Chapter 9

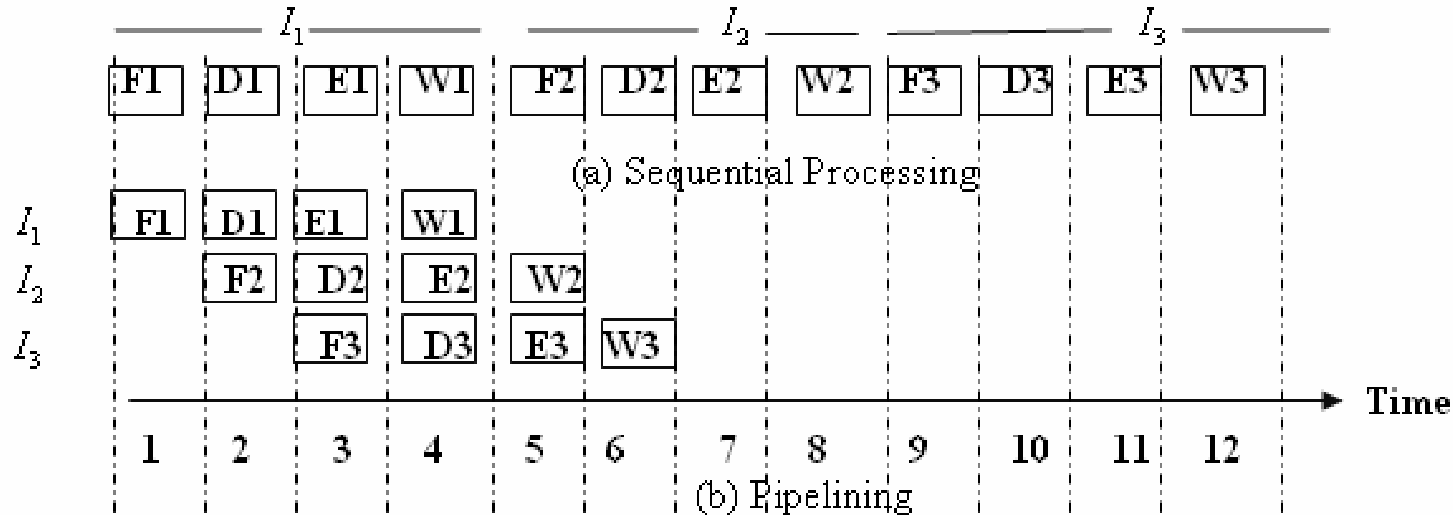
Pipelining Design Techniques

9.1 General Concepts

- Pipelining refers to the technique in which a given task is divided into a number of subtasks that need to be performed in sequence.
- Each subtask is performed by a given functional unit.
- The units are connected in a serial fashion and all of them operate simultaneously.
- The use of Pipelining improves the performance as compared to the traditional sequential execution of tasks.

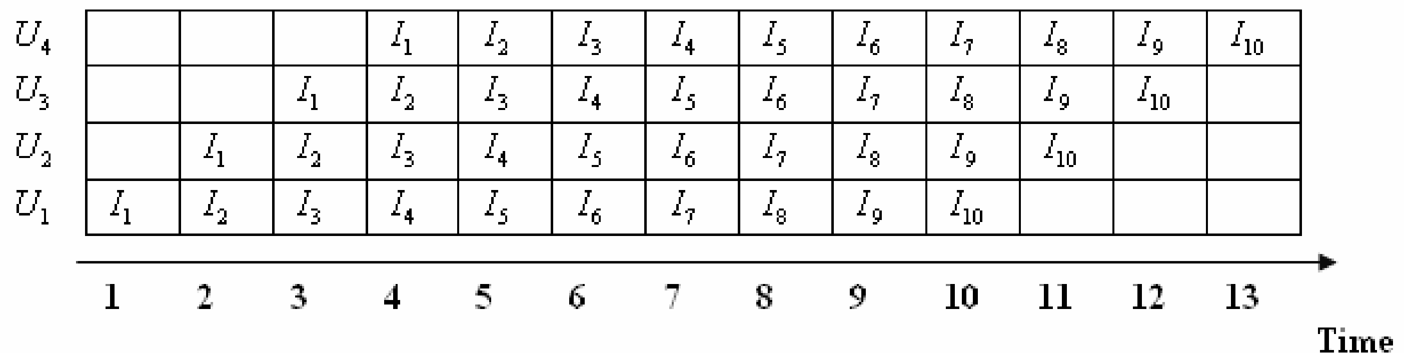
9.1 General Concepts

- Below is an illustration of the basic difference between executing four subtasks of a given instruction (in this case fetching *F*, decoding *D*, execution *E*, and writing the results *W*) using pipelining and sequential processing.



