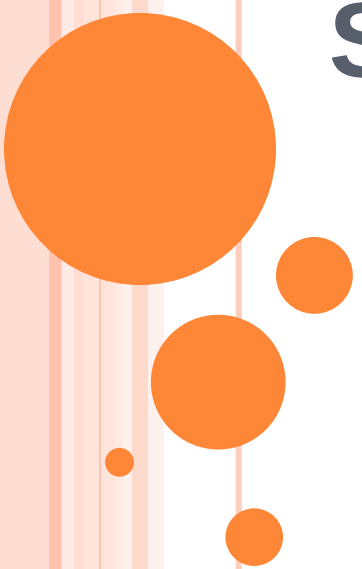
Superscalar processor



Scalar processor

- **Scalar processors** is classified as SISD processor(single instruction,single data) .A scalar processor processes only one datum at a time.
- In a scalar organization, a single pipelined **functional unit** exists for:
- • Integer operations;
- • And one for floating-point operations;
- **Functional unit:**
- • Part of the CPU responsible for calculations

Superscalar processor

- A superscalar processor is a CPU that implements a form of parallelism called instruction-level parallelism within a single processor.
- A superscalar CPU can execute more than one instruction per clock cycle. Because processing speeds are measured in clock cycles per second (megahertz), a superscalar processor will be faster than a scalar processor
- **Ability to execute instructions in different pipelines:**
- • Independently and concurrently;

Pipeline problems:

- Pipeline concept already introduced some problems.
- A **resource hazard** exists when the pipeline required for an instruction is unavailable due to a previous instruction.

- **Data hazards:**
- occur when the **pipeline** changes the order of read/write accesses to operands so that the order differs from the order seen by sequentially executing instructions on the unpipelined machine.

- **WAR:**
- A write after read (WAR) data hazard represents a problem with concurrent execution.
- **Example**
- i1. R4 <- R1 + **R5**
- i2. **R5** <- R1 + R2
- In any situation with a chance that i2 may finish before i1 (i.e., with concurrent execution),

- **WAW;**
- A write after write (WAW) data hazard may occur in a concurrent execution environment.
- **Example**
- i1. **R2** <- R4 + R7
- i2. **R2** <- R1 + R3

- The write back (WB) of i2 must be delayed until i1 finishes executing.

- **RAW:**
- A read after write (RAW) data hazard refers to a situation where an instruction refers to a result that has not yet been calculated or retrieved
- For example:
- i1. **R2** <- R1 + R3
- i2. R4 <- **R2** + R3

