SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE NAME :19IT301 COMPUTER ORGANIZATION AND ARCHITECTURE

II YEAR /III SEMESTER

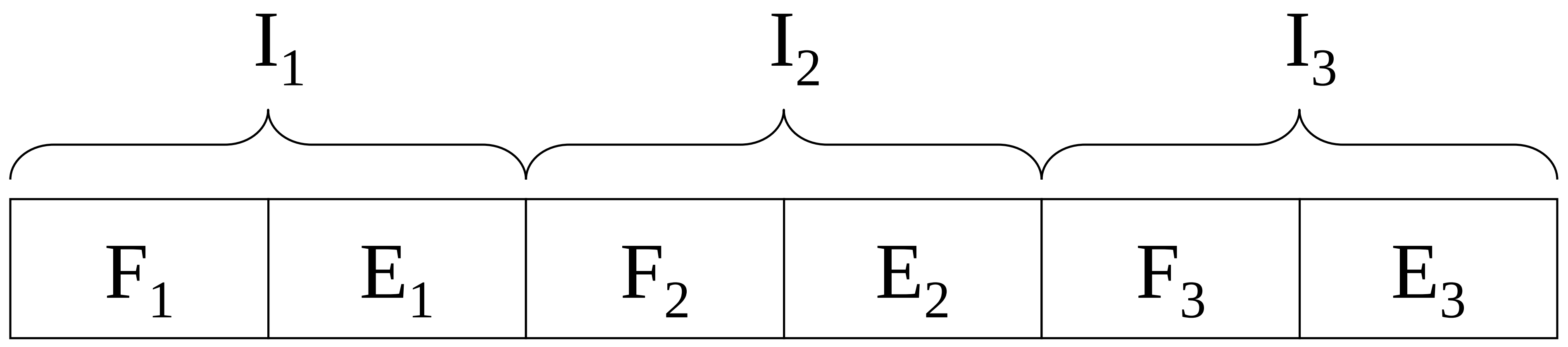
Unit 3- **PROCESSOR AND PIPELINING**

Topics : Pipelining: Basic concepts – Data hazards – Instruction hazards – Influence on Instruction sets – Data path and control consideration.

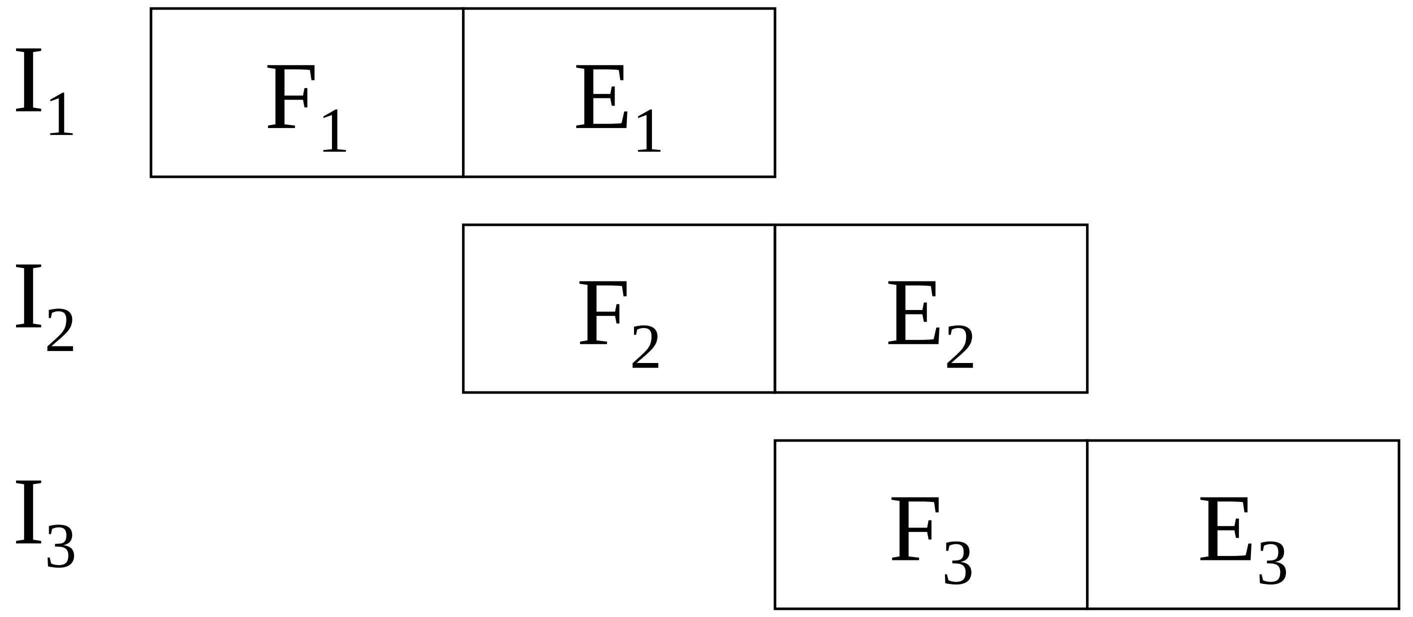
Pipelining

A TECHNIQUE TO IMPROVE THE PERFORMANCE BY OVERLAPPING THE EXECUTION OF SUCCESSIVE INSTRUCTIONS.

