*Course on Advanced Computer Architectures*

# Static Multiple-Issue Processors: VLIW Architectures

POLITECNICO
MILANO 1863

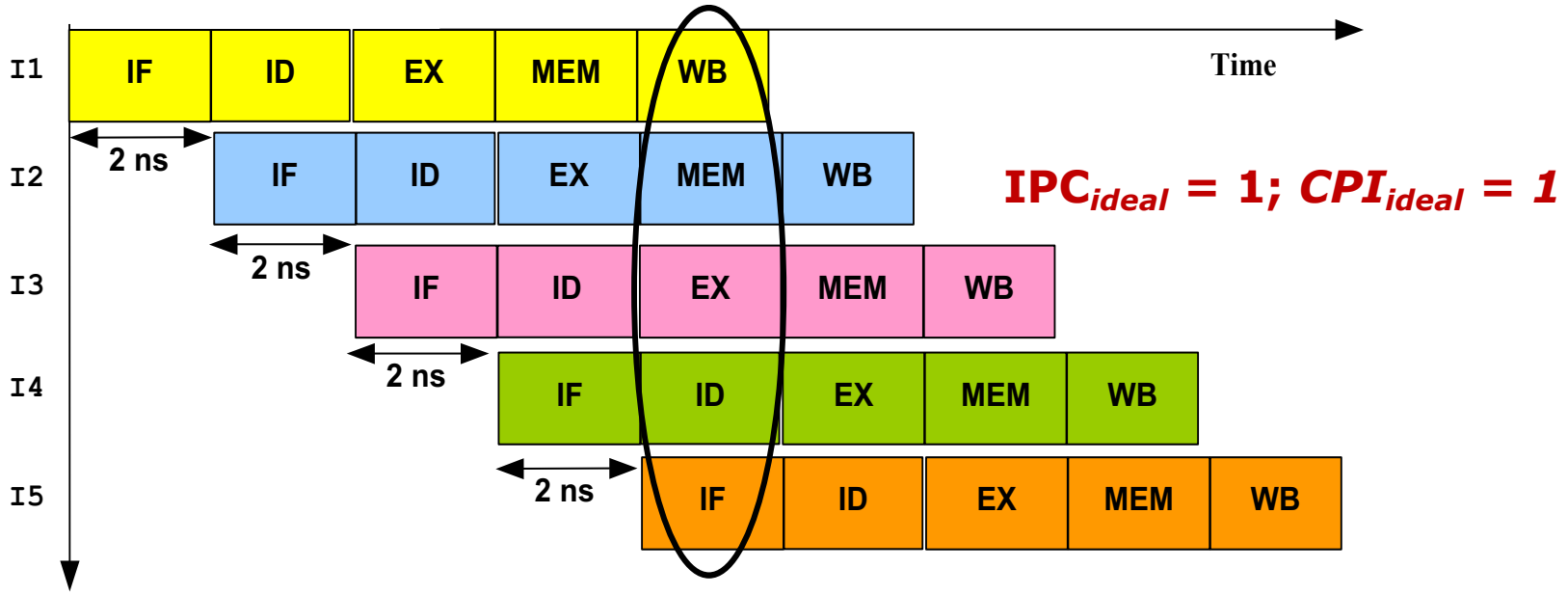
Prof. Cristina Silvano, email: cristina.silvano@polimi.it


# Summary

- Recap on Instruction Level Parallelism
- VLIW Architectures
- Code Scheduling for VLIW Architectures
  - *Next lecture*

