Program Execution

Joannah Nanjekye

July 08, 2024

Announcements

- Lab 1 has been posted on D2L
- Quiz 1 is next Wednesday, 17th July, 10:50 - 11:20
- Labs due every Wednesday, by midnight
- Assignments due every Friday, by midnight

Outline

- Program Execution
- Revisit the Instruction Cycle
- Instruction Format
- Assembly Language Programming

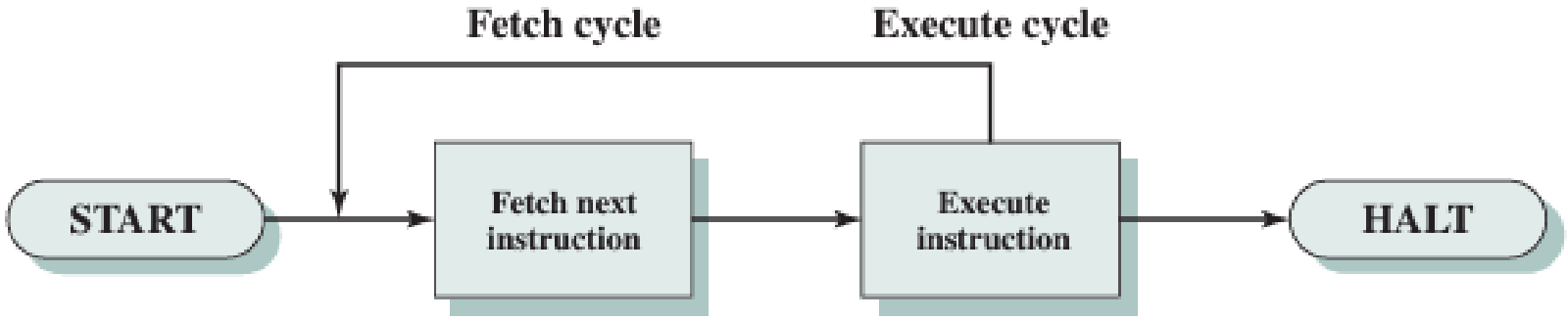
The Instruction Cycle

Fetch cycle: reads instructions from memory one at a time

- Reads the next instruction from the program counter (PC) register
- Loads it in the instruction register (IR)

Execute cycle: interprets the instruction and performs the required action

- Each instruction has a bit that specifies the action
- This action is one of the core functions of the computer like control, processing etc



Revisiting Key Registers

- **r0 - r1:** general-purpose registers, r0, ..., r_n for temporary storage
- **PC:** program counter, contains the address of the next instruction to be execute
- **IR:** instruction register, stores the instruction currently being executed
- **MAR:** memory address register stores the address of the location in main memory that is currently being accessed by a read or write operation
- **MBR:** memory buffer register stores data that has just been read from main memory, or data to be immediately written to main memory

Program Execution: Fetch Phase

1. The PC supplies the address of the next instruction to be executed to the MAR
2. The MAR reads this instruction
3. The PC is incremented by the size of an instruction
4. The instruction is read and loaded in to the MBR
5. Then copied to the IR
6. The ope-code is decoded

Program Execution: Execute Phase

1. A load or store requires sending the memory address to from the IR to the MBR
2. Then a read or write operation is done
3. If the operands are from a register, then they are transferred to the ALU
4. The ALU operates on the operands
5. A result is passed to a destination register

Constants

- Consider an instruction *MUL* $r0, r1, 8$
- The constant operand, 8, is transferred from the operand field in the IR, not from the registers

Flow Control

- An action that modifies the normal instruction sequence
- Conditional behavior allows a processor to select one of two possible courses of actions:
  - Execute the next instruction in sequence
  - Load the PC with a new value and execute a branch to another region of code
- Conditional behaviour depends on the status value in the Condition Code Register (CCR)
- The status in the CCR can be :
  - Zero (Z)
  - Negative in two's complement terms (N)
  - Generated a Carry (C)
  - Generated an arithmetic oVerflow (V)