Pipelining

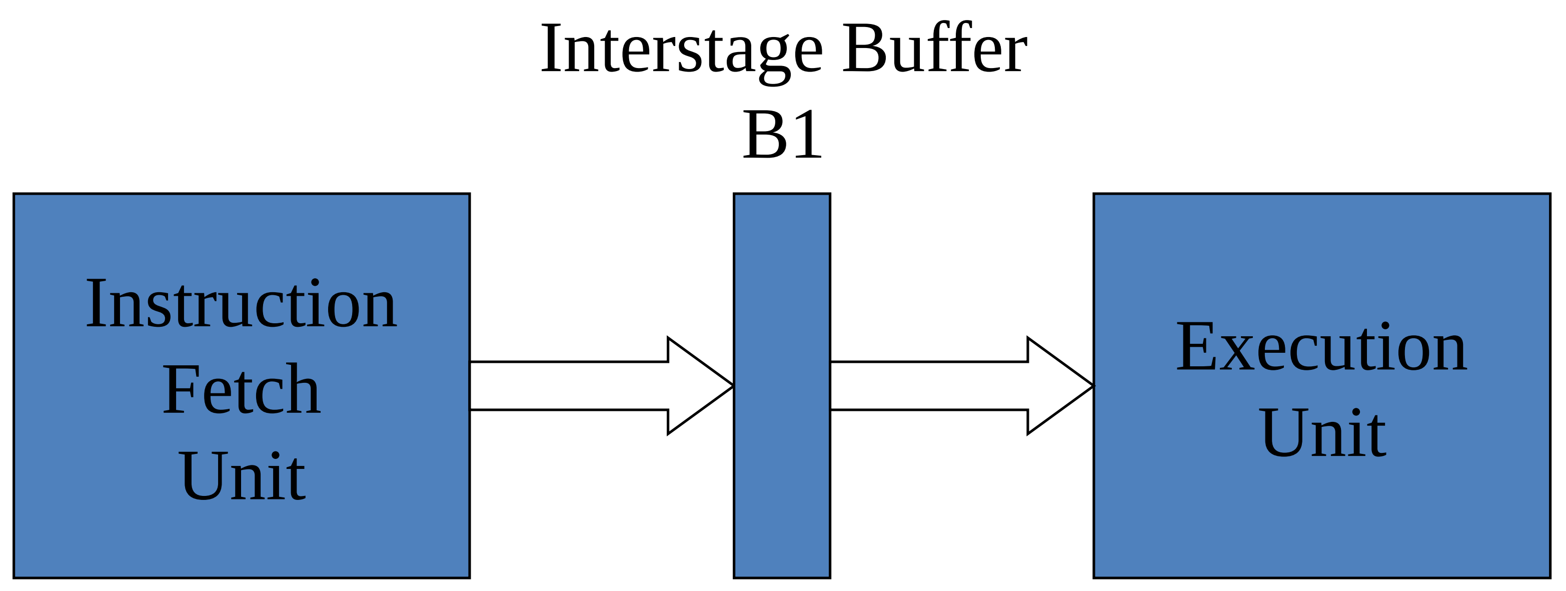


Sequential Execution



Pipelined Execution

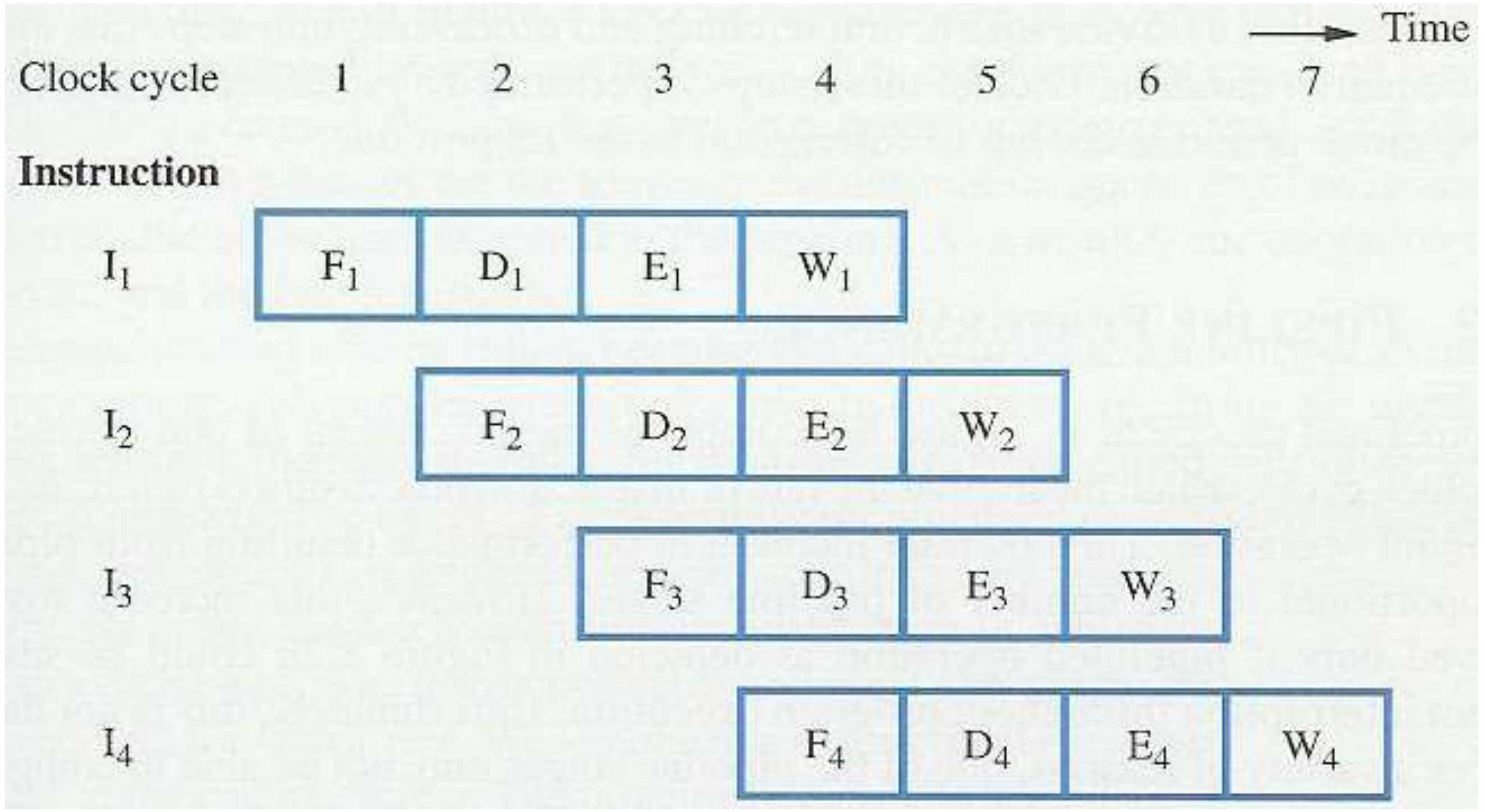
Hardware Organization



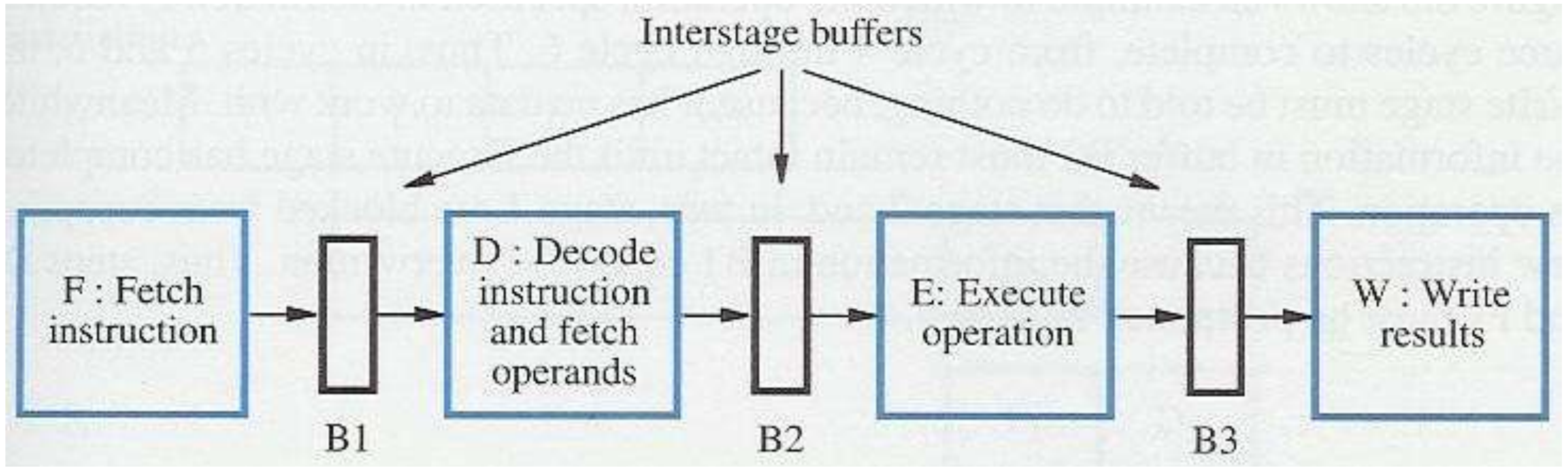
Four State Pipeline

- Fetch (F)
  - Read the instruction from memory
- Decode (D)
  - Decode the instruction and fetch the source operand(s)
- Execute (E)
  - Perform the operation specified by the instruction
- Write (W)
  - Store the result in the destination location

Four Stage Pipeline



Hardware Organization



Role of cache memory

- ✓ Pipelining is more effective in improving performance if the tasks being performed in different stages require about the same amount of time.
- ✓ But access time of main memory is high compared to the access time of registers.
- ✓ This problem is solved by cache memory.

PIPELINE PERFORMANCE

- ✓ Data Hazard
- ✓ Instruction Hazards
- ✓ Structural Hazard

Data Hazard

- ✓ It is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason.
- ✓ Source or destination operands not available at time expected in the pipeline
- ✓ Execution operation taking more than one clock cycle.

Example:

A=5

$A \longleftarrow 3+A$

$B \longleftarrow 4*A$

Data Hazard

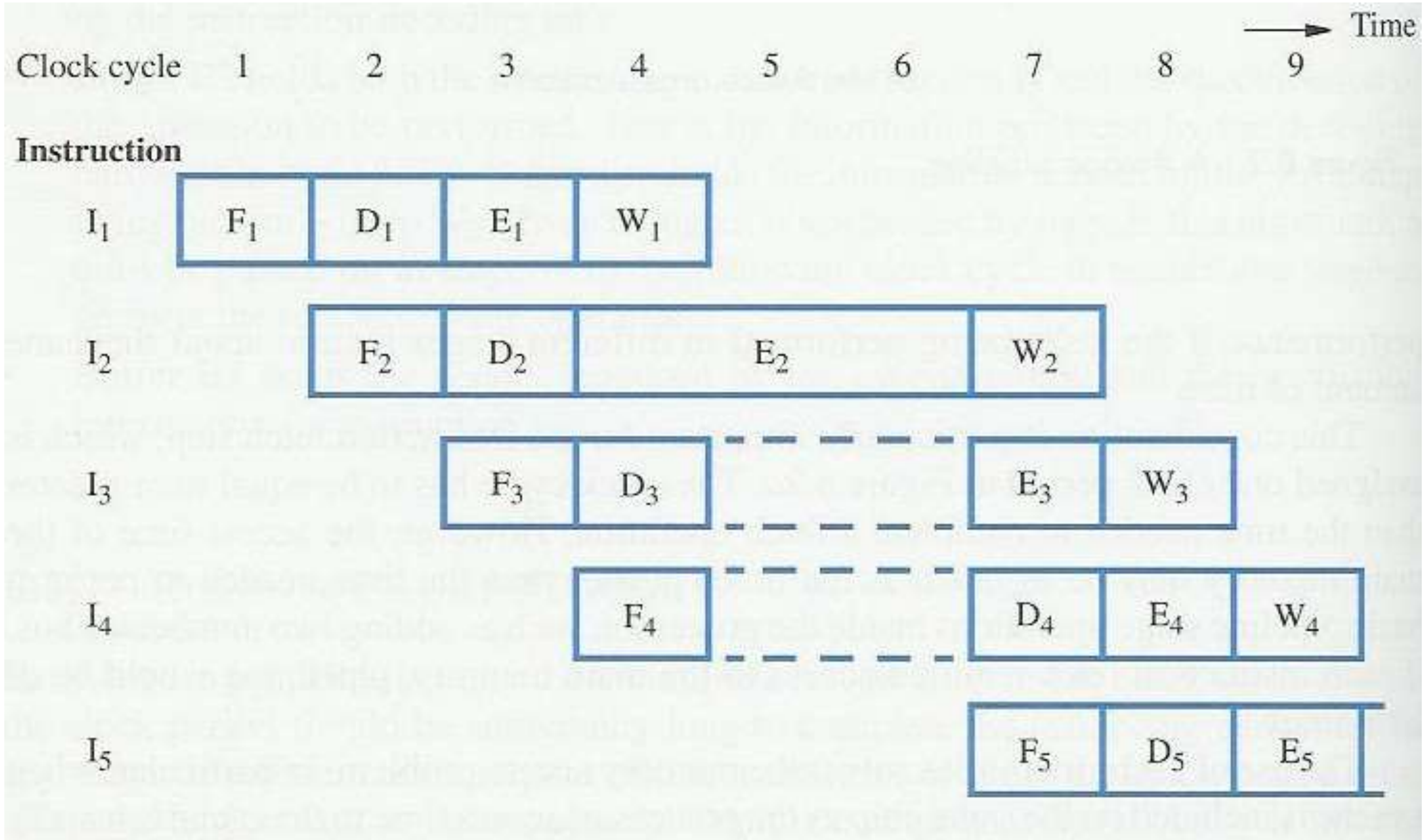

Time
Clock cycle 1 2 3 4 5 6 7 8 9
Instruction
I_1 F_1 D_1 E_1 W_1
I_2 F_2 D_2 E_2 W_2
I_3 F_3 D_3 E_3 W_3
I_4 F_4 D_4 E_4 W_4
I_5 F_5 D_5 E_5

Figure 8.3 Effect of an execution operation taking more than one clock cycle.

