Interrupt

- Mechanism:
  - By which other modules (I/O, Memory) may interrupt the normal processing of the processor
- Primarily used to improve processor efficiency:
  - External devices are slower than processor
  - Exp: Processor writing to a printer (pauses of thousand instruction cycles)
  - After each write operation processor has to wait for printer to catch-up

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation, request service from the processor, or to signal a variety of error conditions.
Hardware failure	Generated by a failure such as power failure or memory parity error.

Table 3.1

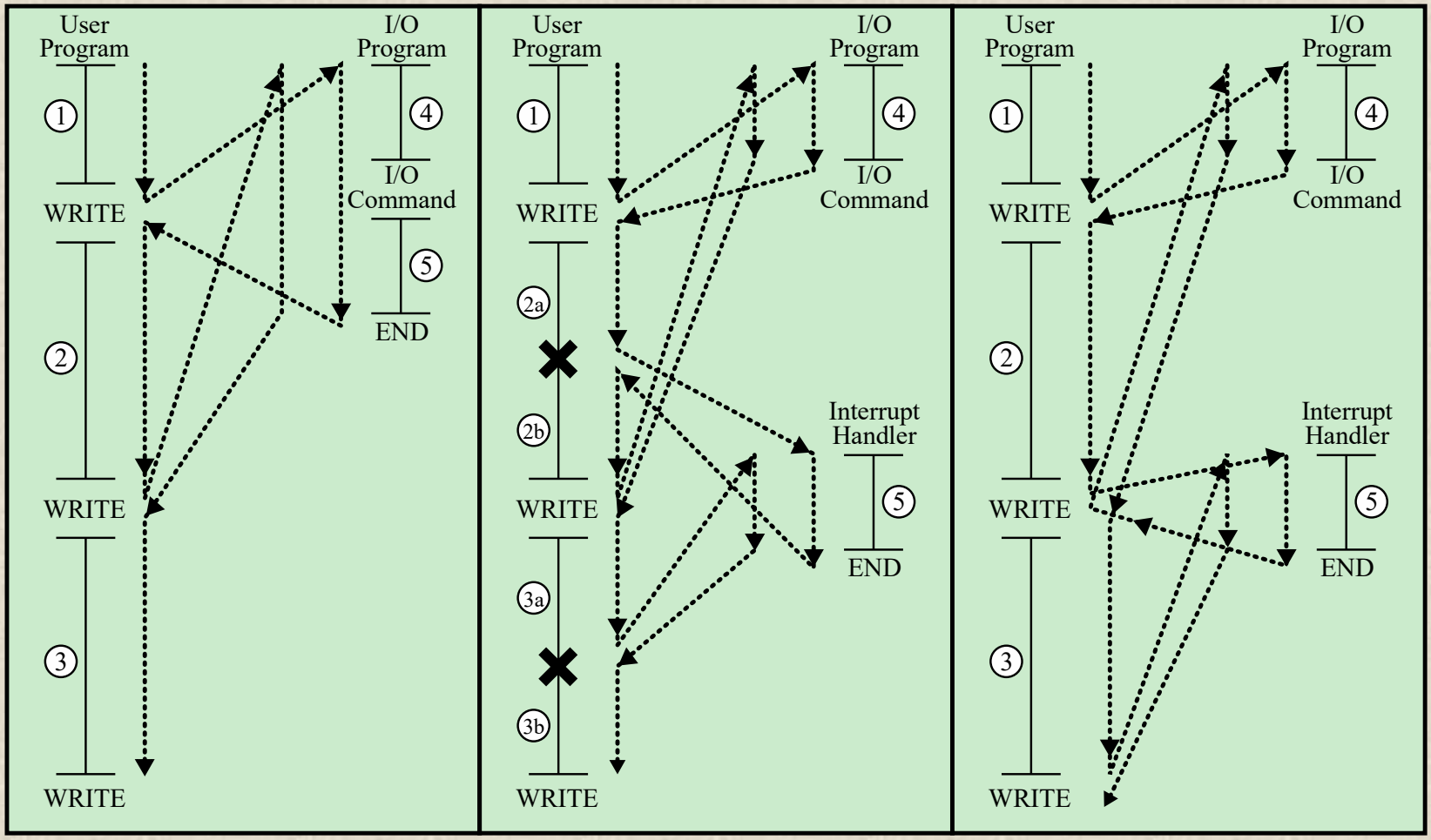
Classes of Interrupts

No Interrupts

- User program contains set of Instructions labelled from 1 to 3
- Write calls are to I/O Program
  - Processor waits until these calls get executed
- I/O Program consists of:
  - Preparation for actual I/O Labelled 4
  - Actual I/O Command to perform action
  - Sequence of Instructions labelled 5 to indicate the status through flags:
    - Complete/Failur

With Interrupts

- Processor can perform executions even when I/O is in progress



(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

✖ = interrupt occurs during course of execution of user program

Figure 3.7 Program Flow of Control Without and With Interrupts

Interrupts and The Instruction Cycle

- Processor is engaged while I/O is in progress
- When device is ready to accept more data from processor:
  - I/O Module sends an interrupt signal
  - Current state of program is saved
  - Control is transferred to Interrupt handler
  - Resume after servicing the Interrupt