9.1 General Concepts

- In order to formulate some performance measures for the goodness of a pipeline in processing a series of tasks, a space time chart (called the Gantt's Chart) is used.
- The chart shows the succession of the sub-tasks in the pipe with respect to time.



9.1 General Concepts

- Three performance measures for the goodness of a pipeline are provided:
  - Speed-up S(n),
  - Throughput U(n), and
  - Efficiency E(n).
- It is assumed that the unit time $T = t$ units.

9.1 General Concepts

- Speedup S(n):
  - Consider the execution of *m* tasks (instructions) using *n*-stages (units) pipeline.
  - *n+m-1 time units* are required to complete *m* tasks.

$$Speed-up\ \mathrm{S(n)} = \frac{\text{Time using sequential processing}}{\text{Time using pipeline processing}} = \frac{m \times n \times t}{(n+m-1) \times t} = \frac{m \times n}{n+m-1}$$

$$\lim_{m \to \infty} \mathrm{S(n)} = \mathrm{n} \qquad \text{i.e., n-fold increase in speed is theoretically possible}$$

- Throughput T(n):

$$Throughput\ U(n) = \#\ \text{of tasks executed per unit time} = \frac{\mathrm{m}}{(\mathrm{n}+\mathrm{m}-1) \times \mathrm{t}}$$

$$\lim_{\mathrm{m} \to \infty} \mathrm{U(n)} = 1$$

9.1 General Concepts

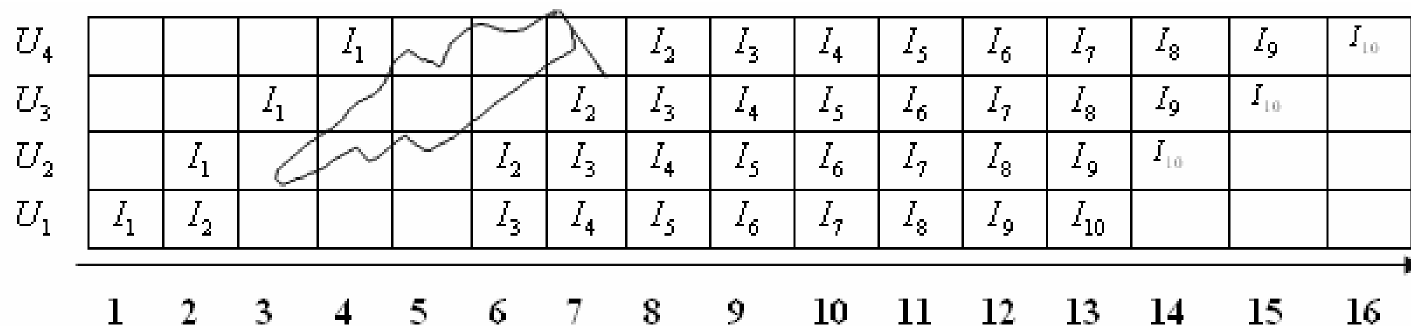
- Efficiency E(n):

$$\textit{Efficiency } E(n) = \text{Ratio of the actual speed} - \text{up to the maximum speed} - \text{up} = \frac{Speed - up}{n} = \frac{m}{n + m - 1}$$

$$\lim_{m \to \infty} E(n) = 1$$

9.2 Instruction Pipeline

- *A pipeline stall*: A Pipeline operation is said to have been stalled if one unit (stage) requires more time to perform its function, thus forcing other stages to become idle.
- Due to the extra time units needed for instruction to be fetched, the pipeline stalls.
- Such situations create what is known as pipeline *bubble* (or pipeline *hazards*).

