file.
  - Program counter.
  - Page table.
- The memory is shared through the virtual memory mechanisms (which already support multiprogramming).
- Thread switching is performed by processor mechanisms, which is much faster than OS context switching.
- The application must contain several threads, either identified by the compiler (tipically by using a language that allows parallel constructs) or by the programmer.

## Three multithreading approaches

- Fine-grain multithreading.
- Coarse-grain multithreading.
- Simultaneous multithreading.

## Fine-grain multithreading

- Thread switching is performed every clock cycle, interleaving the execution of the threads.
- Interleaving is often done in a round-robin fashion, skipping threads that have no ready instructions at that time.
- Requires very fast thread switching.
- Advantage: hides the throughput losses that arise from both short and long latencies.
- Disadvantage: slows down the execution of individual threads, since thread switching is done on a per-cycle basis.

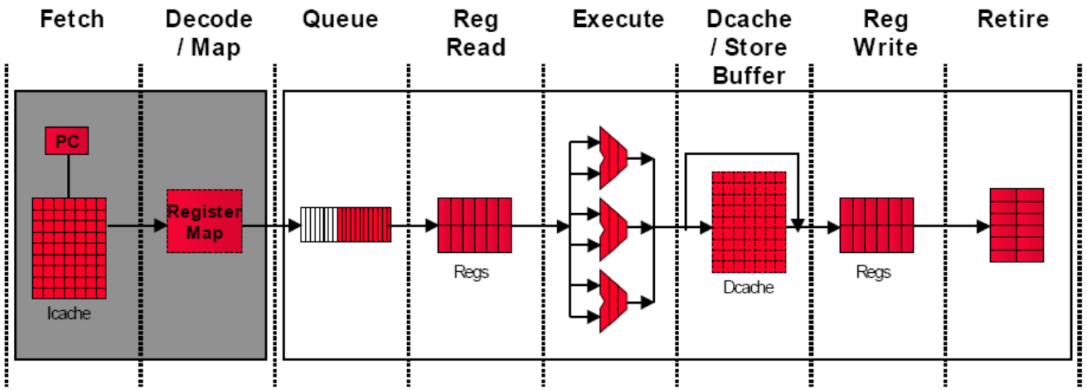
## Coarse-grain multithreading

- Thread switching is only done on long-latency *stalls*, such as L2 or L3 cache misses.
- Advantages:
  - It does not require fast thread switching. Usually, the instruction unit is flushed, and instructions are fetched from the new thread.
  - It does not delay the execution of a single thread, since thread switching only occurs when a thread cannot make forward progress.
- Disadvantage: limited ability to overcome throughput losses, specially from short stalls.

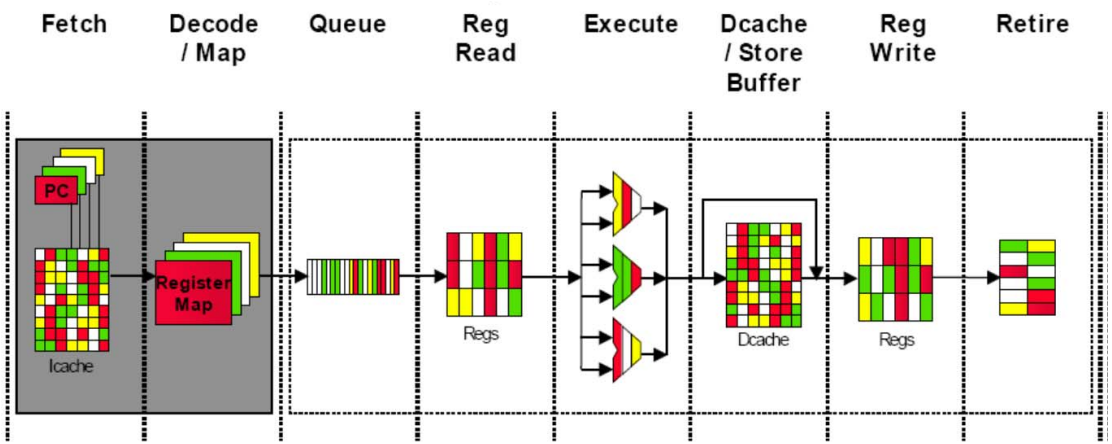
## Simultaneous multithreading (SMT)

- Instructions from different threads can be issued at the same time, trying to use all the available resources.
- Out-of-order superscalar processors already implement mechanisms that support the execution of more than one thread:
  - Register renaming provides a way to uniquely identify all the instruction operands, thus allowing instructions from different threads to be executed without mistaking the operands.
  - Dynamic instruction scheduling takes care of dependencies among instructions within each thread.
  - Out-of-order execution helps improving resource utilization.
  - A private (physical or virtual) ROB for each thread is required.