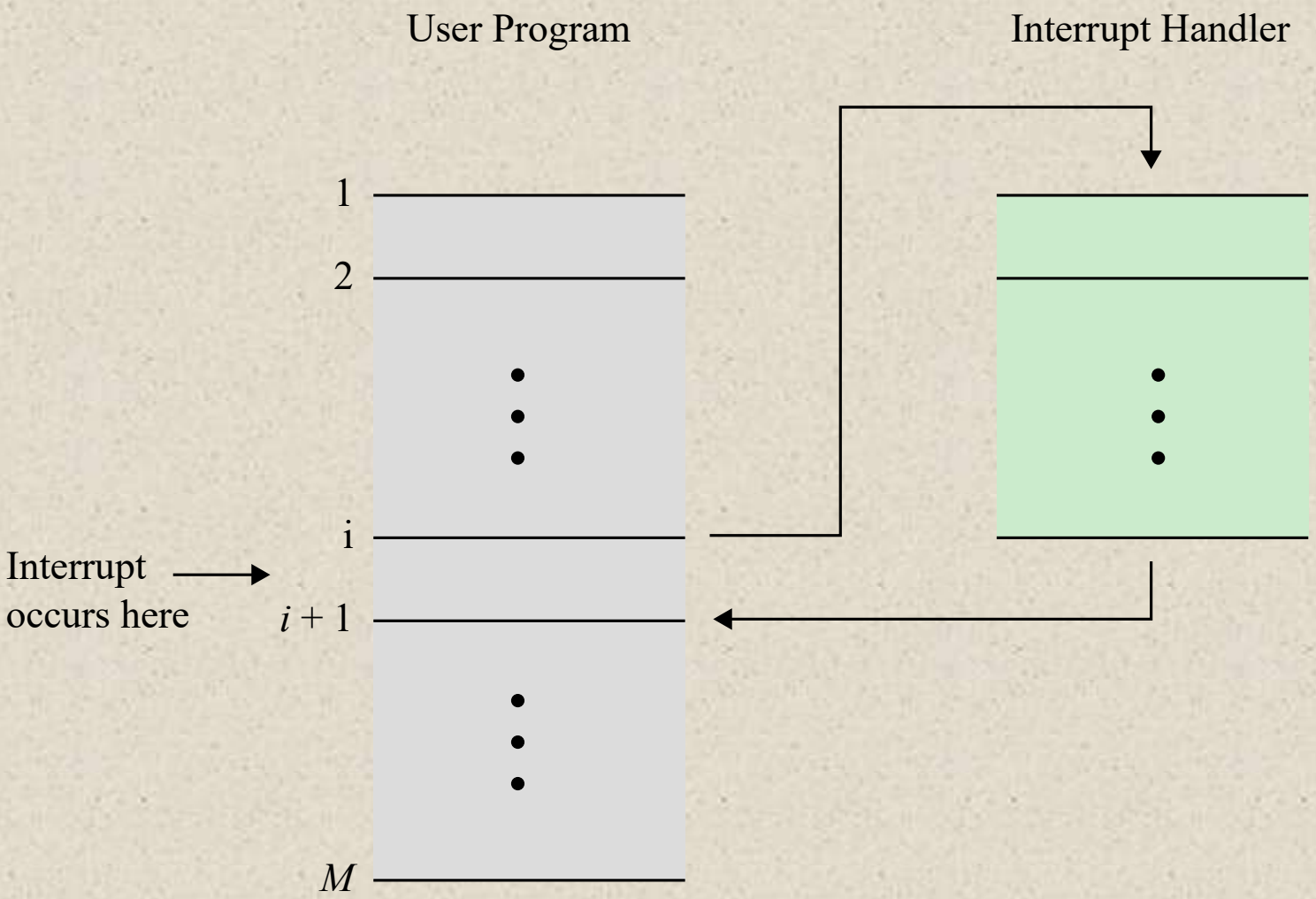


Figure 3.8 Transfer of Control via Interrupts

Addition of Interrupt cycle to Instruction cycle

- Interrupt cycle is added to check for interrupts
- If there is no interrupt:
  - Proceed to fetch next instruction of current program
- But if Interrupt is pending:
  - Processor saves the context of current program
  - Sets the program counter to starting address of Interrupt handler routine and interrupt is serviced
- After servicing interrupts the current program resumes and PC is set to the next instruction