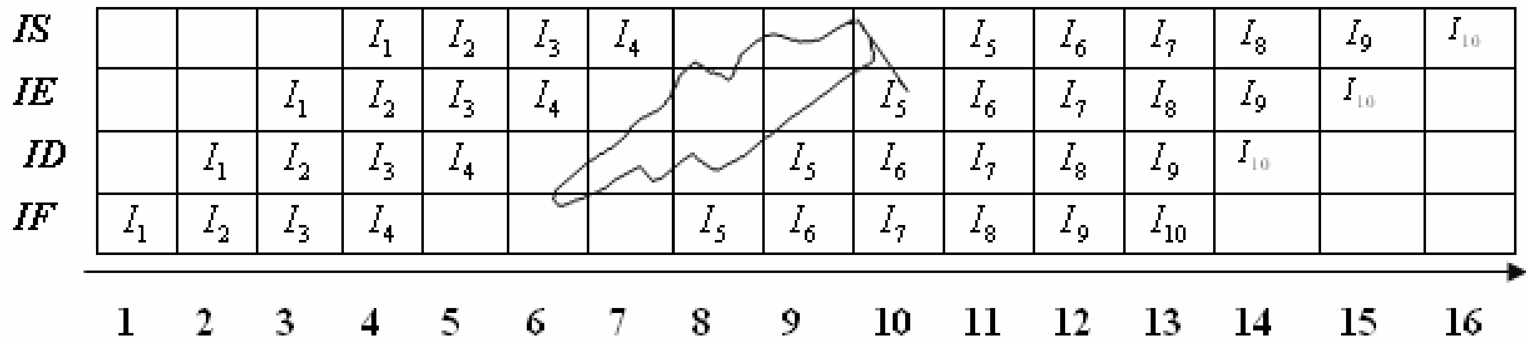


9.2 Instruction Pipeline

- Pipeline "Stall" due to Instruction Dependency:
  - Instruction dependency refers to the case whereby fetching of an instruction depends on the results of executing a previous instruction.
  - Instruction dependency manifests itself in the execution of a conditional branch instruction.
  - For example, in the case of a "branch if negative" instruction, the next instruction to fetch will not be known until the result of executing that "branch if negative" instruction is known.

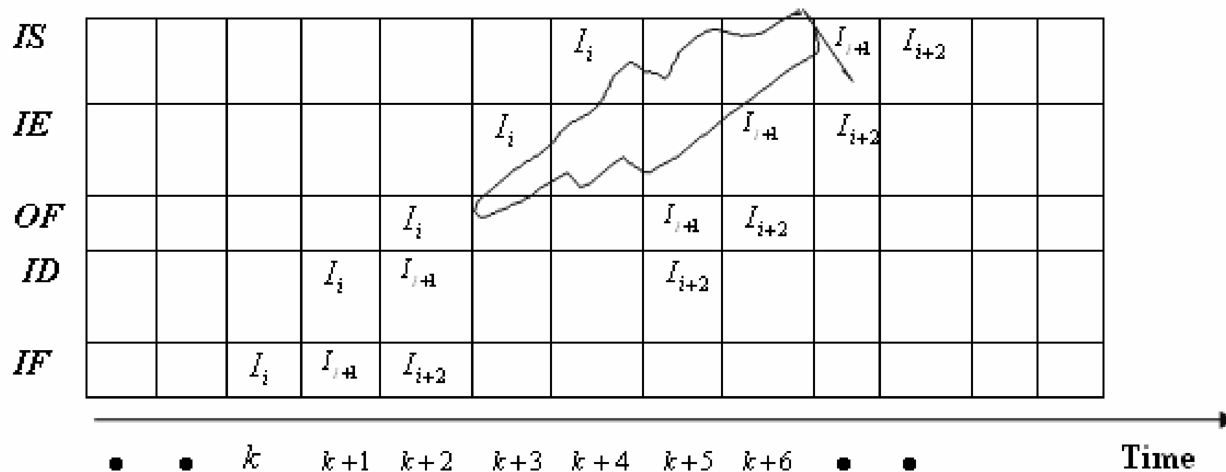
9.2 Instruction Pipeline

- Pipeline "Stall" due to Instruction Dependency:



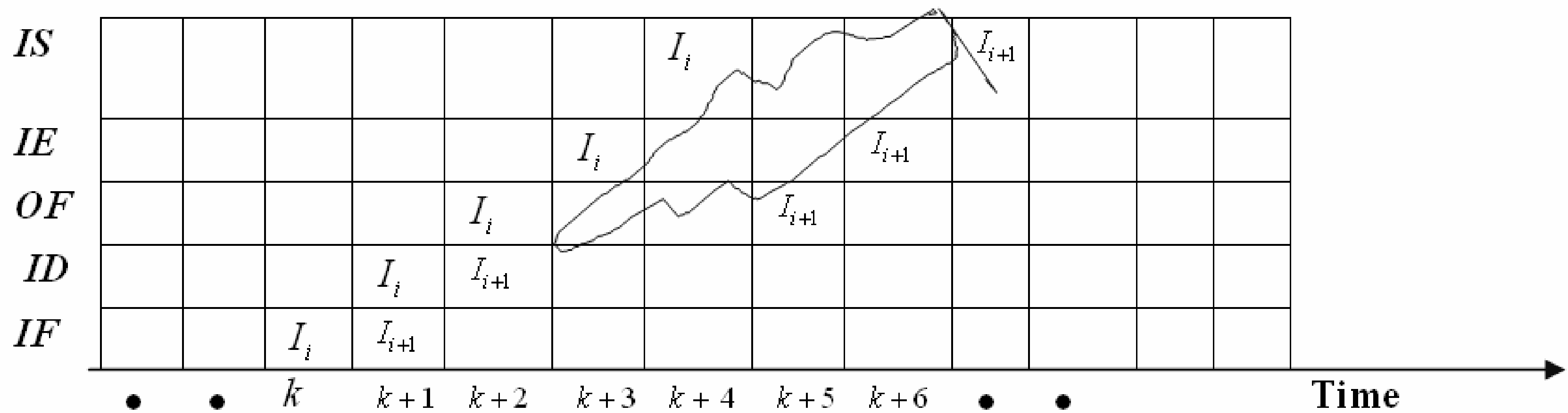
9.2 Instruction Pipeline

- Pipeline "Stall" due to Data Dependency:
  - Data dependency in a pipeline occurs when a source operand of instruction I_i depends on the results of executing a preceding instruction, I_j , $i > j$.
  - Write-after-write data dependency



9.2 Instruction Pipeline

- Pipeline "Stall" due to Data Dependency:
  - Read-after-write data dependency



9.2 Instruction Pipeline

- Method used to prevent fetching the wrong instruction or operand: use of *NOP* (No Operation)
  - In real-life situations, a mechanism is needed to guarantee fetching the appropriate instruction at the appropriate time.
  - Insertion of "*NOP*" instructions will help carrying out this task.
  - A "*NOP*" is an instruction that has no effect on the status of the processor.

9.2 Instruction Pipeline

- Method used to prevent fetching the wrong instruction or operand: use of *NOP* (No Operation)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
IS				I_1	I_2	I_3	I_4	**Nop**	**Nop**	**Nop**	I_5	I_6	I_7	I_8	I_9	I_{10}
IE			I_1	I_2	I_3	I_4	**Nop**	**Nop**	**Nop**	I_5	I_6	I_7	I_8	I_9	I_{10}	
ID		I_1	I_2	I_3	I_4	**Nop**	**Nop**	**Nop**	I_5	I_6	I_7	I_8	I_9	I_{10}		
IF	I_1	I_2	I_3	I_4	**Nop**	**Nop**	**Nop**	I_5	I_6	I_7	I_8	I_9	I_{10}			

9.2 Instruction Pipeline

- Methods used to reduce pipeline stall due to instruction dependency:
  - Unconditional Branch Instructions
    - Reordering of Instructions.
    - Use of Dedicated Hardware in the Fetch Unit.
    - Pre-computing of Branches and Reordering of Instructions.
    - Instruction Pre-fetching.
  - Conditional Branch Instructions
    - Delayed Branch.
    - Prediction of the Next Instruction to Fetch.

