UNIVERSITÀ
DEGLI STUDI DI TRIESTE

dipartimento
di ingegneria
e architettura

Pipeline organization

A.Carini – Microcontrollers

Pipeline

- A *pipeline* is obtained by performing fetching, decoding, and execution simultaneously on different instructions.
- In the most simple case, the instructions are consecutive.
- The pipeline is implemented by making autonomous the functional units of the CPU dedicated to the different phases, to allow them to work at the same time on different instructions without interfering.
- The pipeline organization allows to speed up considerably the CPU, but takes also to an increased circuit complexity, a more complex control. This translates to higher costs of the processor.
- It explain why, while known for a long time, it has been exploited only recently, thanks to the high production volumes and the reduced costs of design and production.

Pipeline

- It is used in many microcontrollers (µCs), based on RISC architectures, and in many DSPs.
- The pipeline complexity in these devices in much smaller than in GPPs. There are two reasons:
  - Industrial devices have stringent cost constraints.
  - Extreme pipeline organizations can make efficient and compact coding much more difficult.

Pipeline concept

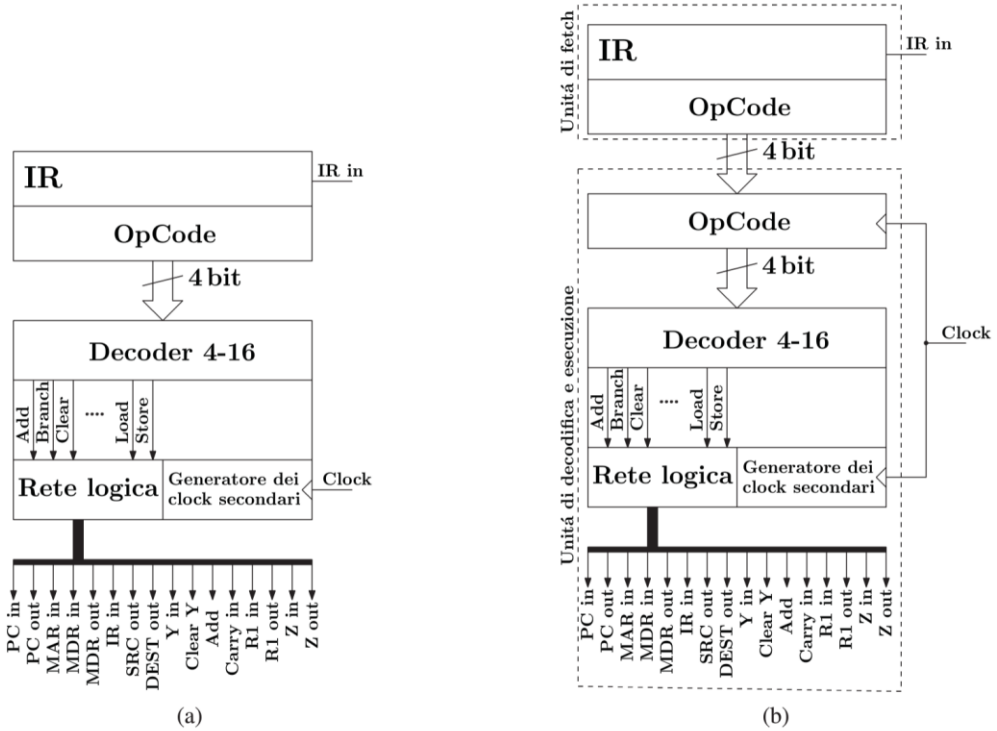


Figura 6.1: Evoluzione di una unità di controllo da organizzazione parallela, o multiciclo (a), a pipeline a due stadi (b).