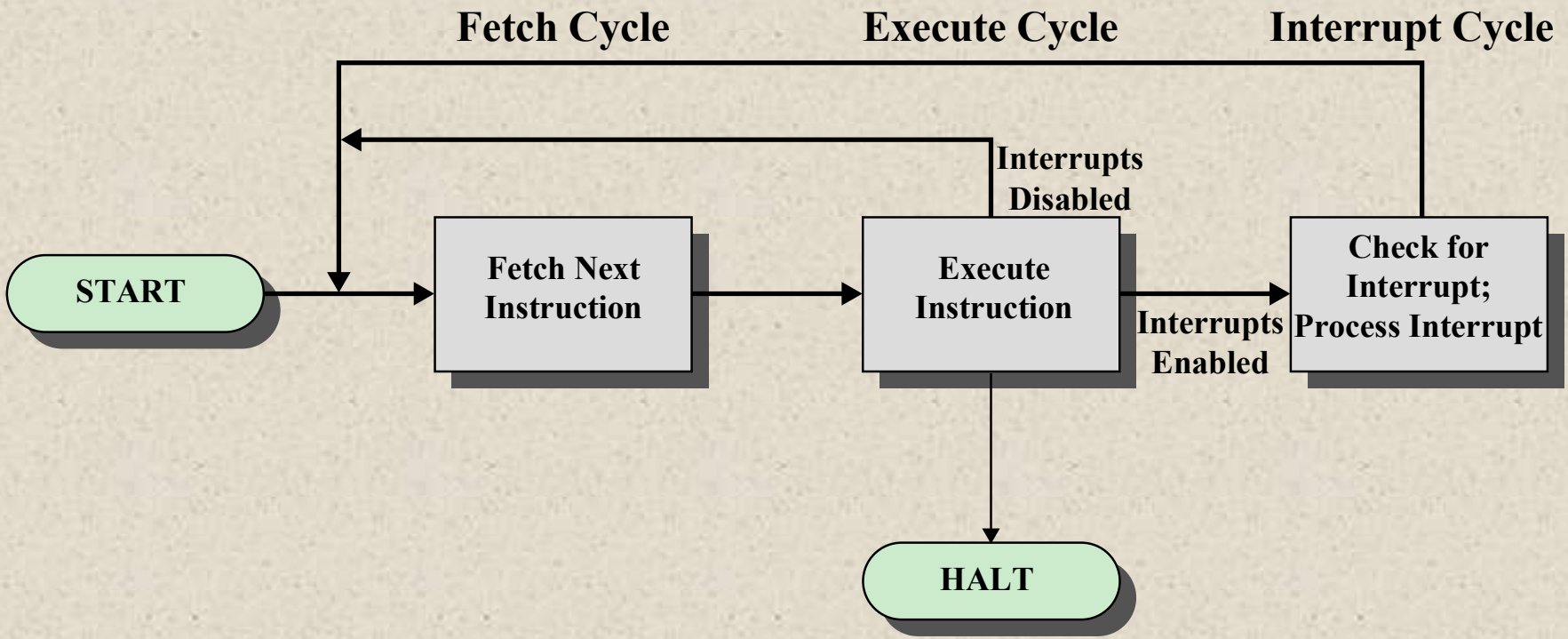


Figure 3.9 Instruction Cycle with Interrupts

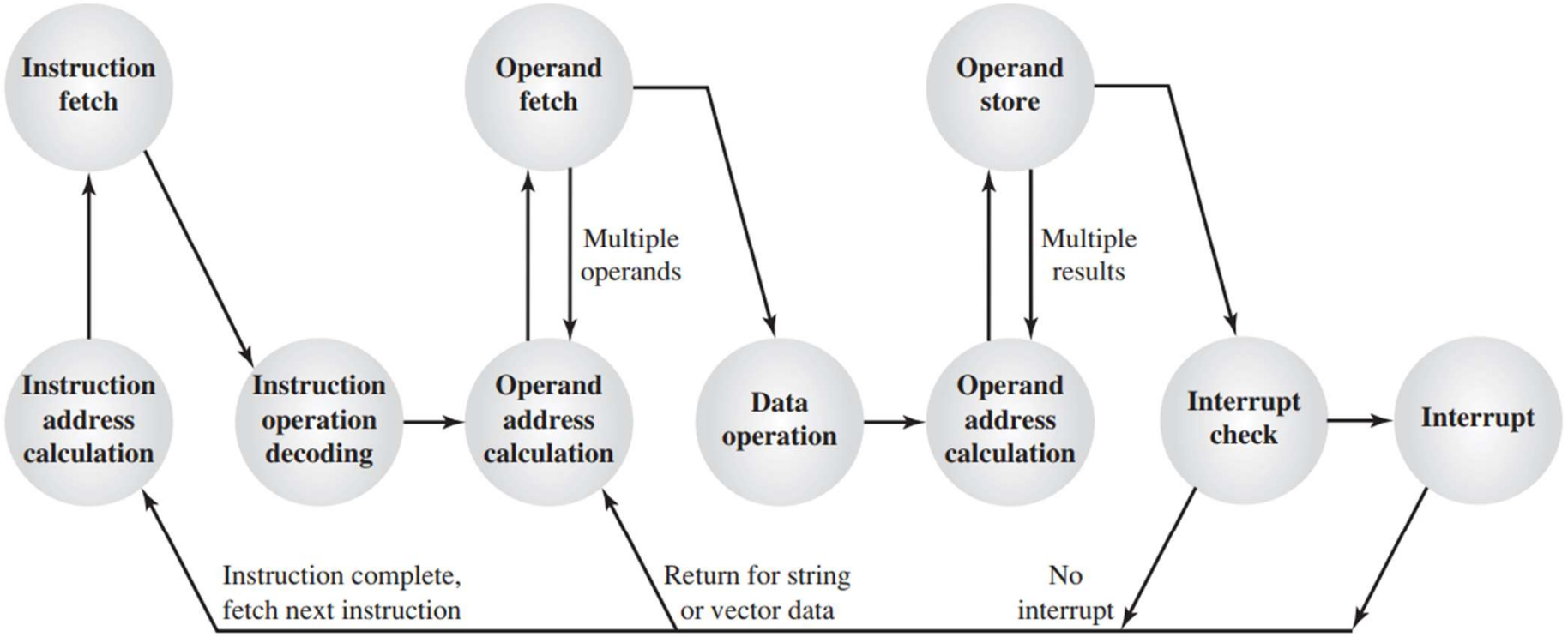


Figure 3.12 Instruction Cycle State Diagram, with Interrupts

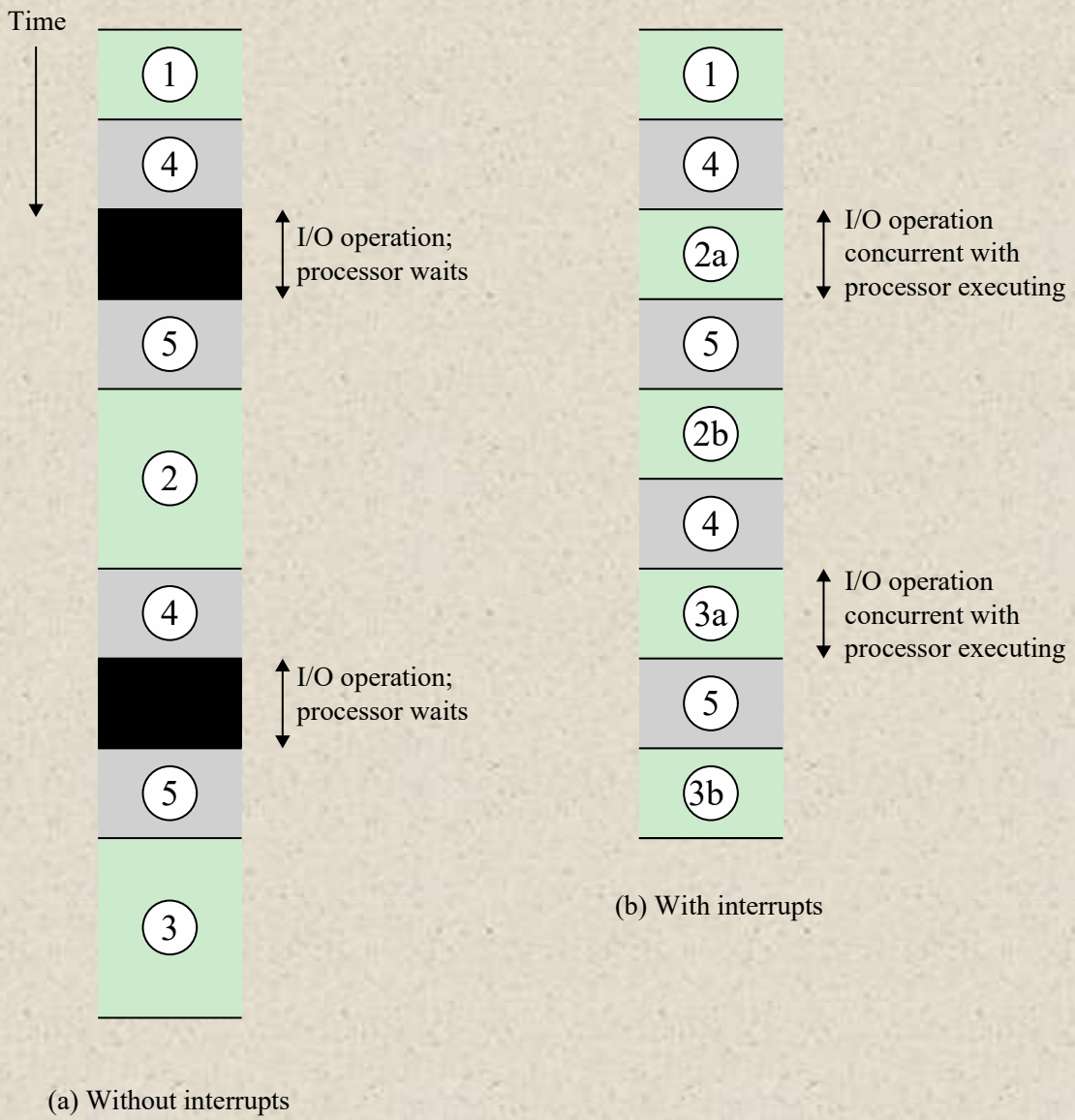


Figure 3.10 Program Timing: Short I/O Wait

