COMPUTER SYSTEMS ARCHITECTURE

Super Scalar Pipeline Design (Malayalam)

As Per KTU Syllabus

Super Scalar Pipeline Design

- Pipeline Design Parameter:

Table 6.1 *Design Parameters for Pipeline Processors*

Machine type	*Scalar base machine of k pipeline stages*	*Superscalar machine of degree m*
Machine pipeline cycle	1 (base cycle)	1
Instruction issue rate	1	*m*
Instruction issue latency	1	1
Simple operation latency	1	1
ILP to fully utilize the pipeline	1	*m*

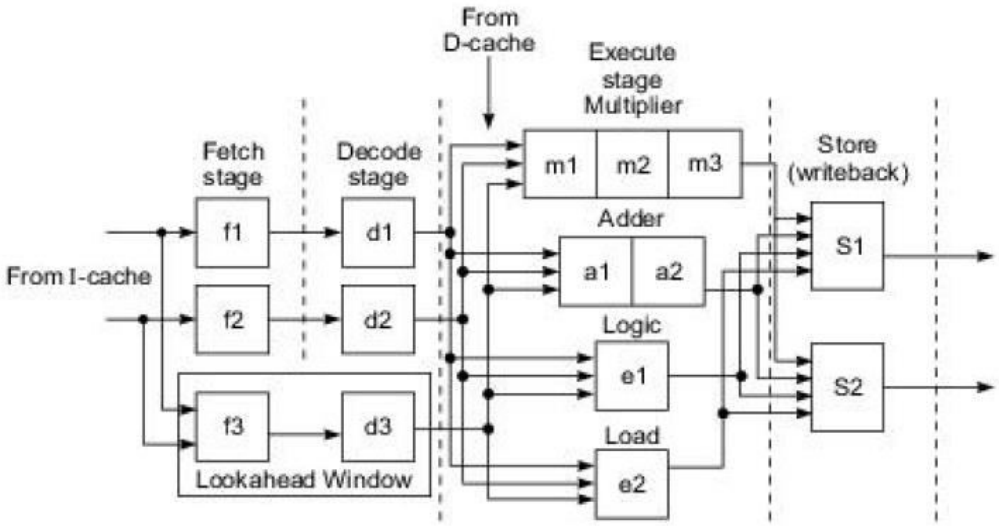
Note: All timing is relative to the base cycle for the scalar base machine. ILP: Instruction level parallelism.

- The pipeline cycles for the scalar base processor is assumed to be 1 time unit, called the base cycle.
- The instruction level Parallelism (ILP) is the maximum number of instructions that can be simultaneously executed in the pipeline.
- For the base processor, all of these parameters have a value of 1.
- All processor types are designed relative to the base processor.
- The ILP is needed to fully utilize a pipeline processor.

- For a superscalar machine of degree m, in instructions are issued per cycle and the ILP should be m in order to fully utilize the pipeline.
- As a matter of fact, the scalar base processor can be considered a degenerate case of a superscalar processor of degree 1

Superscalar Pipeline Structure

- In an m-issue superscalar processor, the instruction decoding and execution resources are increased to form effectively m pipelines operating concurrently.
- At some pipeline stages, the functional units may be shared by multiple pipelines



(a) A dual-pipeline, superscalar processor with four functional units in the execution stage and a lookahead window producing out-of-order issues

2-issue superscalar pipeline processor

- The processor can issue two instructions per cycle if there is no resource conflict and no data dependence problem.
- There are essentially two pipelines in the design.
- Both pipelines have four processing stages labelled fetch, decode, execute, and store, respectively.
- Each pipeline essentially has its own fetch unit. decode unit. and store unit.

- we assume that each pipeline stage requires one cycle, except the execute stage which may require a variable number of cycles.
- Four functional units, multiplier, adder, logic unit, and load unit, are available for use in the execute stage.
- These functional units are shared by the two pipelines on a dynamic basis.
- The multiplier itself has three pipeline stages,
- the adder has two stages, and
- the others each have only one stage.