POLITECNICO
MILANO 1863

# Recap on Single-Issue Processors

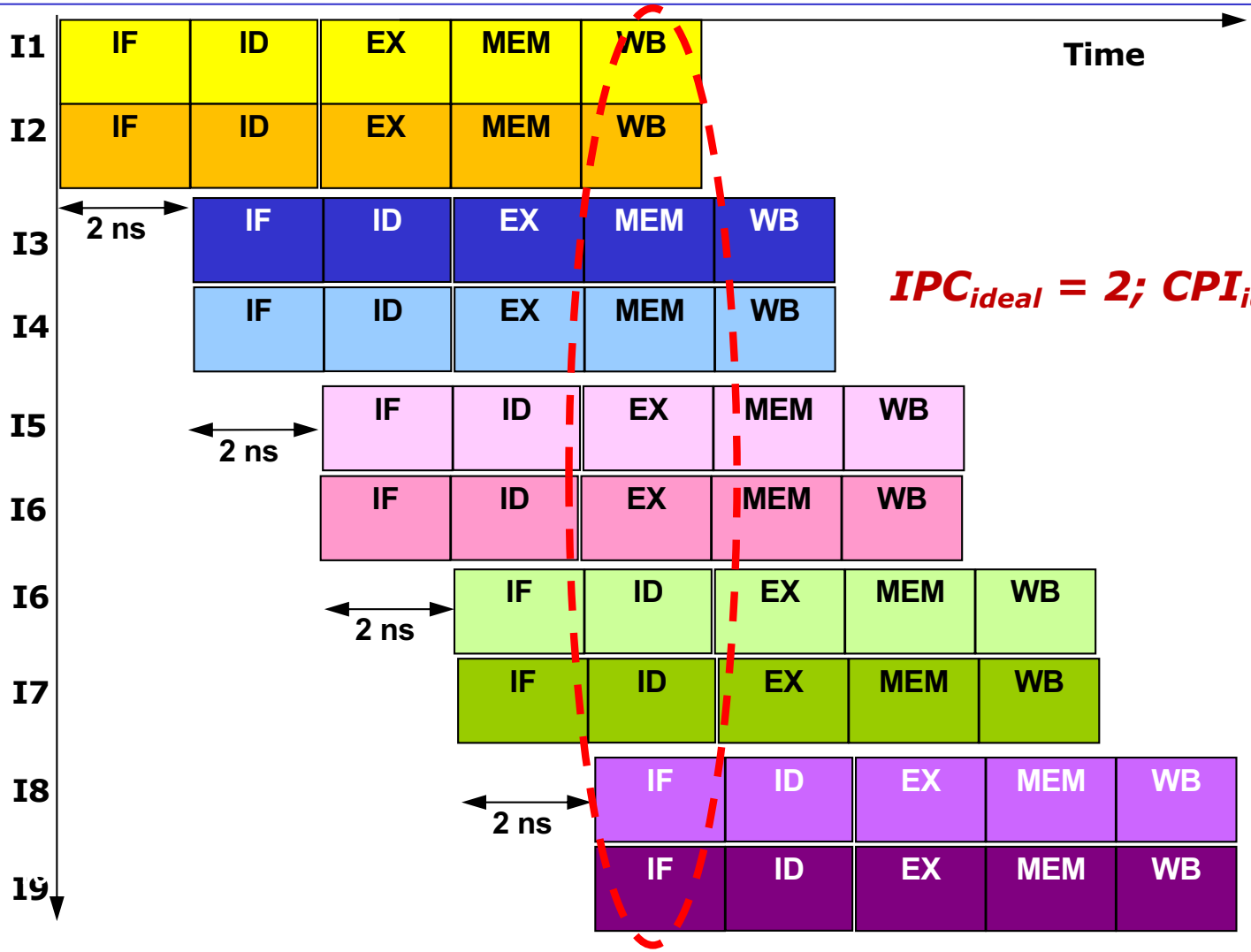



$IPC_{ideal} = 1$; $CPI_{ideal} = 1$

***5-stage pipeline => 5 different instructions overlapped***

POLITECNICO
MILANO 1863

# Recap on Dual-Issue Processors




$IPC_{ideal} = 2;\ CPI_{ideal} = 0.5$

*2-issue 5-stage pipeline => 2 x 5 different instructions overlapped*

POLITECNICO
MILANO 1863

# Towards Multiple-Issue Processors

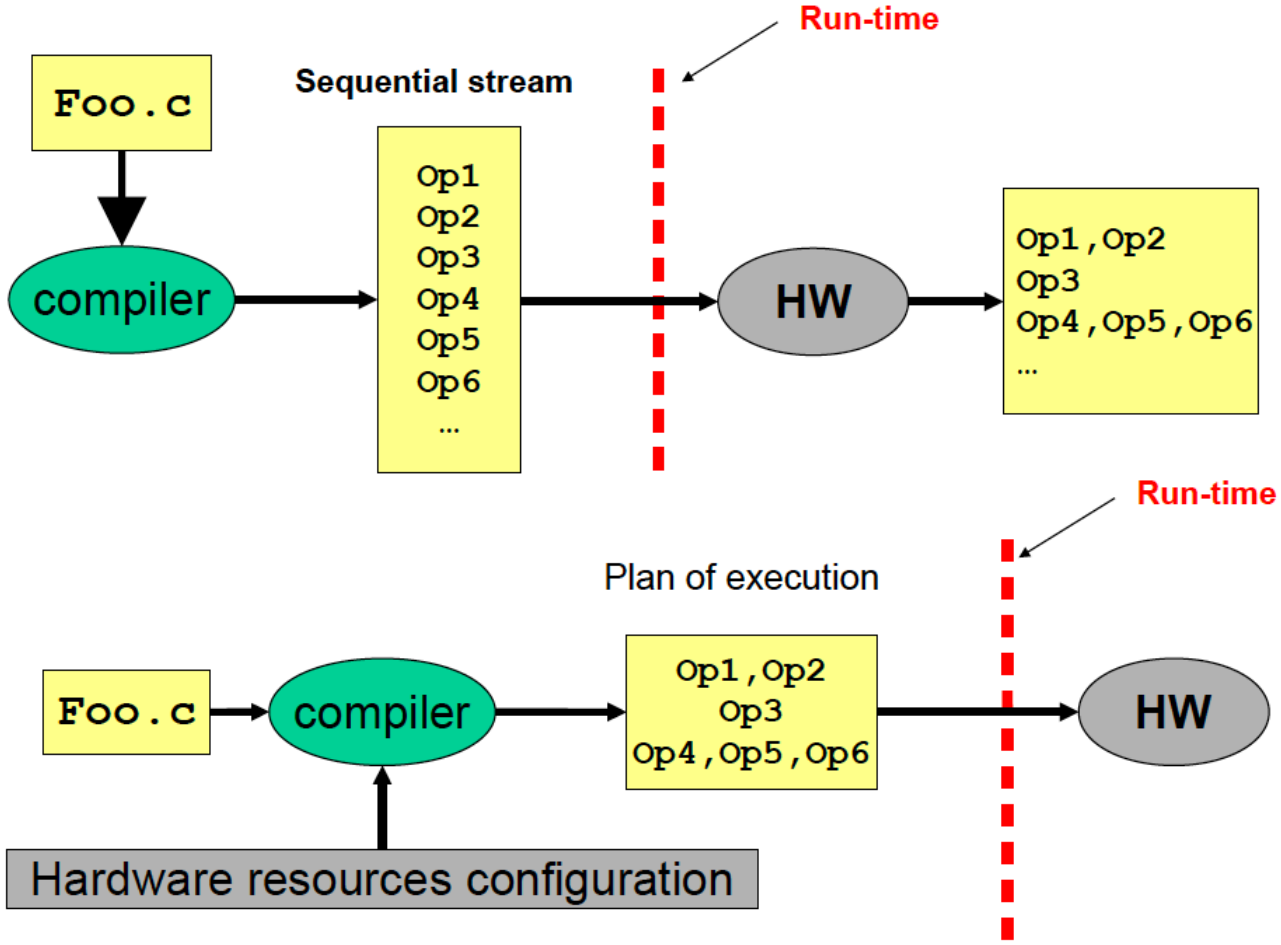
- **Single-Issue Processors:** Scalar processors that fetch and issue max one operation at each clock cycle.

- **Multiple-Issue Processors** require:
  - **To fetch** multiple instructions in a cycle (higher bandwidth from the instruction cache)
  - **To issue** multiple instructions based on:
    - **Dynamic scheduling:** the HW issues at runtime a varying number of instructions at each clock cycle.
    - **Static scheduling:** the compiler issues statically a fixed number of instructions at each clock cycle.