## Simultaneous multithreading (SMT) (cont.)



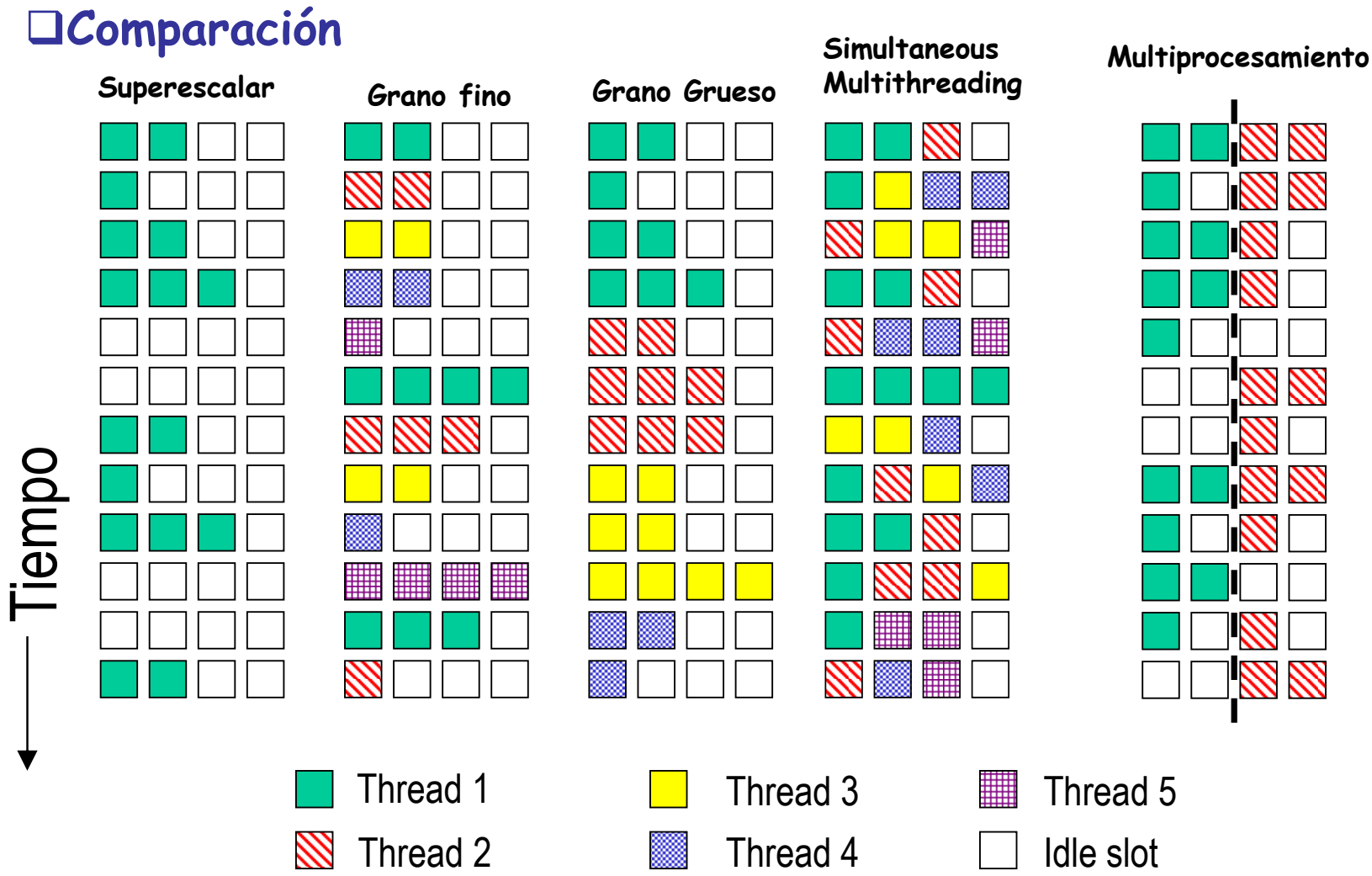

✓ todos los recursos utilizados por un hilo




✓ recursos para distinguir el estado de los hilos

✓ los otros recursos se pueden compartir

## Comparison


❑Comparación
Superescalar
Grano fino
Grano Grueso
Simultaneous Multithreading
Multiprocesamiento
Tiempo
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Idle slot

## Contents

1 Introduction

2 Superscalar processors

3 Multithreaded processors

4 VLIW processors

5 Superpipelined processors

## Concept

VLIW: *Very Long Instruction Word*

Processors with a very long instruction format, coding several ($p$) operations in an instruction.

Static instruction scheduling (compiler-based technique) is applied.