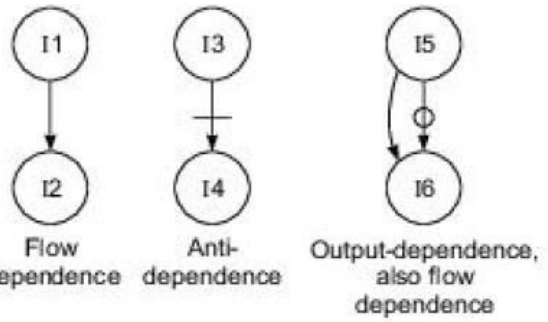
- There is a lookahead window with its own fetch and decoding logic.
- This window is used for instruction lookahead in case out-of-order instruction issue is desired to achieve better pipeline throughput.

Data Dependence:

- To schedule instructions through one or more pipelines, these data dependences must not be violated.
- Otherwise, erroneous results may be produced.

Program order

I1.	Load	R1,	A	/R1 ← Memory (A)/
I2.	Add	R2,	R1	/R2 ← (R2) + (R1)/
I3.	Add	R3,	R4	/R3 ← (R3) + (R4)/
I4.	Mul	R3,	R5	/R4 ← (R4) * (R5)/
I5.	Comp	R6		/R6 ← $(\overline{R6})$ /
I6.	Mul	R6,	R7	/R6 ← (R6) * (R7)/

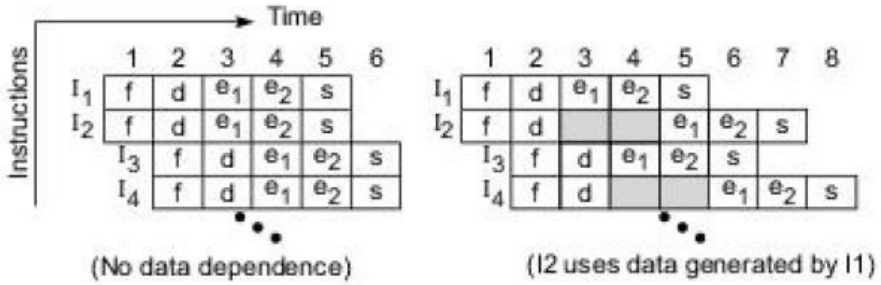


(b) A sample program and its dependence graph, where I2 and I3 share the adder and I4 and I6 share the multiplier

Fig. 6.28 A two-issue superscalar processor and a sample program for parallel execution

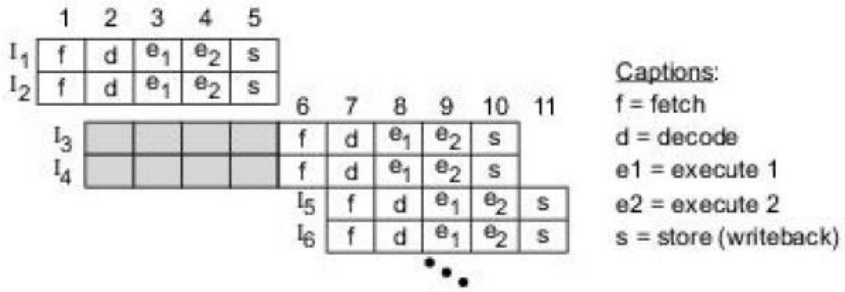
Pipeline Stalling

- a pipeline stall is a delay in execution of an instruction in order to resolve a hazard.
- This is a problem which may seriously lower pipeline utilization.
- Proper scheduling avoids pipeline stalling.
- The problem exists in both scalar and superscalar processors.
- However, it is more serious in a superscalar pipeline.
- Stalling can be caused by data dependences or by resource conflicts among instructions already in the pipeline or about to enter the pipeline.