Data Dependency
Ex: Mul R2,R3,R4

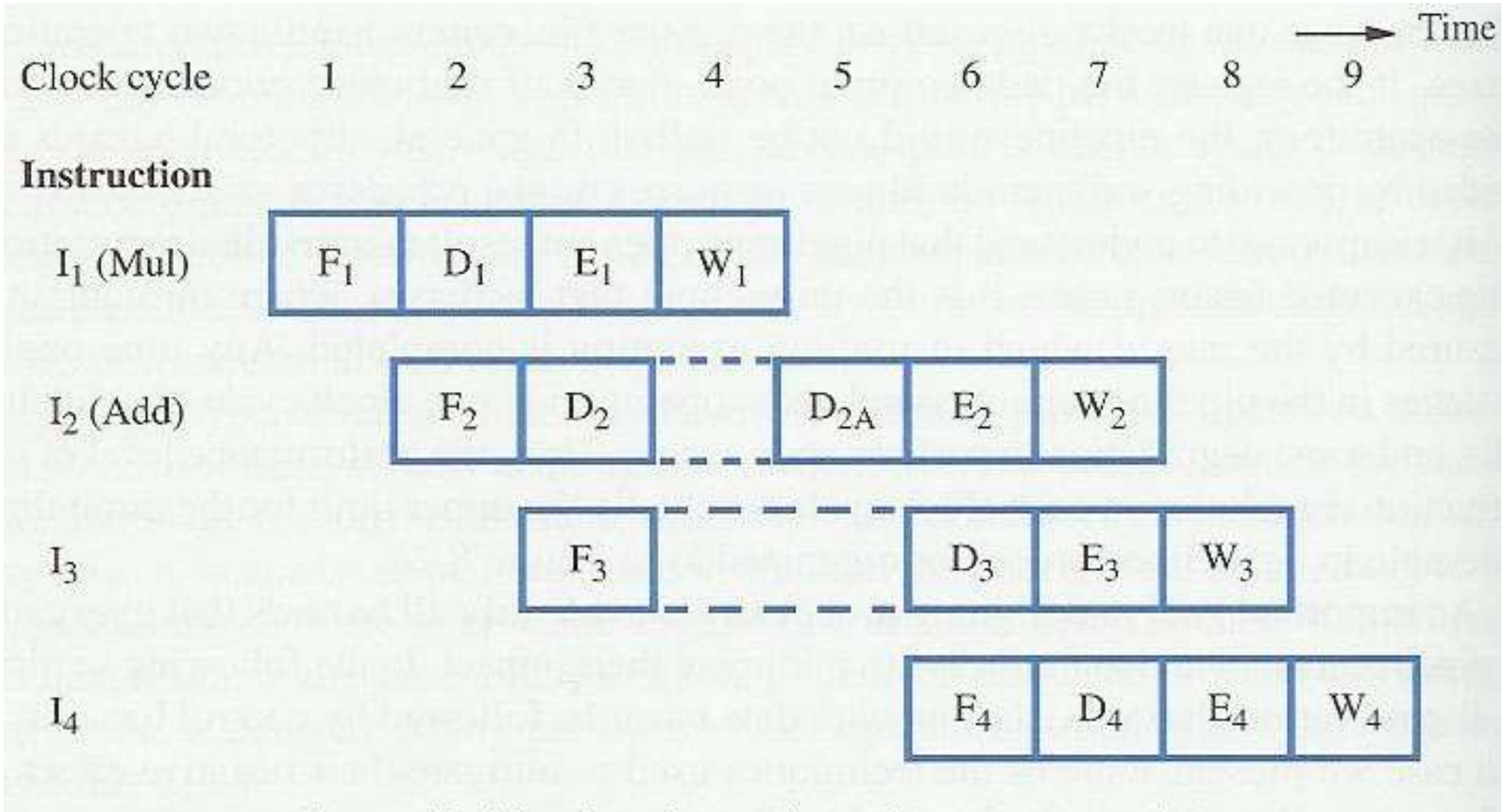
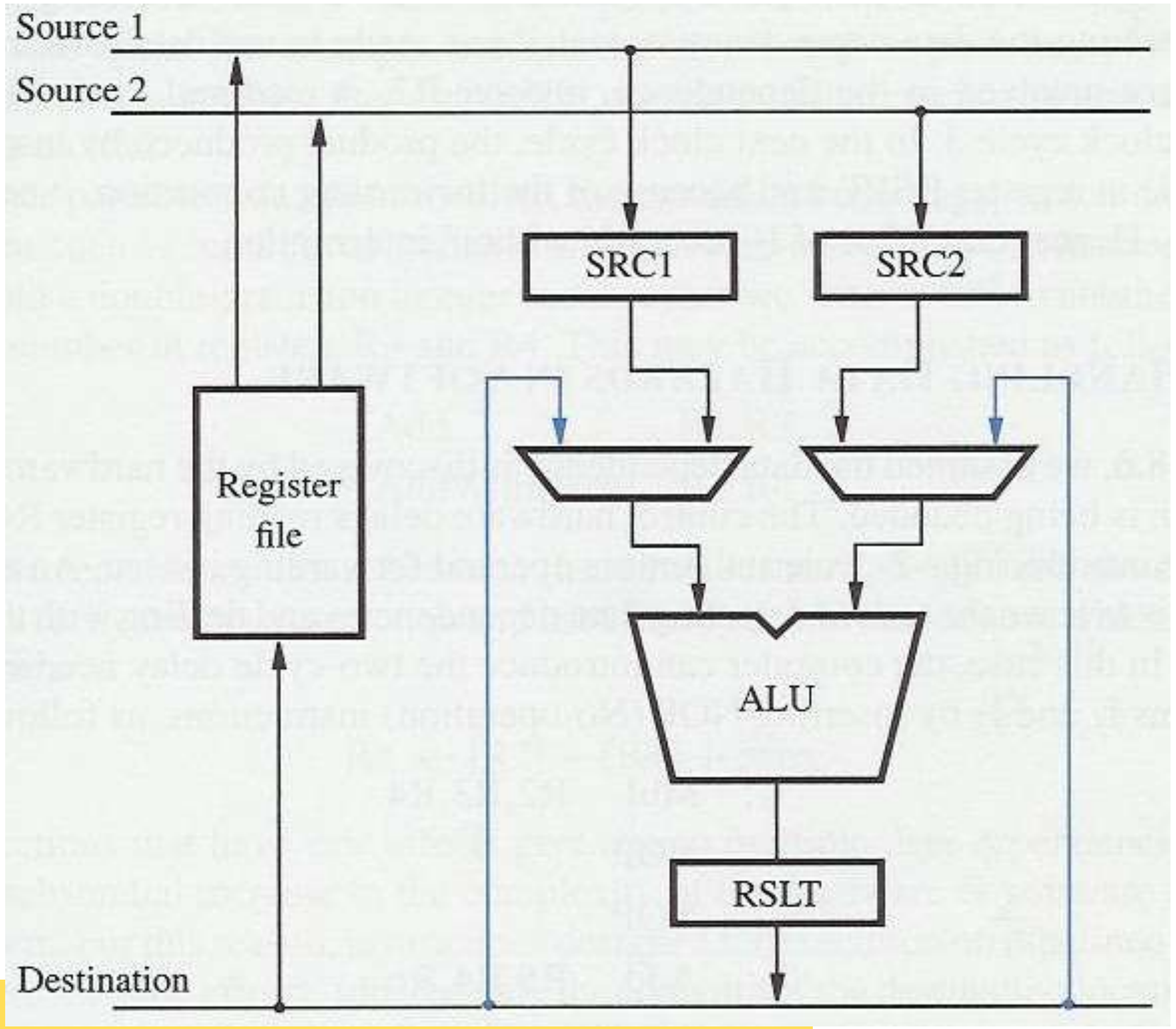
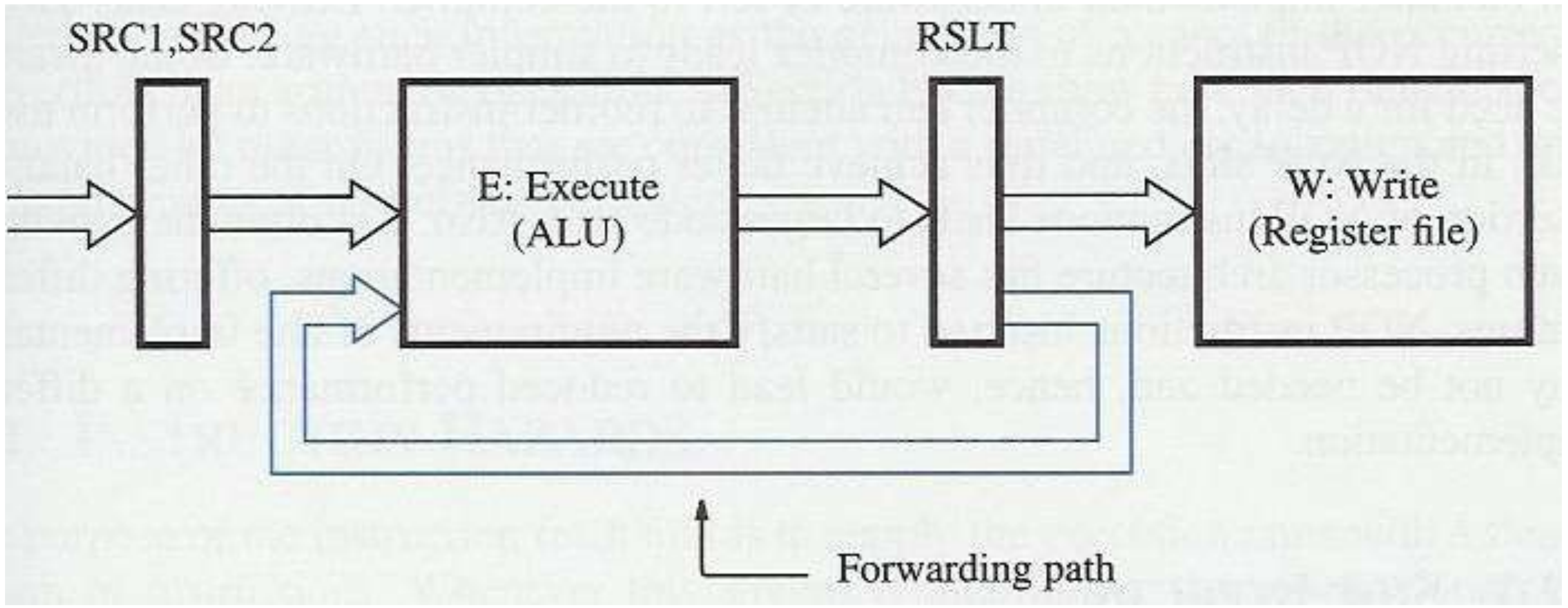

Time
Clock cycle 1 2 3 4 5 6 7 8 9
Instruction
I_1 (Mul) F_1 D_1 E_1 W_1
I_2 (Add) F_2 D_2 D_{2A} E_2 W_2
I_3 F_3 D_3 E_3 W_3
I_4 F_4 D_4 E_4 W_4

Figure 8.6 Pipeline stalled by data dependency between D_2 and W_1.