### Example with $p = 5$:

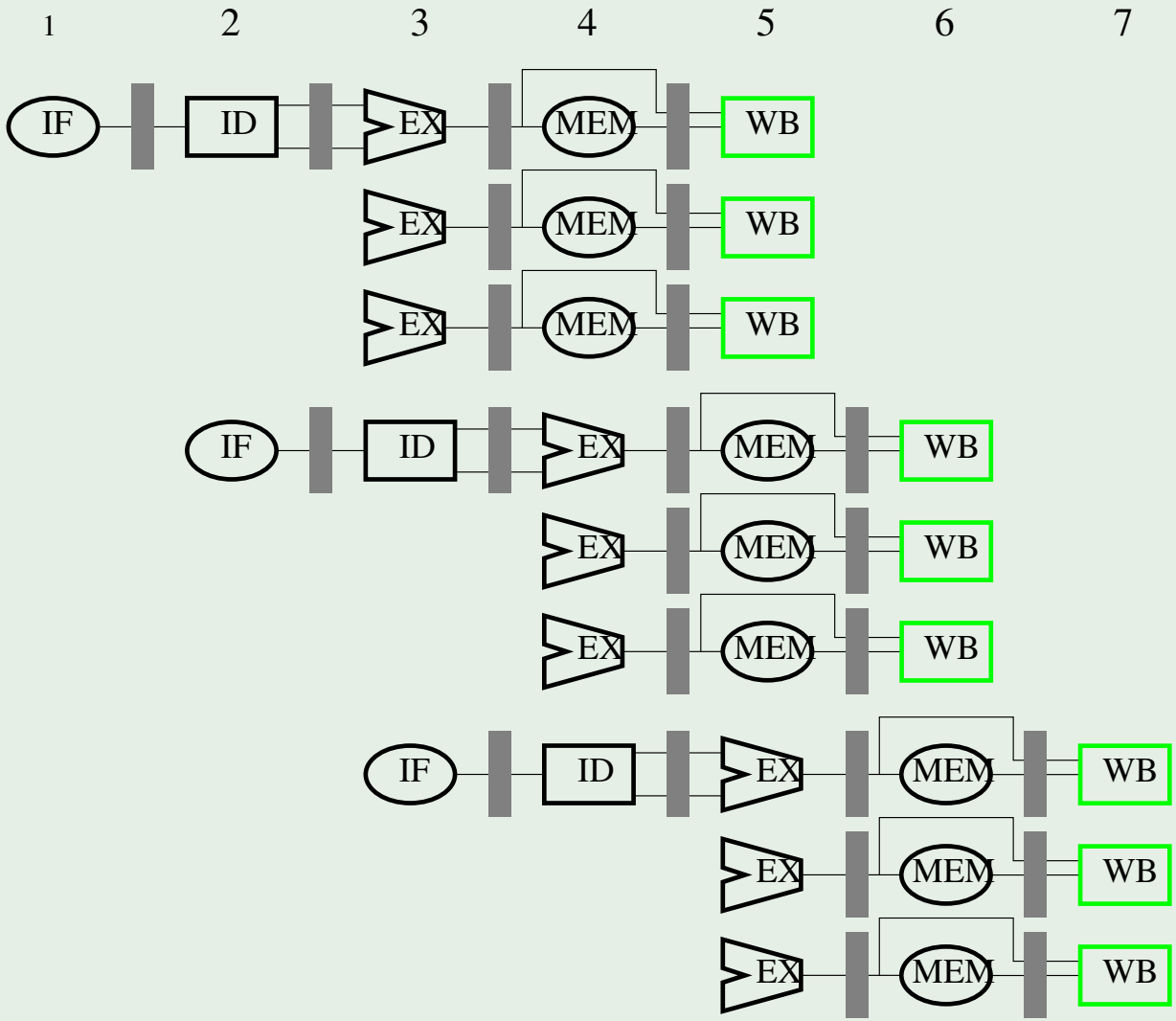
| $\text{Memory}_1$ | $\text{Memory}_2$ | $\text{FP}_1$ | $\text{FP}_2$ | Integer/Branch |
|---|---|---|---|---|

### Comparison with a pipelined processor

- $T_{\text{pipelined}} = I \times CPI \times T$
  $T_{\text{VLIW}} = I' \times CPI' \times T' = \frac{I}{p} \times CPI \times T$

$\Rightarrow$ Performance improvement by a factor of $p$.

## i–t diagram for $p = 3$:



## Consequences:

- The *hardware* does not apply dynamic instruction scheduling.
- The compiler extracts the instruction level parallelism, packing several independent operations (or NOPs, when they are not found) in a single instruction $\rightarrow$ special compilation techniques.
- They are not binary compatible with the original scalar machines, neither with other VLIW machines with different hardware (different # of functional units and/or different operator latencies).
- If the compiler optimization is not good or the code presents low ILP, the generated code size can be larger than the conventional one.

