CSE539: Advanced Computer Architecture

Chapter 6
Pipelining and Superscalar Techniques

Book: "Advanced Computer Architecture – Parallelism, Scalability, Programmability", Hwang & Jotwani

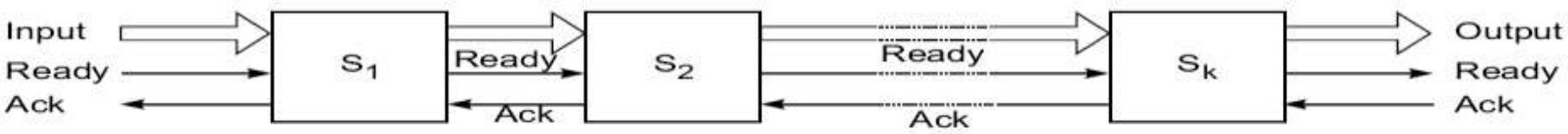
Sumit Mittu
Assistant Professor, CSE/IT
Lovely Professional University
sumit.12735@lpu.co.in

In this chapter…

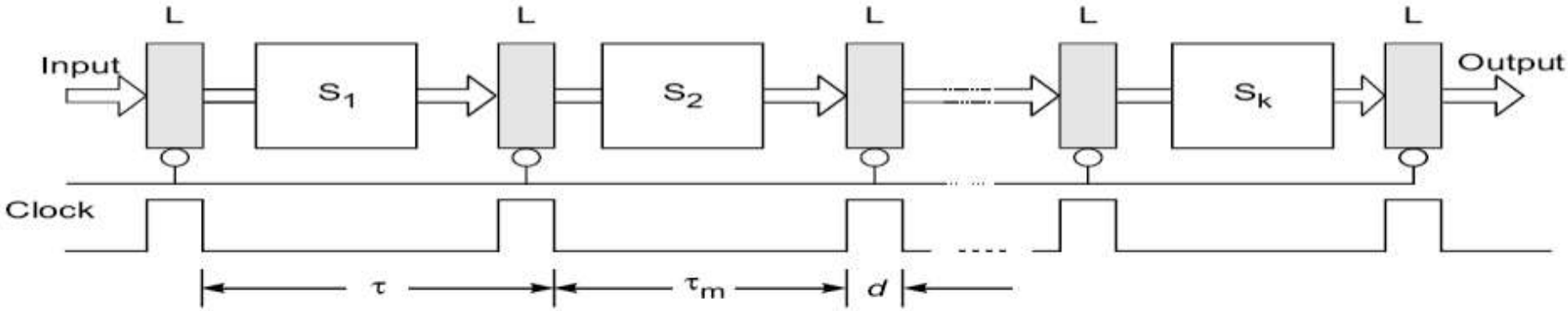
- Linear Pipeline Processors
- Non-linear Pipeline Processors
- Instruction Pipeline Design
- Arithmetic Pipeline Design
- Superscalar Pipeline Design

LINEAR PIPELINE PROCESSORS

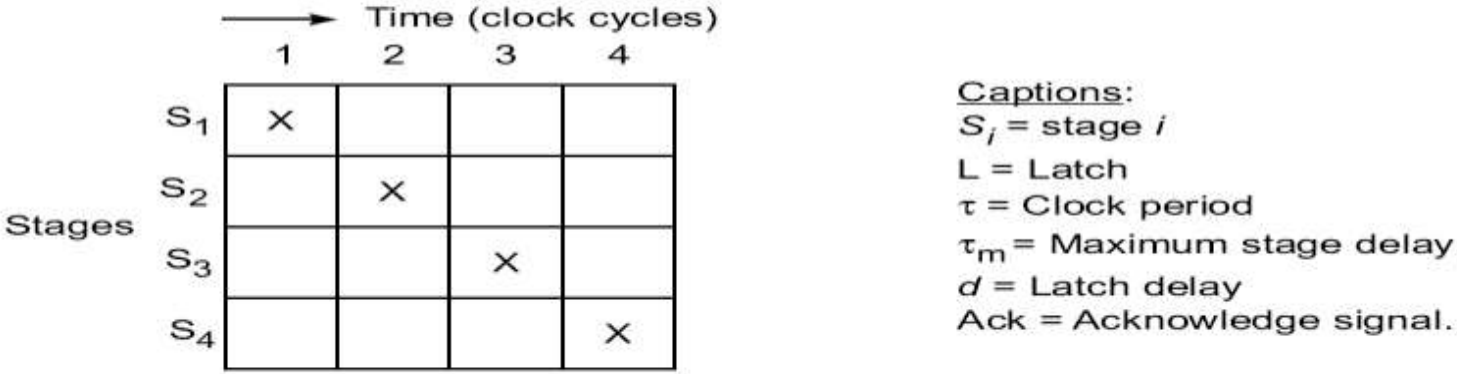
- Linear Pipeline Processor
  - *(Definition)*
- Models of Linear Pipeline
  - Synchronous Model
  - Asynchronous Model
  - *(Corresponding reservation tables)*
- Clocking and Timing Control
  - Clock Cycle
  - Pipeline Frequency
  - Clock skewing
  - Flow-through delay
  - Speedup, Efficiency and Throughput
- Optimal number of Stages and Performance-Cost Ratio (PCR)



(a) An asynchronous pipeline model



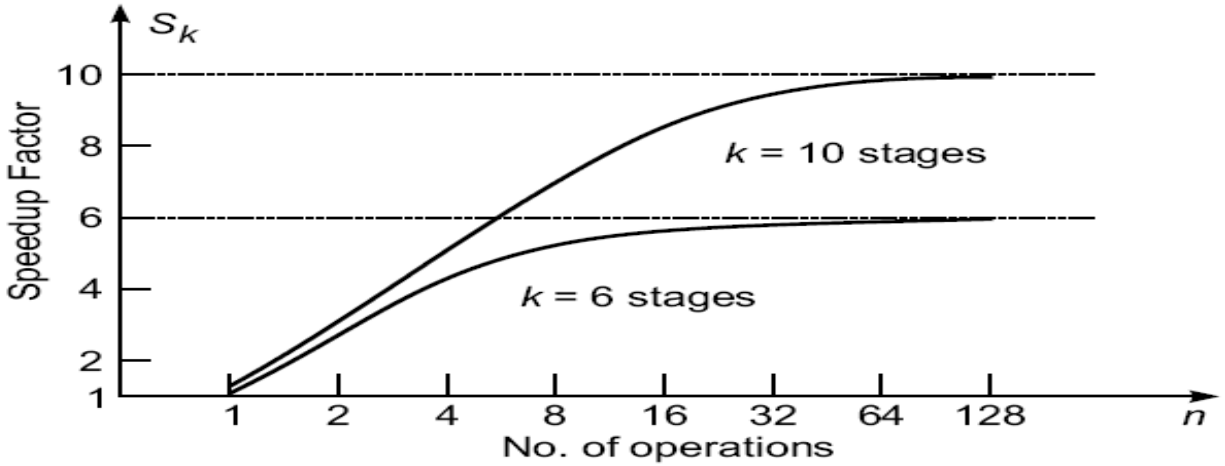
(b) A synchronous pipeline model



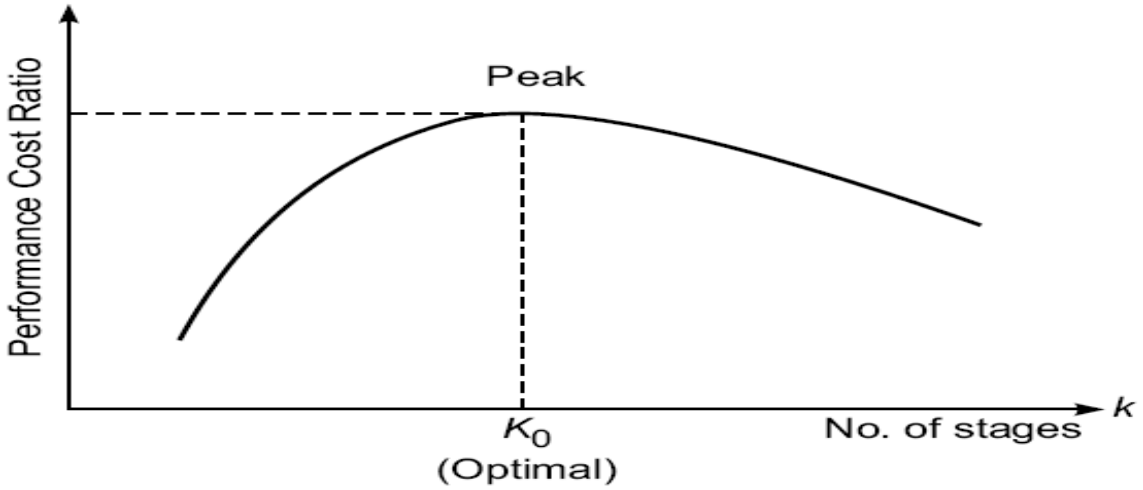
Captions:
S_i = stage i
L = Latch
τ = Clock period
τ_m = Maximum stage delay
d = Latch delay
Ack = Acknowledge signal.

(c) Reservation table of a four-stage linear pipeline

Fig. 6.1 Two models of linear pipeline units and the corresponding reservation table



(a) Speedup factor as a function of the number of operations (Eq. 6.5)



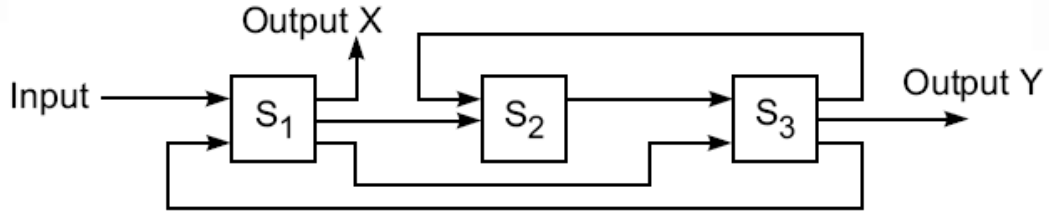
(b) Optimal number of pipeline stages (Eqs. 6.6 and 6.7)

Fig. 6.2 Speedup factors and the optimal number of pipeline stages for a linear pipeline unit

NON-LINEAR PIPELINE PROCESSORS

- Dynamic Pipeline
  - Static v/s Dynamic Pipeline
  - Streamline connection, feed-forward connection and feedback connection
- Reservation and Latency Analysis
  - Reservation tables
  - Evaluation time
- Latency Analysis
  - Latency
  - Collision
  - Forbidden latencies
  - Latency Sequence, Latency Cycle and Average Latency

NON-LINEAR PIPELINE PROCESSORS



(a) A three-stage pipeline

Time

Stages	1	2	3	4	5	6	7	8
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

(b) Reservation table for function X

Time

Stages	1	2	3	4	5	6
S_1	Y				Y	
S_2			Y			
S_3		Y		Y		Y

(c) Reservation table for function Y

Fig. 6.3 A dynamic pipeline with feed forward and feedback connections for two different functions

NON-LINEAR PIPELINE PROCESSORS

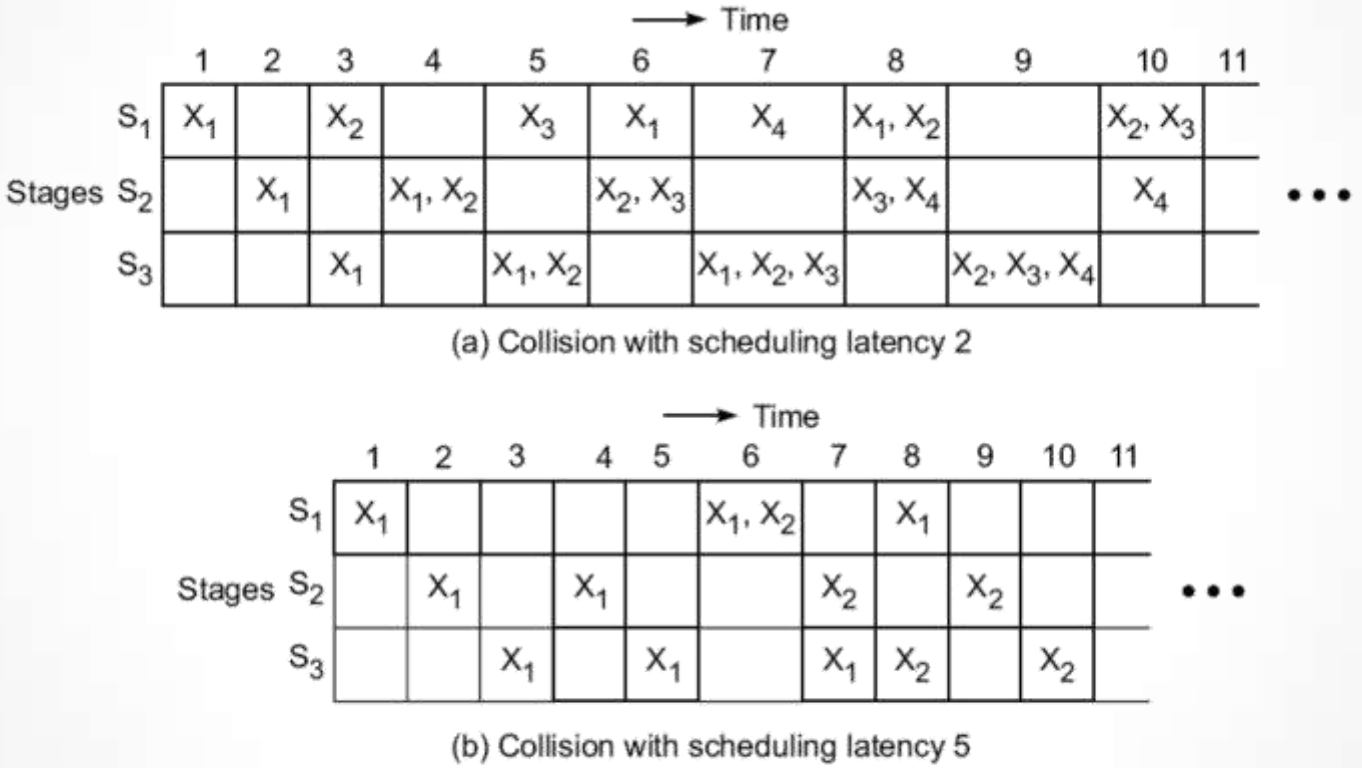


Fig. 6.4 Collisions with forbidden latencies 2 and 5 in using the pipeline in Fig. 6.3 to evaluate the function X