9.2 Instruction Pipeline

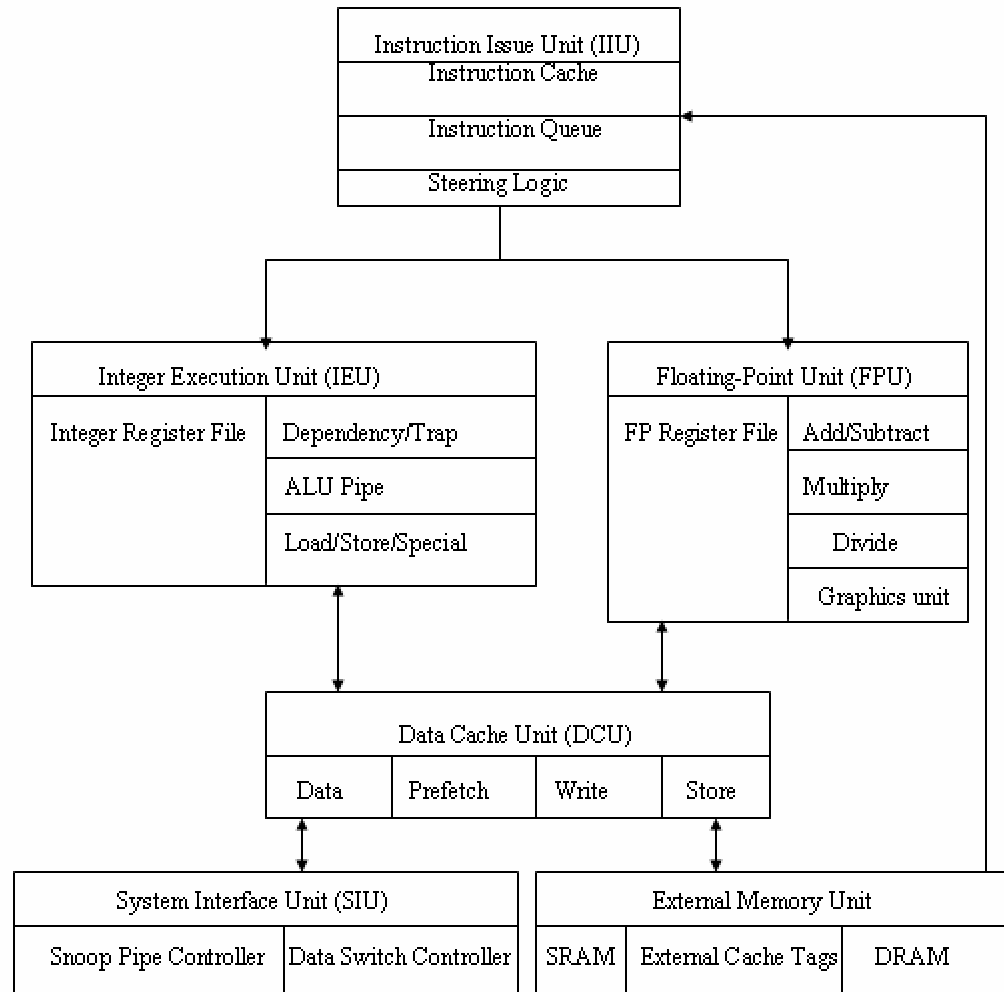
- Methods used to reduce pipeline stall due to data dependency
  - Hardware Operand Forwarding
  - Software Operand Forwarding
    - Store-Fetch.
    - Fetch-Fetch.
    - Store-Store.