Summary

FETCH

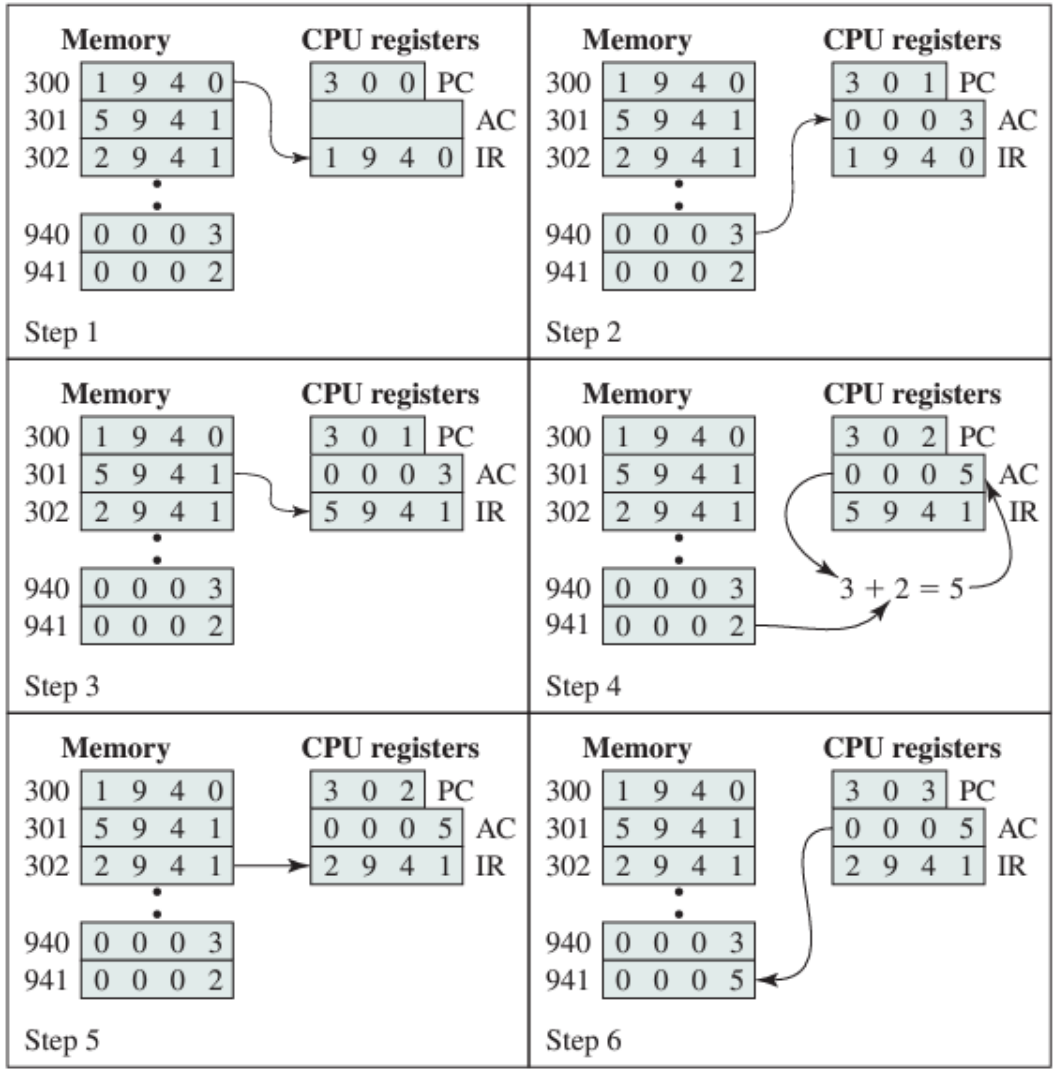
MAR $\leftarrow$ *PC*
PC $\leftarrow$ *PC* + 1
MBR $\leftarrow$ *MAR*
IR $\leftarrow$ *MBR*

EXECUTE

MAR $\leftarrow$ *IR*
MBR $\leftarrow$ *MAR*
r1 $\leftarrow$ *MBR*

Example

Goal: Add the contents at address 940 to the contents at address 941 and store the result at address 941