Pipeline concept

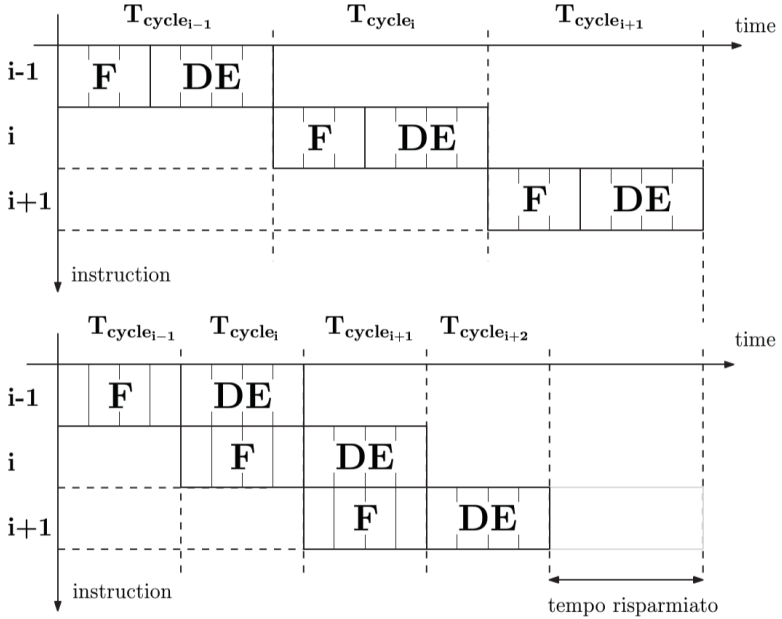


Figura 6.2: Temporizzazione di una sequenza di tre istruzioni in una organizzazione multiciclo senza pipeline (sopra) e con una pipeline a 2 stadi (sotto). Si nota il risparmio di tempo realizzato.

Multi-stage pipeline

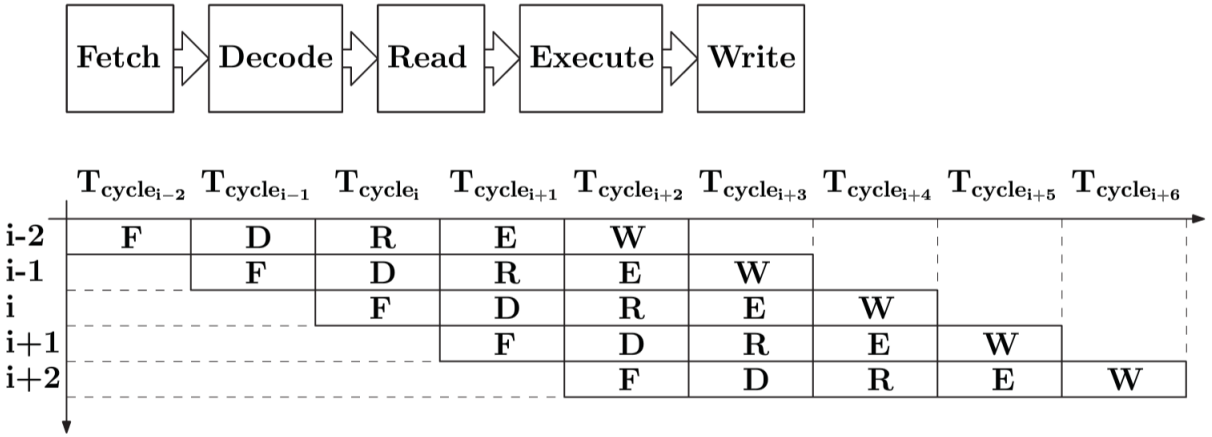


Figura 6.3: Schema logico a blocchi di una pipeline a 5 stadi con il relativo diagramma temporale.

- A multi-stage pipeline acts on a larger block of instructions. There is a larger delay before steady-state operation, but also a larger improvement in the number of operations executed per unit time.

Multi-stage pipeline

- Let us consider:
  - Fetch 5 ns
  - Decode 3 ns
  - Operands read 5 ns
  - Execute 2 ns
  - Result write 5ns
- Then a single cycle execution would require a clock period

$$T_{clock_{SC}} = 5 + 3 + 5 + 2 + 5 = 20\,\text{ns},$$

- With a multi-cycle organization it could work at

$$T_{clock_{MC}} = 5\,\text{ns},$$

- Which correspond to a maximum machine cycle of

$$T_{macchina_{MAX}} = 5 \times 5\,\text{ns} = 25\,\text{ns},$$

Acceleration

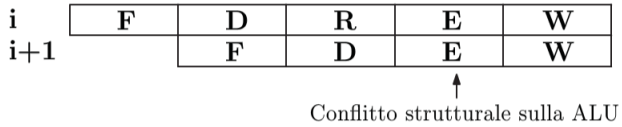
- In a pipeline, the clock period is determined by the slowest functional unit (the slowest pipeline stage) exactly as in multi-cycle organizations.
- Without conflicts, called *hazards,* in every clock period it executes an instruction, as in single-cycle organizations.
- In ideal conditions, a pipeline with *n* stages can accelerate the processor speed by a factor *n,* in comparison with a multi-cycle organization at same conditions.
- In reality, without hazards, the time required for completing *N* instructions is $(n + N - 1) \cdot T_{clock}$ and the number of operation per unit time is