POLITECNICO
MILANO 1863

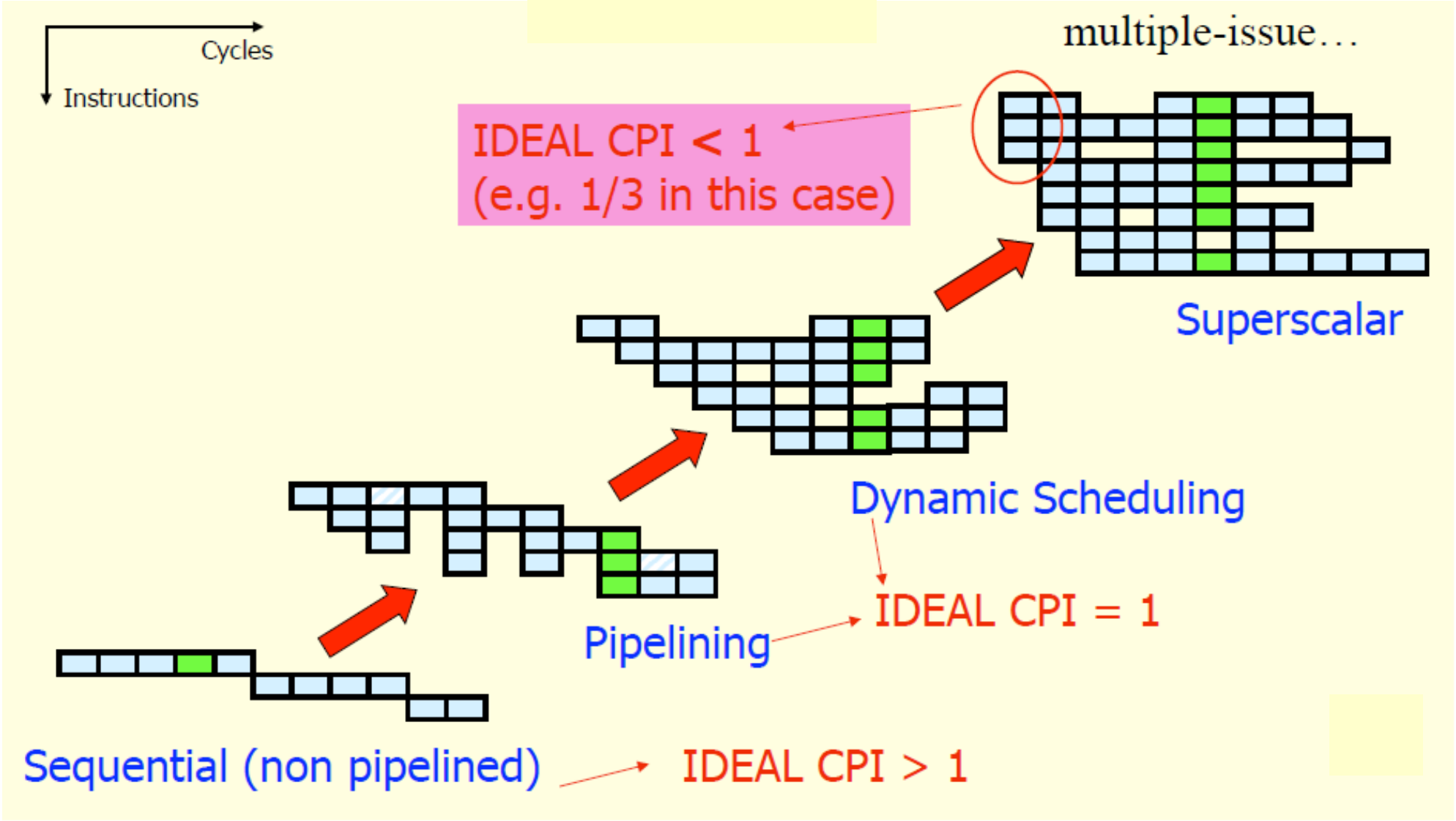
# Dynamic vs Static Scheduling

*Who decides about the ILP?*

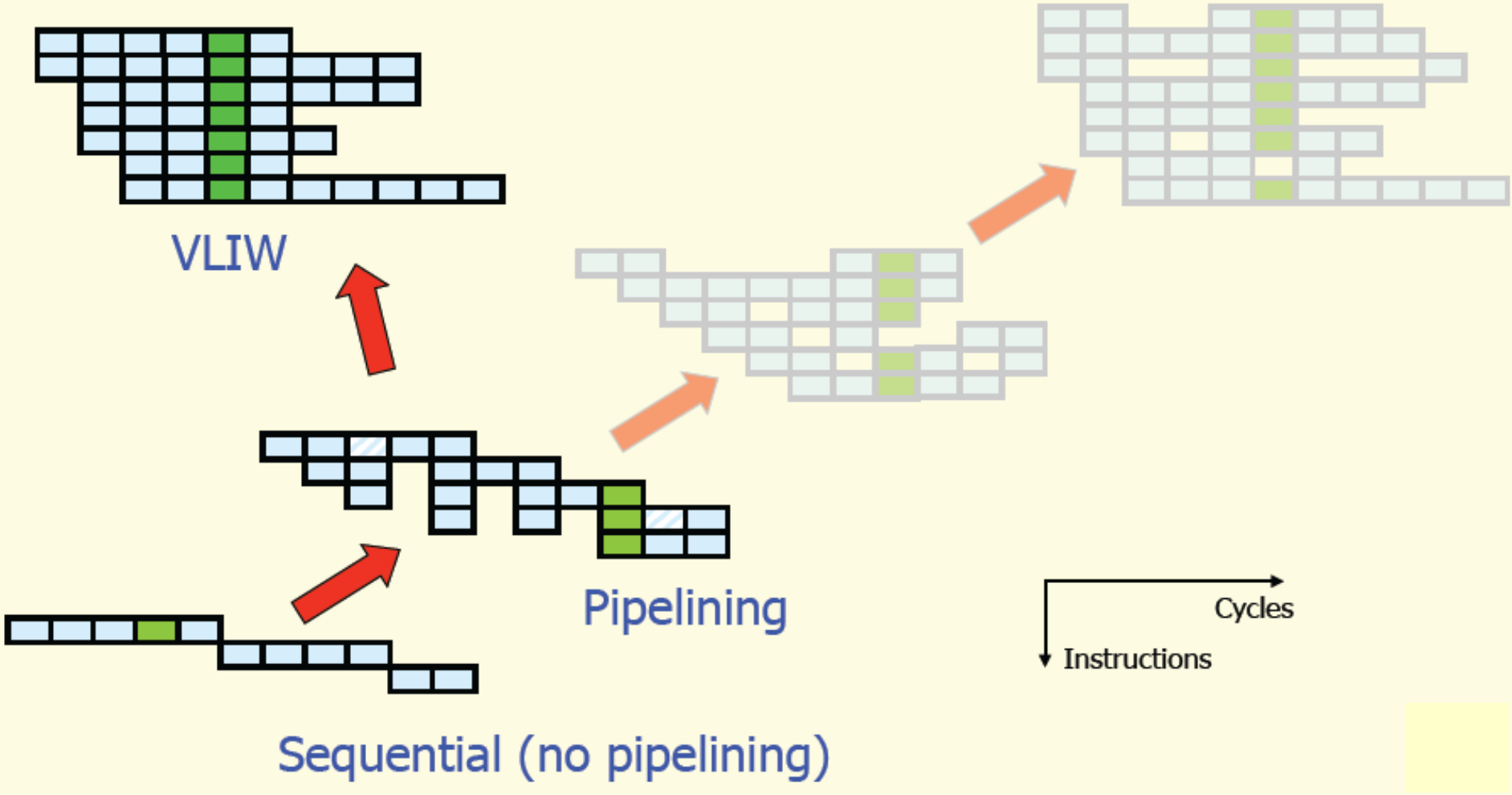


POLITECNICO
MILANO 1863

# Towards multiple-issue superscalar processors



# Very Long Instruction Word: An Alternative Way of Extracting ILP



POLITECNICO
MILANO 1863

# Statically Scheduled Processors

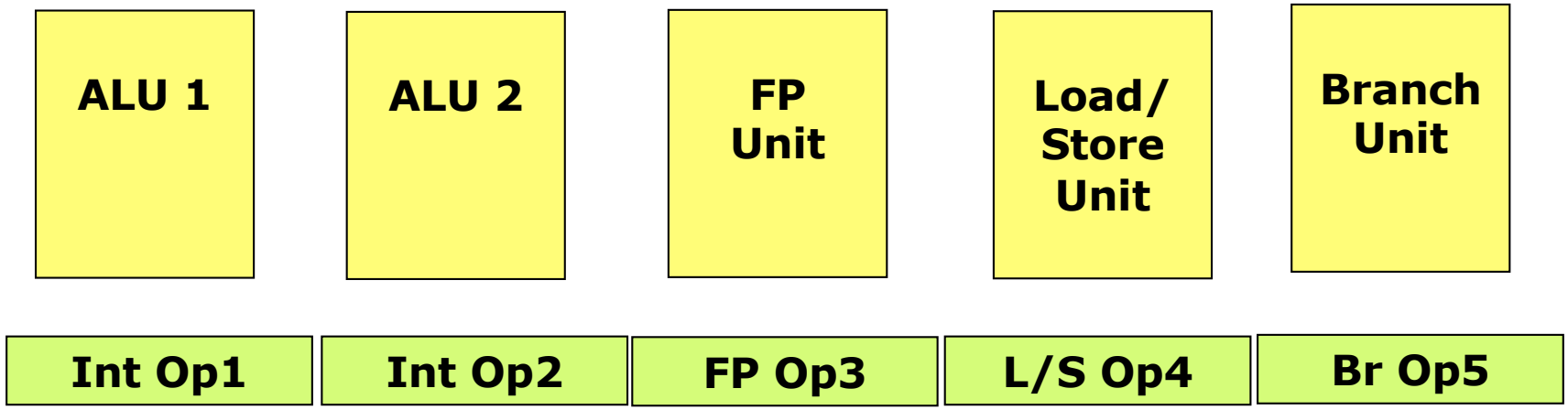
- **Compilers** can use sophisticated algorithms for code scheduling to exploit ILP (Instruction Level Parallelism)
