- **Fetch instruction (FI):** Read the next expected instruction into a buffer.
- **Decode instruction (DI):** Determine the opcode and the operand specifiers.
- **Calculate operands (CO):** Calculate the effective address of each source operand. This may involve displacement, register indirect, indirect, or other forms of address calculation.
- **Fetch operands (FO):** Fetch each operand from memory. Operands in registers need not be fetched.
- **Execute instruction (EI):** Perform the indicated operation and store the result, if any, in the specified destination operand location.
- **Write operand (WO):** Store the result in memory.

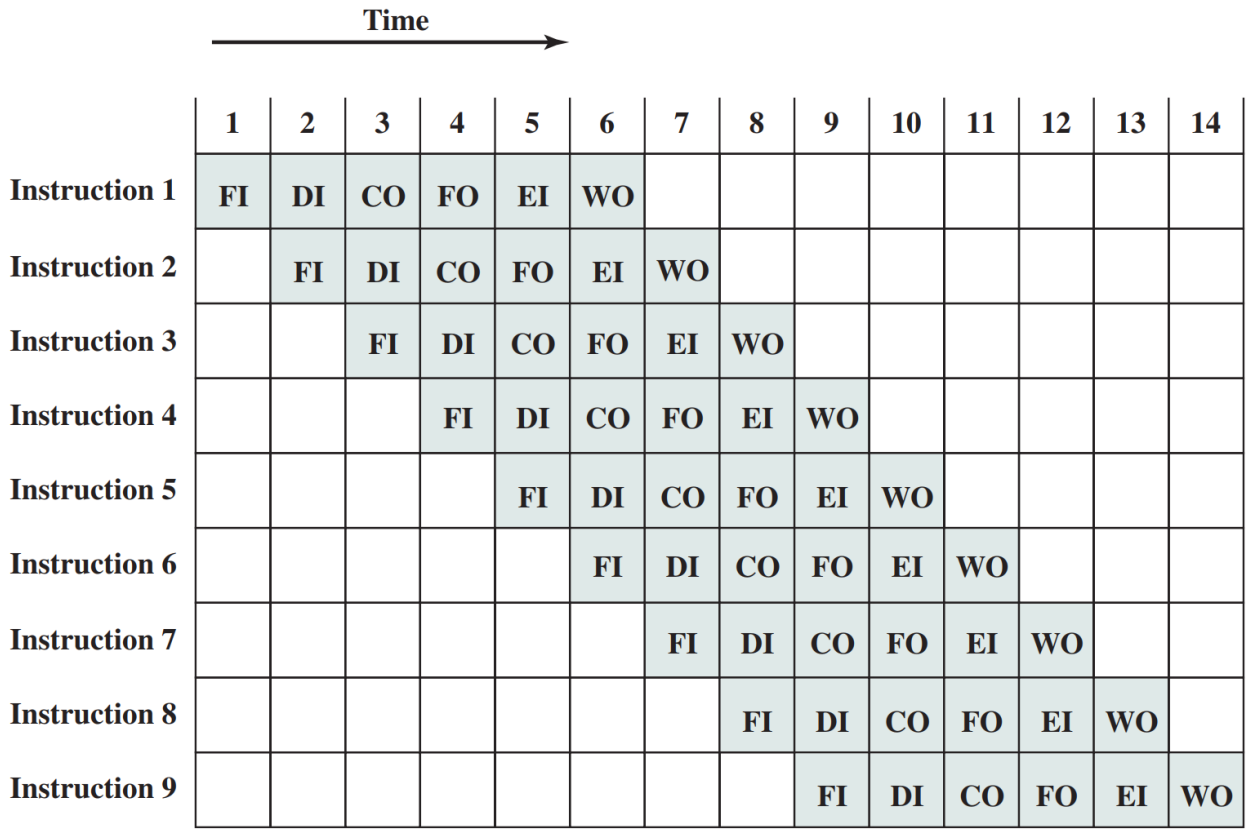


Figure 14.10 Timing Diagram for Instruction Pipeline Operation

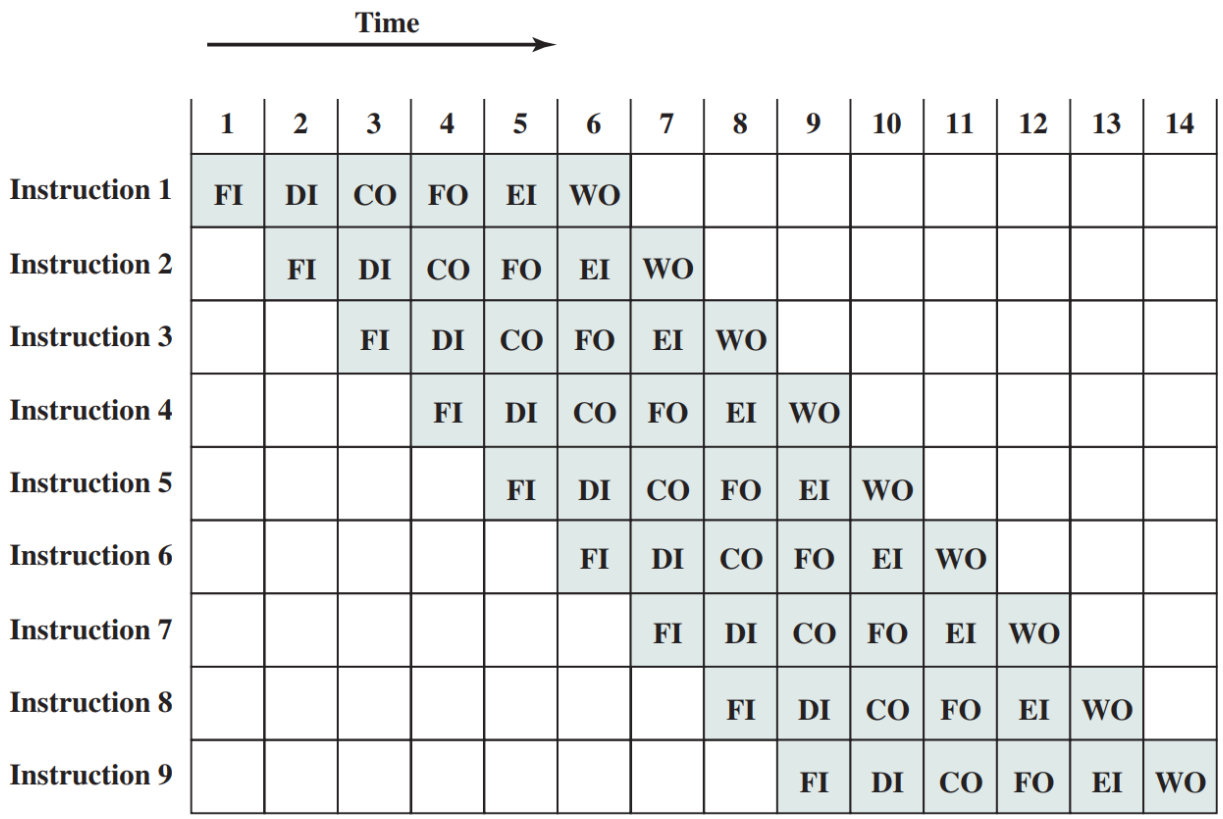
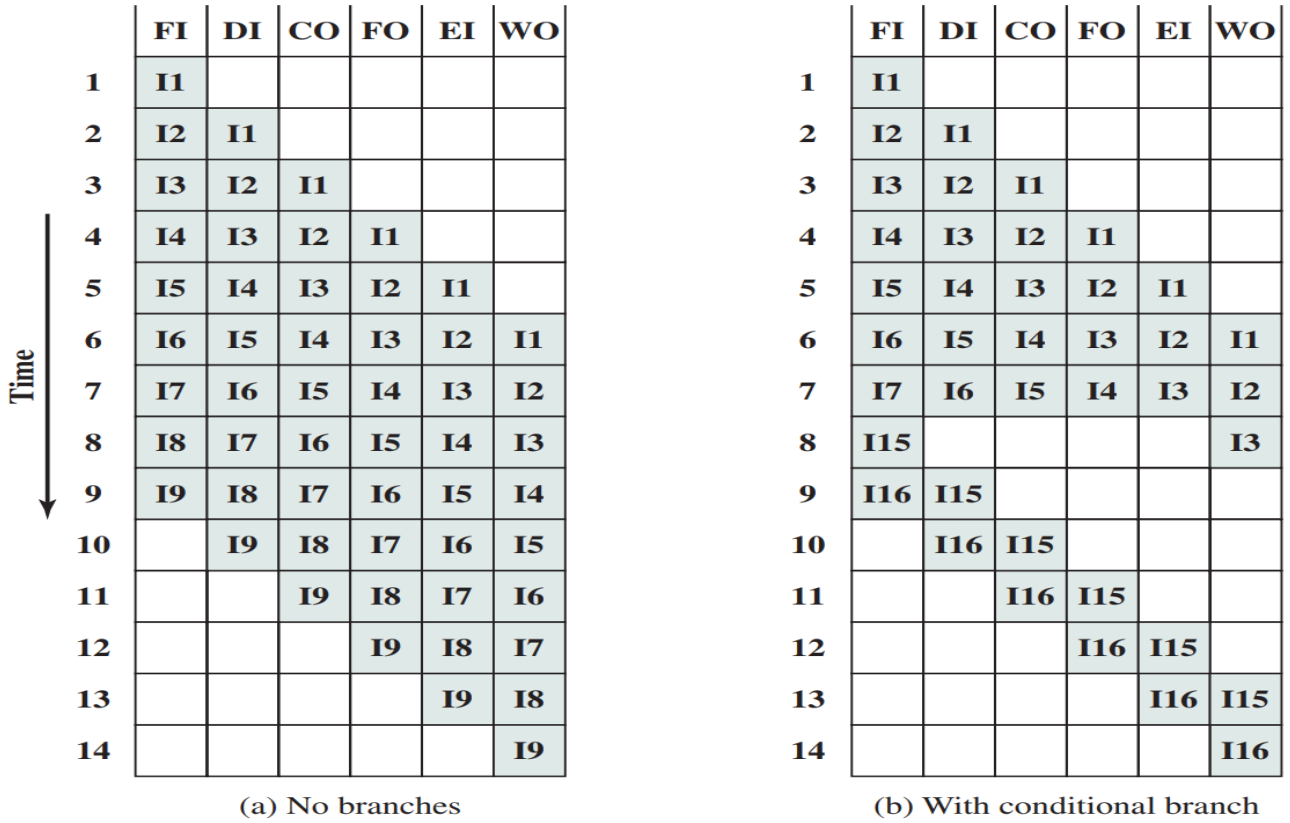