Instruction Format

Consider the instructions in our example program: 1940, 5941, 2941

Load	1	940
ADD	5	941
Store	2	941

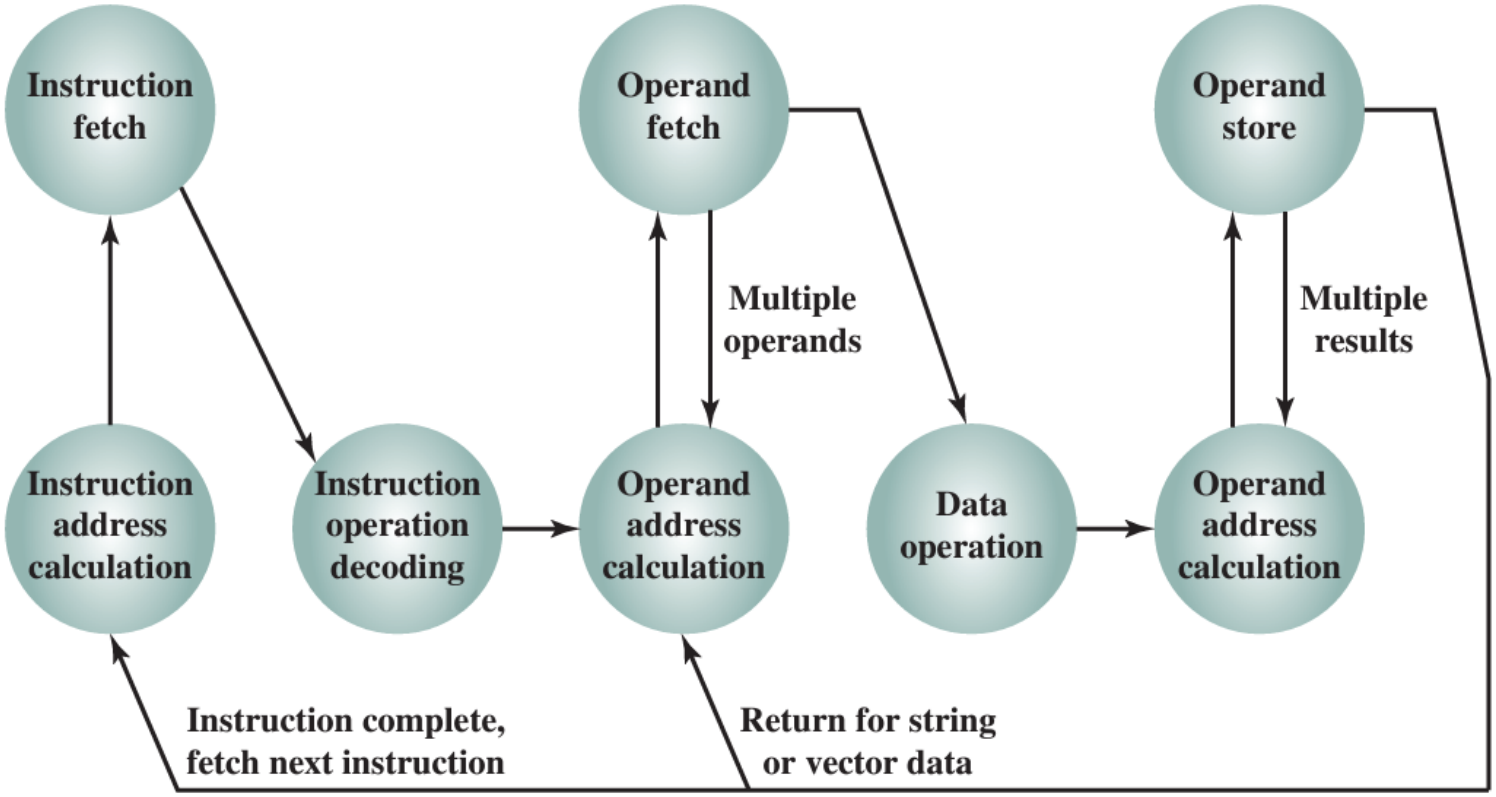
- Size of opcode space = 2^4 = 16
- Size of memory space = 2^6 = 4096 = 4K