  - Detect whether instructions can be parallelized given the hardware resource constraints and the data dependencies.
  - Schedule statically instructions to be executed in parallel, otherwise insert NOPs.

POLITECNICO
MILANO 1863

# VLIW Processors: An Alternative Way of Extracting ILP

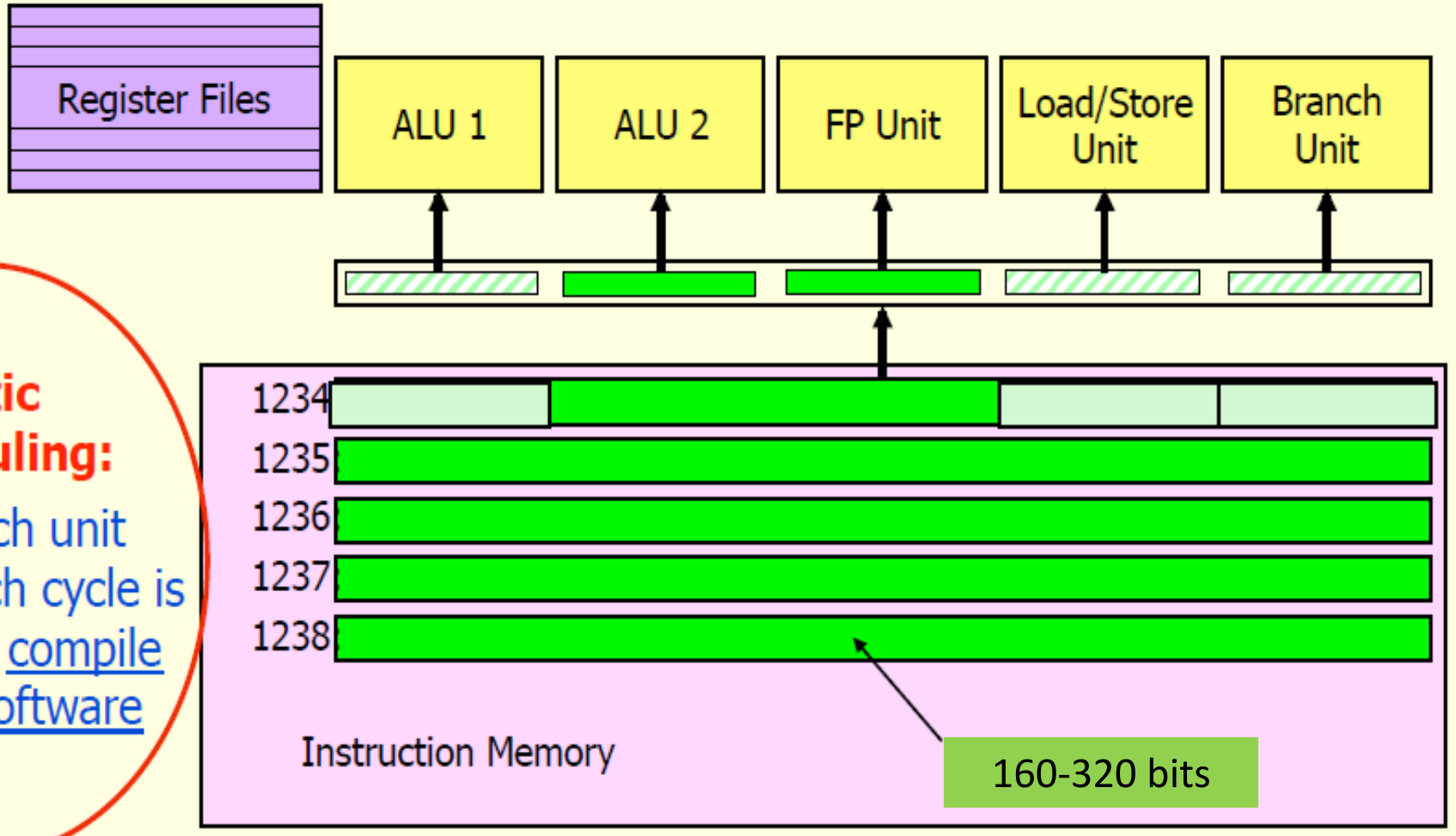
POLITECNICO MILANO 1863

# Multiple-issue VLIW processor




- The long instruction **(bundle)** has a fixed set of **operations** (**slots).**
- Example: a **5-issue VLIW** has a long instruction (bundle) to contain up to **5 operations** corresponding to **5 slots.**