$$NMI_{pipeline} = \frac{N}{(n + N - 1) \cdot T_{clock}}$$

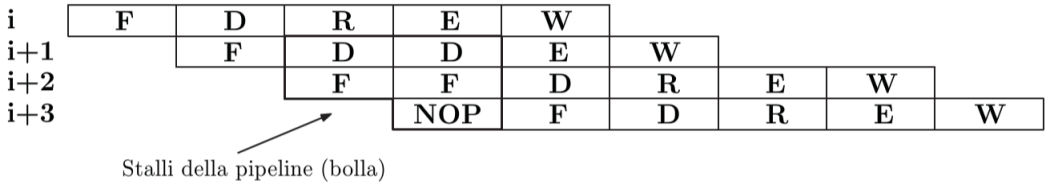
- In µCs and DSPs the number of stages is typically 3 (fetch, decode, execute), but there exists pipelines with 5 stages (e.g., TMS320C54x)

Structural hazards

- Two different instructions of a program could require the use of the same pipeline stage at the same clock cycle, determining a *structural hazard.*



- Avoiding structural hazards requires modifying the HW organization using functional units redundancy. Costs increase and a compromise is necessary.
- Most simple solution to structural hazards involves the introduction of delays in the pipeline control, stalling operations with a technique called *interlocking*.



Other hazards

- There exist also other types of hazards:
  - *Data hazards*
    - When an instruction uses the results of a previous instruction that has not yet concluded.
  - *Control hazards*
    - When a program has *jumps* or *branches.*

Data hazards

- Data hazards between an instruction <i> and a following instruction <j> can be at least of *three types:*
  - <j> try to read a data written by <i> before <i> is completed.
  - <j> try to overwrite a data read by <i> before <i> has read it.
  - <j> try to write a data in the same position of <i> before <i> is completed.

$$
\begin{aligned}
ADD \quad & R1,\ R1,\ numero \quad \Leftrightarrow \quad R1 \leftarrow R1 + M[numero] \\
ADD \quad & R2,\ R1,\ R2 \quad \Leftrightarrow \quad R2 \leftarrow R1 + R2
\end{aligned}
$$

Figura 6.5: Conflitto sui dati generato dalle due istruzioni del codice (6.2).

Data hazards

- In the example, the data hazard can be solved by delaying the operand read, i.e., by interlocking.
- Alternatively, the data hazard can be managed with the following strategies:
  - *Bypass:* the data required by <j> is taken whenever ready, even before being written.
  - *Overlap:* operand reading and writing can be performed on the same cycle, because it is divided in two phases, the first for writing the second for reading.
  - *Reordering:* the sequence of operations is altered (by the compiler or programmer) in order to avoid the hazard.

Data hazards

$$
\begin{aligned}
ADD \quad & R1,\ R1,\ numero \quad \Leftrightarrow \quad R1 \leftarrow R1 + M[numero] \\
ADD \quad & R3,\ R3,\ R2 \quad \Leftrightarrow \quad R3 \leftarrow R3 + R2 \\
ADD \quad & R2,\ R1,\ R2 \quad \Leftrightarrow \quad R2 \leftarrow R1 + R2
\end{aligned}
$$