Instruction Format

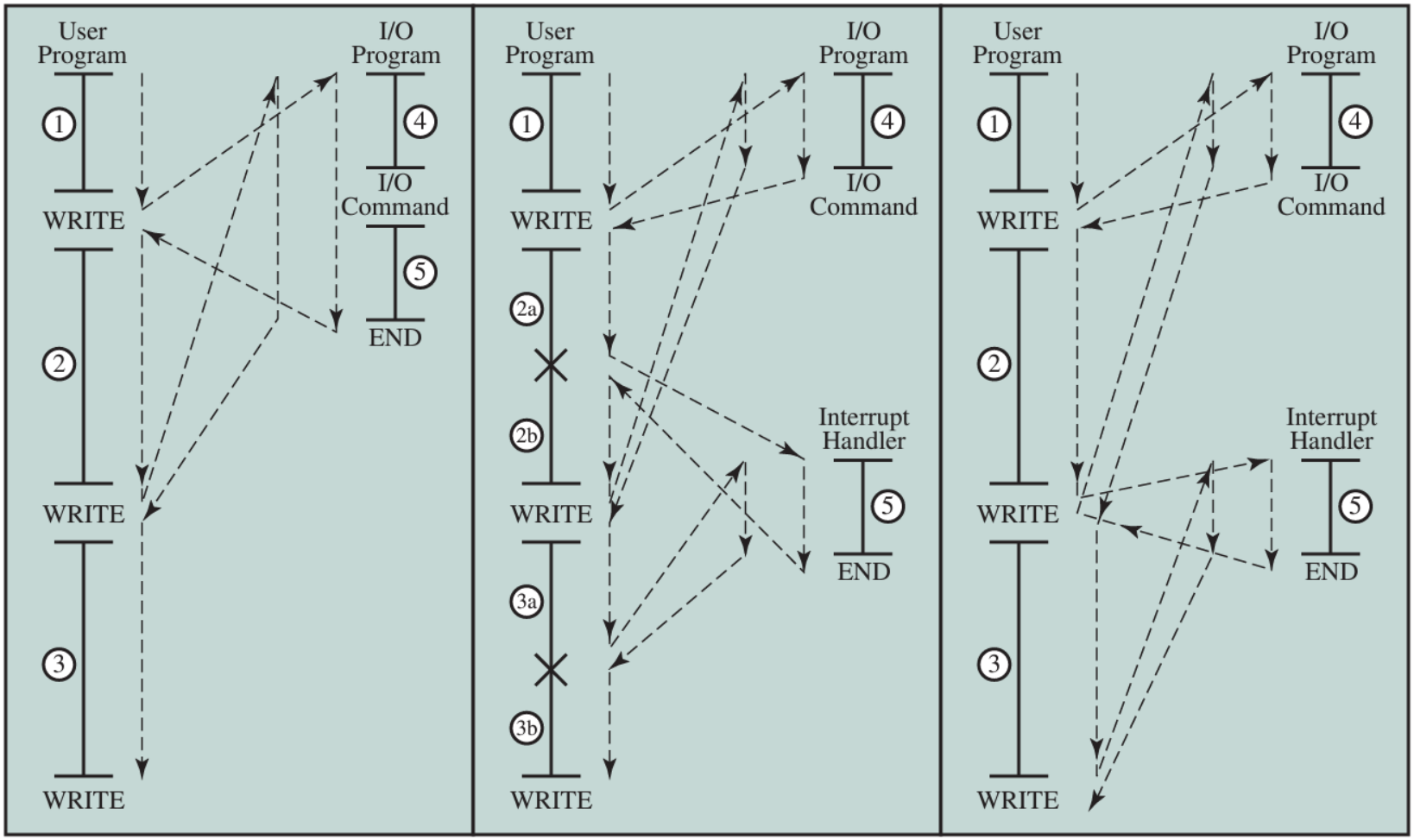
L = M + L	LDA M	1940
	ADD L	5941
	STA L	2941

Instruction	Opcode	Address			Assembly Code
1940	0001	1001	0100	0000	LDA M
5940	0101	1001	0100	0001	ADD L
2940	0010	1001	0100	0001	STA L