# (Statically Scheduled) Very Long Instruction Word Processor (VLIW)




**Static Scheduling:**

What each unit does in each cycle is decided at compile time in software

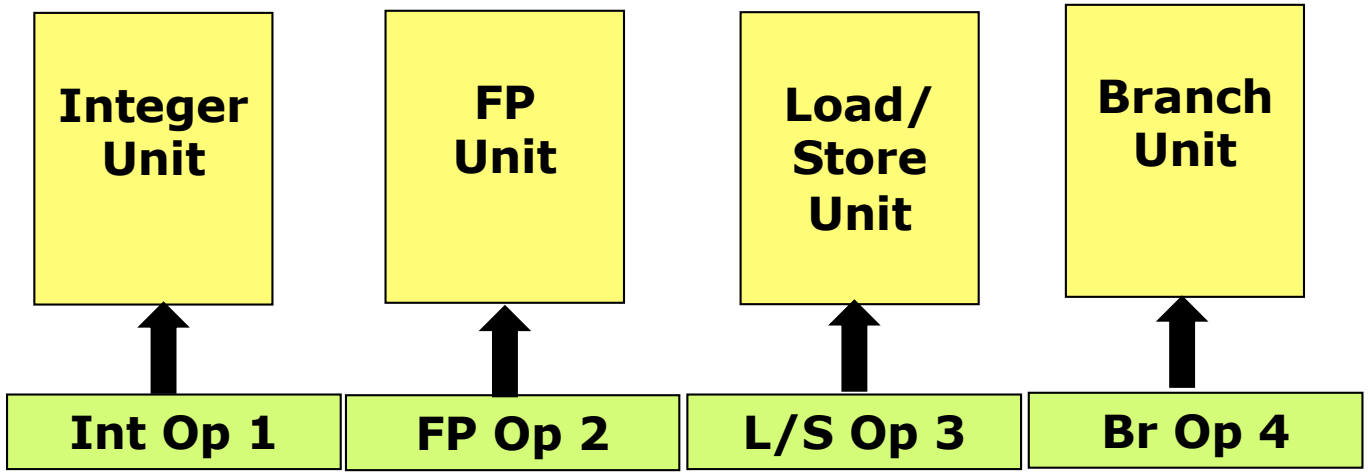
POLITECNICO
MILANO 1863

# Very Long Instruction Word Processors

- The **single-issue packet *(bundle)*** represents a wide instruction with multiple independent operations ***(or syllables)*** per instruction => *named* ***"Very Long Instruction Word"***
- The compiler identifies ***statically*** the multiple independent operations to be executed **in parallel** by the multiple Functional Units.
- The compiler solves ***statically*** the **structural hazards** for the use of HW resources and the **data hazards,** otherwise the compiler inserts NOPs.

POLITECNICO
MILANO 1863

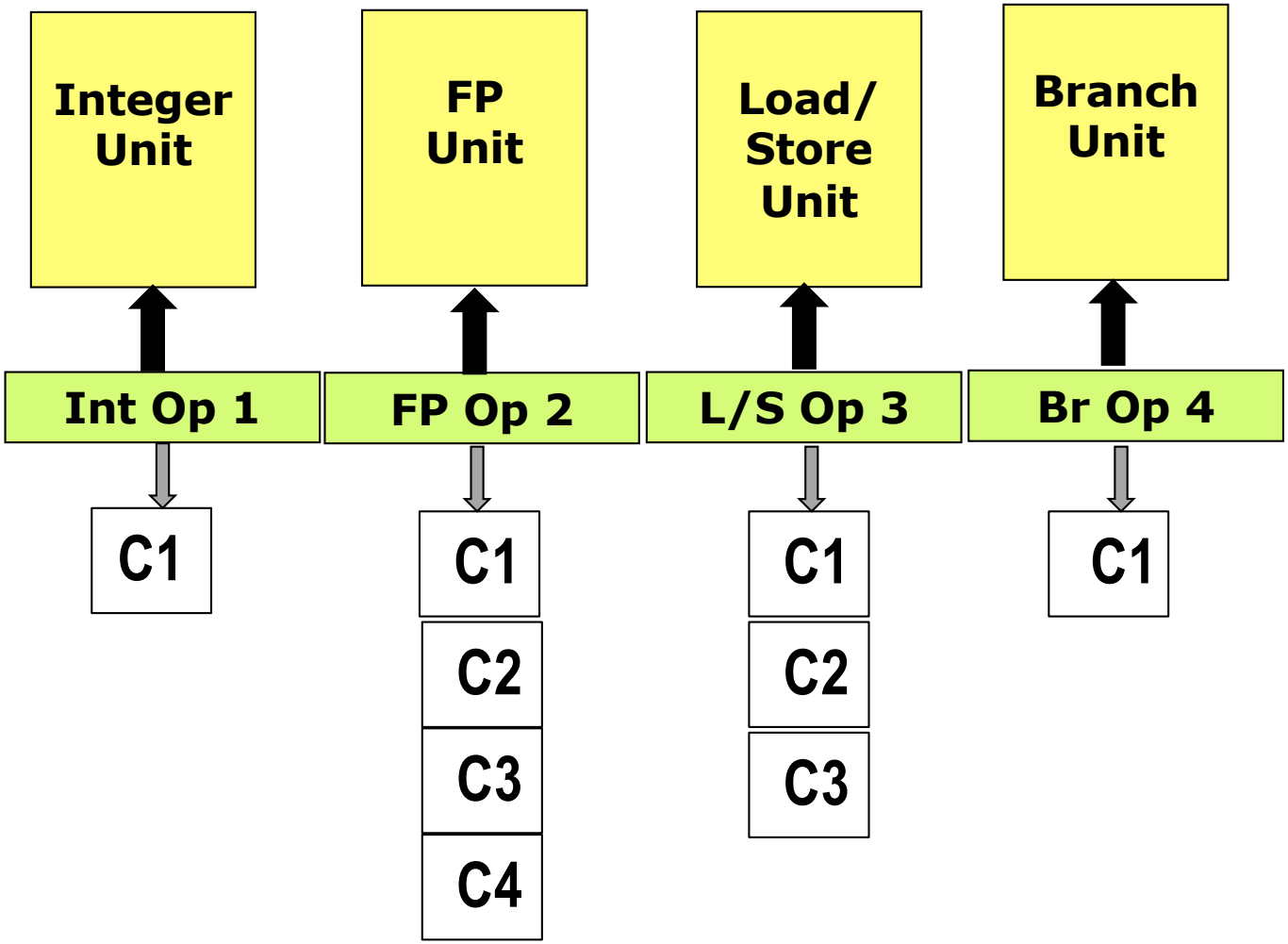
# Multiple-issue VLIW processor




- The long instruction **(bundle)** has a fixed set of **operations** (**slots).**
- Example: a **4-issue VLIW** has a long instruction (bundle) to contain up to **4 operations** corresponding to **4 slots.**