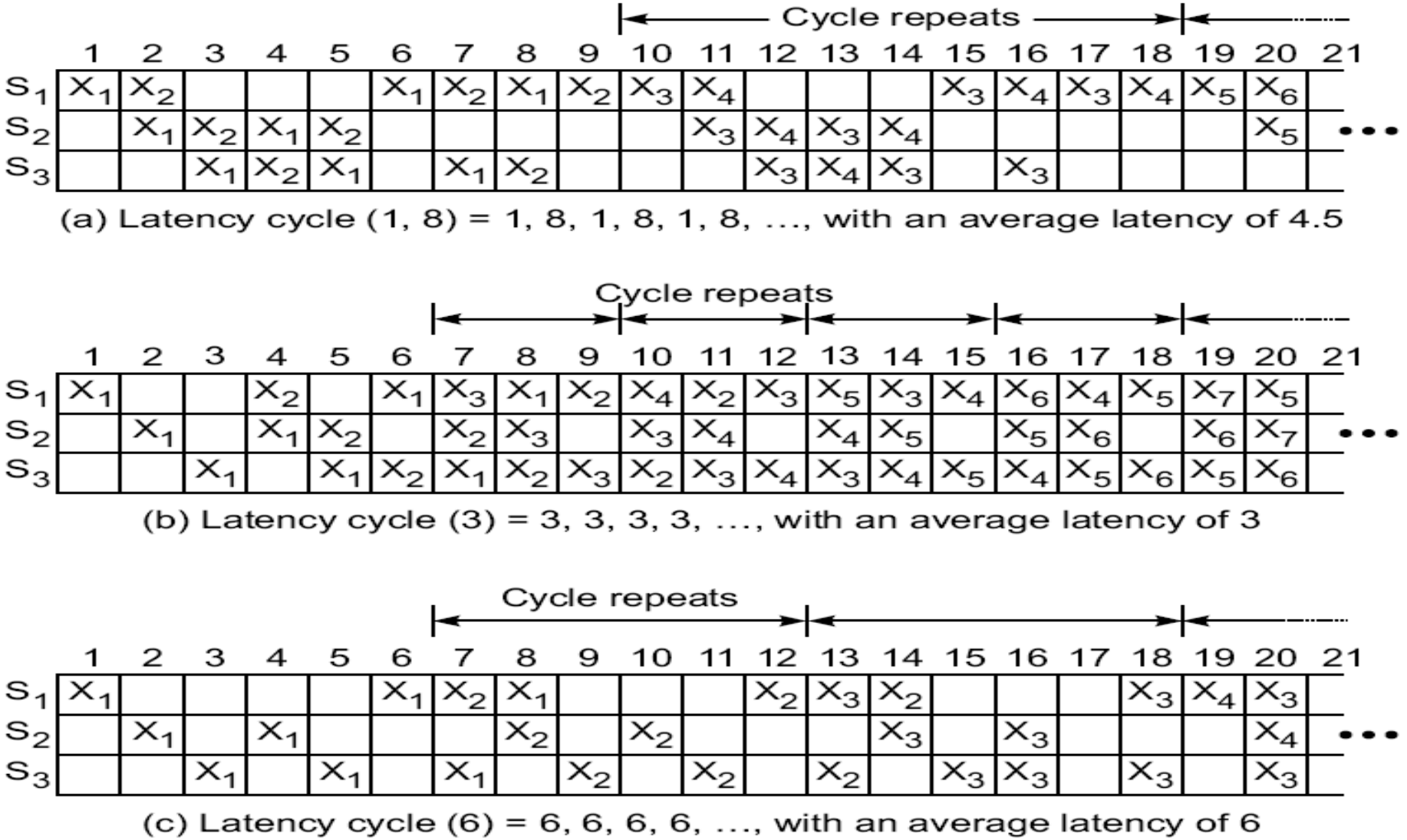
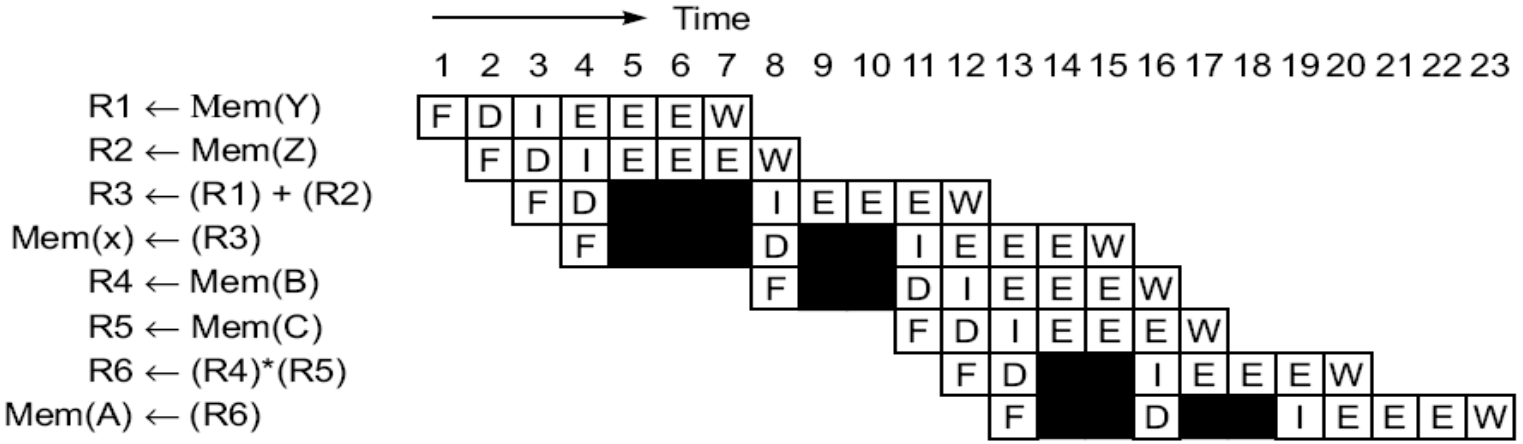


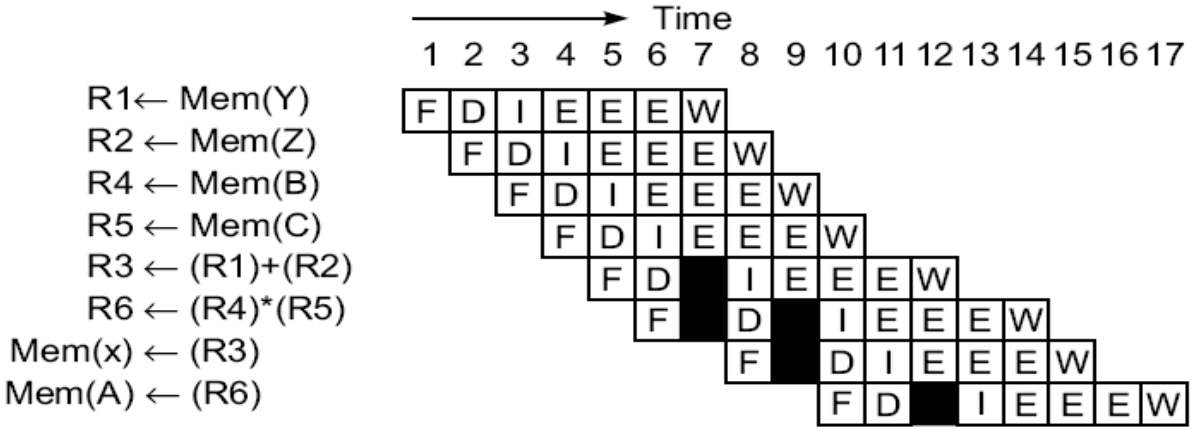
Fig. 6.5 Three valid latency cycles for the evaluation of function X

INSTRUCTION PIPELINE DESIGN

- Instruction Execution Phases
  - E.g. Fetch, Decode, Issue, Execute, Write-back
  - In-order Instruction issuing and Reordered Instruction issuing
    - E.g. **X = Y + Z , A = B x C**
- Mechanisms/Design Issues for Instruction Pipelining
  - Pre-fetch Buffers
  - Multiple Functional Units
  - Internal Data Forwarding
  - Hazard Avoidance
- Dynamic Scheduling
- Branch Handling Techniques



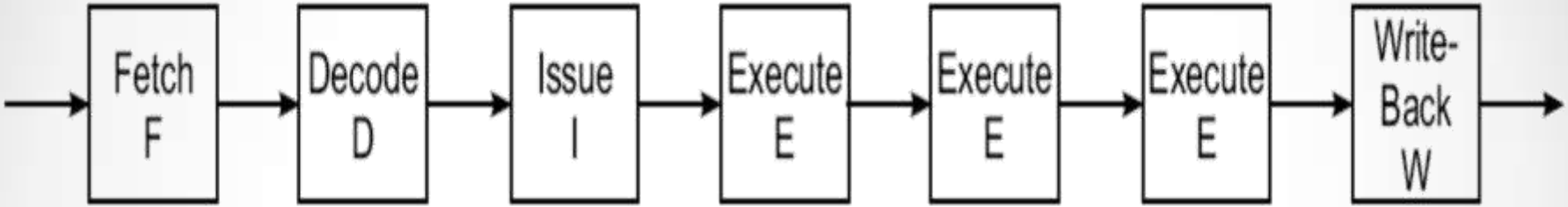
(b) In-order instruction issuing



(c) Reordered instruction issuing

Fig. 6.9 Pipelined execution of X = Y + Z and A = B × C (Courtesy of James Smith; reprinted with permission from *IEEE Computer*, July 1989)