Instruction Cycle State



Interrupts



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

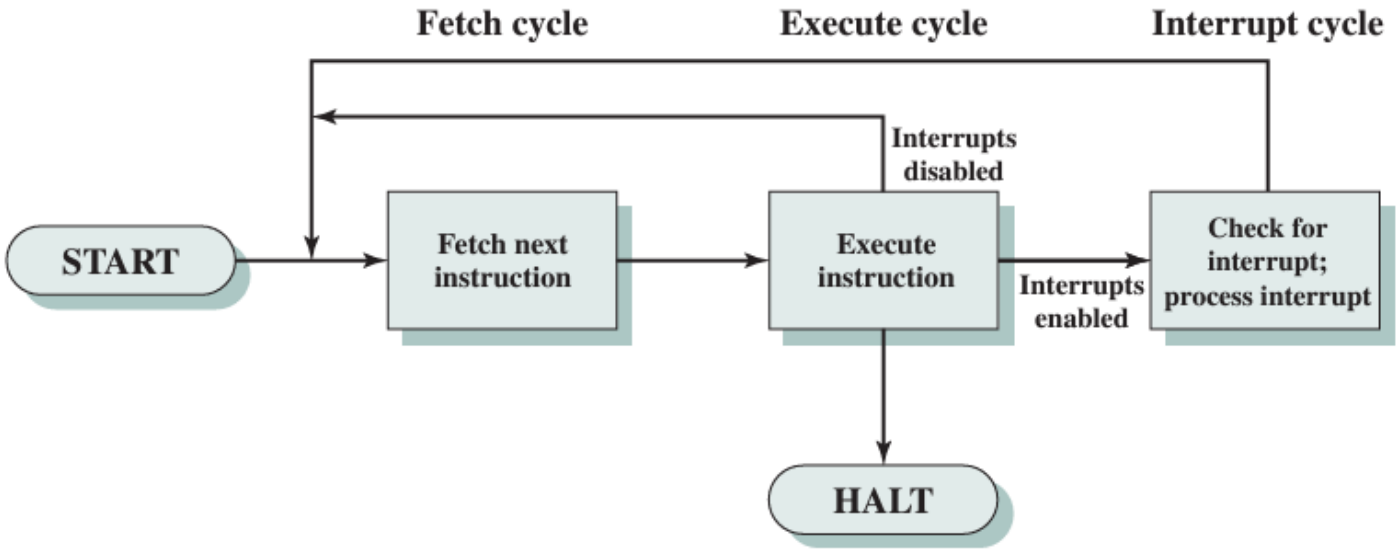
Types of Interrupts

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.

