1. Explain instruction level parallelism and its difficulties in implementing it?

Instruction-level parallelism (ILP) is a measure of how many of the operations in a computer program can be performed simultaneously. The potential overlap among instructions is called instruction level parallelism.

ILP is to Achieve not only instruction overlap, but the actual execution of more than one instruction at a time through dynamic scheduling and how to maximize the throughput of a processor. For typical RISC processors, instructions usually depend on each other too and as a result the amount of overlap is limited.

Instruction level dependencies and hazards during ILP

If two instructions are not dependent, then they can execute simultaneously— assuming sufficient resources that is no structural hazards. if one instruction depends on another, they must execute in order though they may still partially overlap.

1. Data dependencies

Data dependence means that one instruction is dependent on another if there exists a chain of dependencies between them. . Compilers can be of great help in detecting and scheduling around these sorts of hazards; hardware can only resolve these dependencies with severe limitations.

2. Name Dependencies

A name dependency occurs when two instructions use the same register or memory location, called a name, but there is no flow of data between them.

- **Anti-dependence** occurs when j writes a register/memory that i reads.
- **Output dependence** occurs when i and j write to the same register/memory location

The name used in the instruction is changed so that they do not conflict. This technique is known as register renaming (uses temp registers).

3. Data Hazards

Data dependency between instructions and they are close enough to cause the pipeline to stall three types of data hazards:

read after write (RAW)—j tries to read a source before i writes it—this is the most common type and is a true data dependence;

- example sequence logic 1. i=i+1; 2. j=i+1

write after write (WAW)—j tries to write an operand before it is written by i—this corresponds to the output dependence;

- example sequence logic 1. i=i+1; 2. print i; 3. i=j+1

write after read (WAR)—j tries to write a destination before i has read it—this corresponds to an anti-dependency

- example sequence logic 1.read i ; 2. i=j+1

4. Control Dependencies

A control dependency determines the order of an instruction i with respect to a branch,

```
if (p1)
        S1
if (p2)
        S2
```

S1 is control dependent on p1 and S2 is control dependent on p2 speculatively executed instructions do not affect the program state until branch result is determined. This implies that the instructions executed speculatively must not raise an exception or otherwise cause side effects.

Implementation of ILP and overcoming hazards or dependencies.

To implement ILP, 3 methods are used to avoid any delay during ILP

1. score boarding.
2. Tomasulo's solution for dynamic scheduling.
3. Branch prediction.

1. Score boarding.

Instructions to be issued when they are ready, not necessarily in order, hence out of-order execution. To implement out-of-order issue we need to split the instruction decode phase into two:

1. **Issue**—decode instructions and check for structural hazards;
2. **Read operands**—wait until no data hazards obtain then read the operands and start executing.

It dynamically schedules the pipeline. **instructions must pass through the issue phase in order;**

This method can stall or bypass each other, in the read operands phase and enter, or even, complete execution in out of order manner.

Example

CDC6600 used a scoreboard, the goal of a scoreboard is to maintain processor throughput of one instruction per clock cycle (no structural hazard). If the next instruction would stall, then store it on a queue and start with a later instruction and takes full responsibility for instruction issue and execution. It uses as many as 16 separate functional units.

2. Tomasulo's solution for dynamic scheduling.

Executing instructions only when operands are available, waiting instruction is stored in a reservation station. Reservation stations keep track of pending instructions (RAW). WAW can be avoided using Register renaming.(80 reg).

Tomasulo architecture executes instructions in three phases; each phase may take more than one clock cycle:

Three Steps:

1. Issue
    - Get next instruction from FIFO queue
    - If available RS, issue the instruction to the RS with operand values if available
    - If operand values not available, stall the instruction
2. Execute
    - When operand becomes available, store it in any reservation stations waiting for it
    - When all operands are ready, issue the instruction
    - Loads and store maintained in program order through effective address
    - No instruction allowed to initiate execution until all branches that proceed it in program order have completed
3. Write result
    - Write result on CDB into reservation stations and store buffers
    - (Stores must wait until address and value are received)

3. Branch prediction.

It uses predictor is a simple saturating **n-bit counter**. Each time a particular branch is taken its entry is incremented otherwise it is decremented.

If the most significant bit in the counter is set then predict that the branch is taken.

Limitations of ILP

1. An instruction stream needs to be run on an ideal processor with no significant limitations.
2. The ideal processor always predicts branches correctly, has no structural hazards.
3. This eliminates all control and name dependencies. (only data dependencies)
4. Theoretically it is possible for the last dynamically executed instruction in the program to be scheduled on the first cycle.

3. Parallel processing challenges

- Parallel processing is the simultaneous use of more than one CPU to execute a program or multiple computational threads.
- Ideally, parallel processing makes programs run faster because there are more engines (CPUs or cores) running it.
- A parallel computer (or multiple processor system) is a collection of communicating processing elements (processors) that cooperate to solve large computational problems fast by dividing such problems into parallel tasks, exploiting Thread-Level Parallelism (TLP).

Advantages:

- Faster execution time, so higher throughput.

Disadvantages:

- More hardware required, also more power requirements.
- Not good for low power and mobile devices.

Challenges in parallel processing

- Connecting your CPUs
  - Dynamic vs Static—connections can change from one communication to next
  - Blocking vs Nonblocking—can simultaneous connections be present?
  - Connections can be complete, linear, star, grid, tree, hypercube, etc.
- Bus-based routing
  - Crossbar switching—impractical for all but the most expensive super-computers
  - 2X2 switch—can route inputs to different destinations
- Dealing with memory
- Various options:
  - Global Shared Memory
  - Distributed Shared Memory
  - Global shared memory with separate cache for processors
- Potential Hazards:
  - Individual CPU caches or memories can become out of synch with each other. "Cache Coherence"
  - Solutions:
    - UMA/NUMA machines
    - Snoopy cache controllers
    - Write-through protocols