Instruction Level Parallelism (ILP) is used to refer to the architecture in which multiple operations can be performed parallelly in a particular process, with its own set of resources – address space, registers, identifiers, state, and program counters. It refers to the compiler design techniques and processors designed to execute operations, like memory load and store, integer addition, and float multiplication, in parallel to improve the performance of the processors.

Examples of architectures that exploit ILP are VLIWs and superscalar Architecture. ILP processors have the same execution hardware as RISC processors. The machines without ILP have complex hardware which is hard to implement. A typical ILP allows multiple-cycle operations to be pipelined.

Example: Suppose, 4 operations can be carried out in a single clock cycle. So there will be 4 functional units, each attached to one of the operations, branch unit, and common register file in the ILP execution hardware. The sub-operations that can be performed by the functional units are Integer ALU, Integer Multiplication, Floating Point Operations, Load, and Store. Let the respective latencies be 1, 2, 3, 2, 1.

Let the sequence of instructions by

1. y1 = x1*1010
2. y2 = x2*1100
3. z1 = y1+0010
4. z2 = y2+0101
5. t1 = t1+1
6. p = q*1000
7. clr = clr+0010
8. r = r+0001

Sequential Record of Execution vs. Instruction-level Parallel Record of Execution

CYCLE	OPERATION
1	y1 = x1*1010
2	nop
3	nop
4	y2 = x2*1100
5	nop
6	nop
7	z1 = y1+0010
8	z2 = y2+0101
9	t1 = t1+1
10	p = q*1000
11	clr = clr+0010
12	r = r+0001

Fig. a

CYCLE	INT ALU	INT ALU	FLOAT ALU	FLOAT ALU
1	t1 = t1+1	clr = clr+0010	y1 = x1*1010	y2 = x2*1100
2	r = r+0001		p = q*1000	
3	nop			
4	z1 = y1+0010	z2 = y2+0101		

Fig. b

Fig a – Sequential execution of operations. Fig. b – shows the use of the ILP in improving the performance of the processor.

The 'nop's or the 'no operations' in the above diagram is used to show the idle time of the processor. Since the latency of floating-point operations is 3, hence multiplications take 3 cycles and the processor has to remain idle for that time period. However, in Fig. b processor can utilize those nop's to execute other operations while previous ones are still being executed. While in sequential execution, each cycle has only one operation being executed, in the processor with ILP, cycle 1 has 4 operations, and cycle 2 has 2 operations. In cycle 3 there is 'nop' as the next two operations are dependent on the first two multiplication operations. The sequential processor takes 12 cycles to execute 8 operations whereas the processor with ILP takes only 4 cycles.

Instruction Level Parallelism (ILP) Architecture

Instruction Level Parallelism is achieved when multiple operations are performed in a single cycle, which is done by either executing them simultaneously or by utilizing gaps between two successive operations that are created due to the latencies. Now, the decision of when to execute an operation depends largely on the compiler rather than the hardware. However, the extent of the compiler's control depends on the type of ILP architecture where information regarding parallelism given by the compiler to hardware via the program varies.

Classification of ILP Architectures

The classification of ILP architectures can be done in the following ways –

- **Sequential Architecture:** Here, the program is not expected to explicitly convey any information regarding parallelism to hardware, like superscalar architecture.
- **Dependence Architectures:** Here, the program explicitly mentions information regarding dependencies between operations like dataflow architecture.
- **Independence Architecture:** Here, programme m gives information regarding which operations are independent of each other so that they can be executed instead of the 'nops.

In order to apply ILP, the compiler and hardware must determine data dependencies, independent operations, and scheduling of these independent operations, assignment of functional units, and register to store data.

Advantages of Instruction-Level Parallelism

- **Improved Performance:** ILP can significantly improve the performance of processors by allowing multiple instructions to be executed simultaneously or out-of-order. This can lead to faster program execution and better system throughput.
- **Efficient Resource Utilization:** ILP can help to efficiently utilize processor resources by allowing multiple instructions to be executed at the same time. This can help to reduce resource wastage and increase efficiency.
- **Reduced Instruction Dependency:** ILP can help to reduce the number of instruction dependencies, which can limit the amount of instruction-level parallelism that can be exploited. This can help to improve performance and reduce bottlenecks.
- **Increased Throughput:** ILP can help to increase the overall throughput of processors by allowing multiple instructions to be executed simultaneously or out-of-order. This can help to improve the performance of multi-threaded applications and other parallel processing tasks.

Disadvantages of Instruction-Level Parallelism

- **Increased Complexity:** Implementing ILP can be complex and requires additional hardware resources, which can increase the complexity and cost of processors.
- **Instruction Overhead:** ILP can introduce additional instruction overhead, which can slow down the execution of some instructions and reduce performance.
- **Data Dependency:** Data dependency can limit the amount of instruction-level parallelism that can be exploited. This can lead to lower performance and reduced throughput.

- **Reduced Energy Efficiency:** ILP can reduce the energy efficiency of processors by requiring additional hardware resources and increasing instruction overhead. This can increase power consumption and result in higher energy costs.