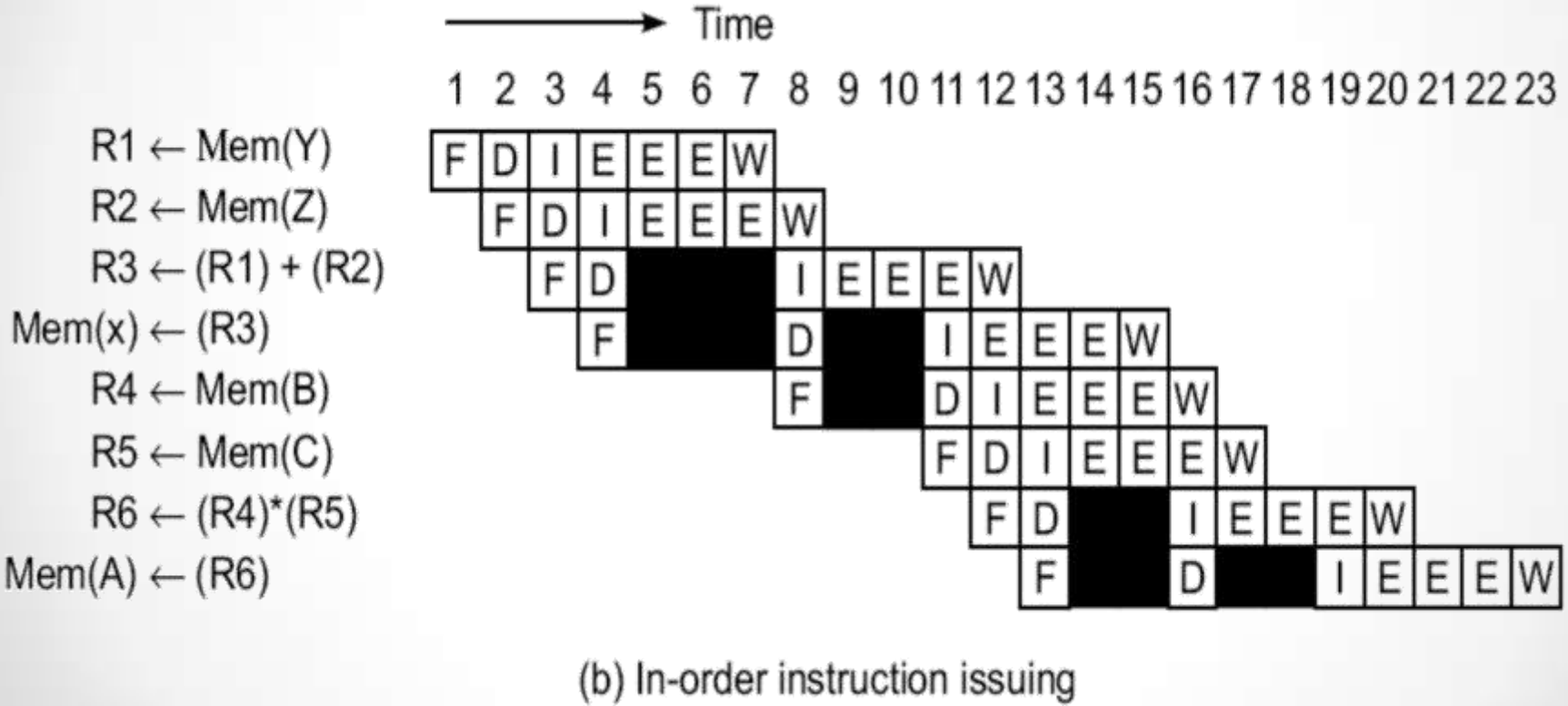
INSTRUCTION PIPELINE DESIGN



(a) A seven-stage instruction pipeline

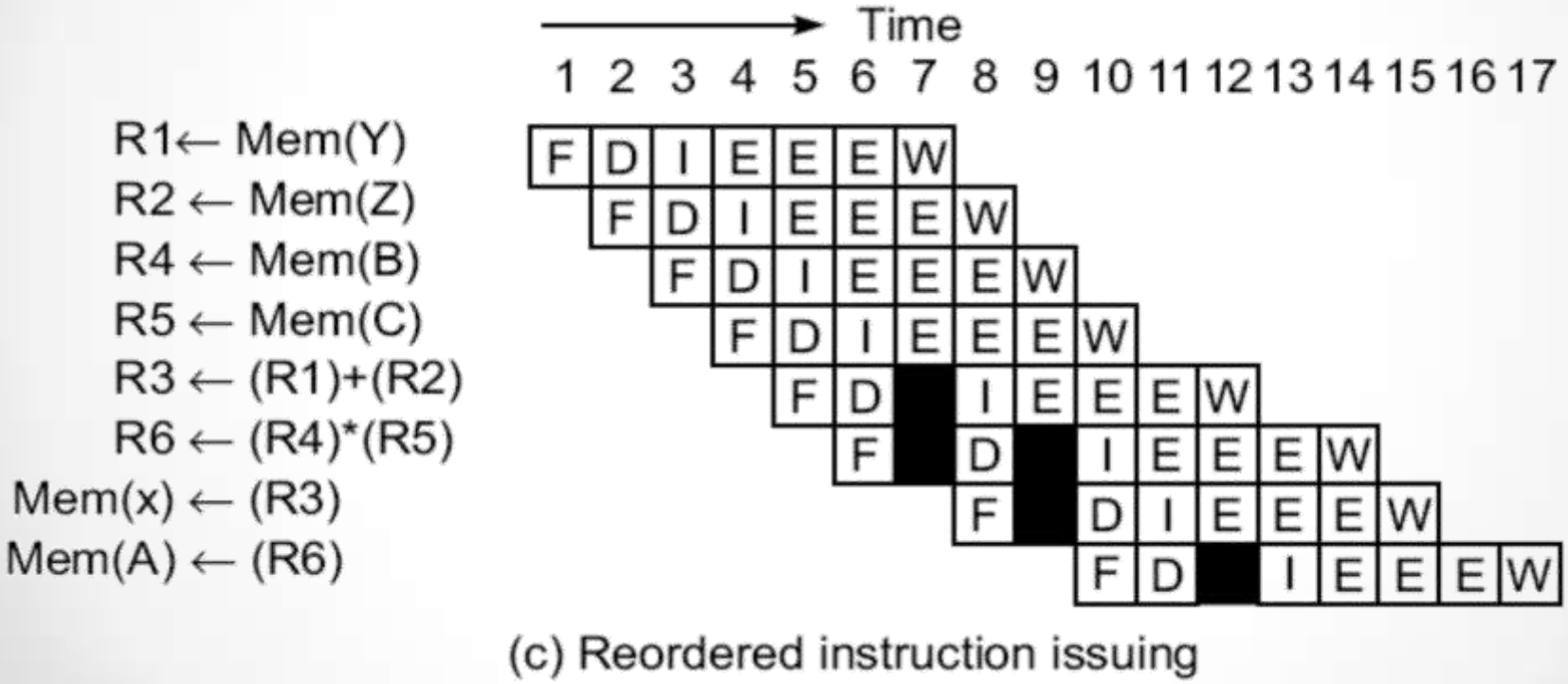
- **Fetch**: *fetches instructions from memory; ideally one per cycle*
- **Decode**: *reveals instruction operations to be performed and identifies the resources needed*
- **Issue**: *reserves the resources and reads the operands from registers*
- **Execute**: *actual processing of operations as indicated by instruction*
- **Write Back**: *writing results into the registers*

INSTRUCTION PIPELINE DESIGN



(b) In-order instruction issuing

INSTRUCTION PIPELINE DESIGN



(c) Reordered instruction issuing

INSTRUCTION PIPELINE DESIGN
Mechanisms/Design Issues of Instruction Pipeline

- Pre-fetch Buffers
  - Sequential Buffers
  - Target Buffers
  - Loop Buffers

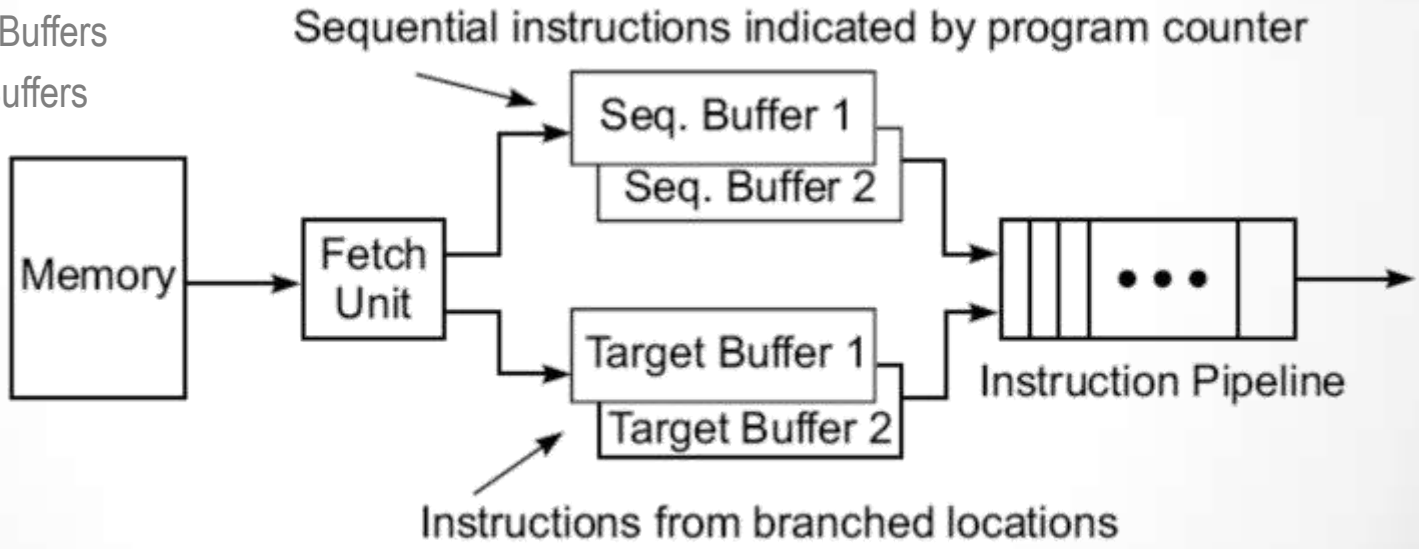


Fig. 6.11 The use of sequential and target buffers

INSTRUCTION PIPELINE DESIGN
Mechanisms/Design Issues of Instruction Pipeline

- Multiple Functional Units
  - Reservation Station and Tags
  - Slow-station as Bottleneck stage
    - Subdivision of Pipeline Bottleneck stage
    - Replication of Pipeline Bottleneck stage
    - *(Example to be discussed)*

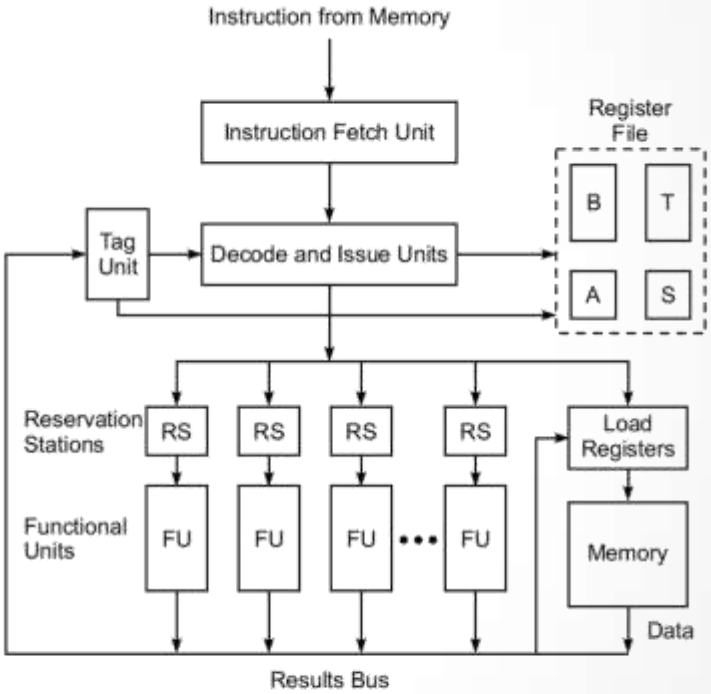


Fig. 6.12 A pipelined processor with multiple functional units and distributed reservation stations supported by tagging (Courtesy of G. Sohi; reprinted with permission from *IEEE Transactions on Computers*, March 1990)

INSTRUCTION PIPELINE DESIGN
Mechanisms/Design Issues of Instruction Pipeline

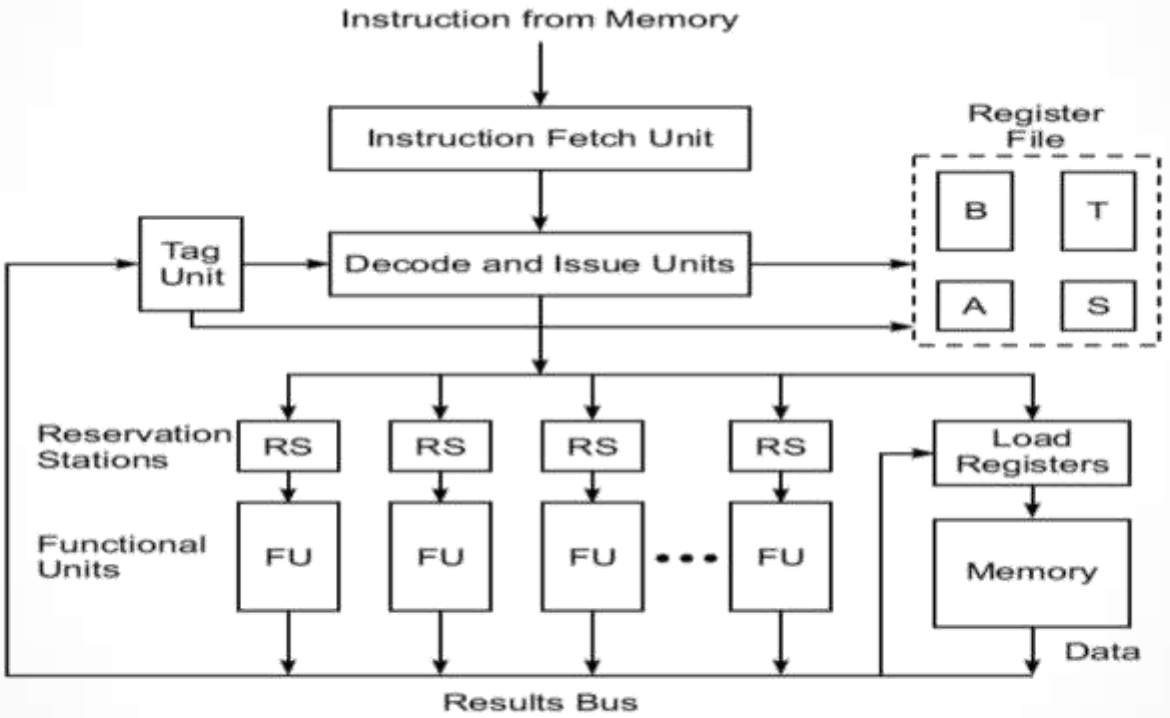


Fig. 6.12 A pipelined processor with multiple functional units and distributed reservation stations supported by tagging (Courtesy of G. Sohi; reprinted with permission from *IEEE Transactions on Computers*, March 1990)

INSTRUCTION PIPELINE DESIGN
Mechanisms/Design Issues of Instruction Pipeline

- Internal Forwarding and Register Tagging
  - Internal Forwarding:
    - *A "short-circuit" technique to replace unnecessary memory accesses by register-register transfers in a sequence of fetch-arithmetic-store operations*
  - Register Tagging:
    - *Use of tagged registers , buffers and reservation stations, for exploiting concurrent activities among multiple arithmetic units*
  - **Store-Fetch Forwarding**
    - *(M ← R1, R2 ← M) replaced by (M ← R1, R2 ← R1)*
  - **Fetch-Fetch Forwarding**
    - *(R1 ← M, R2 ← M) replaced by (R1 ← M, R2 ← R1)*
  - **Store-Store Overwriting**
    - *(M ← R1, M ← R2) replaced by (M ← R2)*