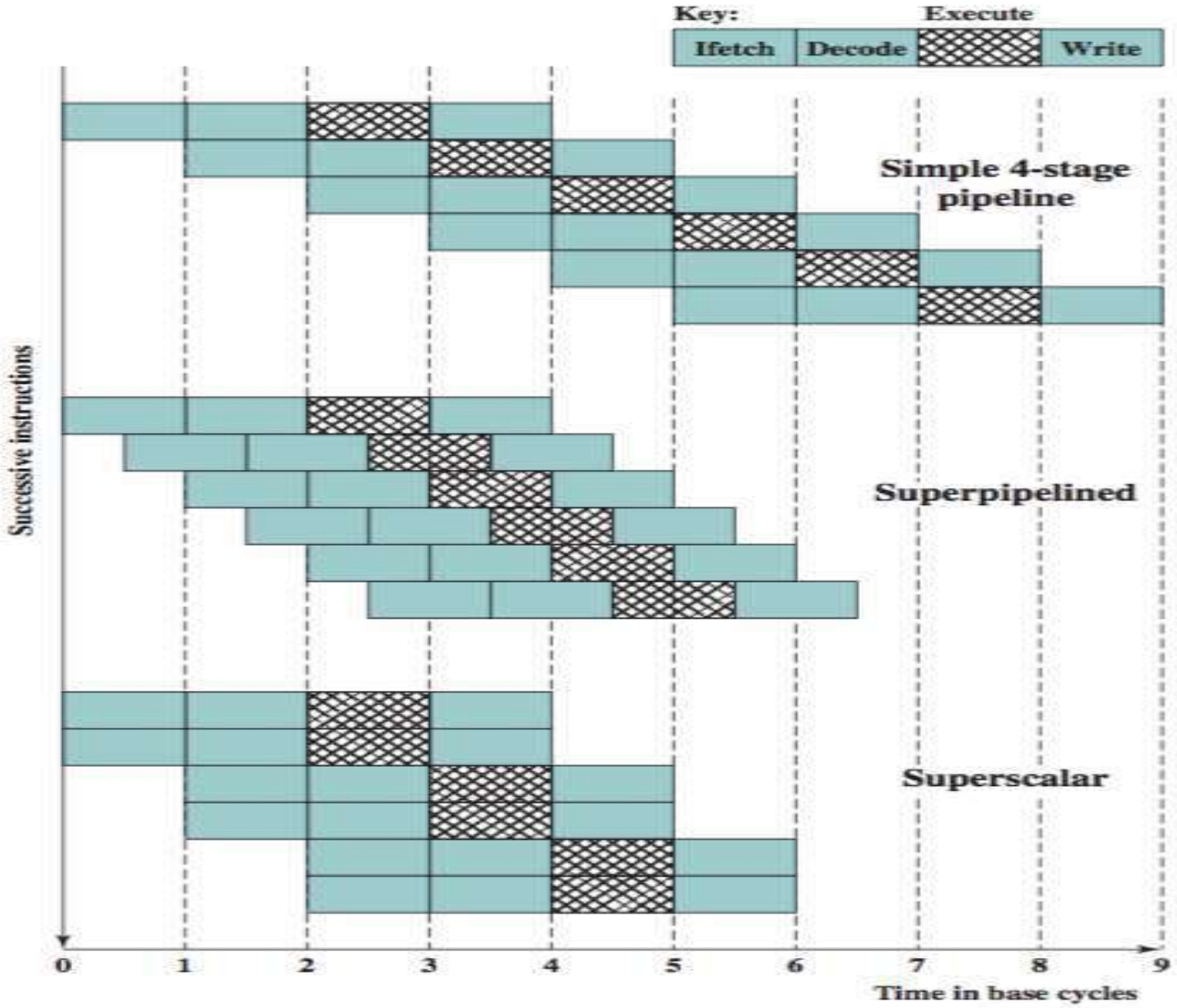
Solution of problems:

- Responsibility of the hardware and the compiler to:
- Assure that parallel execution does not violate program intent.
- Tradeoff between performance and complexity.

Superscalar vs. superpipelined

- Superscalar machines can issue several instructions per cycle. Superpipelined machines can issue only one instruction per cycle, but they have cycle times shorter than the time required for any operation.
- Both of these techniques exploit instruction-level parallelism, which is often limited in many applications. Superpipelined machines are shown to have better performance and less cost than superscalar machines

- Consider the following execution scenario



How much time is required for the normal pipeline approach? Why?

How much time is required for the superpipeline approach? Why?

How much time is required for the superscalar approach? Why?

- From the previous figure, **base pipeline**:
- • Issues one instruction per clock cycle;
- • Can perform one pipeline stage per clock cycle;
- • Although several instructions are executing concurrently:
- • Only one instruction is in its execution stage at any one time.
- • Total time to execute 6 instructions: 9 cycles.

- From the previous figure, **superpipelined** implementation:
- • Capable of performing two pipeline stages per clock cycle;
- • Each stage can be split into two non-overlapping parts:
- • With each executing in half a clock cycle;
- • Total time to execute 6 instructions: 6.5 cycles.

- From the previous figure, **superscalar** implementation:
- • Capable of executing two instances of each stage in parallel;
- • Total time to execute 6 instructions: 6 cycles