(a) Data dependence stalls the second pipeline in shaded cycles

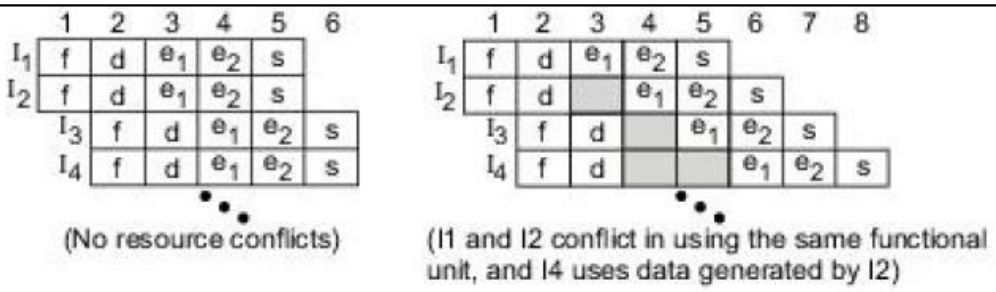
- shows the case of no data dependence on the left and flow dependence I1→I2 on the right.
- Without data dependence all pipeline stages are utilised without idling.
- With dependence, instruction I2 entering the second pipeline must wait for two cycles before entering the execution stages.
- This delay may also pass to the next instruction I4 entering the pipeline.



(b) Branch instruction I2 causes a delay slot of length 4 in both pipelines

- show the effect of branching (instruction I2).
- A delay slot of four cycles results from a branch taken by I2 at cycle 5.
- Therefore, both pipelines must be flushed before the target instructions I3 and I4 can enter the pipelines from cycle 6.
- Here, delayed branch or other amending actions are not taken.

13



(c) Resource conflicts and data dependences cause the stalling of pipeline operations for some cycles

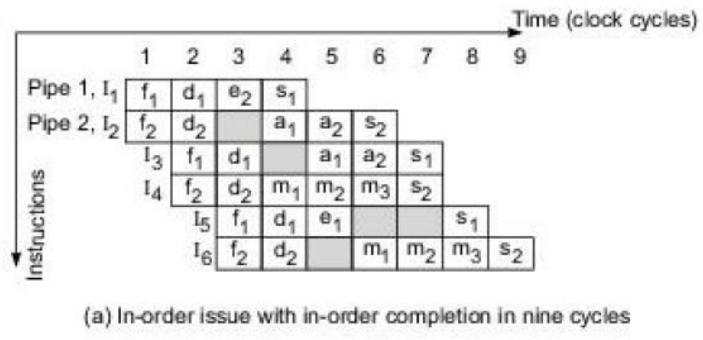
- show a combined problem involving both resource conflict and data dependence.
- Instructions I1 and I2 need to use the same functional unit, and I2→I4 exists.
- The net effect is that I2 must be scheduled one cycle behind because the two pipeline stages (e1 and e2) of
- the same functional unit must be used by I1 and I2 in an overlapped fashion.
- For the same reason, I3 is also delayed by one cycle.
- Instruction I4 is delayed by two cycles due to the flow dependence on I2.

14

Superscalar Pipeline Scheduling

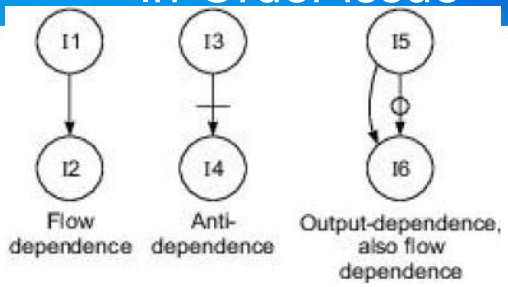
- Three scheduling policies are introduced below

1. In-order issue with in-order completion
2. In-order issue and out-of-order completion
3. Out of-order issue and out-of-order completion