INSTRUCTION PIPELINE DESIGN
Mechanisms/Design Issues of Instruction Pipeline

- Internal Forwarding and Register Tagging

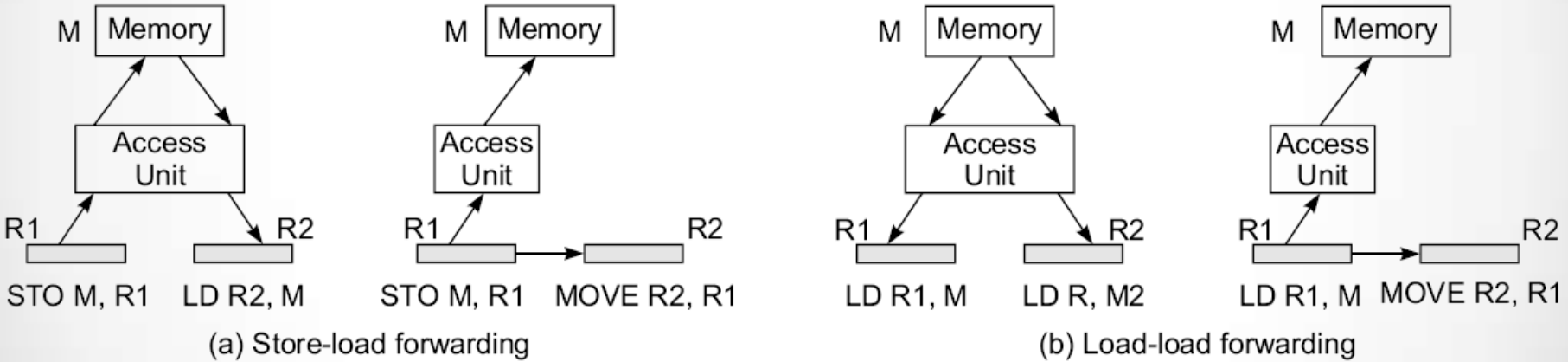
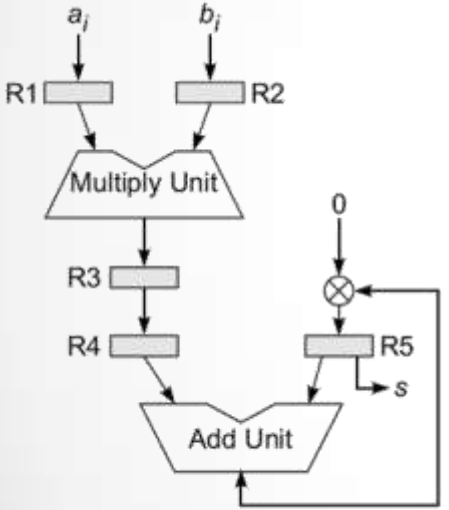


Fig. 6.13 Internal data forwarding by replacing memory-access operations with register transfer operations

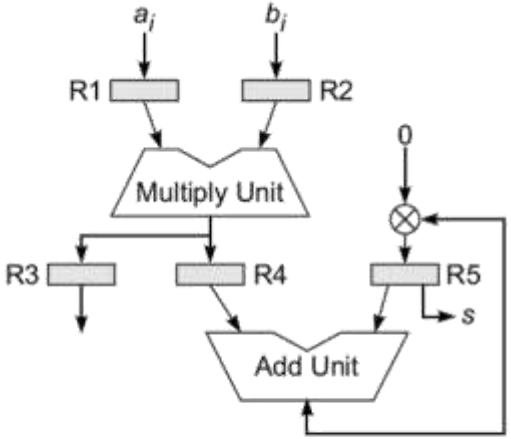
INSTRUCTION PIPELINE DESIGN
Mechanisms/Design Issues of Instruction Pipeline

- Internal Forwarding and Register Tagging



(a) Without data forwarding

I_1 : R3 ← (R1) * (R2)
I_2 : R4 ← (R3)
I_3 : R5 ← (R5) + (R4)



(b) With internal data forwarding

I'_1 : R3 ← (R1) * (R2)
I'_2 : R4 ← (R1) * (R2)
I'_3 : R5 ← (R4) + (R5)

I'_1 and I'_2 can be executed simultaneously with internal data forwarding.

Fig. 6.14 Internal data forwarding for implementing the dot-product operation

INSTRUCTION PIPELINE DESIGN
Mechanisms/Design Issues of Instruction Pipeline

- Hazard Detection and Avoidance
  - Domain or Input Set of an instruction
  - Range or Output Set of an instruction
  - Data Hazards: **RAW**, **WAR** and **WAW**
  - Resolution using Register Renaming approach

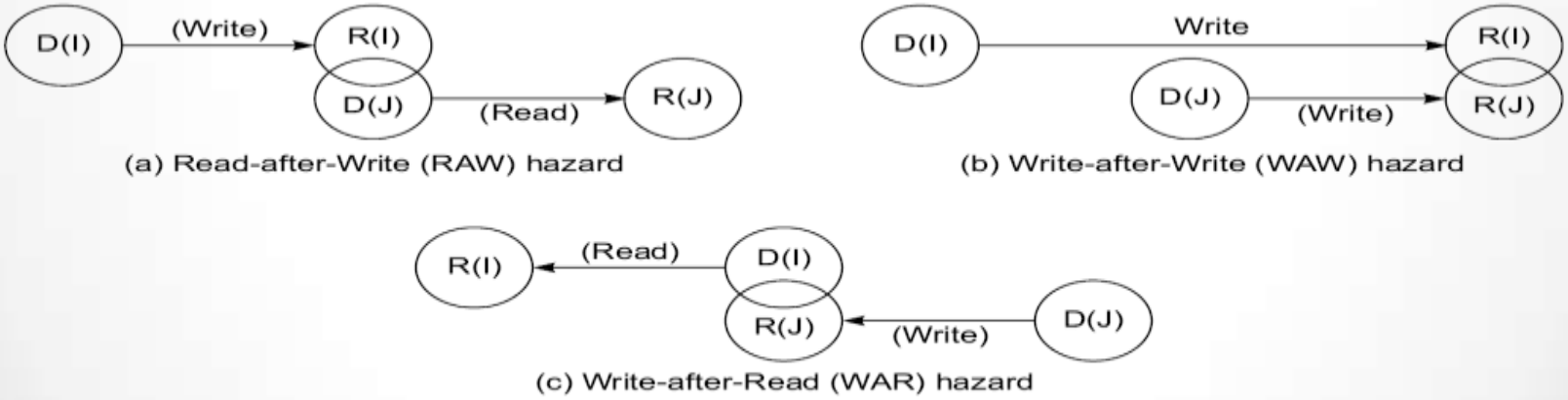


Fig. 6.15 Possible hazards between read and write operations in an instruction pipeline (instruction I is ahead of instruction J in program order)

INSTRUCTION PIPELINE DESIGN
Dynamic Instruction Scheduling

- Idea of Static Scheduling
  - Compiler based scheduling strategy to resolve Interlocking among instructions
- Dynamic Scheduling
  - **Tomasulo's Algorithm** (Register-Tagging Scheme)
    - Hardware based dependence-resolution
  - **Scoreboarding** Technique
    - Scoreboard: the centralized control unit
    - A kind of data-driven mechanism

INSTRUCTION PIPELINE DESIGN
Dynamic Instruction Scheduling

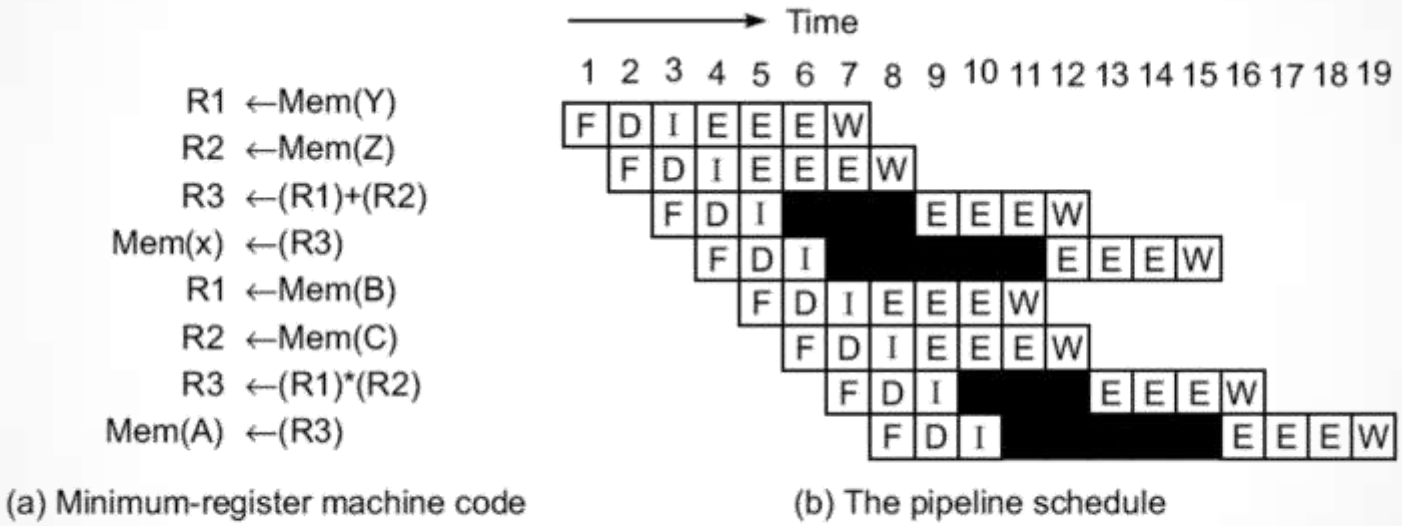
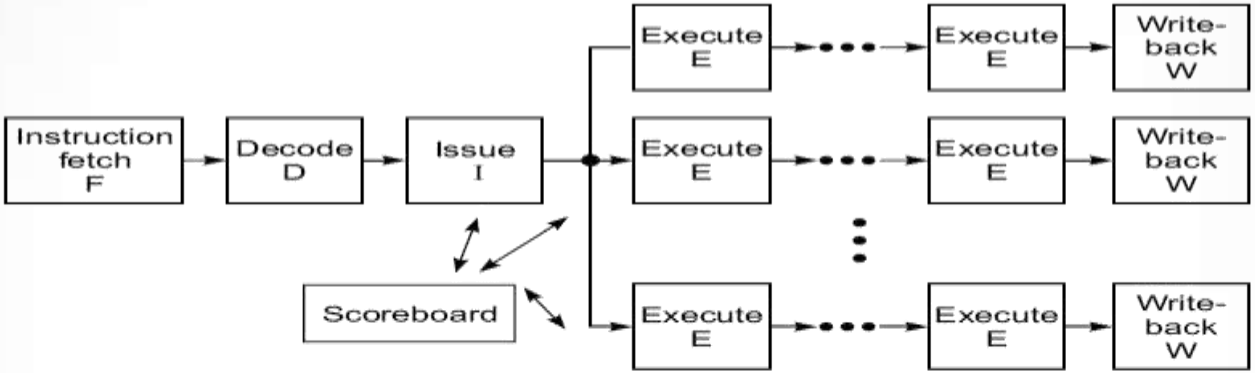
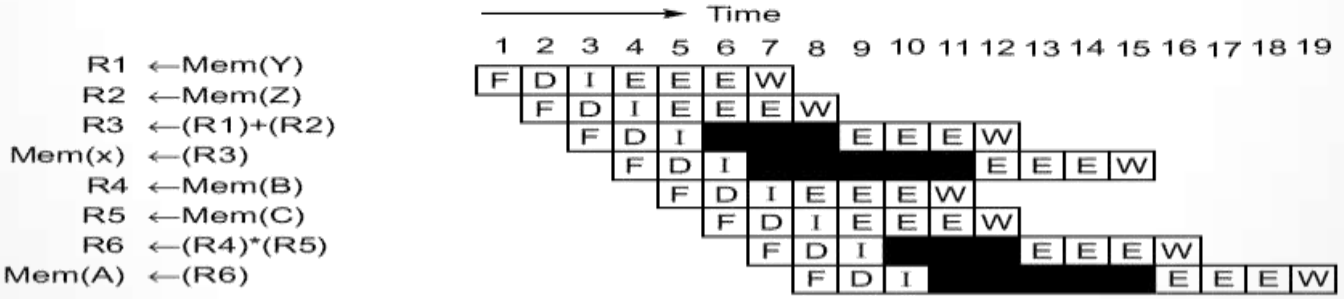


Fig. 6.16 Dynamic instruction scheduling using Tomasulo's algorithm on the processor in Fig. 6.12 (Courtesy of James Smith; reprinted with permission from *IEEE Computer*, July 1989)

INSTRUCTION PIPELINE DESIGN
Dynamic Instruction Scheduling



(a) A CDC 6600-like processor



(b) The improved schedule from Fig. 6.9b

Fig. 6.17 Hardware scoreboarding for dynamic instruction scheduling (Courtesy of James Smith; reprinted with permission from *IEEE Computer*, July 1989)

INSTRUCTION PIPELINE DESIGN
Branch Handling Techniques

- Branch Taken, Branch Target, Delay Slot
- Effect of Branching
  - Parameters:
    - **k** : No. of stages in the pipeline
    - **n** : Total no. of instructions or tasks
    - **p** : Percentage of Brach instructions over **n**
    - **q** : Percentage of successful branch instructions (branch taken) over **p**.
    - **b** : Delay Slot
    - τ : Pipeline Cycle Time
  - Branch Penalty = **q of (p of n) * bτ = pqnbτ**
  - Effective Execution Time:
    - T_{eff} **= [k + (n-1)]** τ **+ pqnb**τ **= [k + (n-1) + pqnb]**τ

INSTRUCTION PIPELINE DESIGN
Branch Handling Techniques

- Effect of Branching
  - Effective Throughput:
    - $H_{eff} = n/T_{eff}$
    - $H_{eff} = n / \{[k + (n-1) + pqnb]\tau\} = nf / [k + (n-1) + pqnb]$
    - As $n \rightarrow Infinity$ and $b = k-1$
      - $H^*_{eff} = f / [pq(k-1)+1]$
    - If $p=0$ and $q=0$ (no branching occurs)
      - $H^{**}_{eff} = f = 1/\tau$
  - Performance Degradation Factor
    - $D = 1 - H^*_{eff} / f = pq(k-1) / [pq(k-1)+1]$