## Example

Conventional code:

```
loop:   L.D F1,0(R1)
        ADD.D F2,F1,F0
        S.D F2,0(R1)
        DSUB R1,R1,#8
        BNEZ R1,loop
```

VLIW Code (2-cycle L.D and 3-cycle ADD.D, pipelined; 16 FP regs):

| $Memory_1$ | $Memory_2$ | Arithmetic $FP_1$ | Arithmetic $FP_2$ | Integer/Branch |
|---|---|---|---|---|
| L.D F1,0(R1) | L.D F3,-8(R1) | | | |
| L.D F5,-16(R1) | L.D F7,-24(R1) | | | |
| L.D F9,-32(R1) | L.D F11,-40(R1) | ADD.D F2,F1,F0 | ADD.D F4,F3,F0 | |
| L.D F13,-48(R1) | | ADD.D F6,F5,F0 | ADD.D F8,F7,F0 | |
| | | ADD.D F10,F9,F0 | ADD.D F12,F11,F0 | |
| S.D F2,0(R1) | S.D F4,-8(R1) | ADD.D F14,F13,F0 | | |
| S.D F6,-16(R1) | S.D F8,-24(R1) | | | |
| S.D F10,-32(R1) | S.D F12,-40(R1) | | | DSUB R1,R1,#56 |
| S.D F14,8(R1) | | | | BNEZ R1,loop |

## Contents

1 Introduction

2 Superscalar processors

3 Multithreaded processors

4 VLIW processors

5 Superpipelined processors

## Concept

The clock cycle is $t$ times smaller than the one of the original pipelined computer $\rightarrow$ The number of stages in the instruction cycle is $kt$ $\rightarrow$ CPI=1 (but the cycle is shorter!)

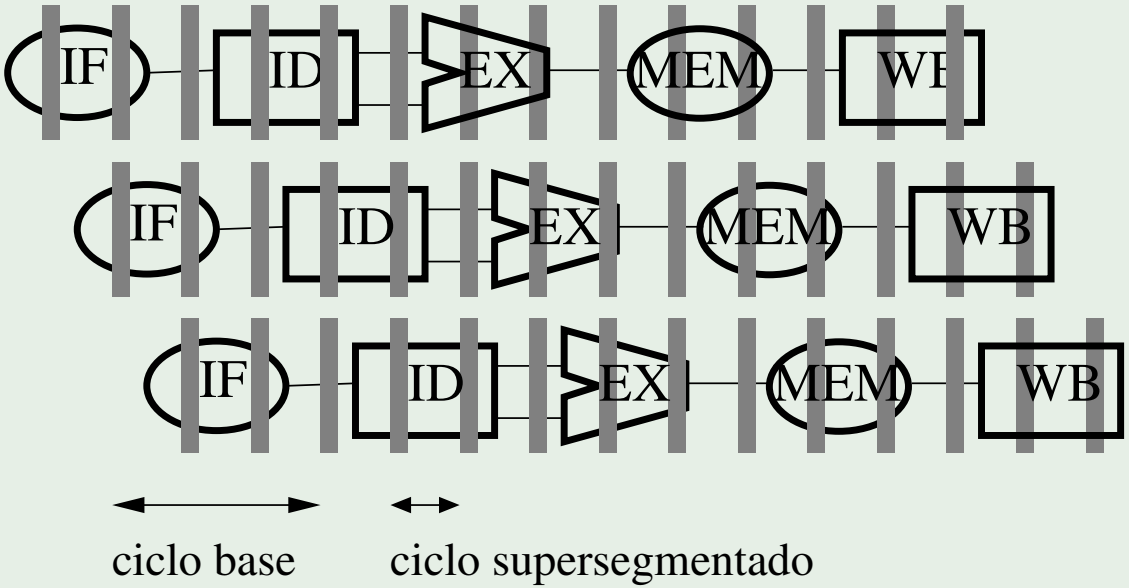
$t$: superpipelined **degree** (number of substages each original stage is divided into).

### Comparison with a pipelined computer

- $T_{\text{pipelined}} = I \times CPI \times T$
  $T_{\text{super-pipelined}} = I' \times CPI' \times T' = I \times CPI \times \frac{T}{t}$

$\Rightarrow$ Performance is increased by a factor of $t$.

## Example for $t = 3$:



## Consequences:

- There is no need for replicating execution units, but pipelining becomes more “complex” (operators and memories must be pipelined in several stages).
- Higher clock frequency $\rightarrow$ higher overhead on intermediate registers and **clock skew** problems.
- Indeed, not all the stages of the original pipelined processor must be superpipelined but only the slowest ones.
  Example: MIPS
  IF(10ns) ID(5ns) EX(10ns) M(10ns) WB(5ns)
  $\downarrow$
  IF1(5ns) IF2(5) ID(5) EX1(5) EX2(5) M1(5) M2(5) WB(5)