9.3 Example Pipeline Processors

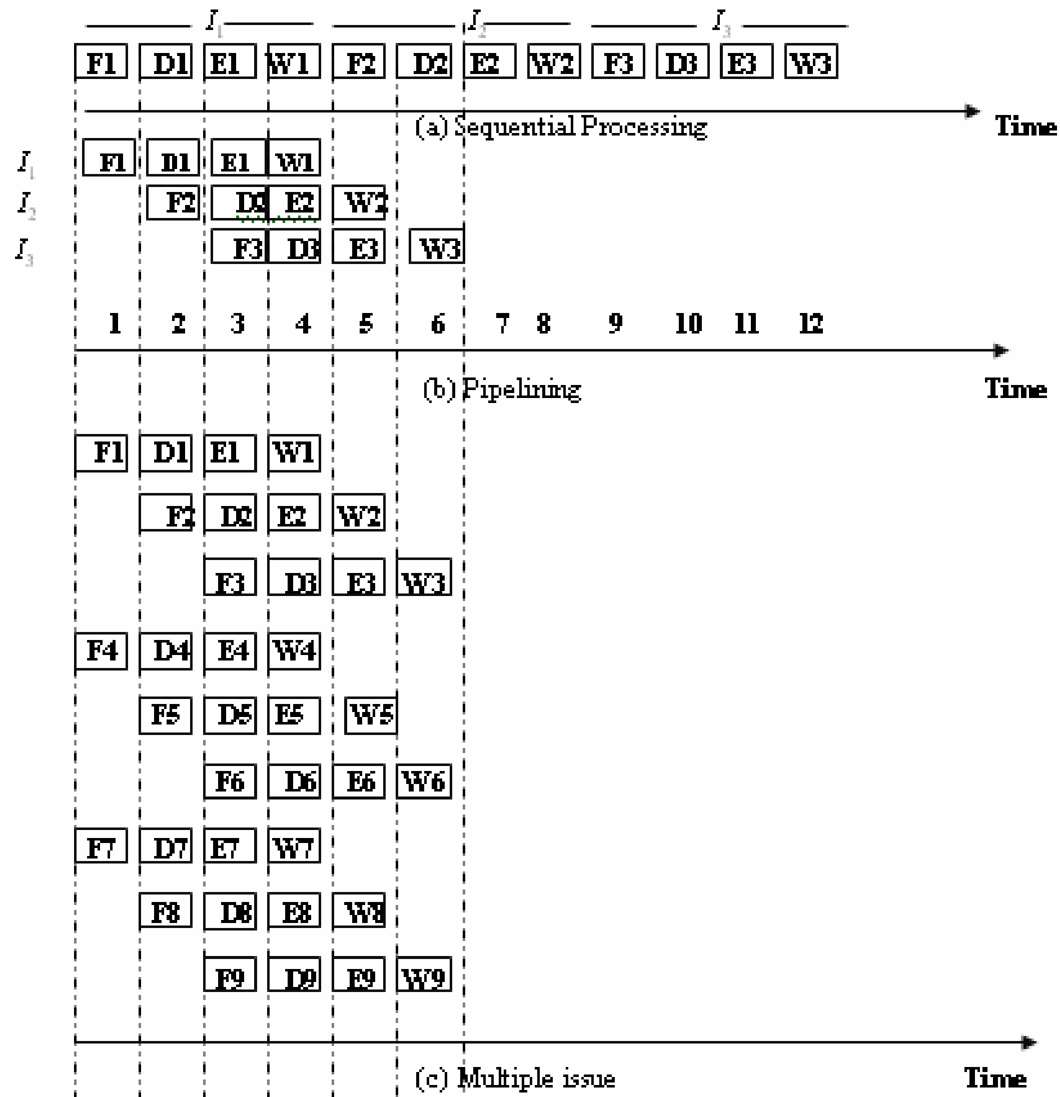
- ARM 1026EJ-S Processor
  - ARM 1022EJ-S is a pipeline processor whose ALU consists of six stages:
    - Fetch Stage: for instruction cache access and branch prediction for instructions that have already been fetched.
    - Issue Stage: for initial instruction decoding.
    - Decode Stage: for final instruction decode, register read for ALU operations, forwarding, and initial interlock resolution.
    - Execute Stage: for data access address calculation, data processing shift, shift & saturate, ALU operations, first stage multiplication, flag setting, condition code check, branch mis-predict detection, and store data register read.
    - Memory Stage: for data cache access, second stage multiplication, and saturations.
    - Write Stage: for register write and instruction retirement.