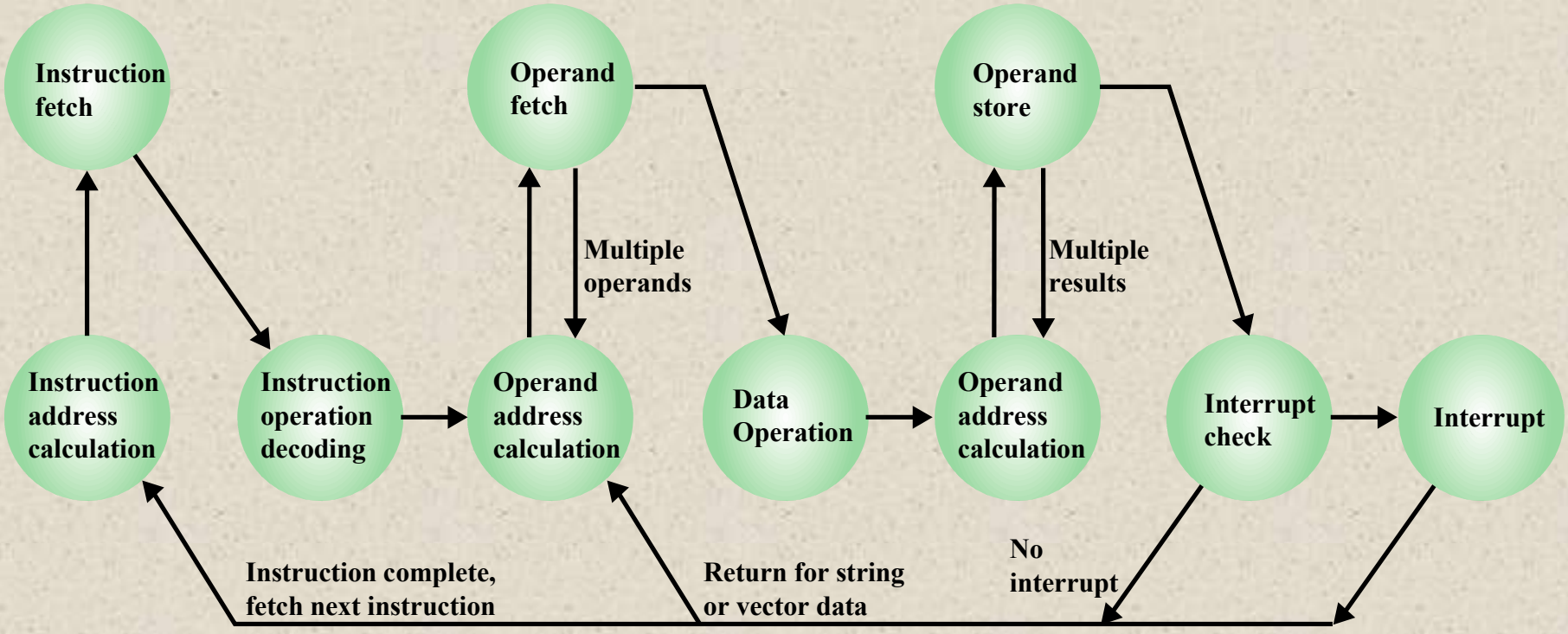


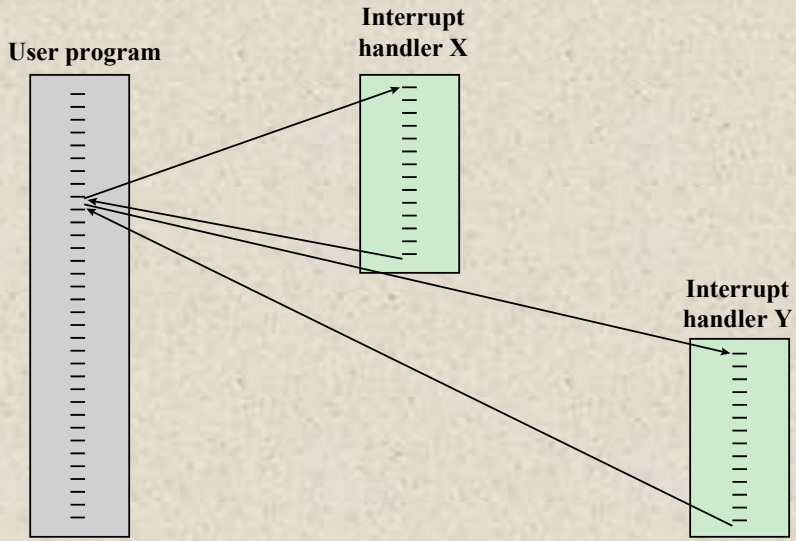
Figure 3.12 Instruction Cycle State Diagram, With Interrupts

Multiple Interrupts

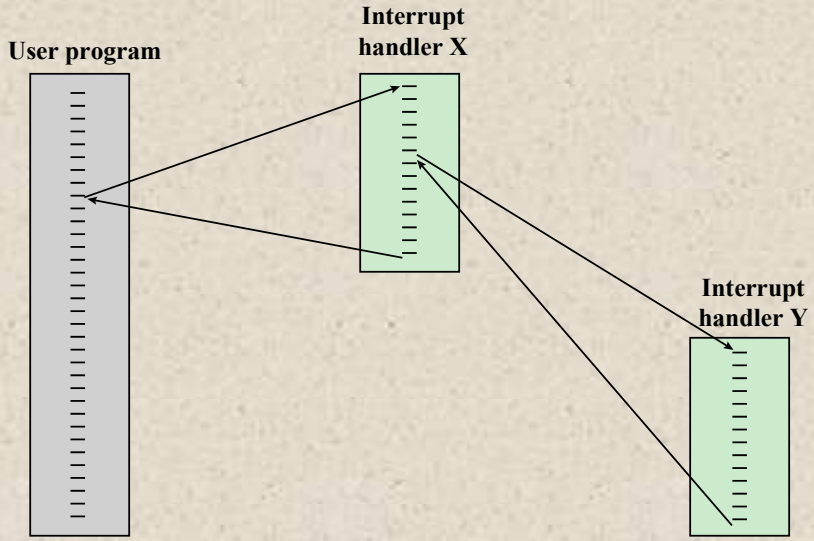
- Possibility of more than one interrupt:
  - Communication line and printer data
- Two approaches to deal with multiple interrupts:
  - **1.Disabled Interrupts:**
    - After occurrence of first interrupt, interrupts are disabled
    - Processor ignores all incoming interrupts
    - Once interrupt is handled all interrupts are enabled
    - Drawback:
      - Not suitable for time critical applications

Multiple Interrupts

- Two approaches to deal with multiple interrupts:
  - **2. Priority Interrupts:**
    - Higher priority interrupts are given preferences
    - Printer=2, communication=4
    - At time t=1, printer interrupt occurred and serviced via Interrupt handler
    - At time t=2 communication interrupt occurred, printer interrupt service routine pushed onto the stack, and communication interrupt is serviced