Operand Forwarding



Operand Forwarding



Handling Data Hazards in SW

Compiler detect data dependencies and deal with them

- Insert NOPs
- Attempt to reorder instructions to perform useful tasks in NOP slots.
- Ex:

```
I1:     Mul R2,R3,R4
        NOP
        NOP
I2:    Add R5,R4,R6
```

Side effects

- ✓ Instruction changes contents of a register other than the named destination
  - ✓ Autoincrement/autodecrement addressing modes. Ex: PUSH & POP OPERATIONS.
  - ✓ Condition code flags   Ex: Add R1,R2
  - ✓   Addwithcarry R3,R4
- ✓ Give rise to multiple dependencies
- ✓ Should be minimized

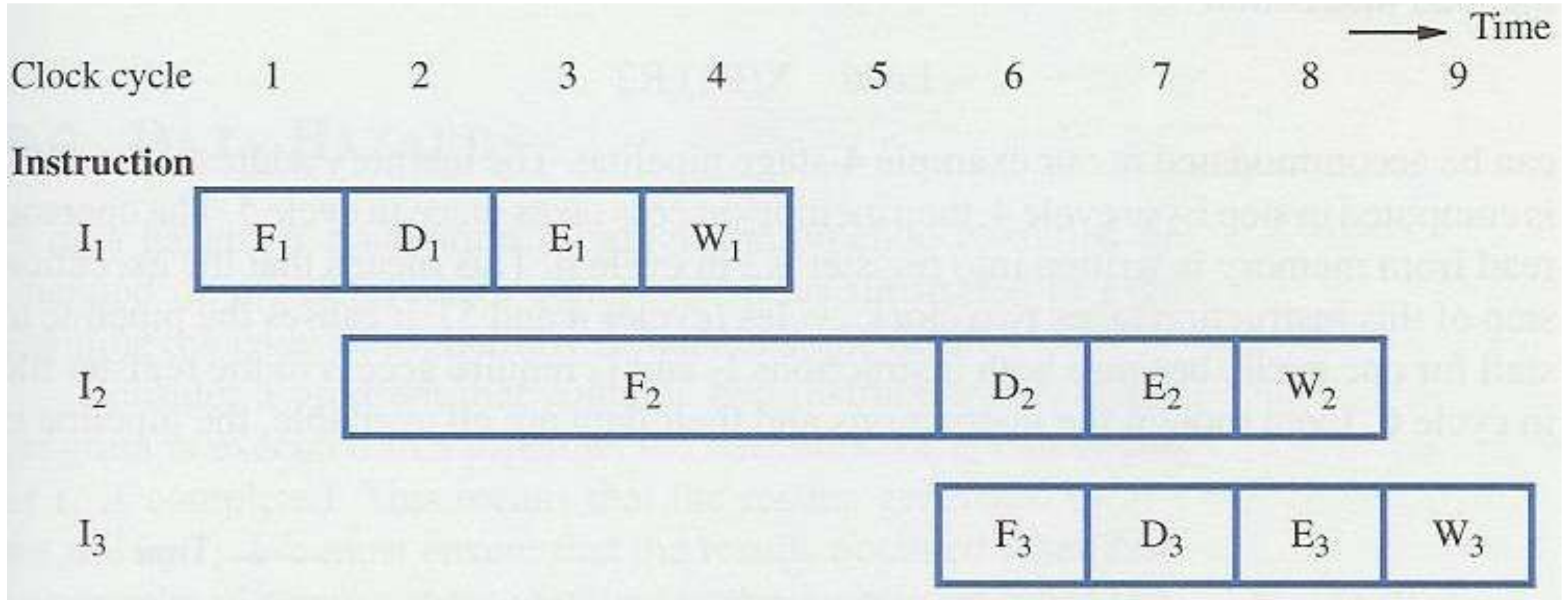
Instruction Hazards

It is a situation in which the pipeline is stalled because of a delay in the availability of an instruction.

Delay in the availability of an instruction

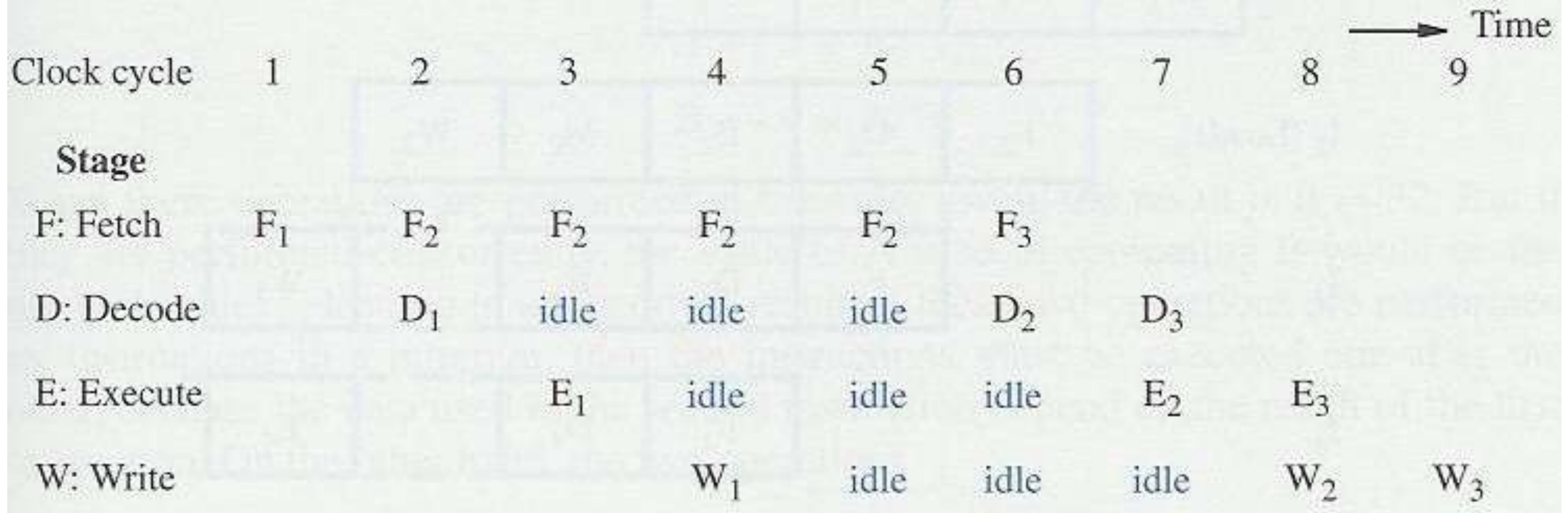
- ✓ Cache miss
- ✓ Branch instructions

Instruction Hazard or Control Hazard---CACHE MISS



Time
Clock cycle 1 2 3 4 5 6 7 8 9
Instruction
I_1 F_1 D_1 E_1 W_1
I_2 F_2 D_2 E_2 W_2
I_3 F_3 D_3 E_3 W_3

(a) Instruction execution steps in successive clock cycles



Time

Clock cycle	1	2	3	4	5	6	7	8	9
Stage									
F: Fetch	F_1	F_2	F_2	F_2	F_2	F_3			
D: Decode		D_1	idle	idle	idle	D_2	D_3		
E: Execute			E_1	idle	idle	idle	E_2	E_3	
W: Write				W_1	idle	idle	idle	W_2	W_3

(b) Function performed by each processor stage in successive clock cycles

Figure 8.4 Pipeline stall caused by a cache miss in F2.