9.3 Example Pipeline Processors

- UltraSPARC-III Processor



9.4 Instruction-Level Parallelism



9.4 Instruction-Level Parallelism

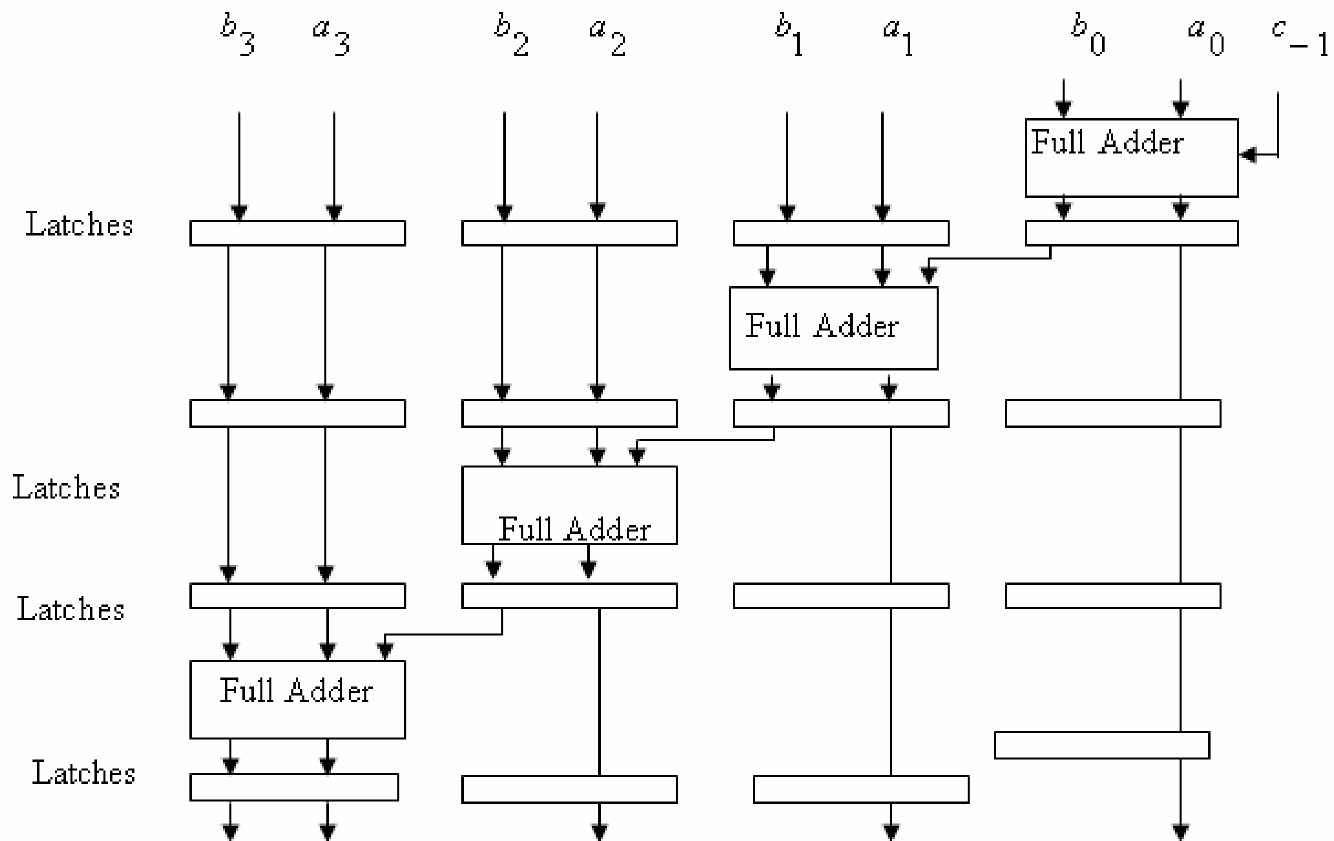
- Superscalar Architectures
  - A scalar machine is able to perform only one arithmetic operation at once.
  - A superscalar architecture (SPA) is able to fetch, decode, execute, and store results of several instructions at the same time.
  - In a SPA instruction processing consists of the fetch, decode, issue, and commit stages.
  - The most crucial step in processing instructions in SPAs is the dependency analysis.
    - The complexity of such analysis grows quadratically with the instruction word size.
    - This puts a limit on the degree of parallelism that can be achieved with SPAs such that a degree of parallelism higher than four will be impractical.

9.4 Instruction-Level Parallelism

- Very Long Instruction Word (VLIW)
  - The compiler performs dependency analysis and determines the appropriate groupings/scheduling of operations.
  - Operations that can be performed simultaneously are grouped into a Very Long Instruction Word (VLIW).
  - Therefore, the instruction word is made long enough in order to accommodate the maximum possible degree of parallelism.
  - In VLIW, resource binding can be done by devoting each field of an instruction word to one and only one functional unit.
  - However, this arrangement will lead to a limit on the mix of instructions that can be issued per cycle.
  - A more flexible approach is to allow a given instruction field to be occupied by different kinds of operations.