(a) Sequential interrupt processing



(b) Nested interrupt processing

Figure 3.13 Transfer of Control with Multiple Interrupts

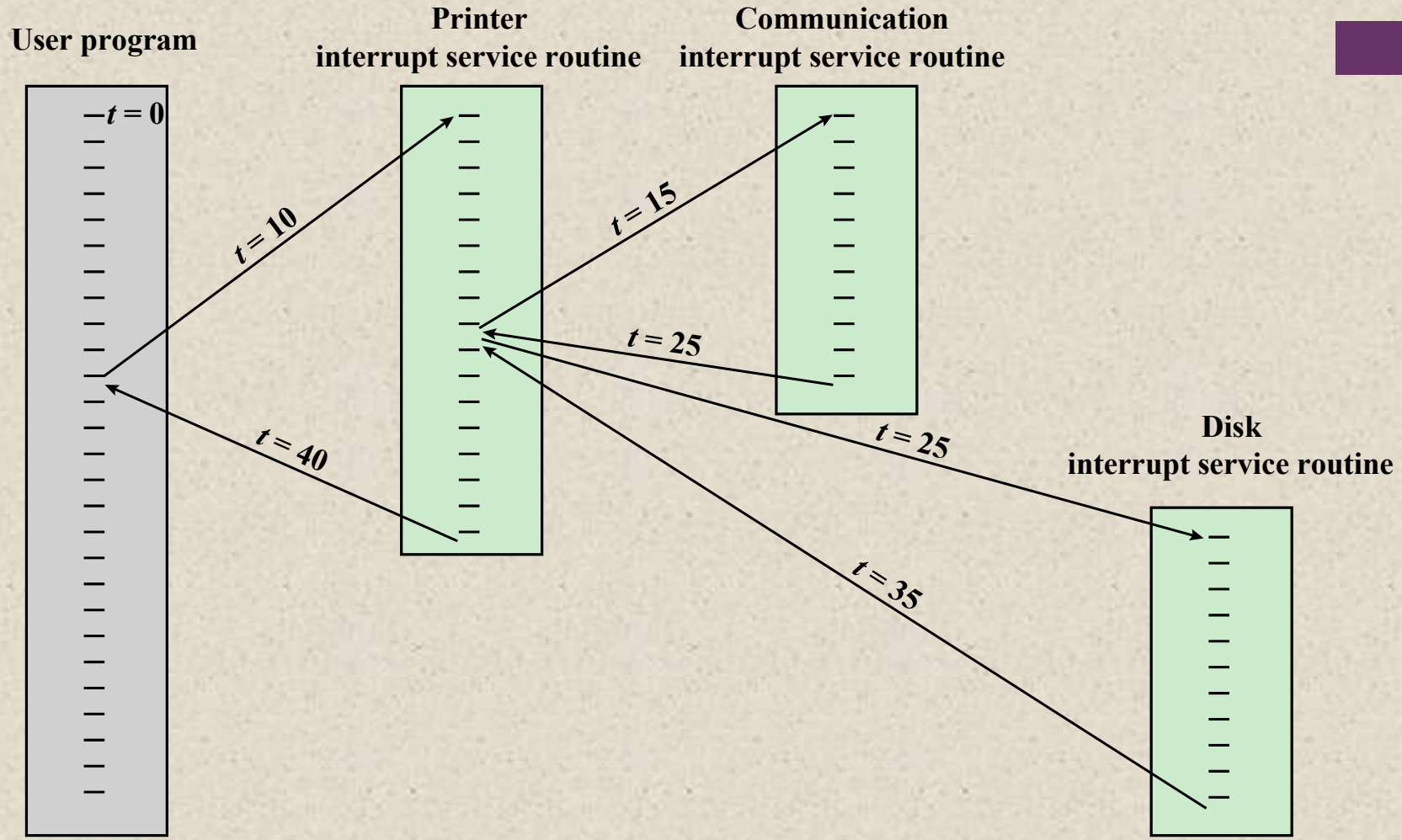


Figure 3.14 Example Time Sequence of Multiple Interrupts