Figure 14.10 Timing Diagram for Instruction Pipeline Operation



(a) No branches

(b) With conditional branch

Figure 14.13 An Alternative Pipeline Depiction

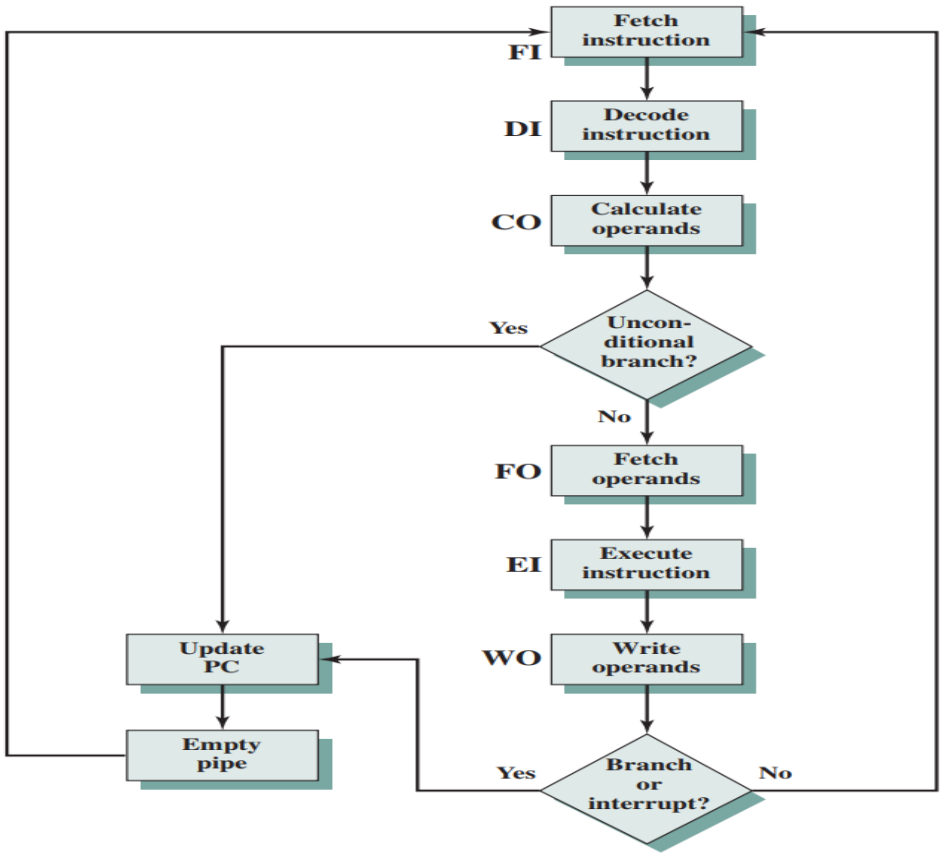
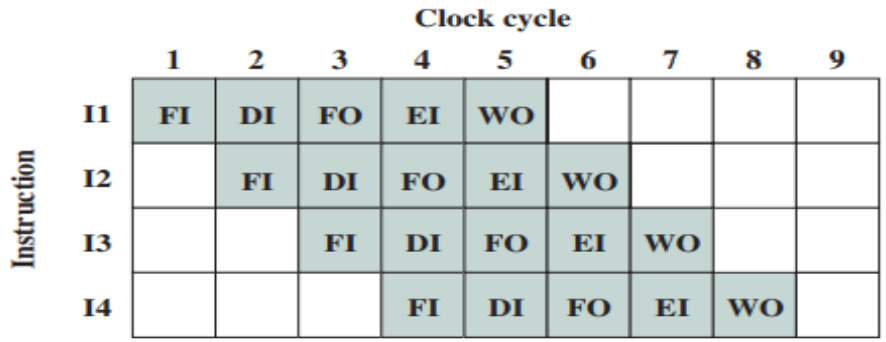
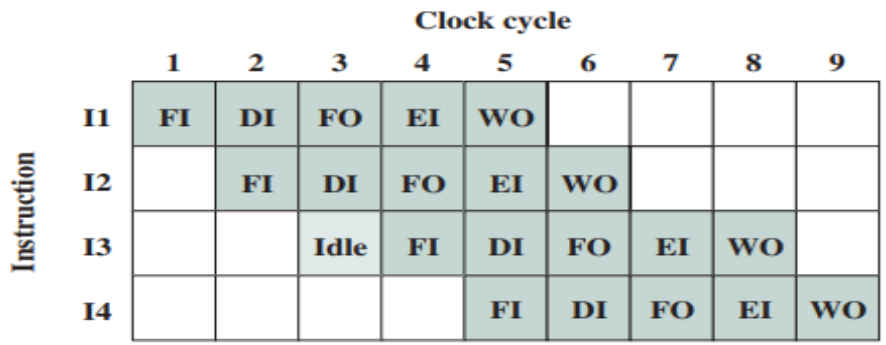


Figure 14.12 Six-Stage CPU Instruction Pipeline



(a) Five-stage pipeline, ideal case



(b) I1 source operand in memory

Figure 14.15 Example of Resource Hazard

As an example, consider the following x86 machine instruction sequence:

```
ADD EAX, EBX /* EAX = EAX + EBX
SUB ECX, EAX /* ECX = ECX - EAX
```

The first instruction adds the contents of the 32-bit registers EAX and EBX and stores the result in EAX. The second instruction subtracts the contents of EAX from ECX and stores the result in ECX. Figure 14.16 shows the pipeline behavior.

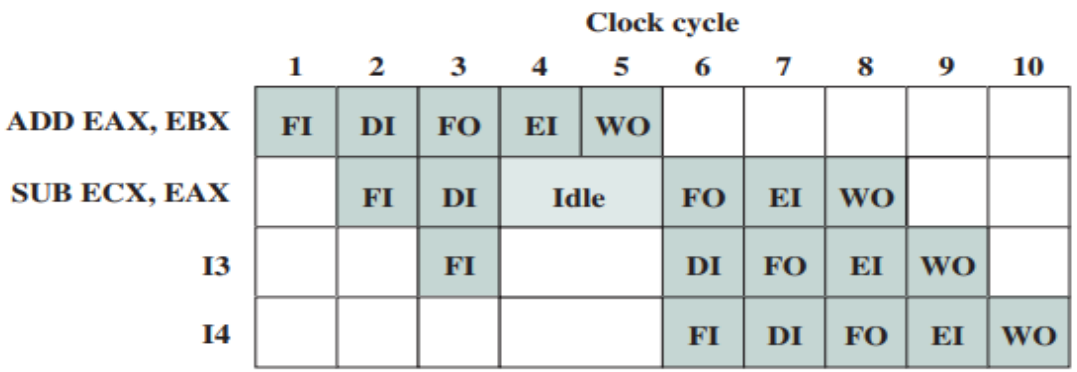


Figure 14.16 Example of Data Hazard

Speed up factor

- S = (unpipelined time)/pipelined time