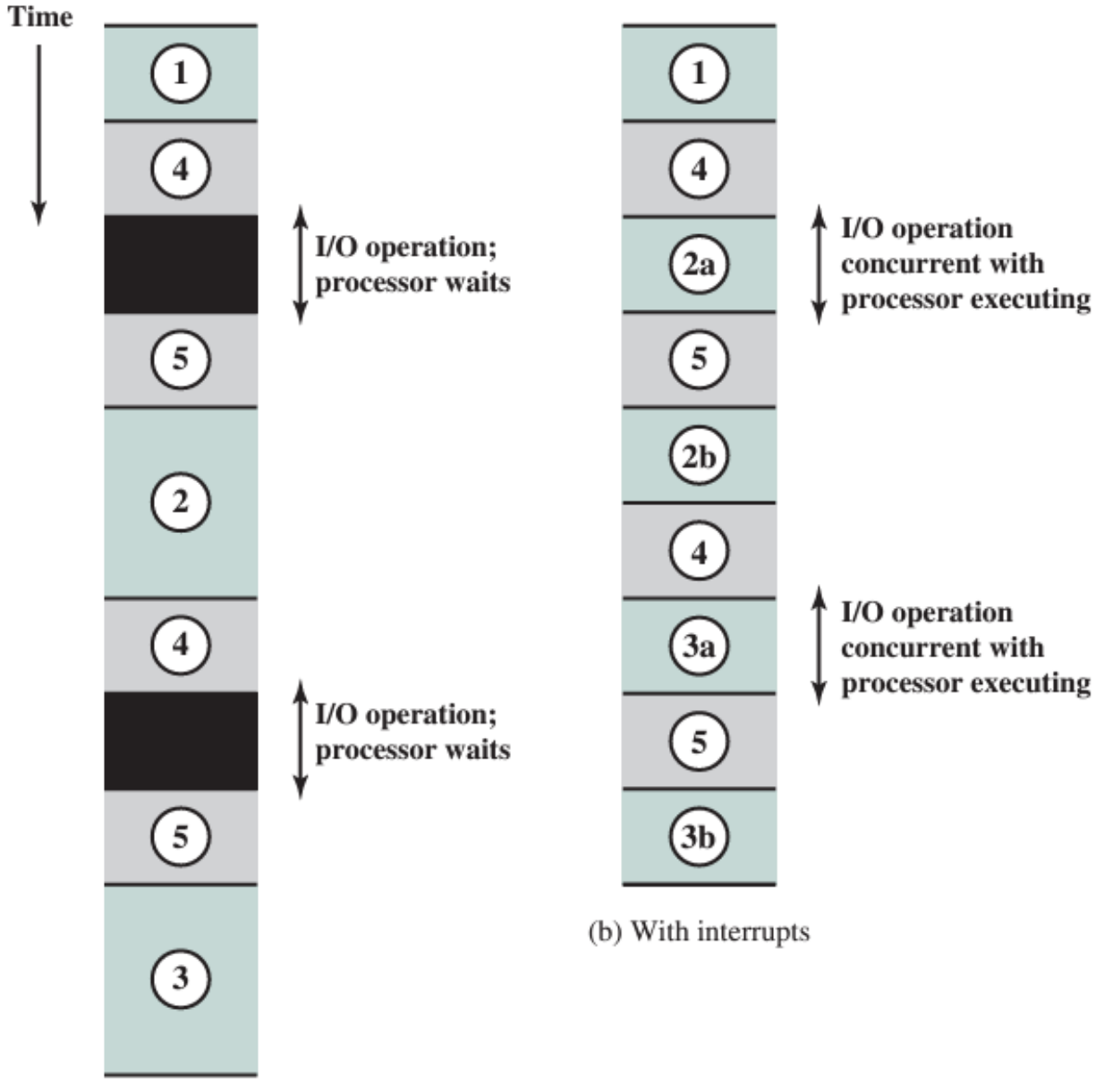
Instruction Cycle with Interrupts

During an interrupt, the processor does the following:

- It suspends execution of the current program being executed
- And saves its context
- It sets the program counter to the starting address of an interrupt handler routine