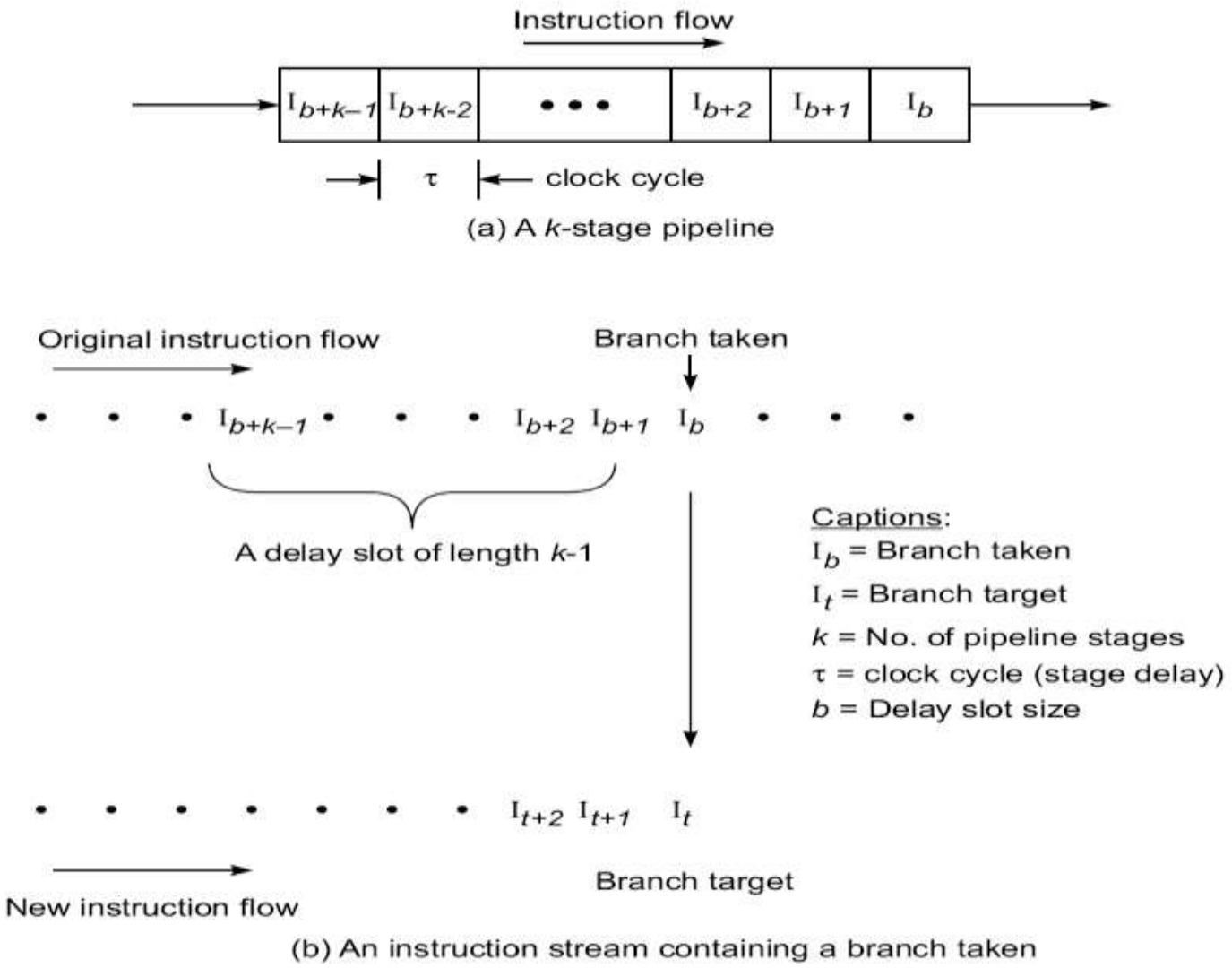
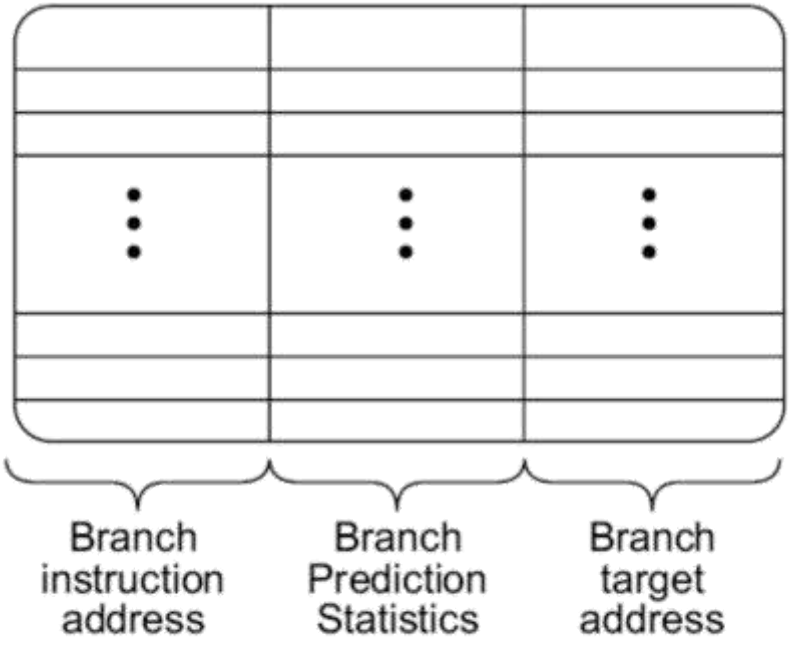


Fig. 6.18 The decision of a branch taken at the last stage of an instruction pipeline causes $b \leq k - 1$ previously loaded instructions to be drained from the pipeline

INSTRUCTION PIPELINE DESIGN
Branch Handling Techniques

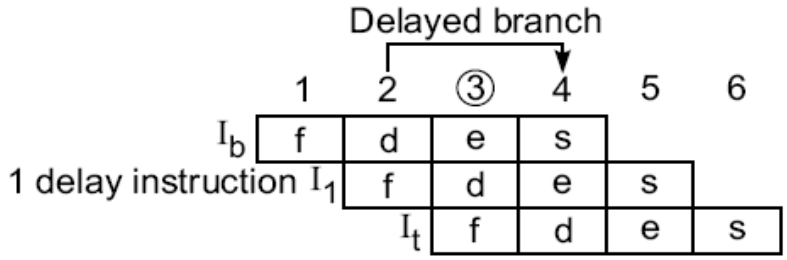
- Branch Prediction
  - **Static Branch Prediction:** based on branch code types
  - **Dynamic Branch prediction:** based on recent branch history
    - **Strategy 1:** Predict the branch direction based on information found at decode stage.
    - **Strategy 2:** Use a cache to store target addresses at effective address calculation stage.
    - **Strategy 3:** Use a cache to store target instructions at fetch stage
  - **Brach Target Buffer Organization**
- Delayed Branches
  - A *delayed branch* of ***d*** cycles allows at most ***d-1*** useful instructions to be executed following the *branch taken*.
  - Execution of these instructions should be independent of branch instruction to achieve a **zero branch penalty**

INSTRUCTION PIPELINE DESIGN
Branch Handling Techniques

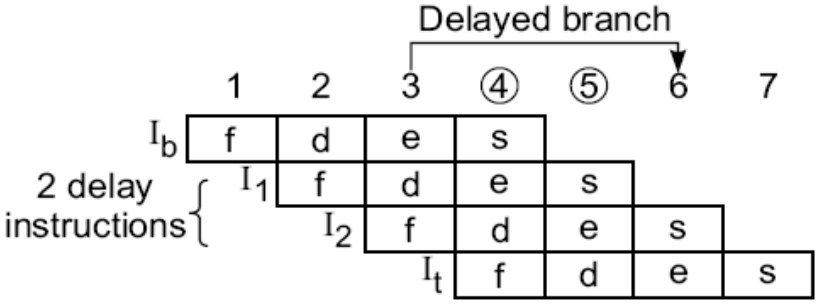


(a) Branch target buffer organization

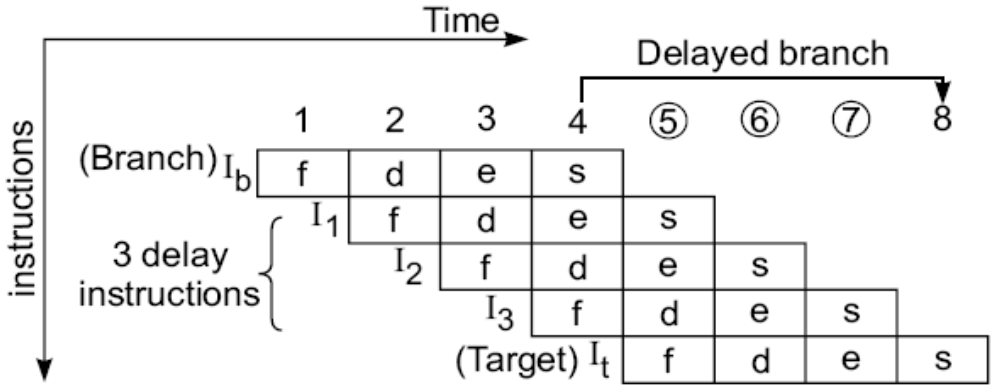
Fig. 6.19 Branch history buffer and a state transition diagram used in dynamic branch prediction (Courtesy of Lee and Smith, *IEEE Computer*, 1984)



(a) A delayed branch for 2 cycles when the branch condition is resolved at the decode stage



(b) A delayed branch for 3 cycles when the branch condition is resolved at the execute stage



(c) A delayed branch for 4 cycles when the branch condition is resolved at the store stage

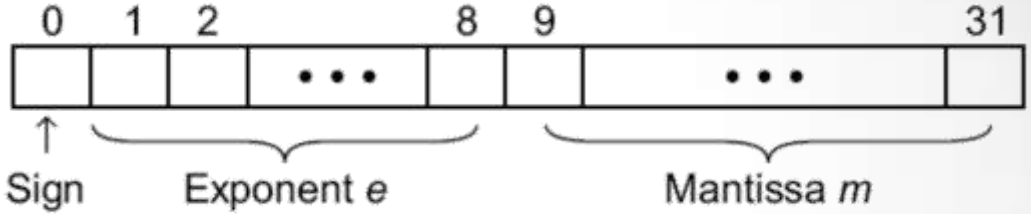
Fig. 6.20 The concept of delayed branch by moving independent instructions or NOP fillers into the delay slot of a four-stage pipeline

ARITHMETIC PIPELINE DESIGN
Computer Arithmetic Operations

- Finite-precision arithmetic
- Overflow and Underflow
- Fixed-Point operations
  - Notations:
    - *Signed-magnitude, one's complement* and *two-complement* notation
  - Operations:
    - *Addition*: (n bit, n bit) ➔ (n bit) Sum, 1 bit output carry
    - *Subtraction*: (n bit, n bit) ➔ (n bit) difference
    - *Multiplication*: (n bit, n bit) ➔ (2n bit) product
    - *Division*: (2n bit, n bit) ➔ (n bit) quotient, (n bit) remainder

ARITHMETIC PIPELINE DESIGN
Computer Arithmetic Operations

- Floating-Point Numbers
  - ***X = (m, e)*** representation
    - ***m***: mantissa or fraction
    - ***e***: exponent with an implied base or radix ***r***.
    - Actual Value ***X = m * r*** ^{e}



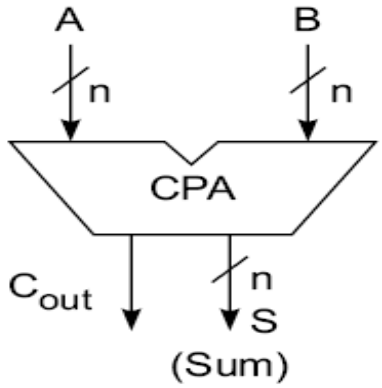
  - Operations on numbers $X = (m_x, e_x)$ and $Y = (m_y, e_y)$
    - *Addition*: $(m_x * r^{ex-ey} + m_y, e_y)$
    - *Subtraction*: $(m_x * r^{ex-ey} - m_y, e_y)$
    - *Multiplication*: $(m_x * m_y, e_x+e_y)$
    - *Division*: $(m_x / m_y, e_x - e_y)$
- Elementary Functions
  - Transcendental functions like: Trigonometric, Exponential, Logarithmic, etc.