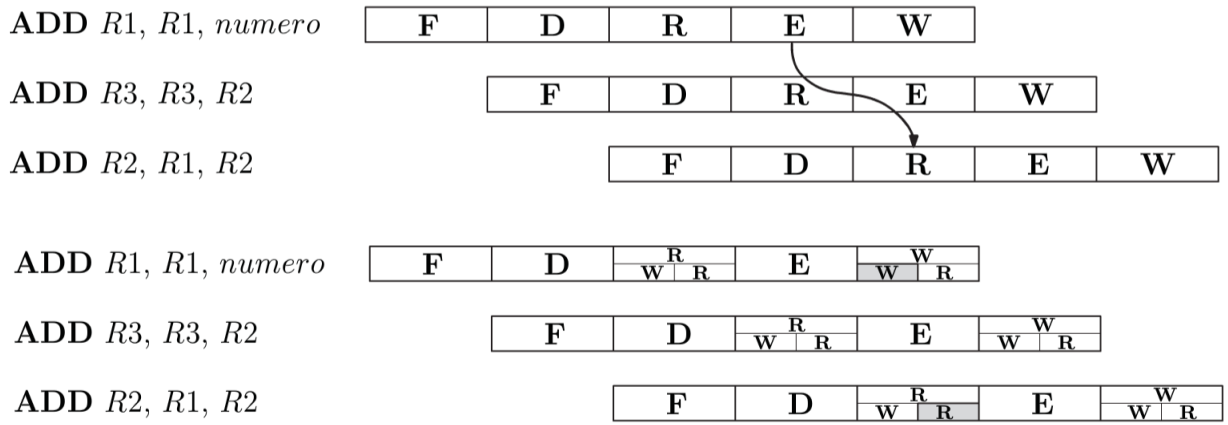


Figura 6.6: Risoluzione del conflitto sui dati grazie ad "internal forwarding" (in alto) e sovrapposizione (in basso).

Control hazards

- In jumps or branches, the instructions that follow must be *discarded,* since they belong to a piece of code that has not to be executed.
- This is known as a control hazard and creates a *bubble* in the pipeline.

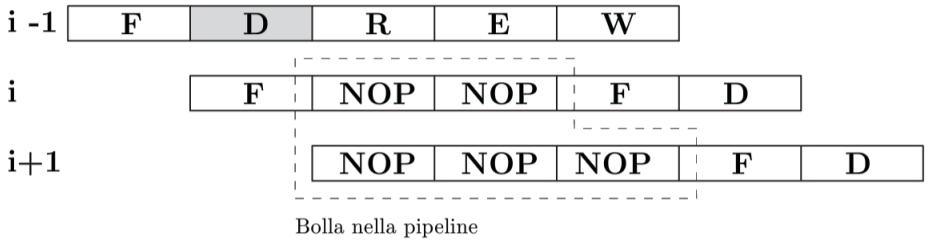


Figura 6.7: Conflitto di controllo con generazione di una bolla di stalli della pipeline.

Control hazards

- To avoid these hazards, some processors allow *delayed jumps*, which consist in completing the pipeline instructions before executing the jump. It requires anticipating the jump in the program.

$$
\begin{array}{lll}
LDR \quad R1,\, [R2] & \Leftrightarrow & R1 \leftarrow M[R2] \\
ADD \quad R3,\, R3,\, R4 & \Leftrightarrow & R3 \leftarrow R3 + R4 \\
SUB \quad R5,\, R1,\, R3 & \Leftrightarrow & R5 \leftarrow R1 - R3 \\
JMP \quad altrove & \Leftrightarrow & PC \leftarrow altrove
\end{array}
$$

$$
\begin{array}{lll}
LDR \quad R1,\, [R2] & \Leftrightarrow & R1 \leftarrow M[R2] \\
JMP \quad altrove & \Leftrightarrow & PC \leftarrow altrove \\
ADD \quad R3,\, R3,\, R4 & \Leftrightarrow & R3 \leftarrow R3 + R4 \\
SUB \quad R5,\, R1,\, R3 & \Leftrightarrow & R5 \leftarrow R1 - R3
\end{array}
$$

Control hazards

- In the most simple µCs and DSPs, the jump execution is always performed with interlocking, i.e., with the generation of a bubble.
- Some DSPs implements *only* delayed jumps, others implements both delayed jumps and jumps with interlocking, according to the coder preferences.
- In *branches,* it is better to avoid the immediate introduction of a bubble, because, if the branch condition is false, we have to execute all instructions of the pipeline. Only when the condition is true, we will have to eliminate all the pipeline instructions that follows.
- *Interrupts* also produce effects similar to jumps. They require a sudden change of context for servicing the *interrupt subroutine.*