Unconditional branch instructions

Branch Penalty: The time lost as a result of a branch instruction

Branch Penalty

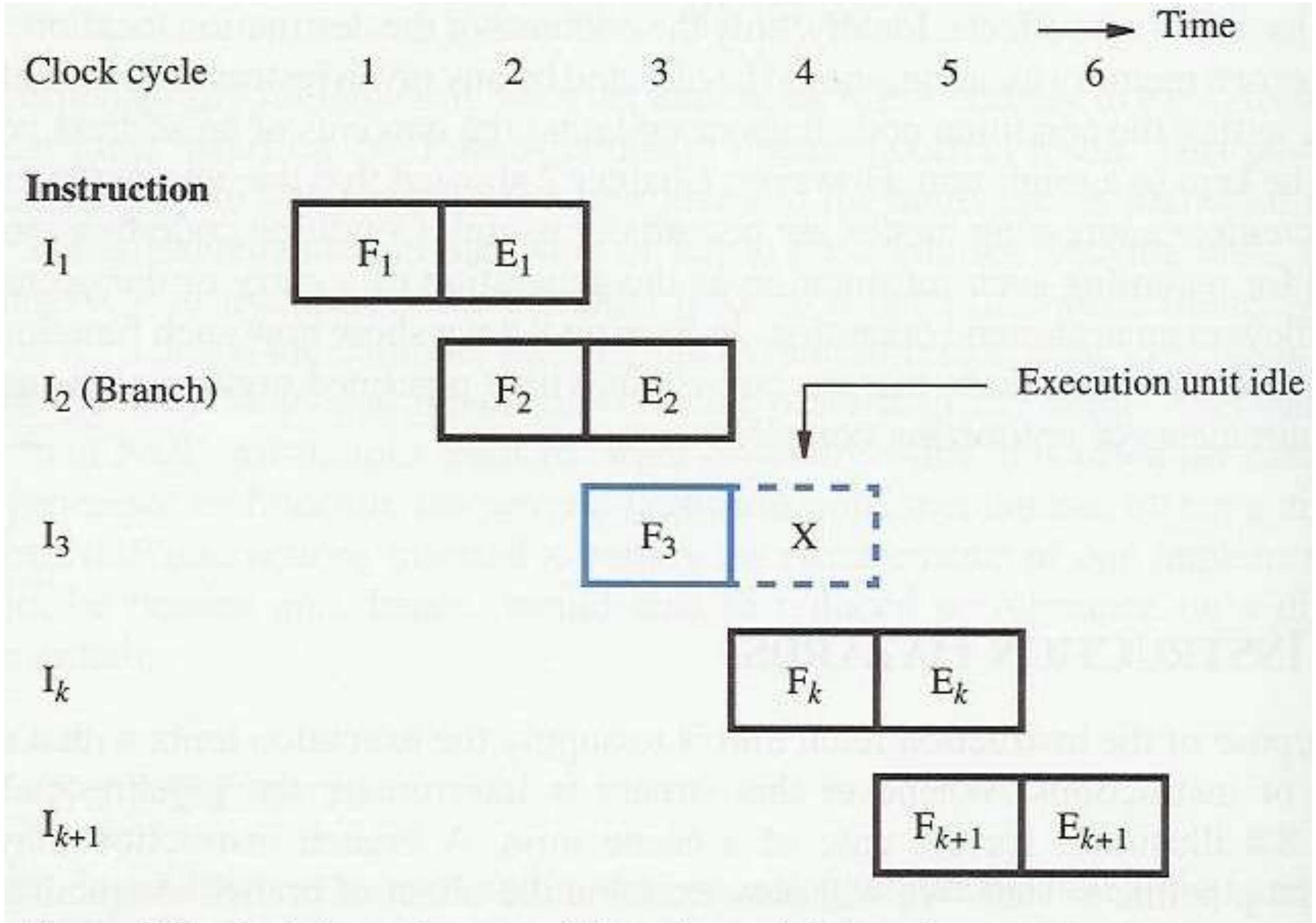
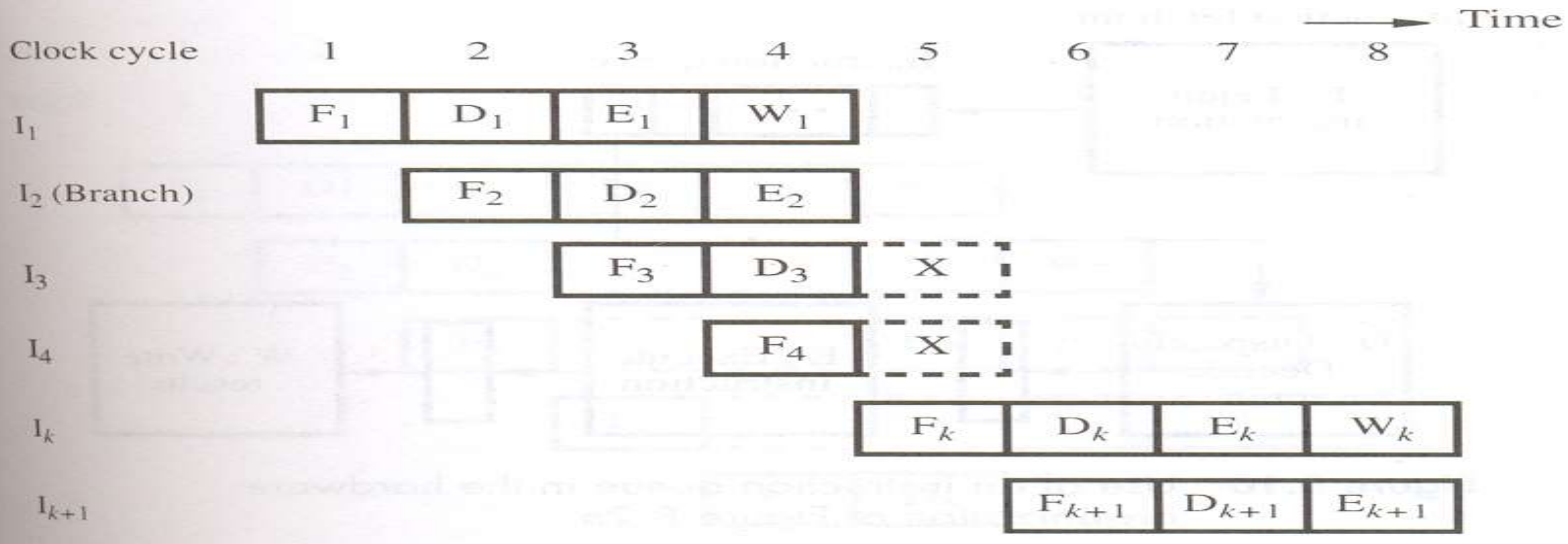
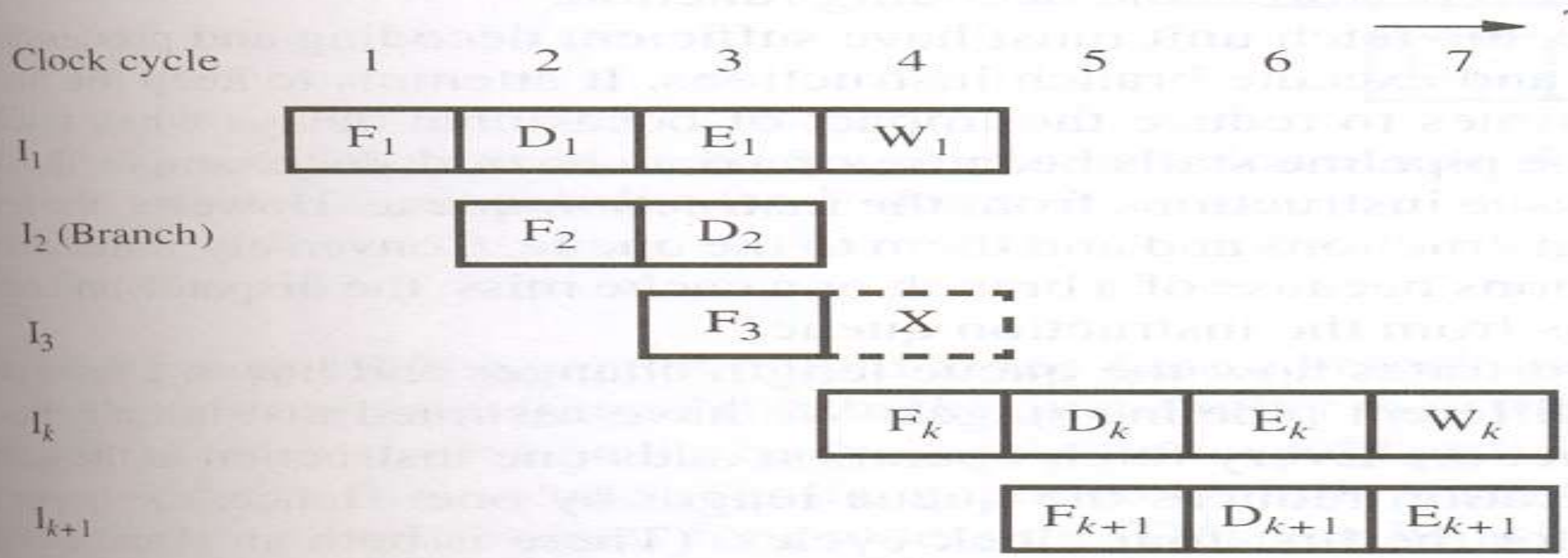


Figure 8.8 An idle cycle caused by a branch instruction.

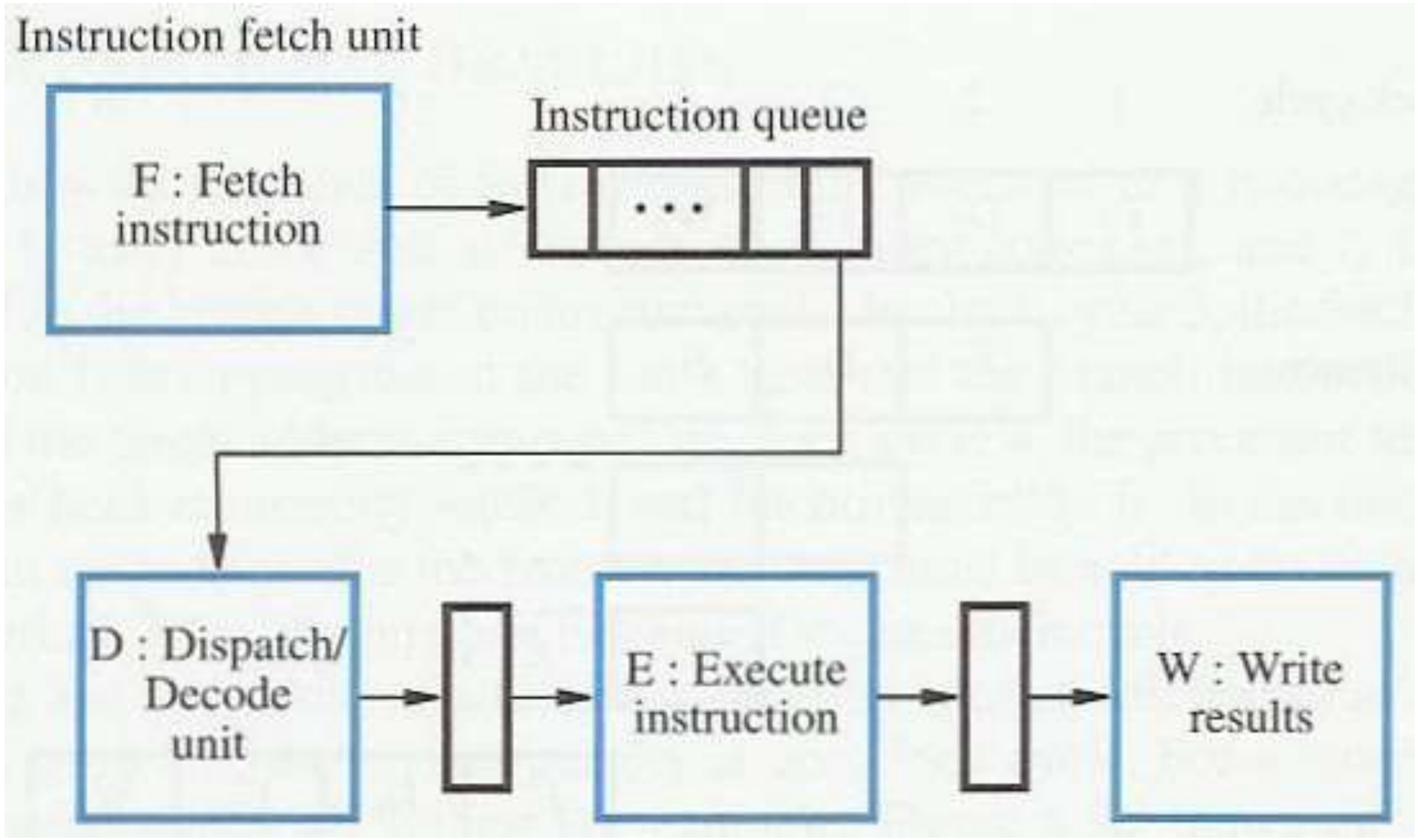

Time
Clock cycle 1 2 3 4 5 6 7 8
I_1: F_1 D_1 E_1 W_1
I_2 (Branch): F_2 D_2 E_2
I_3: F_3 D_3 X
I_4: F_4 X
I_k: F_k D_k E_k W_k
I_{k+1}: F_{k+1} D_{k+1} E_{k+1}

(a) Branch address computed in Execute stage


Time
Clock cycle 1 2 3 4 5 6 7
I_1: F_1 D_1 E_1 W_1
I_2 (Branch): F_2 D_2
I_3: F_3 X
I_k: F_k D_k E_k W_k
I_{k+1}: F_{k+1} D_{k+1} E_{k+1}