ARITHMETIC PIPELINE DESIGN
Static Arithmetic Pipelines

- Separate units for fixed point operations and floating point operations
- Scalar and Vector Arithmetic Pipelines
- Uni-functional or Static Pipelines
- Arithmetic Pipeline Stages
  - Majorly involve hardware to perform: **Add** and **Shift** micro-operations
  - **Addition** using: *Carry Propagation Adder (CPA)* and *Carry Save Adder (CSA)*
  - **Shift** using: *Shift Registers*
- Multiplication Pipeline Design
  - E.g. To multiply two 8-bit numbers that yield a 16-bit product using CSA and CPA *Wallace Tree.*

e.g. n=4

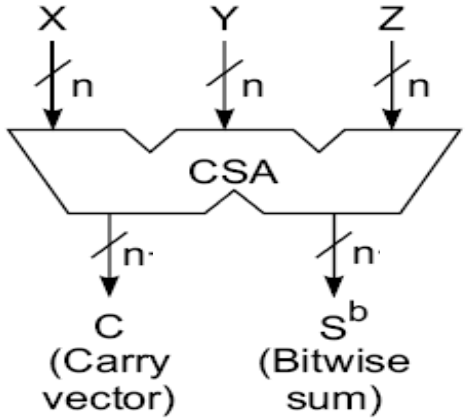
```
      A =   1 0 1 1
+)    B =   0 1 1 1
-------------------
      S = 1 0 0 1 0 = A + B
```



(a) An *n*-bit carry-propagate adder (CPA) which allows either carry propagation or applies the carry-lookahead technique

e.g. n=4

```
      X =   0 0 1 0 1 1
      Y =   0 1 0 1 0 1
⊕     Z =   1 1 1 1 0 1
-------------------------
      S^b = 0 1 0 0 0 1 1
+)    C = 0 1 1 1 0 1 0
-------------------------
      S = 1 0 1 1 1 0 1 = S^b+C = X+Y+Z
```



(b) An *n*-bit carry-save adder (CSA), where S^b is the bitwise sum of X, Y, and Z, and C is a carry vector generated without carry propagation between digits

Fig. 6.22 Distinction between a carry-propagate adder (CPA) and a carry-save adder (CSA)

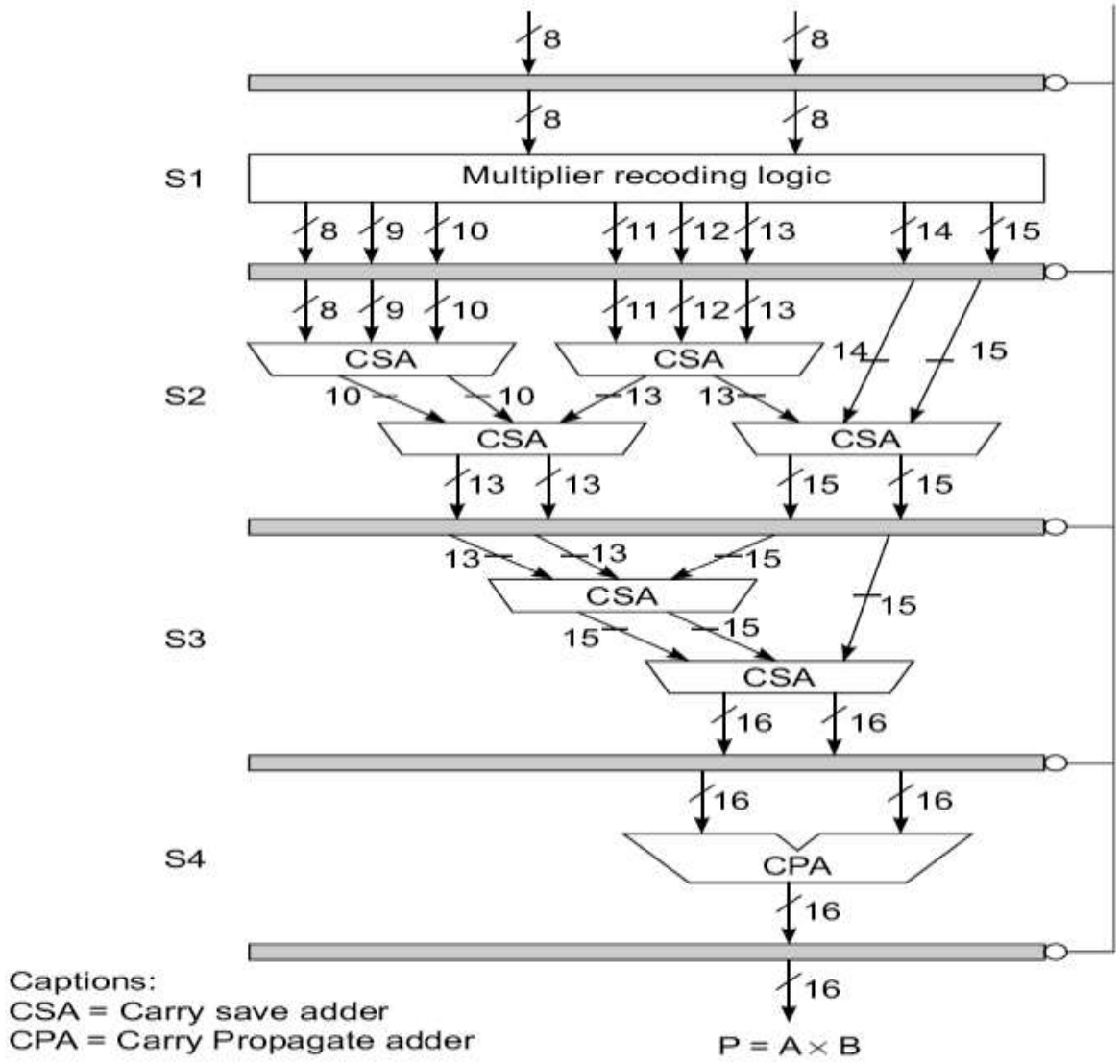


Fig. 6.23 A pipeline unit for fixed-point multiplication of 8-bit integers (The number along each line indicates the line width.)

ARITHMETIC PIPELINE DESIGN
Multifunctional Arithmetic Pipelines

- Multifunctional Pipeline:
  - **Static** multifunctional pipeline
  - **Dynamic** multifunctional pipeline
- Case Study: T1/ASC static multifunctional pipeline architecture

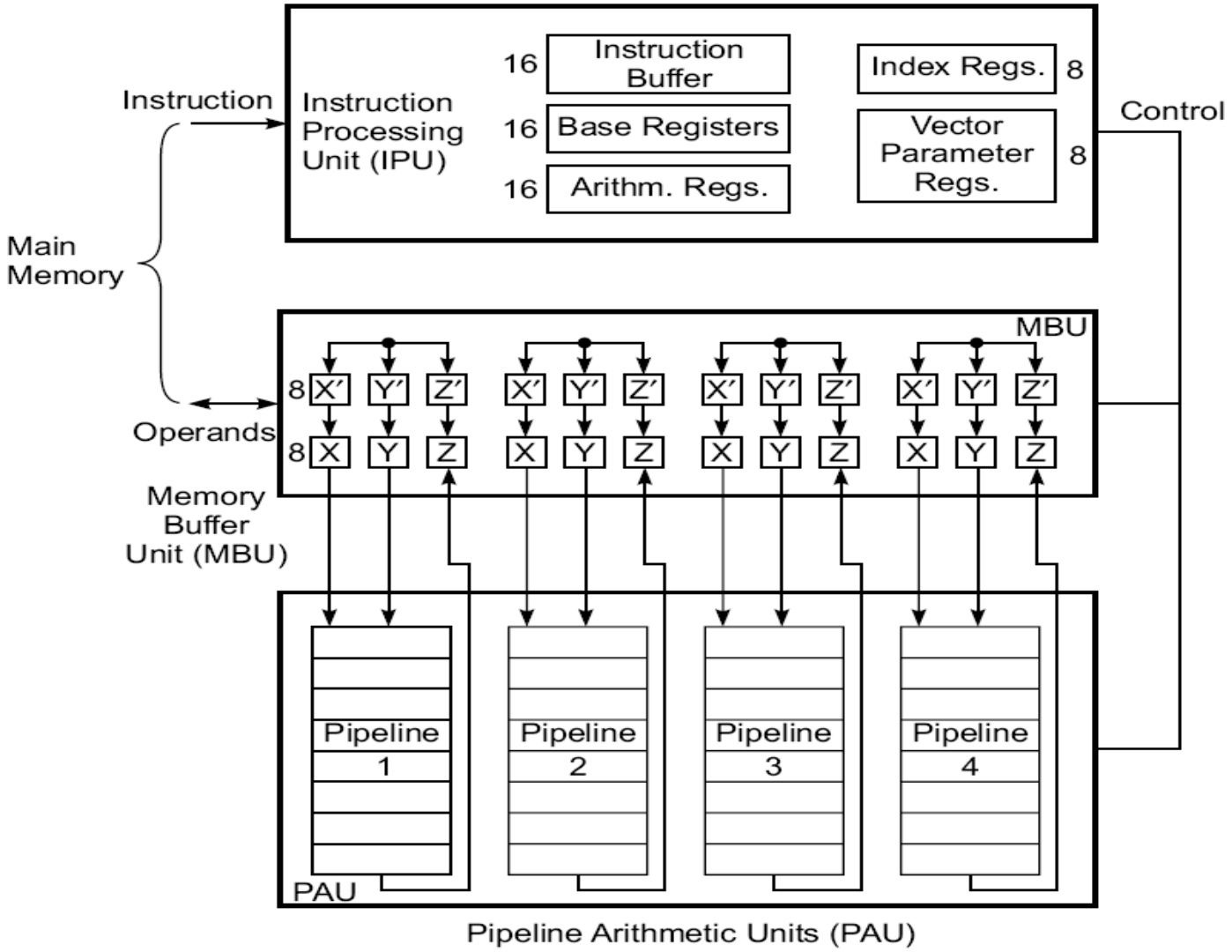
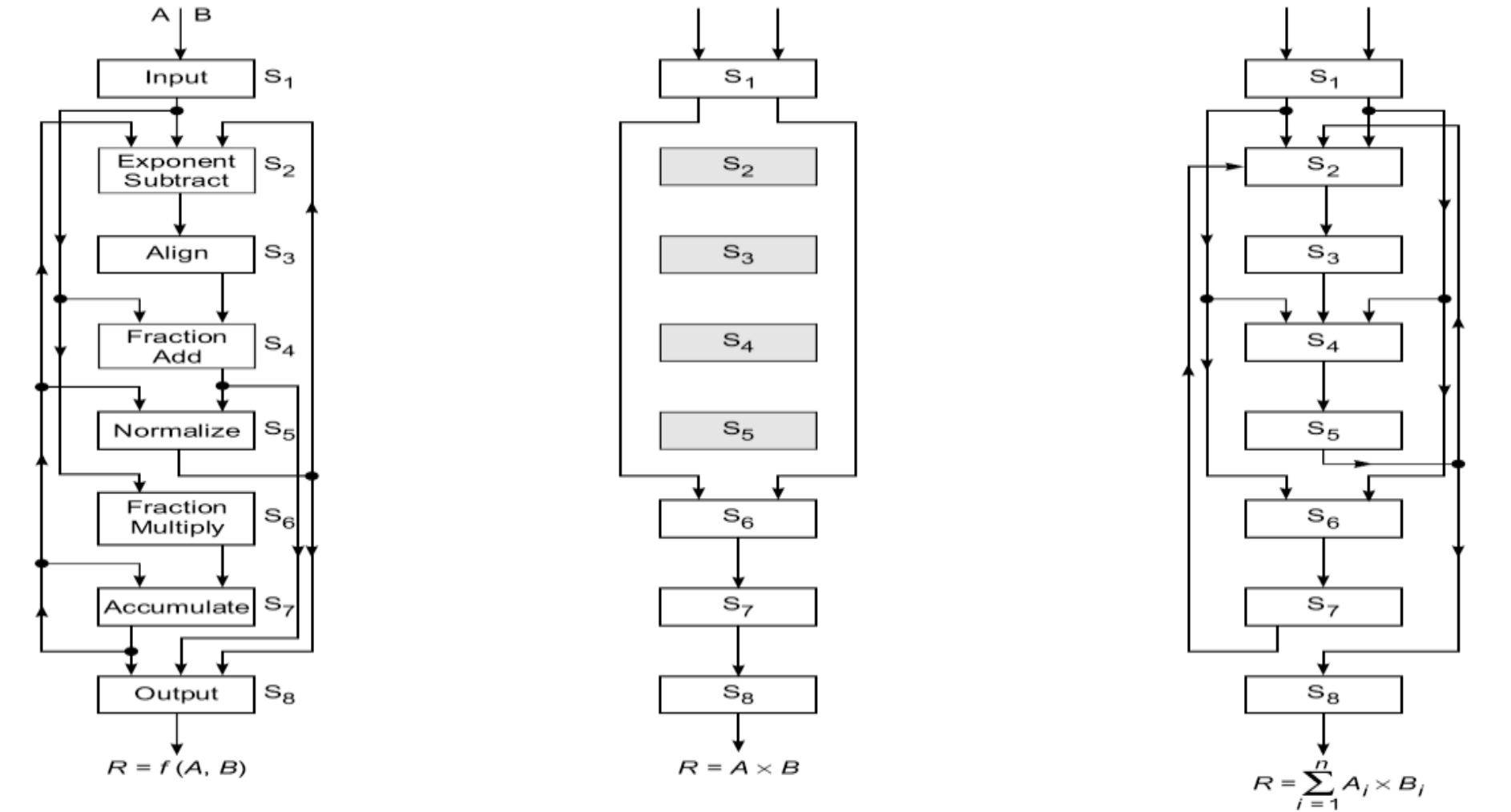


Fig. 6.26 The architecture of the TI Advanced Scientific Computer (ASC) (Courtesy of Texas Instruments, Inc.)