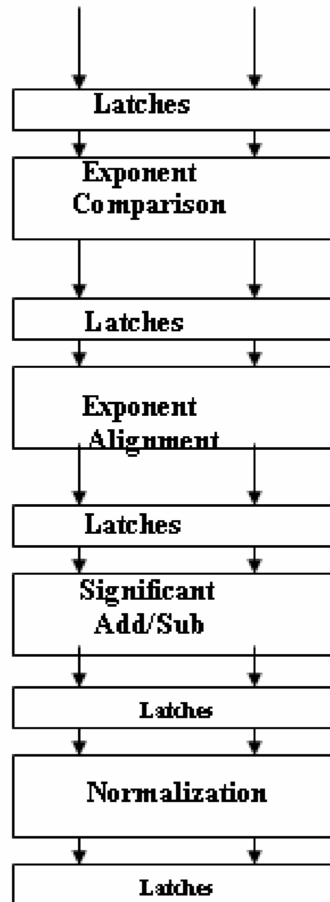
9.5 Arithmetic Pipeline

- Fixed-Point Arithmetic Pipelines



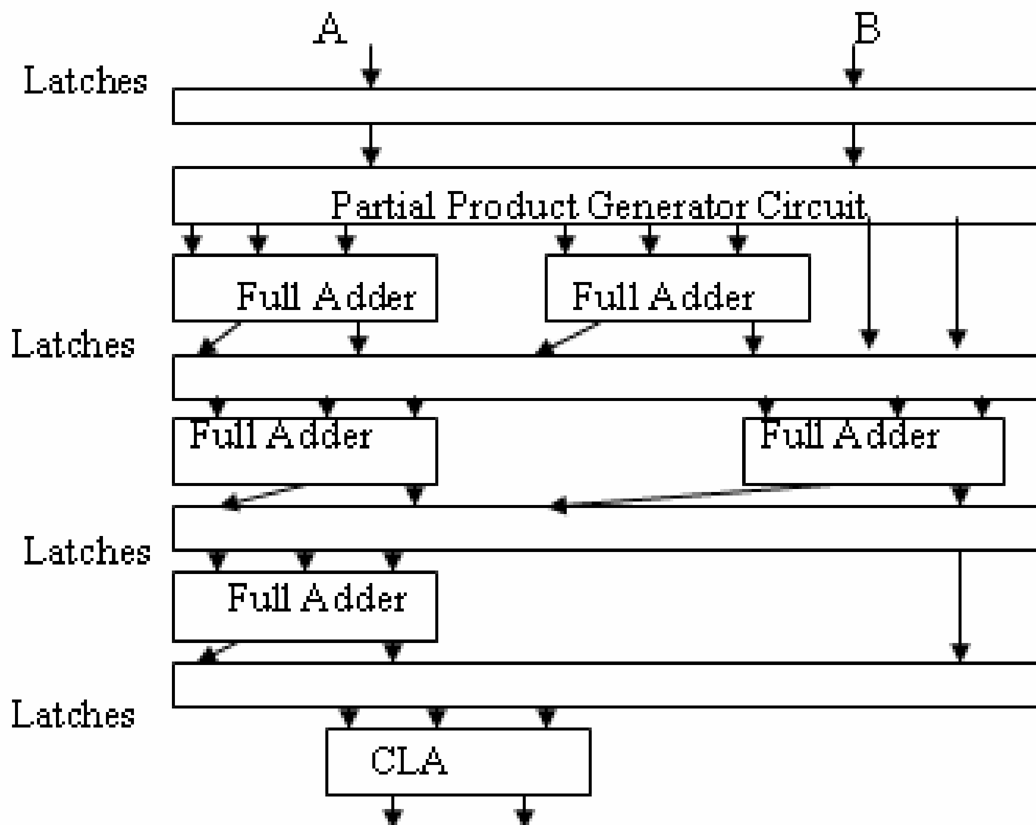
9.5 Arithmetic Pipeline

- Floating-Point Arithmetic Pipelines



9.5 Arithmetic Pipeline

- Pipelined Multiplication using Carry-Save Addition



9.6 Summary

- In this chapter, the basic principles involved in designing pipeline architectures were considered.
- Our coverage started with a discussion on a number of metrics that can be used to assess the goodness of a pipeline.
- We then moved to present a general discussion on the main problems that need to be considered in designing a pipelined architecture.
  - In particular we considered two main problems: Instruction and data dependency.

9.6 Summary

- The effect of these two problems on the performance of a pipeline has been elaborated.
- Some possible techniques that can be used to reduce the effect of the instruction and data dependency have been introduced and illustrated.
- An example of a recent pipeline architecture, the ARM 11 microarchitecture, has been presented.