(b) Branch address computed in Decode stage

Instruction Queue and Prefetch



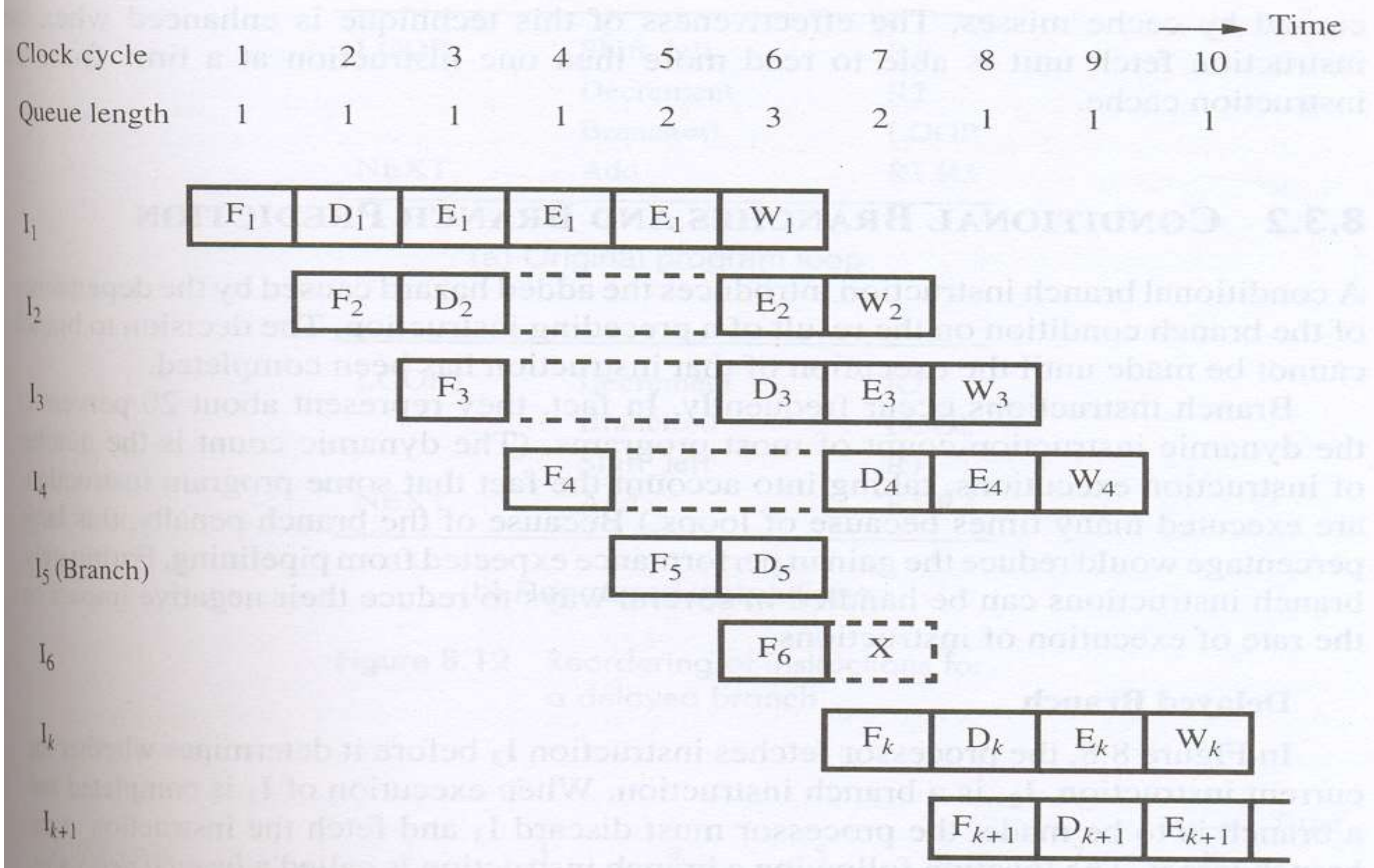


Figure 8.11 Branch timing in the presence of an instruction queue. Branch target address is computed in the D stage.

Conditional branch instructions

Techniques for reducing the branch penalty associated with conditional branch instruction:

1. Delayed branching
2. Branch prediction

```
LOOP    Shift_left    R1
        Decrement     R2
        Branch=0      LOOP
NEXT    Add           R1,R3
```

(a) Original program loop

```
LOOP    Decrement     R2
        Branch=0      LOOP
        Shift_left    R1
NEXT    Add           R1,R3
```

(b) Reordered instructions

Figure 8.12 Reordering of instructions for a delayed branch.

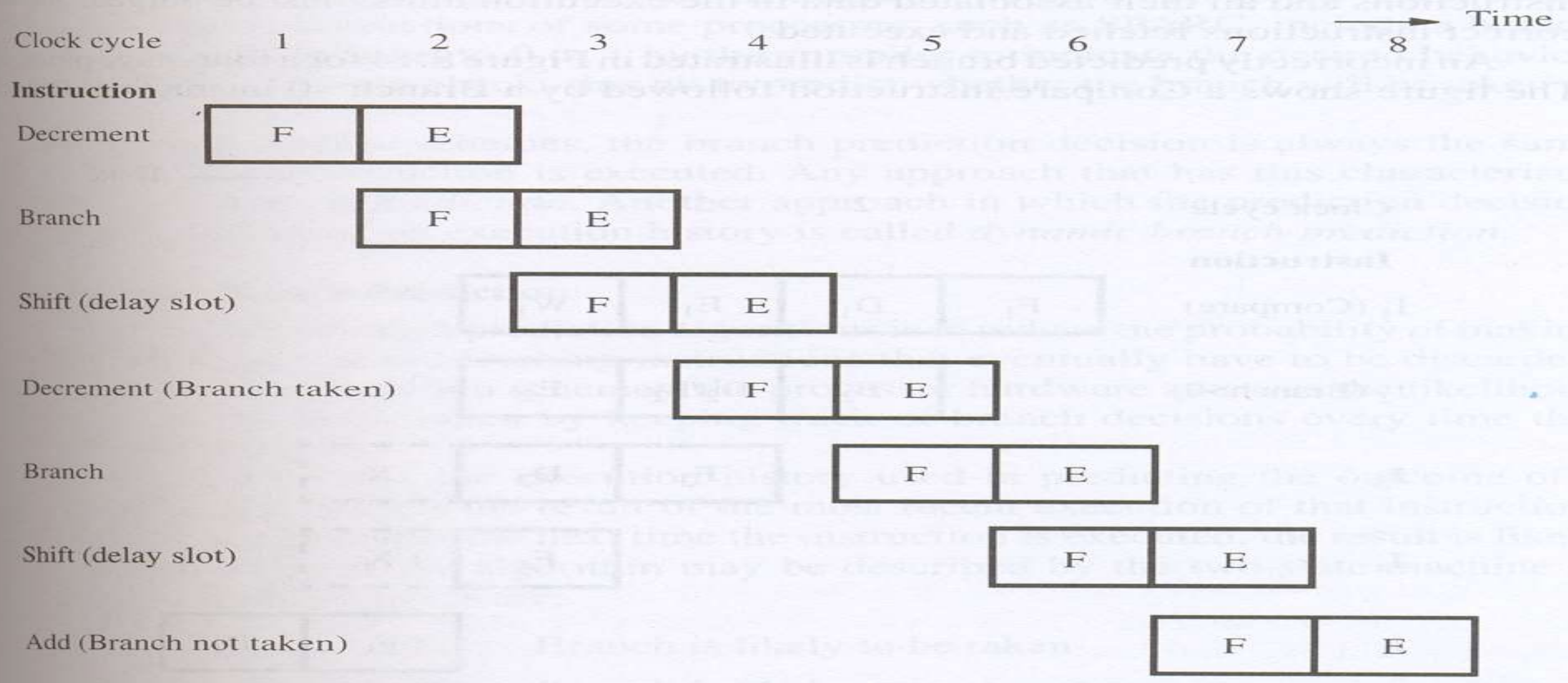


Figure 8.13 Execution timing showing the delay slot being filled during the last two passes through the loop in Figure 8.12*b*.

Branch Prediction

Attempt to predict whether or not a particular branch will be taken

Static Branch Prediction

The result of the branch instruction may be made in hardware by observing whether the target address of the branch is lower than or higher than the address of the branch instruction.

A more flexible approach is to have the compiler decide whether a given branch instruction should be predicted taken or not taken. That is branch instructions include a branch prediction bit, which is set to 0 or 1 by the compiler to indicate the desired behavior.

Dynamic Branch Prediction

Based on execution history

The figure shows a Compare

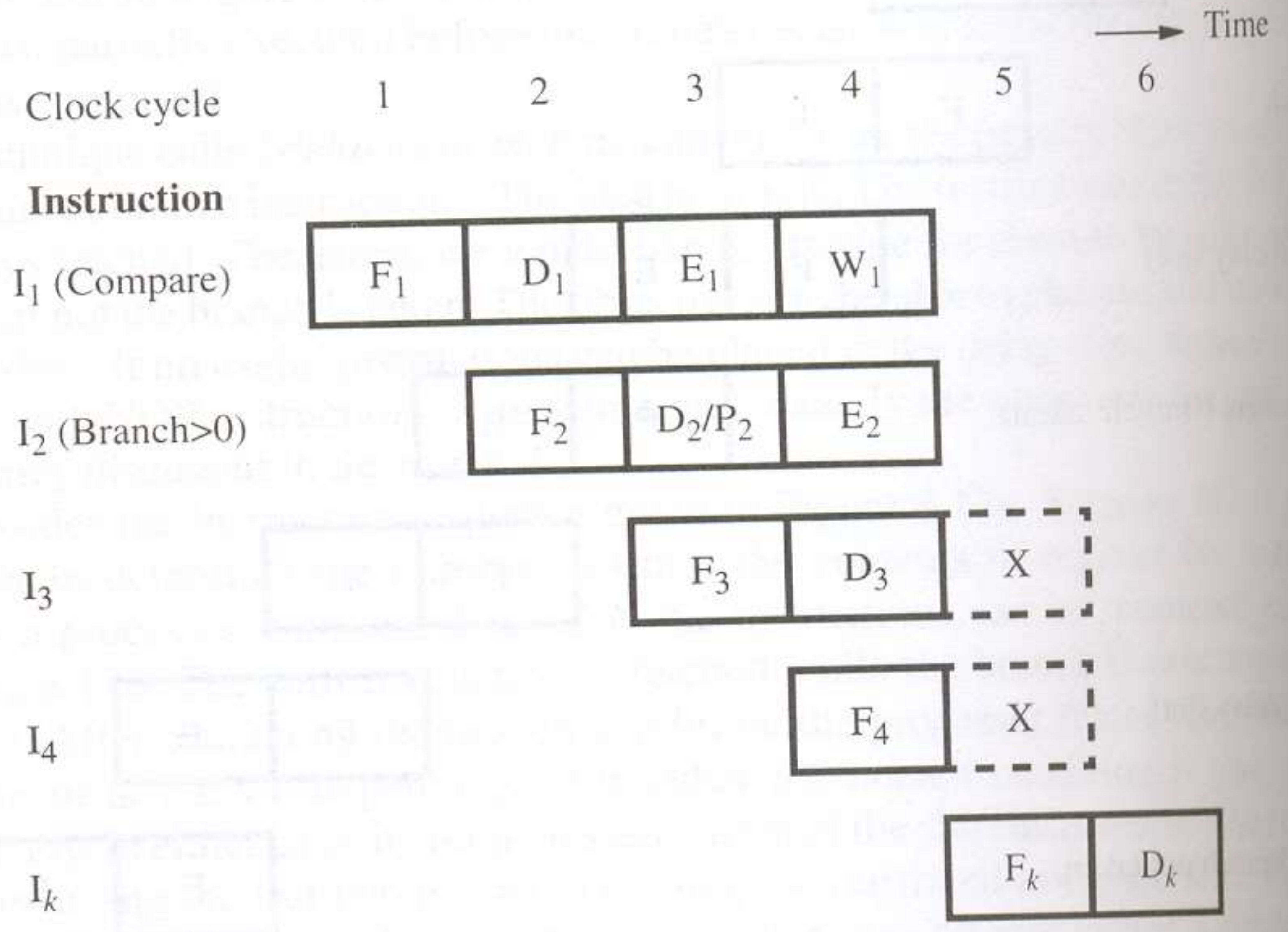


Figure 8.14 Timing when a branch decision has been incorrectly predicted as not taken.