POLITECNICO
MILANO 1863

# Multiple-issue VLIW: Operation latencies



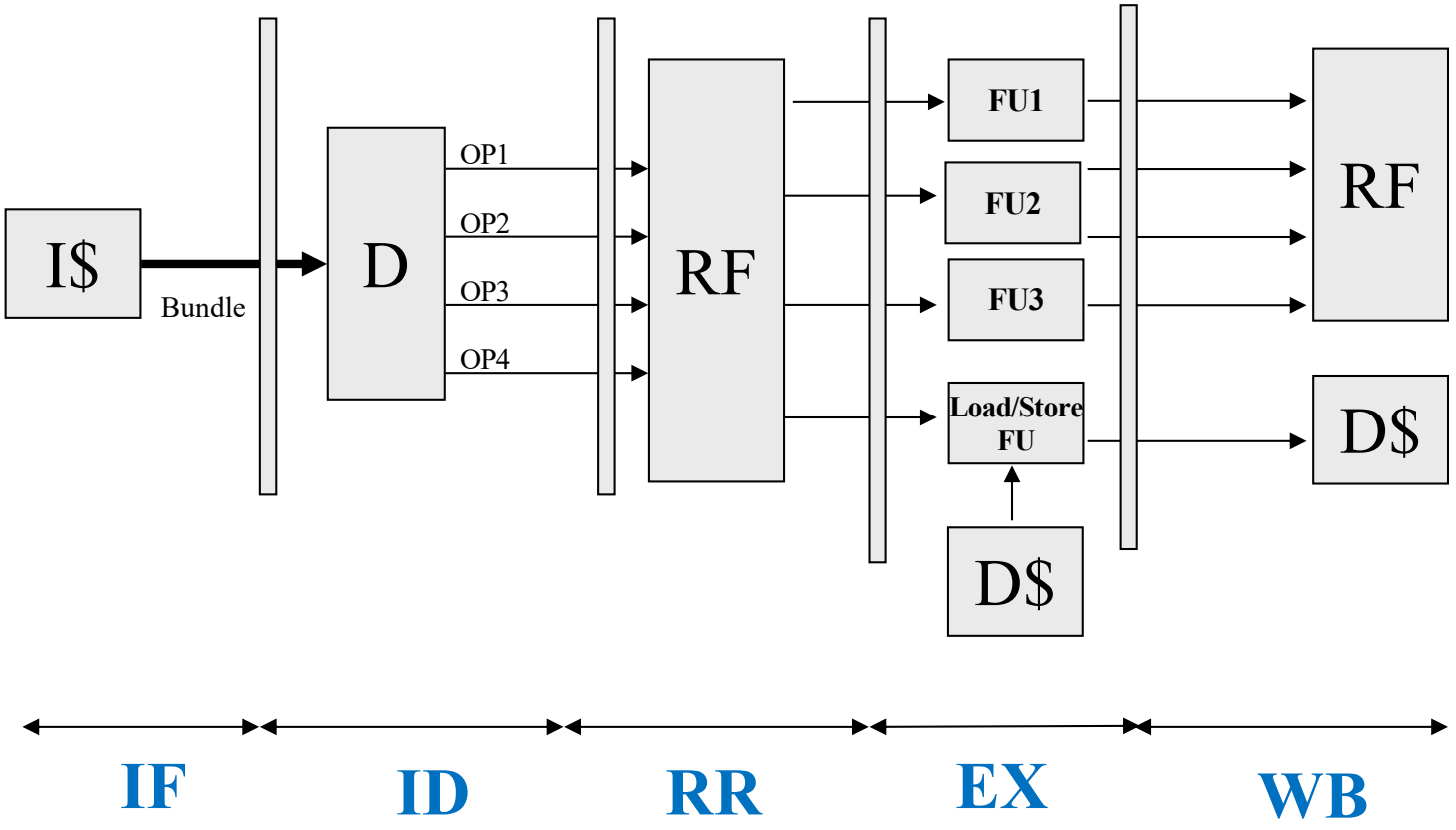
POLITECNICO
MILANO 1863

# VLIW processors: basic concepts (1)

- There is a **single PC** to fetch a long instruction (bundle).
- **Only one branch for each bundle** to modify the control flow.
- There is a **Shared Multi-ported Register File:**
  If the bundle has **4 slots** => we need (**2 x 4)** read ports and **4** write ports to read 8 source registers per cycle and to write 4 destination registers per cycle.
- To keep busy the FUs, there must be enough parallelism in the source code to fill in the available **4** operation slots. Otherwise, **NOPs are inserted.**

POLITECNICO
MILANO 1863

# 5-stage pipelined 4-issue VLIW Architecture



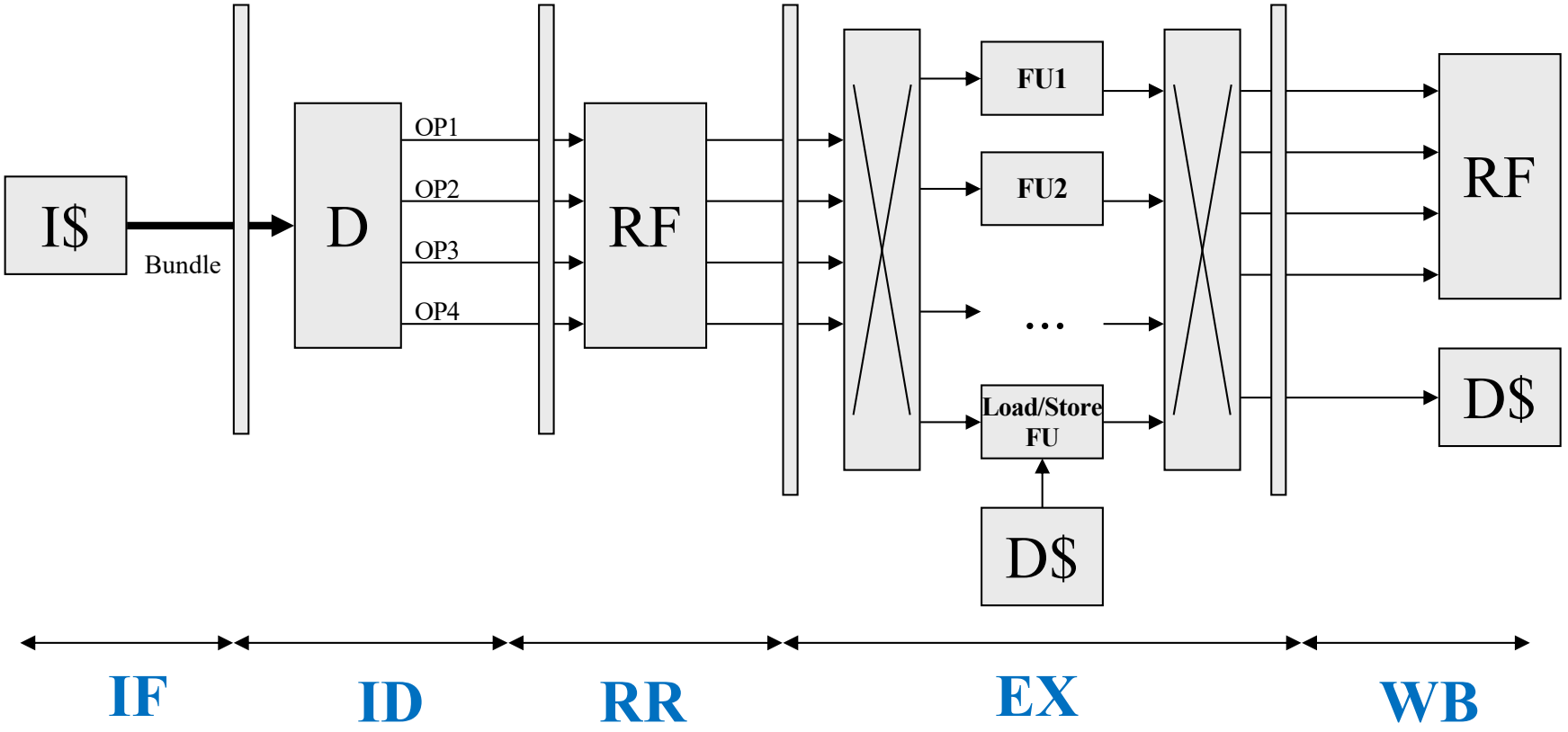
POLITECNICO
MILANO 1863

# VLIW processors: basic concepts (2)

- If each slot is assigned to a Functional Unit => the decode unit is a simple decoder and each op is passed to the corresponding FU to be executed.
- **If there are more parallel FUs than the number of issues (slots)** => the architecture must have a **dispatch network** to redirect each op and the related source operands to the target FU.