(a) Pipeline stages and interconnections

(b) Fixed-point multiplication

(c) Floating-point dot product

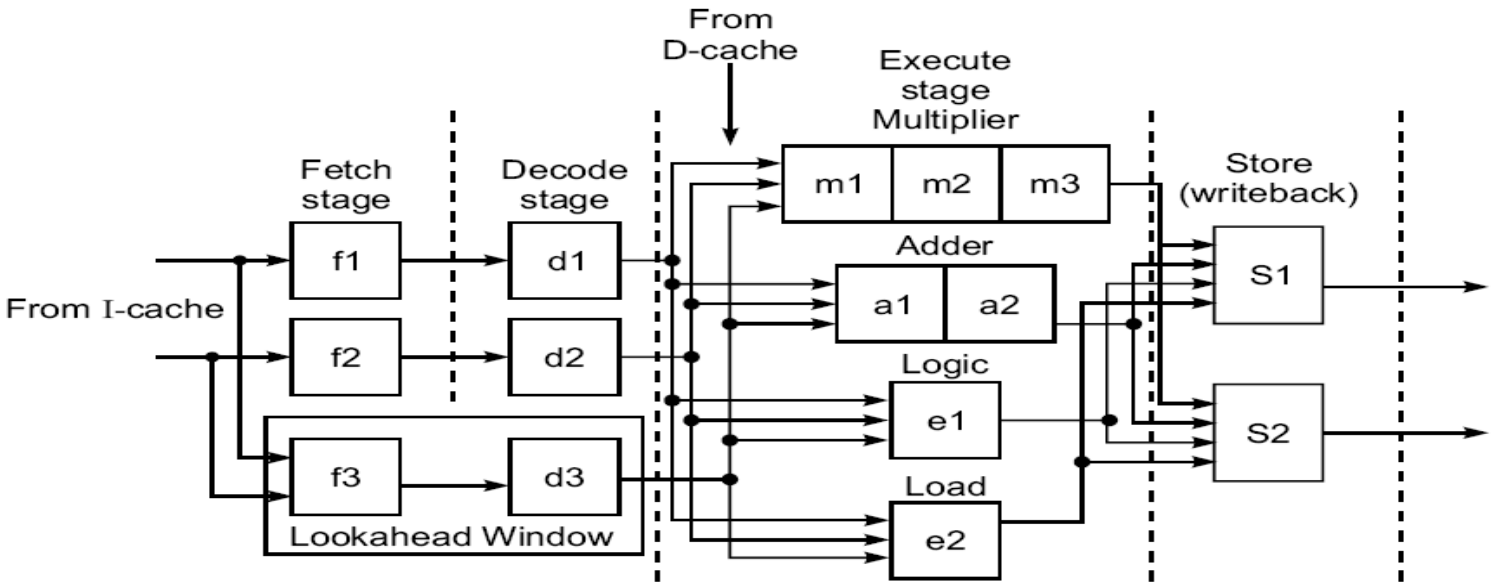
Fig. 6.27 The multiplication arithmetic pipeline of the TI Advanced Scientific Computer and the interstage connections of two representative functions (Shaded stages are unutilized)

SUPERSCALAR PIPELINE DESIGN

- Pipeline Design Parameters
  - Pipeline cycle, Base cycle, Instruction issue rate, Instruction issue Latency, Simple Operation Latency
  - ILP to fully utilize the pipeline
- Superscalar Pipeline Structure
- Data and Resource Dependencies
- Pipeline Stalling
- Superscalar Pipeline Scheduling
  - In-order Issue and in-order completion
  - In-order Issue and out-of-order completion
  - Out-of-order Issue and out-of-order completion
- Superscalar Performance

SUPERSCALAR PIPELINE DESIGN

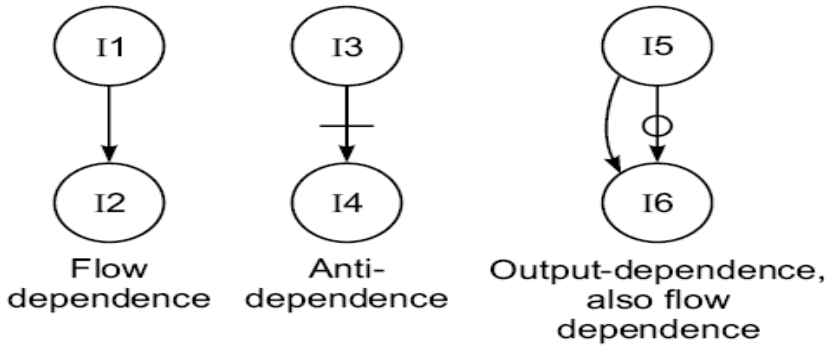
Parameter	Base Scalar Processor	Super Scalar Processor (degree = K)
Pipeline Cycle	1 (base cycle)	K
Instruction Issue Rate	1	K
Instruction Issue Latency	1	1
Simple Operation Latency	1	1
ILP to fully utilize pipeline	1	K



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

Program order

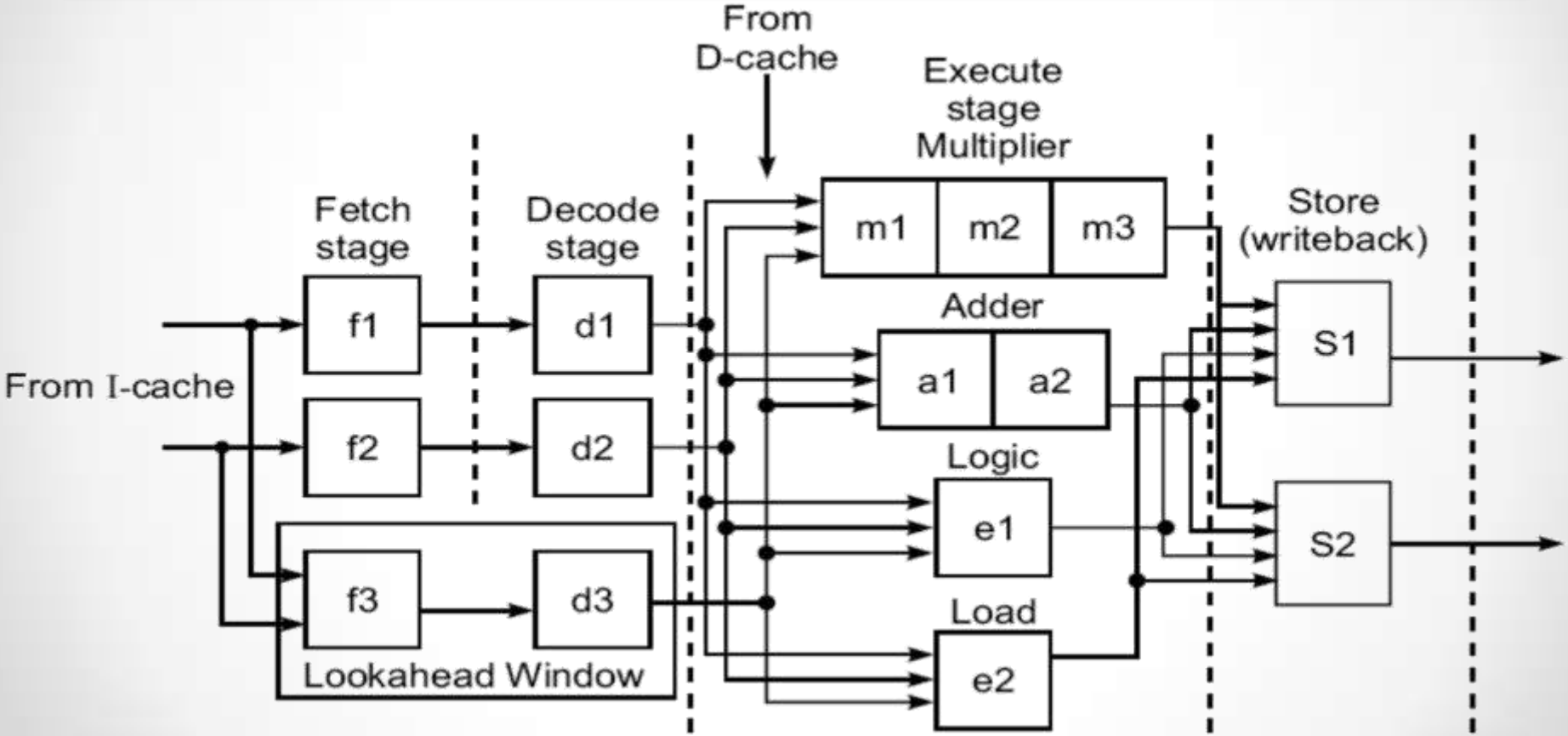
```
I1.  Load  R1,  A    / R1 ← Memory (A) /
I2.  Add   R2,  R1   / R2 ← (R2) + (R1) /
I3.  Add   R3,  R4   / R3 ← (R3) + (R4) /
I4.  Mul   R4,  R5   / R4 ← (R4) * (R5) /
I5.  Comp  R6        / R6 ← (R6) /
I6.  Mul   R6,  R7   / R6 ← (R6) * (R7) /
```



(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

SUPERSCALAR PIPELINE DESIGN



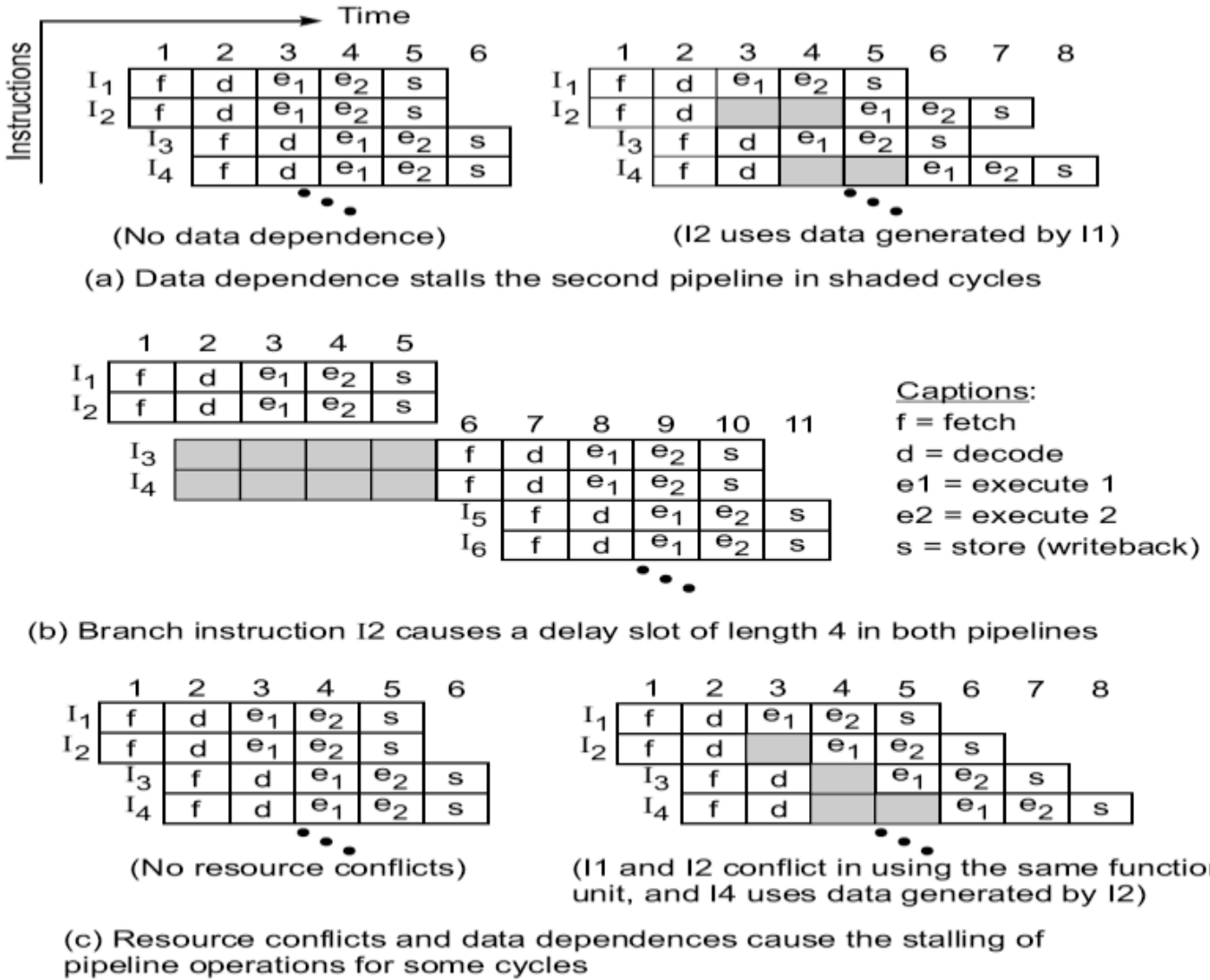
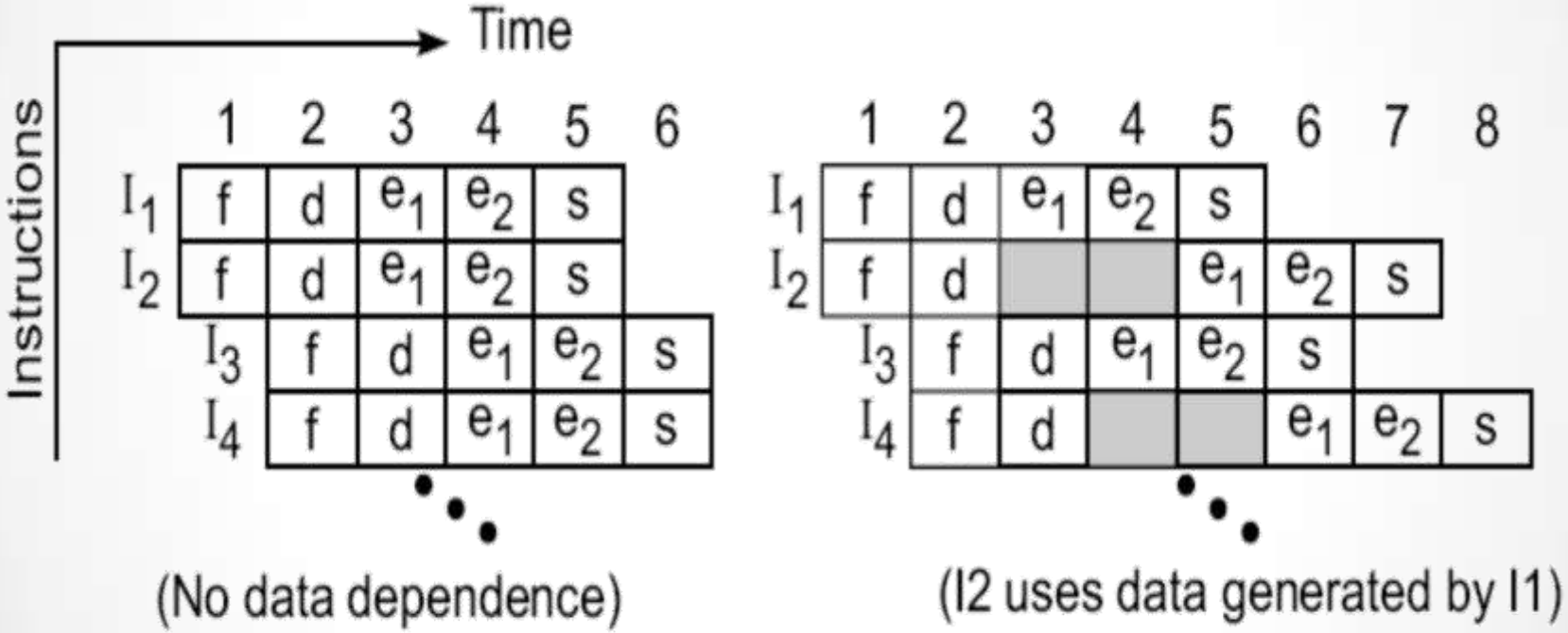