ST: strongly likely to be taken
LT: likely to be taken
LNT: likely not to be taken
SNT: strongly likely not to be taken

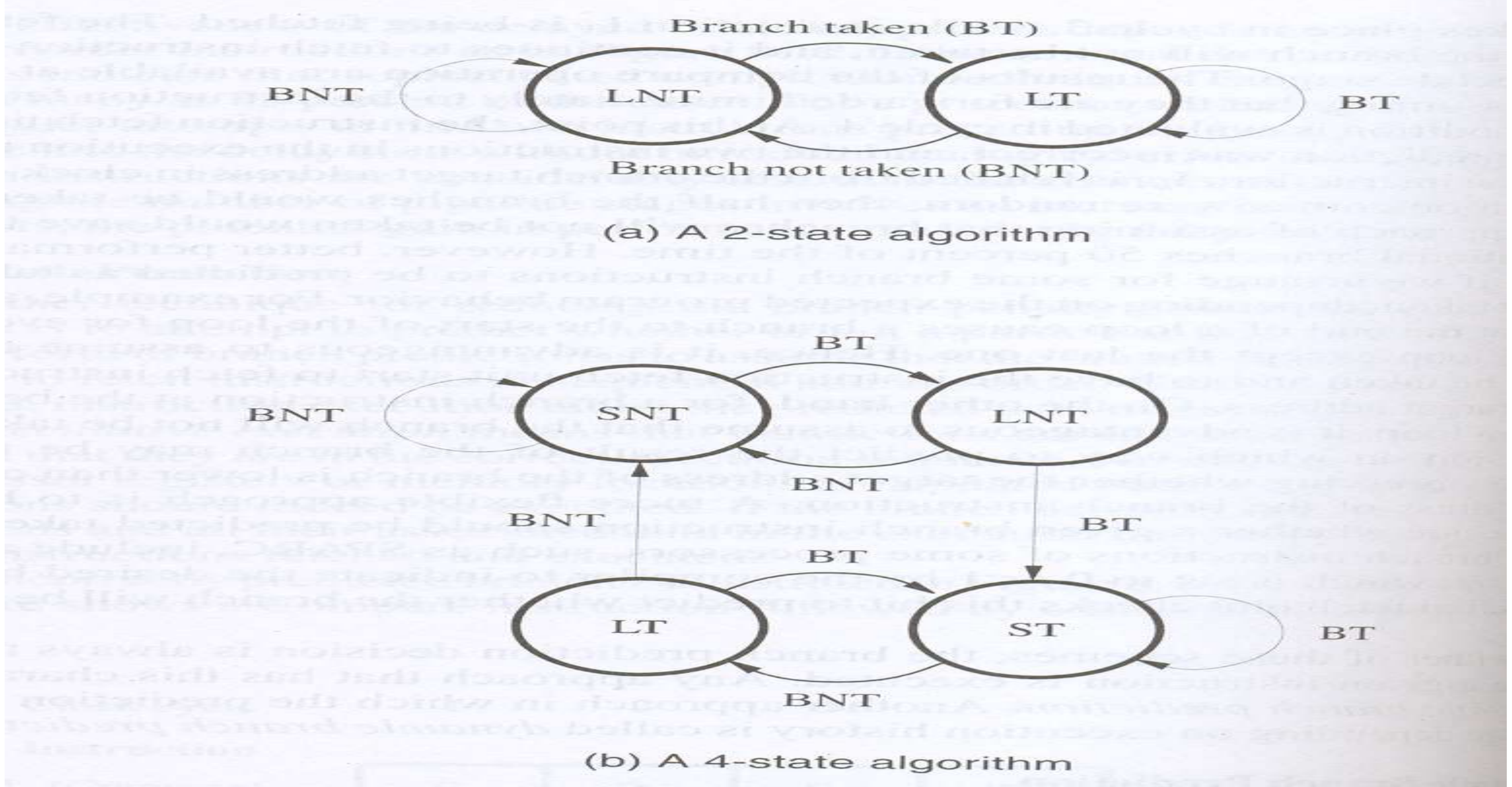


Figure 8.15 State-machine representation of branch-prediction algorithms.

INFLUENCE ON INSTRUCTION SETS

Two key aspects of machine instructions:

- ✓ Addressing modes
- ✓ Condition code flags

Addressing modes

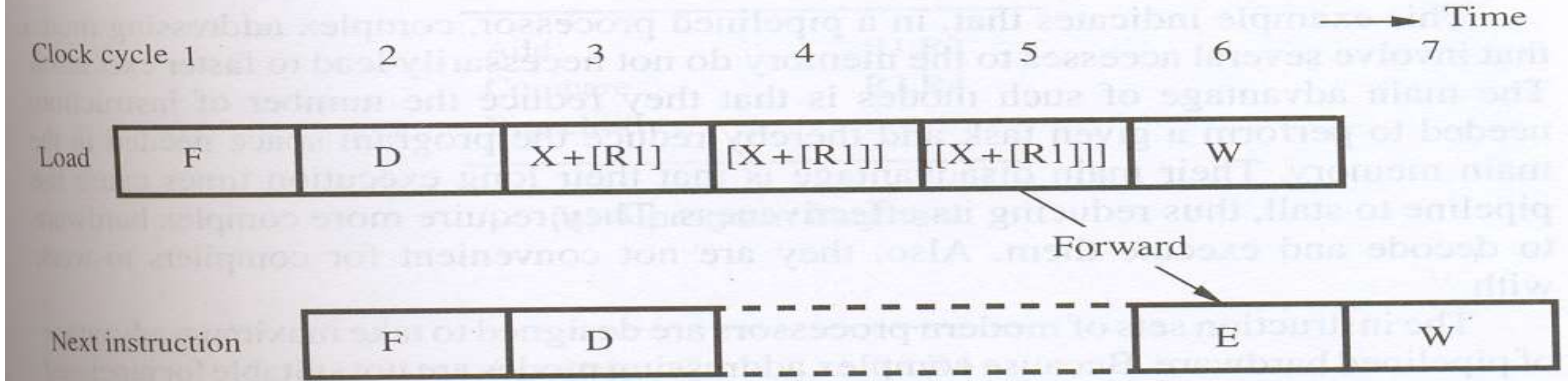
Complex addressing :Load (x(R1)),R2
Simple addressing   :Add #x,R1,R2
                                Load (R2),R2
                                Load (R2),R2

*Complex addressing modes are not suitable for piplined execution.

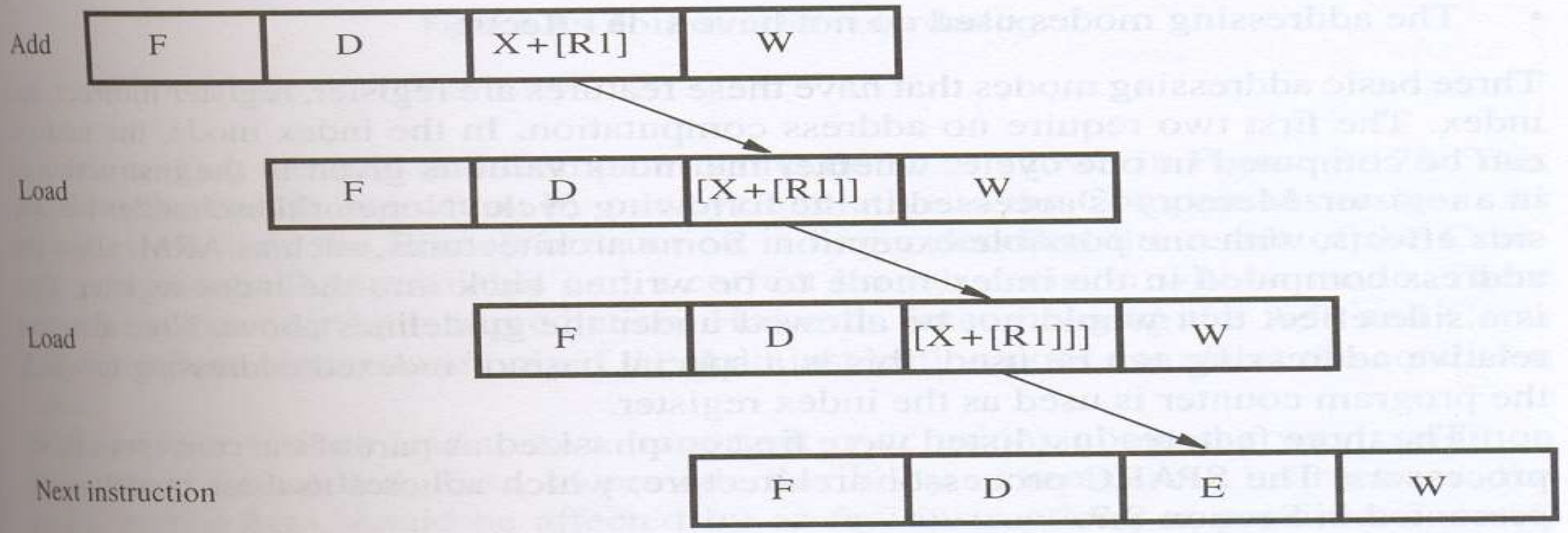
The addressing modes used in modern processors often have the following features to support pipeline hardware:

1. Access to an operand does not require more than one access to the memory.
2. Only load and store instructions access memory operands.
3. The addressing modes used do not have side effects.

*Register, register indirect, index addressing modes are having all these features.