POLITECNICO
MILANO 1863

# 5-stage pipelined 4-issue VLIW Architecture



POLITECNICO
MILANO 1863

# VLIW processors: data dependences (1)

- True, anti and output data dependencies are solved **statically** by the compiler by considering the FU latency.

- **To solve RAW hazards in VLIW processors:**
  - The compiler during the scheduling phase reorders **statically** instructions (not involved in the dependencies)
  - Otherwise, the compiler introduces some **NOPs**

POLITECNICO MILANO 1863

# VLIW processors: operation latency

- **Operation latencies** and **data dependencies** must be exposed to the compiler:

```
I      [C = A*B, ....]; latency 2 cycles
I+1    [  NOP,   ....]; inserted by compiler
I+2    [X = C*F, ....];
```

  - Otherwise, correct instruction execution is compromised
  - True even in the case of a pipelined multiplier

POLITECNICO
MILANO 1863

# VLIW processors: data dependences (2)

- **WAR** and **WAW** hazards are statically solved by the compiler by correctly selecting temporal slots for the operations or by using Register Renaming.
- **Structural hazards** are also solved by the compiler.
- Compiler can also provide useful **hints** on how **to statically predict branches.**
- However, a **mispredicted branch** must be solved dynamically by the hardware by **flushing** the execution of the speculative instructions in the pipeline.

POLITECNICO MILANO 1863

# VLIW processors: data dependences (3)

- To keep **in-order execution,** the Write Back phase of the parallel ops in a bundle must occur **at the same clock cycle,** to avoid structural hazards accessing the RF and WAR/WAW hazards.
  - Ops in a bundle are constrained to the **latency** of the longer latency op in the bundle.
- Otherwise, we get **out-of-order execution** and we need to check RF write accesses and WAR/WAW hazards.

POLITECNICO
MILANO 1863

# VLIW processors: Register pressure

- **Register pressure:** the multicycle latency of the operations together with the multiple issue of instructions generates an increment of the number of register occupied over time.
- Example: If one bundle has 2 FP operations requiring 5 clock cycles, the 2 destination registers are allocated for 5 long instructions => these registers cannot be reused by the corresponding (5 x 4) operations.
- **Register Renaming** used to solve **WAR** and **WAW** hazards by the compiler also increases the register pressure.

POLITECNICO
MILANO 1863

# VLIW processors: Dynamic events

- The compiler at static time does not know the behavior of some **dynamic events** such as:
  - **Data Cache Misses:** Stalls are introduced at runtime (latency of a data cache miss is known at compile time)
  - **Branch Mispredictions:** Need of flushing at runtime the execution of speculative instructions in the pipeline.

POLITECNICO
MILANO 1863

# Scalar vs. 2-issue VLIW Scheduling: Example

| INSTRUCTIONS | HAZARDS |
|---|---|
| L1: lw $2, BASEA($4) | |
| addi $2,$2,INC1 | raw $2, waw $2 |
| lw $3, BASEB($4) | |
| addi $3,$3,INC2 | raw $3, waw $3 |
| addi $4, $4, 4 | war $4 |
| bne $4,$7, L1 | raw $4 |

POLITECNICO MILANO 1863

# Scalar vs. 2-issue VLIW Scheduling: Example

| INSTRUCTIONS | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| L1: lw $2, BASEA(s4) | IF | ID | EX | M | WB | | | | | | | | |
| addi $2,$2,INC1 | | IF | IDS | ID | EX | M | WB | | | | | | |
| lw $3, BASEB($4) | | | | IF | ID | EX | M | WB | | | | | |
| addi $3,$3,INC2 | | | | | IF | IDS | ID | EX | M | WB | | | |
| addi $4, $4, 4 | | | | | | | IF | ID | EX | M | WB | | |
| bne $4,$7, L1 | | | | | | | | IF | IDS | ID | EX | M | WB |

**MIPS 5-stage pipelined scalar scheduling with forwarding and early branch resolution in ID stage + dynamic stall insertion to solve RAW hazards**

POLITECNICO MILANO 1863

# Scalar vs. 2-issue VLIW Scheduling: Example

| | INSTRUCTIONS | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| L/S | lw $2, BASEA(s4) | IF | ID | EX | M | WB | | | | | | |
| A/B | NOP | IF | ID | EX | M | WB | | | | | | |
| L/S | lw $3, BASEB($4) | | IF | ID | EX | M | WB | | | | | |
| A/B | NOP | | IF | ID | EX | M | WB | | | | | |
| L/S | NOP | | | IF | ID | EX | M | WB | | | | |
| A/B | addi $2,$2,INC1 | | | IF | ID | EX | M | WB | | | | |
| L/S | NOP | | | | IF | ID | EX | M | WB | | | |
| A/B | addi $3,$3,INC2 | | | | IF | ID | EX | M | WB | | | |
| L/S | NOP | | | | | IF | ID | EX | M | WB | | |
| A/B | addi $4, $4, 4 | | | | | IF | ID | EX | M | WB | | |
| L/S | NOP | | | | | | IF | ID | EX | M | WB | |
| A/B | NOP | | | | | | IF | ID | EX | M | WB | |
| L/S | NOP | | | | | | | IF | ID | EX | M | WB |
| A/B | bne $4,$7, L1 | | | | | | | IF | ID | EX | M | WB |

POLITECNICO MILANO 1863

# Scalar vs. 2-issue VLIW Scheduling: Example

| | INSTRUCTIONS | C1 | C2 | C3 | C4 | C5 | C6 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| L/S | lw $2, BASEA(s4) | IF | ID | EX | M | WB | | | | | | |
| A/B | NOP | IF | ID | EX | M | WB | | | | | | |
| L/S | lw $3, BASEB($4) | | IF | ID | EX | M | WB | | | | | |
| A/B | NOP | | IF | ID | EX | M | WB | | | | | |
| L/S | NOP | | | IF | ID | EX | M | | | | | |
| A/B | addi $2,$2,INC1 | | | IF | ID | EX | M | | | | | |
| L/S | NOP | | | | IF | ID | EX | M | WB | | | |
| A/B | addi $3,$3,INC2 | | | | IF | ID | EX | M | WB | | | |
| L/S | NOP | | | | | IF | ID | EX | M | WB | | |
| A/B | addi $4, $4, 4 | | | | | IF | ID | EX | M | WB | | |
| L/S | NOP | | | | | | IF | ID | EX | M | WB | |
| A/B | NOP | | | | | | IF | ID | EX | M | WB | |
| L/S | NOP | | | | | | | IF | ID | EX | M | WB |
| A/B | bne $4,$7, L1 | | | | | | | IF | ID | EX | M | WB |

| | SLOT 1: LD/ST Ops | SLOT 2: Int/BR Ops |
|---|---|---|
| C1 | lw $2,BASEA($4) | NOP |
| C2 | lw $3,BASEB($4) | NOP |
| C3 | NOP | addi $2,$2,INC1 |
| C4 | NOP | addi $3,$3,INC2 |
| C5 | NOP | addi $4, $4, 4 |
| C6 | NOP | NOP |
| C7 | NOP | bne $4,$7, L1 |