Pipeline acceleration

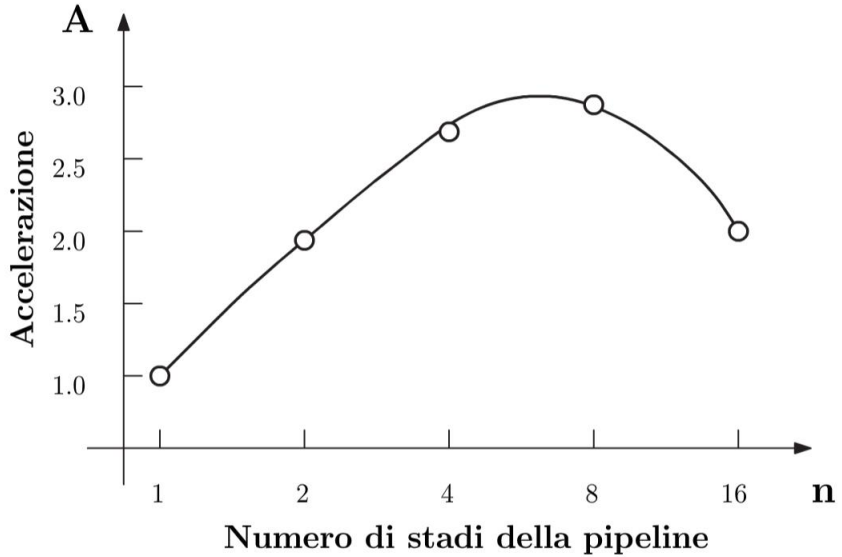


Figura 6.8: Diagramma che correla l'incremento di prestazioni misurabile al numero di stadi n (profondità) di una pipeline. Si noti la taratura *logaritmica* dell'asse delle ascisse.

Pipeline acceleration

- Factors that limit the performance improvement offered by a pipeline are
  1. *Data hazards*: the larger the number of stages, the larger is the penalty caused by an hazard.
  2. *Delay in jump execution:* the larger the number of stages, the larger the delay.
  3. *CPU complexity:* it forces to reduce the clock frequency, due to the pipeline support structures.

Pipeline in ARM processors

- ARM processors exploit a pipeline with three stages, *F – D – E* , corresponding to Fetch, Decode, and Execute.
- The machine cycle coincides with the clock period.
- The pipeline allows managing control hazards, due to jumps, and structural hazards, due to memory read/write operations.
- The pipeline can stall the instructions in the stages, or can remove them in case of jumps.

Pipeline in ARM processors

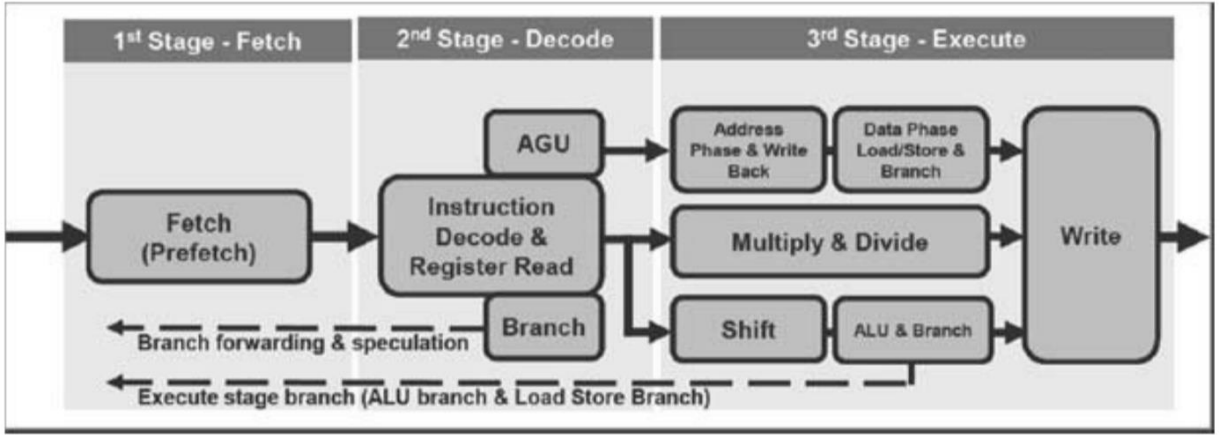


Figura 6.9: Pipeline nei processori ARM: si compone di tre stadi, Fetch, Decode e Execute, F-D-E.

Pipeline in ARM processors

Ciclo macchina	1	2	3	4	5	6	7	8	9
Operazione									
SUB	F	D	E						
AND		F	D	E					
ORR			F	D	E				
ADD				F	D	E			
ORR					F	D	E		
EOR						F	D	E	

Ciclo macchina		1	2	3	4	5	6	7	8	9
Indirizzo	Operazione									
0x80A0	BX r3	F	D	E						
0x80A2	ADD		F	D						
0x80A4	ORR			F						
0x80EC	AND				F	D	E			
0x80EE	ORR					F	D	E		
0x80F0	EOR						F	D	E	