(a) Complex addressing mode



(b) Simple addressing mode

Condition code flags

Compiler for a pipelined processor attempts to reorder instructions to avoid stalling the pipeline when branches or data dependencies between successive instructions occur.

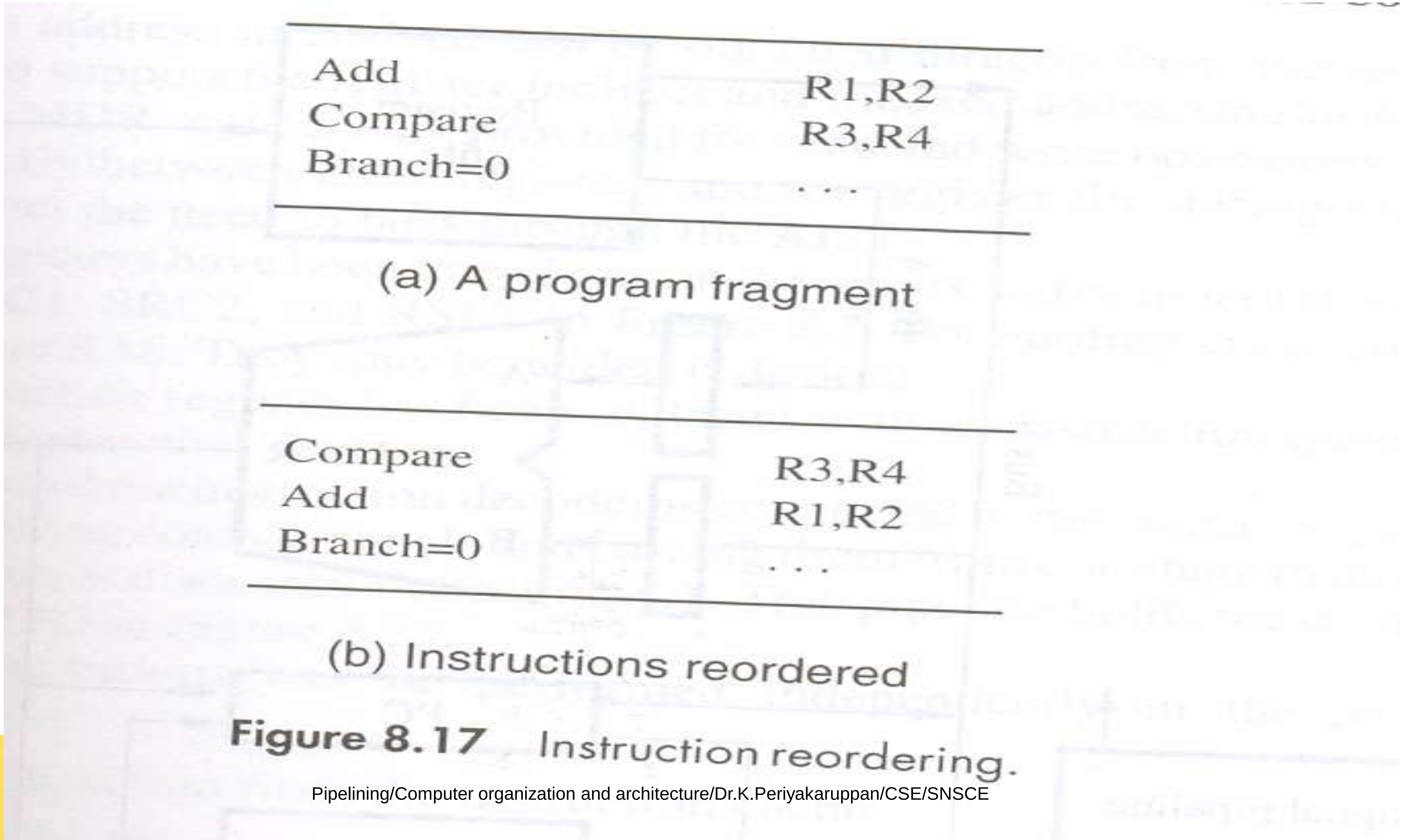
The dependency introduced by the condition code flags reduces the flexibility available for the compiler to reorder instructions.

To provide flexibility in reordering instructions,

1. the condition code flags should be affected by as few instructions as possible.
2.the compiler should be able to specify in which instructions of a program the condition codes are affected and in which they are not.

An instruction set designed with pipelining in mind usually provides the desired flexibility.

Condition code flags



Add	R1,R2
Compare	R3,R4
Branch=0	. . .

(a) A program fragment

Compare	R3,R4
Add	R1,R2
Branch=0	. . .

(b) Instructions reordered

Figure 8.17 Instruction reordering.

Datapath and control considerations

Several important changes as compared to the previous diagram (three bus organization):

1.Separate instruction and data caches.

2.PC is directly connected to IMAR(independent ALU operation)

3.The data address in DMAR can be obtained directly from the register file or from the ALU to support the register indirect and indexed addressing modes.

4. Separate MDR registers are provided for read and write operations. Data can be transferred directly between these registers and the register file during load and store operations without the need to pass through the ALU.

5. Buffer registers have been introduced at the inputs and output of the ALU. These registers SRC1, SRC2, and RSLT in Figure 8.7.

6. The instruction register has been replaced with an instruction queue, which is loaded from the instruction cache.

7. The output of the instruction decoder is connected to the control signal pipeline.

The following operations can be performed independently in the processor of Figure 8.18:

- ✓ Reading an instruction from the instruction cache
- ✓ Incrementing the PC
- ✓ Decoding an instruction
- ✓ Reading from or writing into the data cache
- ✓ Reading the contents of up to two registers from the register file
- ✓ Writing into one register in the register file
- ✓ Performing an ALU operation

Because these operations do not use any shared resources, they can be performed simultaneously in any combination.

The structure provides the flexibility required to implement the four-stage pipeline . For example, let 11,12,13, and 14 be a sequence of four instructions.

The following actions all happen during clock cycle 4:
Write the result of instruction I1 into the register file
Read the operands of instruction I2 from the register file .
Decode instruction I3
Fetch instruction I4 and increment the Pc.

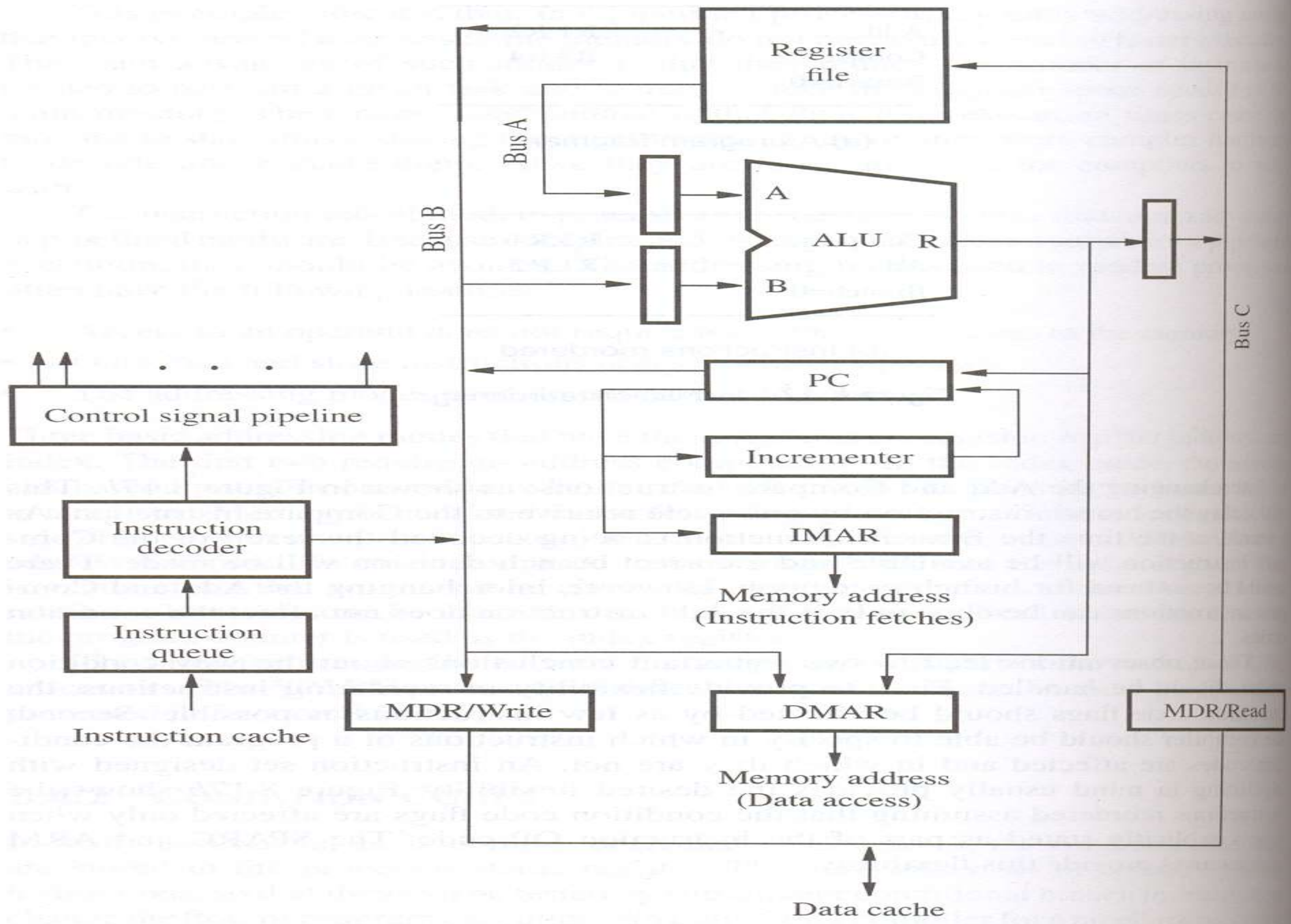


Figure 8.18 Datapath modified for pipelined execution with interstage buffers at the input and output of the ALU.

Assessment

a).What is Pipelining?

____

____

____

b) Mention the purpose of

1.Instruction fetch unit_____

2. Execution unit ____

3.Decode unit _____

4.Write unit ______

5. Data path ______

Reference

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", McGraw-Hill, 6th Edition 2012.
2. David A. Patterson and John L. Hennessey, "Computer organization and design", MorganKauffman /Elsevier, 5th edition, 2014.
3. William Stallings, "Computer Organization and Architecture designing for Performance", Pearson Education 8th Edition, 2010
4. John P.Hayes, "Computer Architecture and Organization", McGraw Hill, 3rd Edition, 2002
5. M. Morris R. Mano "Computer System Architecture" 3rd Edition 2007