POLITECNICO
MILANO 1863

# Scalar vs. 2-issue VLIW Scheduling: Example

| INSTRUCTIONS |
|---|
| L1: lw $2, BASEA($4) |
| addi $2,$2,INC1 |
| lw $3, BASEB($4) |
| addi $3,$3,INC2 |
| addi $4, $4, 4 |
| bne $4,$7, L1 |

**2-issue VLIW with 2 fully pipelined FUs with :**
**- LD/ST Unit with latency 2**
**- INT Unit with latency 1 & latency 2 to BR**

| | SLOT 1: LD/ST Ops | SLOT 2: Int/BR Ops |
|---|---|---|
| C1 | lw $2,BASEA($4) | NOP |
| C2 | lw $3,BASEB($4) | NOP |
| C3 | NOP | addi $2,$2,INC1 |
| C4 | NOP | addi $3,$3,INC2 |
| C5 | NOP | addi $4, $4, 4 |
| C6 | NOP | NOP |
| C7 | NOP | bne $4,$7, L1 |

POLITECNICO
MILANO 1863

# Scalar vs. 4-issue VLIW Scheduling: Example

| INSTRUCTIONS |
|---|
| L1: lw $2, BASEA($4) |
| addi $2,$2,INC1 |
| lw $3, BASEB($4) |
| addi $3,$3,INC2 |
| addi $4, $4, 4 |
| bne $4,$7, L1 |

**4-issue VLIW with 4 fully pipelined FUs with:**
**- 2 LD/ST Units with latency 2**
**- 1 INT Unit with latency 1 & latency 2 to BR**
**- 1 INT/BR Unit with latency 1 & latency 2 to BR**

| | SLOT 1: LD/ST Ops | SLOT 2: LD/ST Ops | SLOT 3: Integer Ops | SLOT 4: Int./Branch Ops |
|---|---|---|---|---|
| C1 | | | | |
| C2 | | | | |
| C3 | | | | |
| C4 | | | | |
| C5 | | | | |
| C6 | | | | |

POLITECNICO
MILANO 1863

# Scalar vs. 4-issue VLIW Scheduling: Example

| INSTRUCTIONS |
|---|
| L1: lw $2, BASEA($4) |
| addi $2,$2,INC1 |
| lw $3, BASEB($4) |
| addi $3,$3,INC2 |
| addi $4, $4, 4 |
| bne $4,$7, L1 |

**4-issue VLIW with 4 fully pipelined FUs with:**
**- 2 LD/ST Units with latency 2**
**- 1 INT Unit with latency 1 & latency 2 to BR**
**- 1 INT/BR Unit with latency 1 & latency 2 to BR**

| | SLOT 1: LD/ST Ops | SLOT 2: LD/ST Ops | SLOT 3: Integer Ops | SLOT 4: Int./Branch Ops |
|---|---|---|---|---|
| C1 | lw $2,BASEA($4) | lw $3,BASEB($4) | NOP | NOP |
| C2 | NOP | NOP | NOP | NOP |
| C3 | NOP | NOP | addi $2,$2,INC1 | add $3,$3,INC2 |
| C4 | NOP | NOP | addi $4, $4, 4 | NOP |
| C5 | NOP | NOP | NOP | NOP |
| C6 | NOP | NOP | NOP | bne $4,$7, L1 |

POLITECNICO
MILANO 1863

# Statically Scheduled Processors

- **Compilers** can use sophisticated algorithms for code scheduling to exploit ILP (Instruction Level Parallelism)
- **Basic Block:** a straight-line code sequence with no branches in/out except at the entry/exit point.
- **Problem:** the amount of parallelism available within a basic block is small (about 3 instructions in generic code)
  - **Example:** For typical MIPS programs the average branch frequency is between 15% and 25% ⊠ from 4 to 7 instructions execute between a pair of branches

POLITECNICO
MILANO 1863

# Statically Scheduled Processors (2)

- **Data dependences** can further limit the amount of ILP we can exploit within a **basic block** to much less than the average basic block size
  - True data dependences force sequential execution of instructions
- To obtain substantial performance enhancements, we must **exploit ILP across multiple basic blocks** (i.e. across branches).

POLITECNICO
MILANO 1863

# Main advantages of VLIW

- Good performance through extensive **compiler optimizations** to schedule the code statically (exploiting the intrinsic program parallelism)
  - The compiler can analyze the code on a **wider instruction window** than the hardware
  - The compiler has more time to analyze data dependencies and to extract more parallelism
- **Reduced hardware complexity** because the complexity is moved to the compiler level.
  - Smaller die area => Cheaper processor cost and lower power consumption
  - Easily extended to larger number of FUs
  - Instructions have fixed fields => simpler decode logic

POLITECNICO
MILANO 1863

# Open Challenges of VLIW

- Need for strong compiler technology
  - To detect and exploit parallelism
  - Need to manage parallelism beyond the basic block
- Code size increase
  - Explicit NOPs may be numerous
  - Possible solution: code compression techniques
    - Used in modern VLIWs, but require added circuitry
- Huge number of registers needed for register renaming
  - Complexity of register file to FU is increased
  - Possible solution: clustered VLIWs

POLITECNICO
MILANO 1863

# Open Challenges of VLIW (2)

- **Binary incompatibility => Low code portability**
  - Architectures with same ISA but different VLIW bundle size are binary incompatible.
  - Also, same ISA and same VLIW bundle size, but different architecture (number and types of FUs and FU latencies) requires code recompilation.
  - There is no fully satisfactory solution to this issue
  - *Just In Time Compilation* is a possible, if costly, solution → attempted in Transmeta Crusoe processor.
  - In most cases, VLIW are simply employed in embedded systems, where binary compatibility is less important

POLITECNICO
MILANO 1863

# Early VLIW Architectures

- Multiflow Trace Scheduling (1987)
  - Designed by Josh Fisher
  - Has several direct descendants
- Intel Itanium IA-64 EPIC → the only major general-purpose VLIW processor.
- STMicroelectronics ST200 (based on HP/STM Lx) → digital audio and video processing for multimedia and Nokia phones
- Texas Instruments TMS320C64x Digital Signal Processors
- AMD Radeon R600 series → unified shaders using VLIW cores
- Used as ASIPs (Application-Specific Instruction set Processors), Domain-specific Processors and as hardware accelerators.

POLITECNICO
MILANO 1863

# Conclusions

- VLIW architectures employ statically scheduled multiple issues to:
  - Reduced hardware complexity and power consumption;
  - Decreased CPI by exploiting ILP statically.
- VLIW architectures are more suitable for application-specific processors used in embedded and IoT systems
  - Multimedia processing
- VLIW architectures face significant limitations when dealing with legacy code, due to poor binary portability.
- VLIW architectures strongly depend on compiler quality.