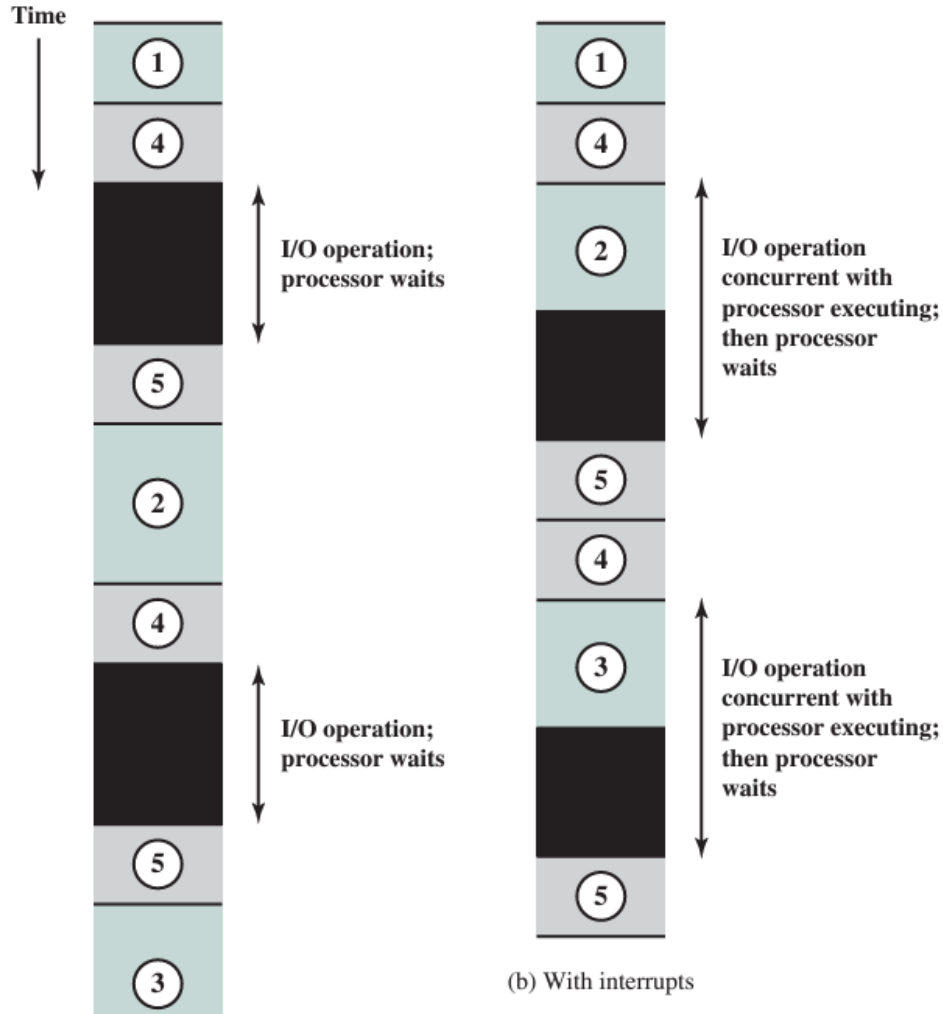
Efficiency of Interrupts: Short Wait



(a) Without interrupts

(b) With interrupts

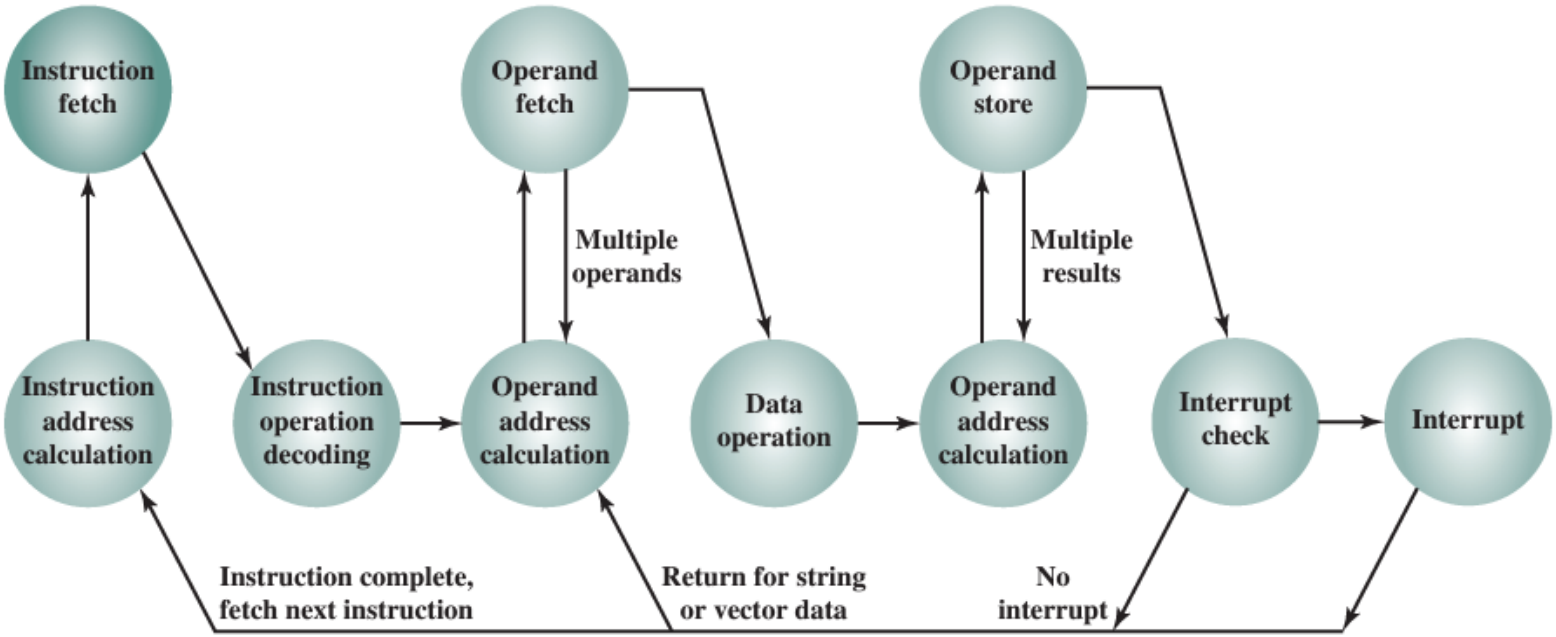
Efficiency of Interrupts: Long Wait



(a) Without interrupts

(b) With interrupts

Instruction Cycle State with Interrupts



Multiple Interrupts

There are two ways of dealing with multiple interrupts:

- **A disabled interrupt:** the processor ignores that interrupt request signal. Interrupts are handled in strict sequential order
- **Define interrupt priority:** allow an interrupt of higher priority to cause a lower-priority interrupt handler to be itself interrupted