(a) In-order issue with in-order completion in nine cycles

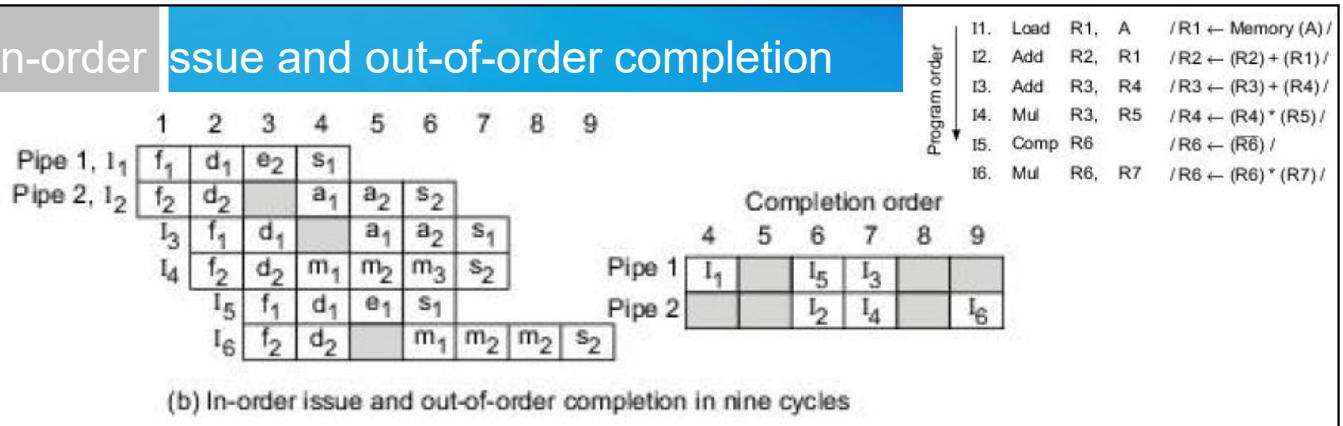
In-Order issue



- a schedule for the six instructions being issued in program order I1, I2, I6.
- Pipeline 1 receives I1, I3, and I5, and
- pipeline 2 receives I2, I4, and I6 in three consecutive cycles.
- Due to I1→I2, I2 has to wait one cycle to use the data loaded in by I1.

- I3 is delayed one cycle for the same adder used by I2.
- I6 has to wait for the result of I5 before it can enter the multiplier stages.
- In order to maintain in-order completion I5 is forced to wait for two cycles to come out of pipeline 1.
- In total, nine cycles are needed and five idle cycles (shaded boxes) are observed.

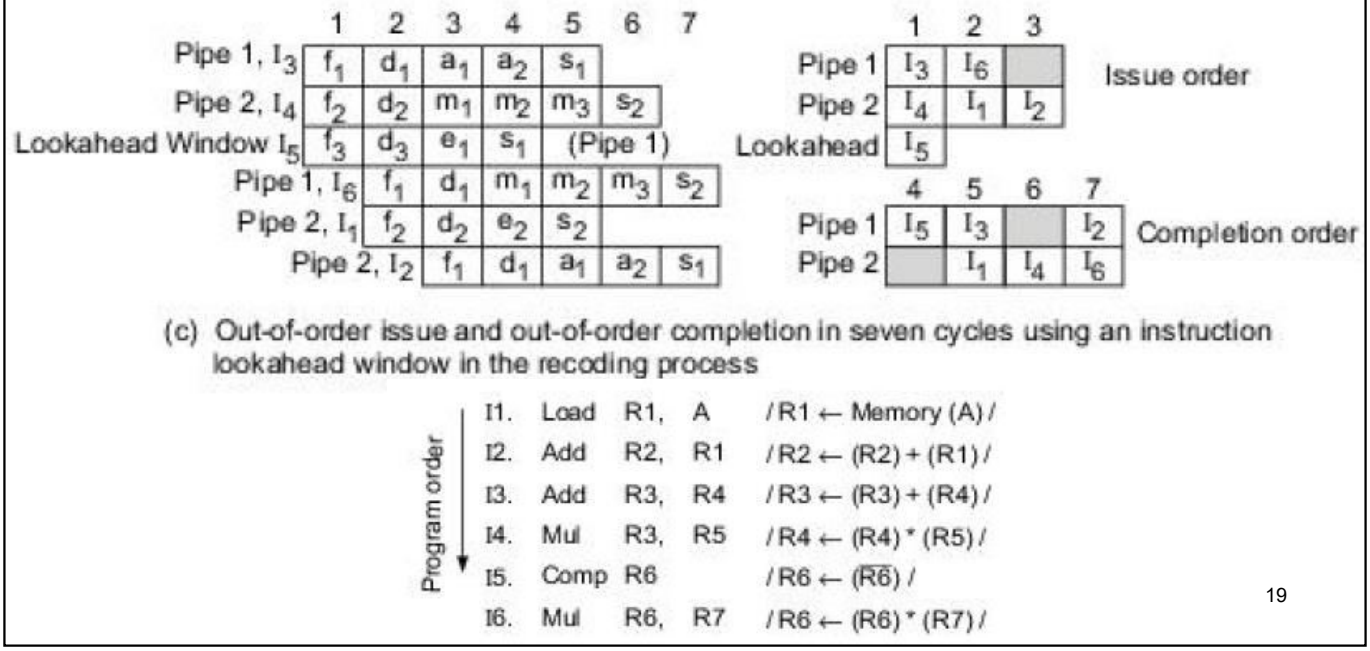
In-order issue and out-of-order completion