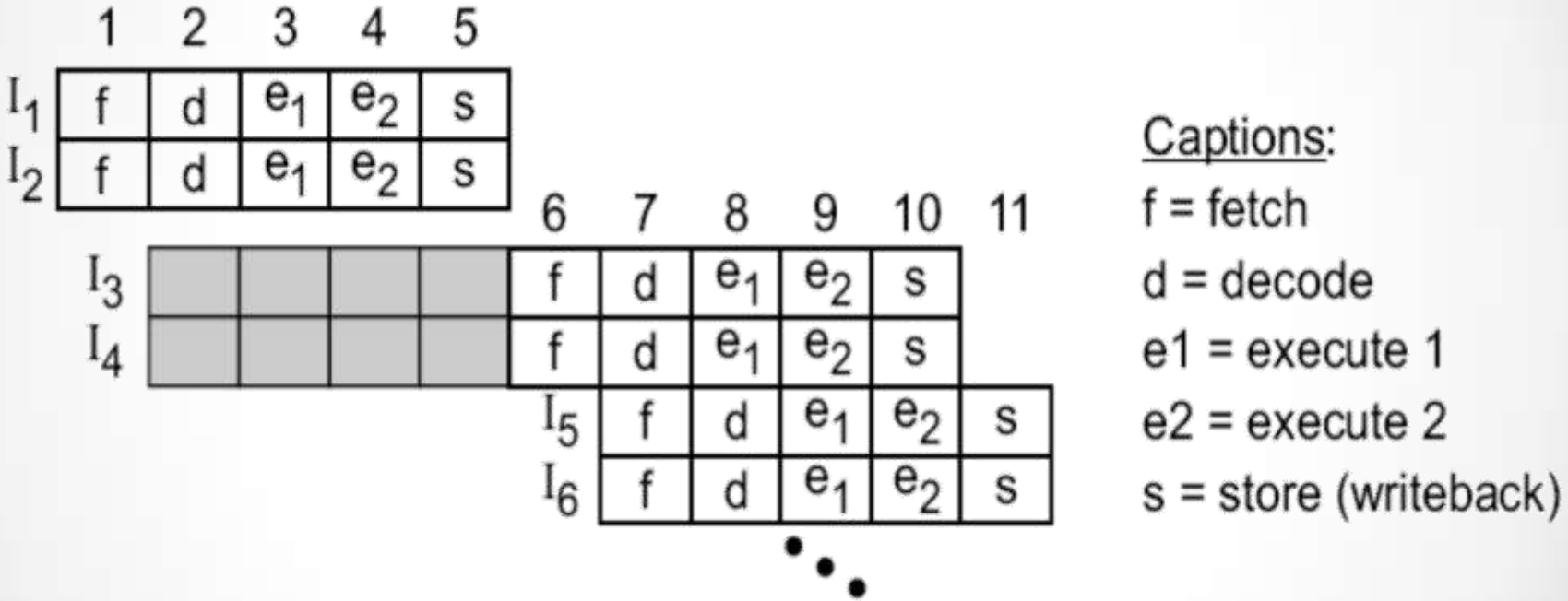
Fig. 6.29 Dependences and resource conflicts may stall one or two pipelines in a two-issue superscalar processor

SUPERSCALAR PIPELINE DESIGN



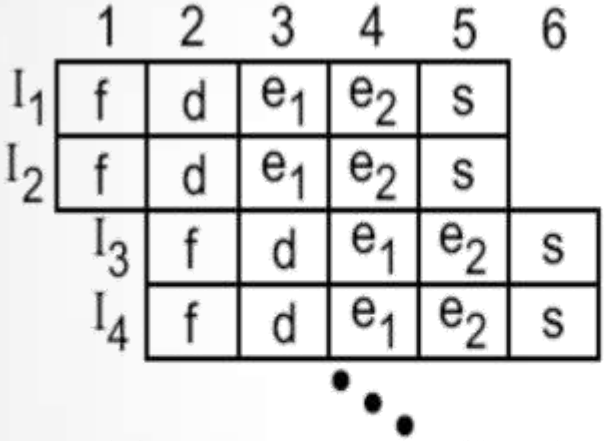
(a) Data dependence stalls the second pipeline in shaded cycles

SUPERSCALAR PIPELINE DESIGN

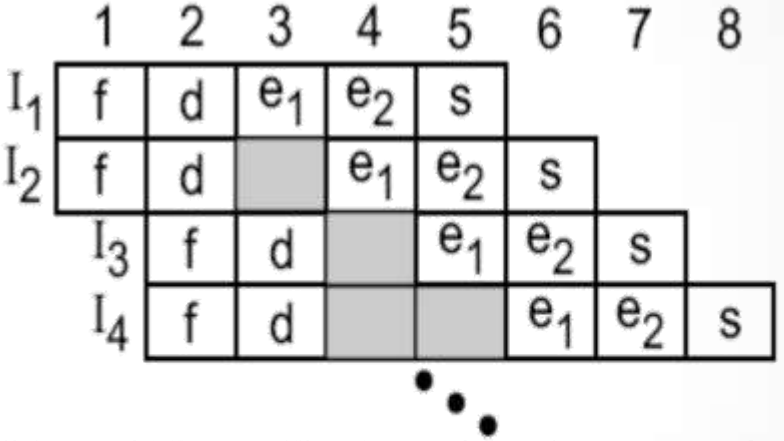


(b) Branch instruction I2 causes a delay slot of length 4 in both pipelines

SUPERSCALAR PIPELINE DESIGN



(No resource conflicts)



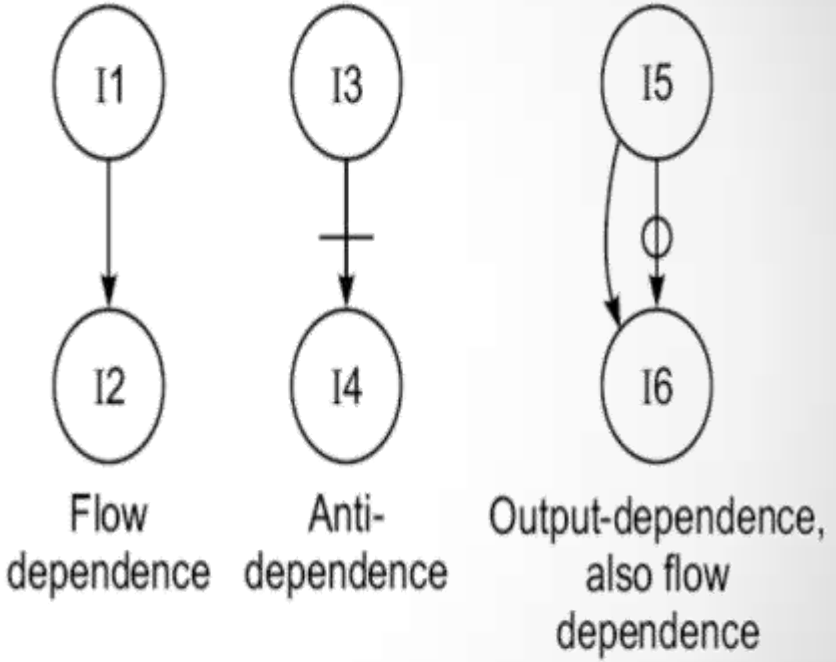
(I1 and I2 conflict in using the same functional unit, and I4 uses data generated by I2)

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

SUPERSCALAR PIPELINE DESIGN

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   R4,  R5   / R4 ← (R4) * (R5) /
I5.  Comp  R6        / R6 ← (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

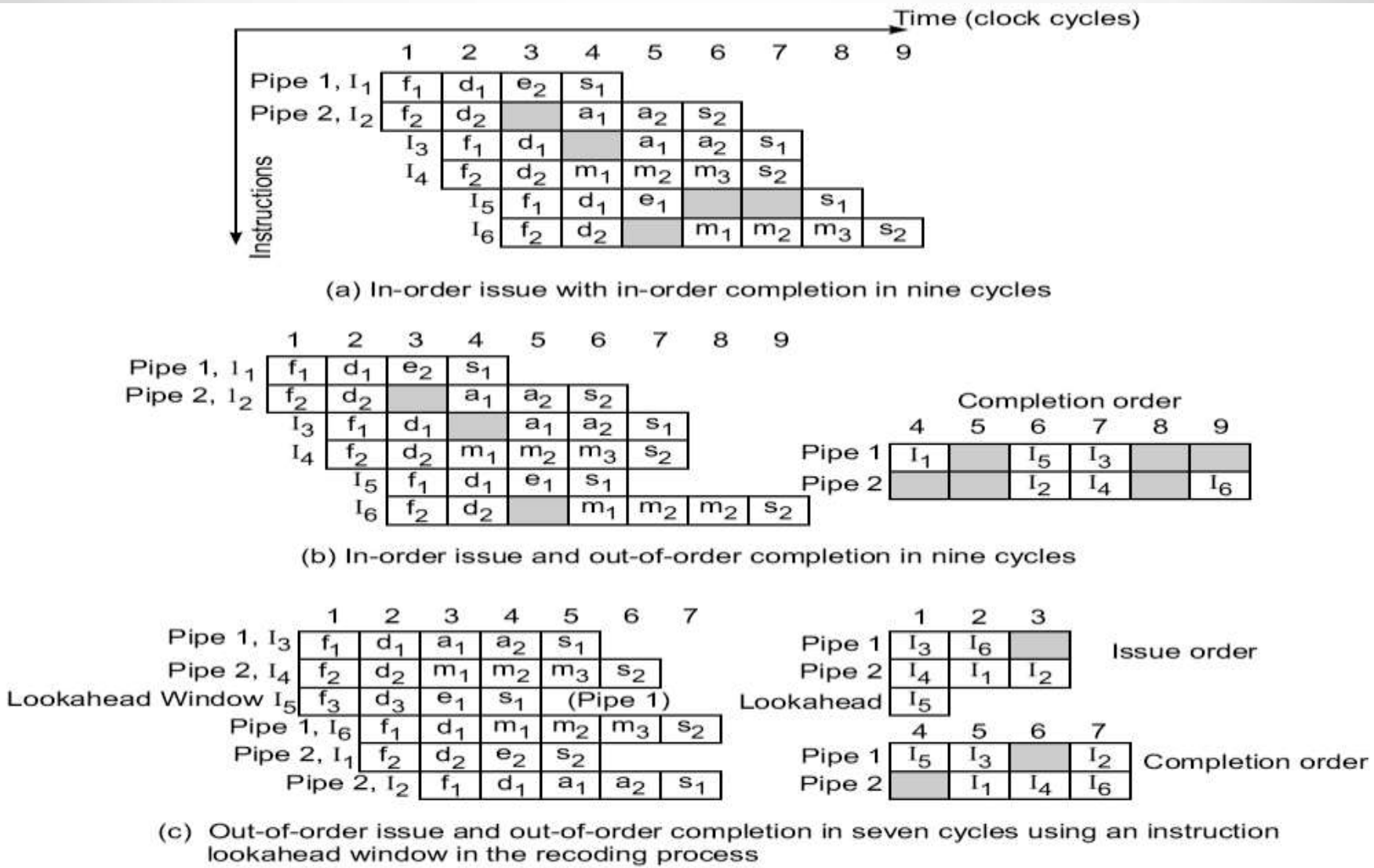


Fig. 6.30 Instruction issue and completion policies for a superscalar processor with and without instruction lookahead support (Timing charts correspond to parallel execution of the program in Fig. 6.28)

SUPERSCALAR PIPELINE DESIGN

- Time required by base scalar machine:
  - T(1,1) = K + N – 1
- The ideal execution time required by m-issue superscalar machine:
  - T(m,1) = K + (N – m)/m
  - Where,
    - K is the time required to execute first m instructions through m pipelines of k-stages simultaneously
    - Second term corresponds to time required to execute remaining N-m instructions , m per cycle through m pipelines
- The ideal speedup of superscalar machine
  - S(m,1) = T(1,1)/T(m,1) = m(N + k – 1)/[N+ m(k – 1)]
- As n → *infinity*, S(m,1) →m