Pipeline in ARM processors

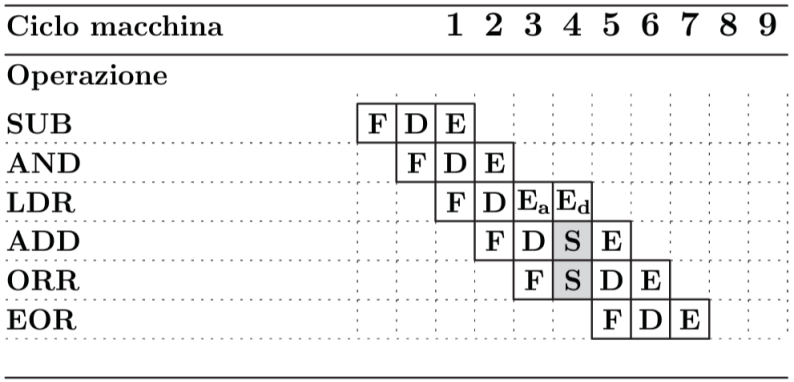


Figura 6.10: Esempio di funzionamento di una pipeline F-D-E per processore ARM. Dall'alto verso il basso: funzionamento imperturbato, senza conflitti; conflitto di controllo dovuto ad una istruzione di diramazione, BX, con generazione di una bolla; conflitto strutturale, con introduzione di un ciclo di stallo S, dovuto a una istruzione di lettura dalla memoria, LDR.

Advanced DSP organizations

- DSPs of the last generation introduce often solutions taken from GPPs, like superscalar (or vector) architectures.
- These solutions consist in replicating (2 to 4 times) the functional units (ALU and AGU) of the CPU to allow the *parallel execution of many instructions*.
- The corresponding control strategies consider also:
  - Branch condition prediction,
  - Advanced memory managements (with DRAM and multilevel cache);
  - Dynamic pipeline management.
- The instruction set is typically RISC, but SIMD architectures are also frequent.

Advanced DSP organizations

- The dynamic pipeline management allows to fetch and execute instructions in an order different from the code sequence.
- The typical structure of a dynamic pipeline includes three units:
  - The fetch and decode unit (FDU).
  - The execution unit (EXU).
  - The write-back unit (WBU).
- The first and last follow the natural instruction order, the second not.
- Instructions are fetched in blocks by FDU; are distributed to different ALU by the EXU, and executed in parallel, without considering their logic order unless the WBU detect hazards. WBU manages write back and provides a coherent output.

Example

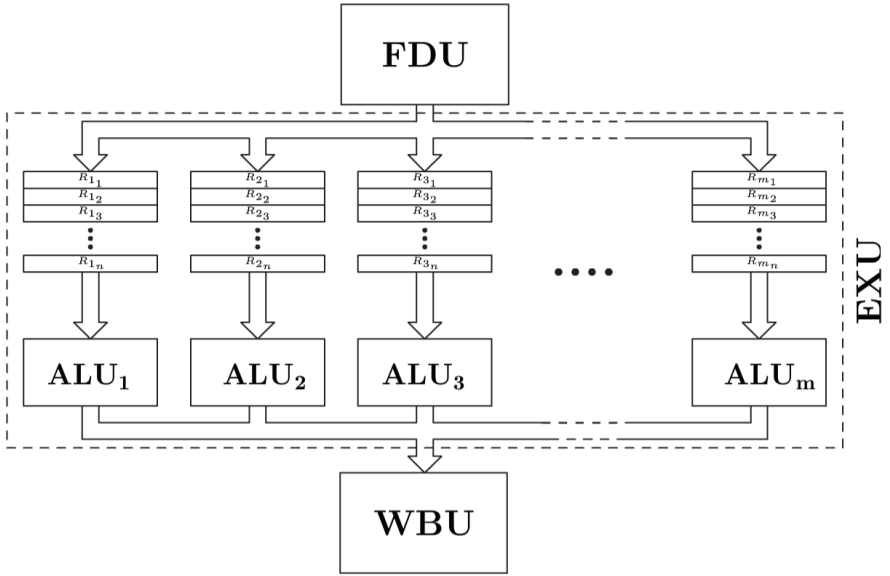


Figura 6.11: Esempio di pipeline dinamica. Il prelievo e la decodifica coinvolgono molte istruzioni simultaneamente. L'unità di esecuzione esegue le istruzioni in parallelo e risolve gli eventuali conflitti sui dati ritardando solo le istruzioni coinvolte per il tempo minimo necessario. L'unità di scrittura memorizza i risultati una volta che essi siano assestati.

See:

- Simone Buso, «Introduzione alle applicazioni industriali di Microcontrollori e DSP» Società editrice Esculapio, 2018
  - Chapter 6