(b) In-order issue and out-of-order completion in nine cycles

- The only difference between this out-of-order schedule and the in-order schedule is that I5 is allowed to complete ahead of I3 and I4, which are totally independent of I5.
- The total execution time does not improve.
- However, the pipeline utilization rate does.

Out-of-Order Issue



(c) Out-of-order issue and out-of-order completion in seven cycles using an instruction lookahead window in the recoding process

Program order

```
I1.  Load  R1,  A    /R1 ← Memory (A)/
I2.  Add   R2,  R1   /R2 ← (R2) + (R1)/
I3.  Add   R3,  R4   /R3 ← (R3) + (R4)/
I4.  Mul   R3,  R5   /R4 ← (R4) * (R5)/
I5.  Comp  R6        /R6 ← (R6)/
I6.  Mul   R6,  R7   /R6 ← (R6) * (R7)/
```


- By using the lookahead window instruction I5 can he decoded in advance because it is independent of all the other instructions.
- The six instructions are issued in three cycles as shown:
  - I5 is fetched and decoded by the lookahead window,
  - while I3 and I4 and decoded concurrently
  - It is followed by issuing I6 and I1 at cycle 2,
  - and I2 at cycle 3.
- Because the issue is out of order, the completion is also out of order
- the total execution time has been reduced to seven cycles with no idle stages during the execution of these sis instructions.

Conclusion

- The in-order issue and completion is the simplest one to implement.
  - rarely used today even in a conventional scalar processor due to some unnecessary delays in maintaining program order.
  - in a multiprocessor environment, this policy is still attractive.
- Allowing out-of-order completion can be found in both scalar and superscalar processors.
  - better performance
  - more freedom to exploit parallelism, and thus pipeline efficiency is enhanced

Superscalar Performance

- To compare the relative performance of a superscalar processor with that of a scalar base machine, we estimate the ideal execution time of N independent instructions through the pipeline.
- The time required by the scalar base machine is

$$T(1, 1) = k + N - 1 \text{ (base cycles)}$$

- The ideal execution time required by an m-issue superscalar machine is

$$T(m, 1) = k + \frac{N - m}{m} \text{ (base cycles)}$$

- where k is the time required to execute the first m instructions through the in pipelines simultaneously and
- the second term corresponds to the time required to execute the remaining N-m instructions,
- m per cycle through m pipelines.

The ideal speedup of the superscalar machine over the base machine is

$$S(m, 1) = \frac{T(1, 1)}{T(m, 1)} = \frac{N + k - 1}{N/m + k - 1} = \frac{m(N + k - 1)}{N + m(k - 1)}$$

As $N \to \infty$, the speedup limit $S(m, 1) \to m$, as expected.