Lecture 01

Computer System Fundamentals

Basic Elements · Registers · Instruction Cycle · Interrupts

SENG 21213 · Computer Architecture & Operating Systems

Learning Objectives — By the end of this lecture you should be able to:

- 1. Identify the four basic computer components and describe the role of each
- 2. Explain the function of MAR, MBR, I/OAR, and I/OBR with register-transfer notation
- 3. Trace a program's execution step-by-step through the full fetch-execute cycle showing all register transfers
- 4. Classify instructions into the four categories: processor-memory, processor-I/O, data processing, and control
- 5. Describe the complete 9-step interrupt processing sequence from IRQ to ISR return
- 6. Compare sequential and nested (priority) multiple-interrupt strategies with their trade-offs

Section 1 · Basic Elements of a Computer System

📖 Stallings Reference

- Chapter 1: Computer System Overview
- Key figure: Figure 1.1 — Computer Components: Top-Level View of Computer Function and Interconnection

All modern computers — from microcontrollers to supercomputers — share the same four fundamental building blocks. Understanding their roles and how they communicate is the foundation of computer architecture.

1.1 The Four Basic Elements

Component	Function	Key Characteristics
Processor (CPU)	Controls the operation of the computer and performs data processing. Also called the Central Processing Unit.	Single or multiple cores; contains the ALU and Control Unit; executes the instruction cycle
Main Memory	Stores both instructions (program) and data currently being used.	Volatile (contents lost at power-off); also called primary memory or RAM; organised as addressable words

Component	Function	Key Characteristics
I/O Modules	Transfer data between the computer and external devices.	Contains internal data buffers; examples: disk controller, NIC, GPU, keyboard controller
System Bus	Provides communication pathway between CPU, memory, and I/O modules.	Carries address, data, and control signals; shared resource managed by bus arbitration

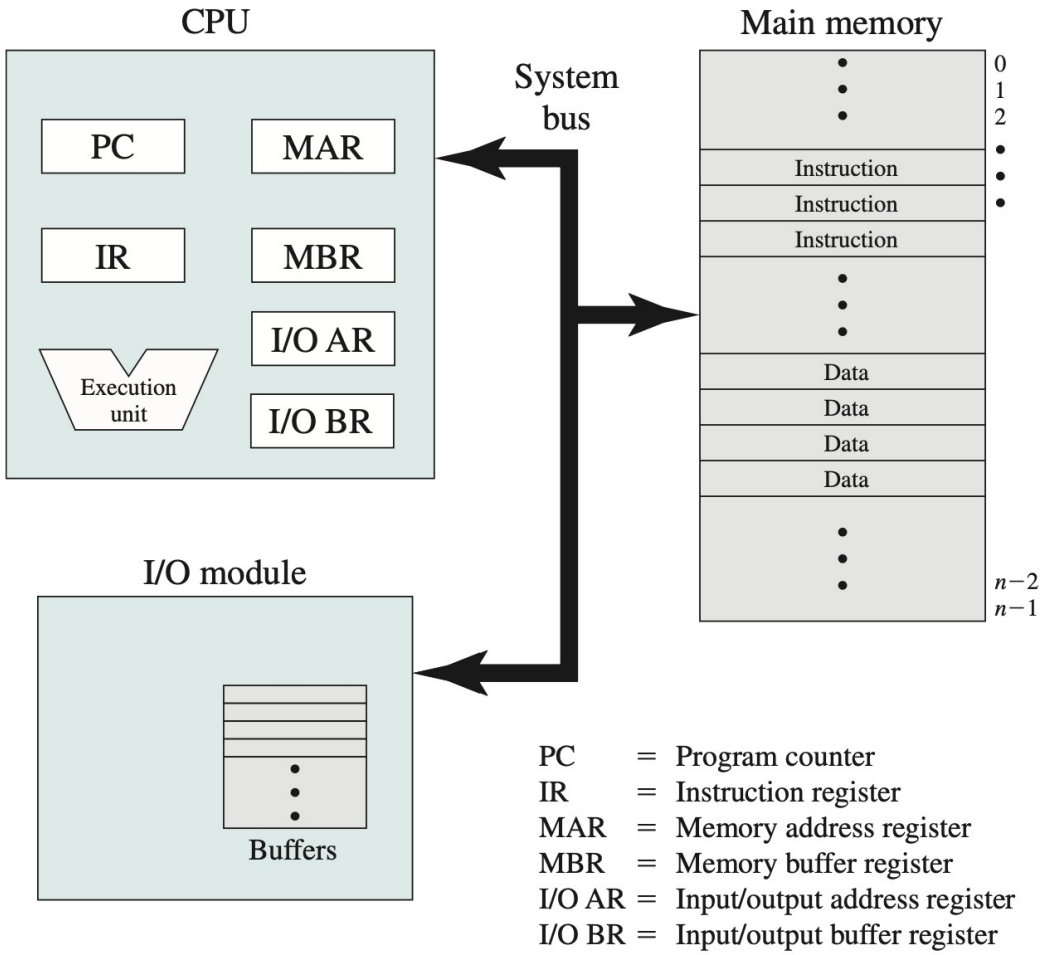


Figure: Top-level view of computer components connected via the System Bus (Stallings Fig. 1.1)

1.2 CPU Registers for Memory and I/O Exchange

The processor uses dedicated internal registers to manage every transfer between the CPU and memory or I/O devices. These registers act as temporary holding locations so that the slower external memory access does not stall internal CPU processing.

Register	Full Name	Role
MAR	Memory Address Register	Holds the address of the memory location to be read from or written to in the next memory operation. CPU sets MAR before initiating any memory

Register	Full Name	Role
		transfer.
MBR	Memory Buffer Register	Holds the data just read from memory (on a fetch) or the data about to be written to memory (on a store). Acts as a buffer between the internal CPU bus and the external memory bus.
I/OAR	I/O Address Register	Specifies which particular I/O device (port address) the current I/O transfer targets.
I/OBR	I/O Buffer Register	Holds data exchanged between the processor and the selected I/O module, analogous to MBR for I/O.

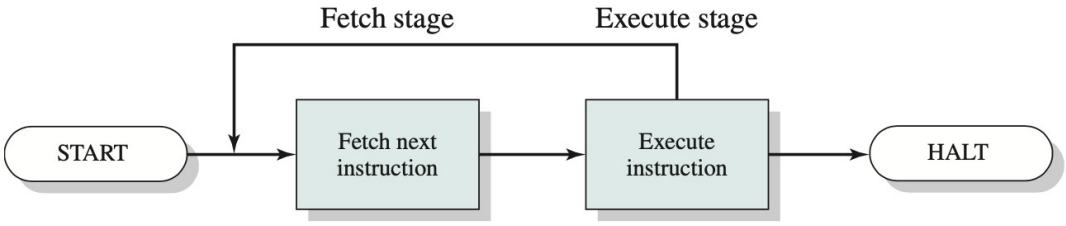


Figure: Internal CPU registers and their roles in memory and I/O data transfer (Stallings Fig. 1.3 region)

★ Key Concept — Register Transfer Notation

- MAR ← PC means: copy the value of the Program Counter into the Memory Address Register.
- MBR ← Memory[MAR] means: read the memory cell at address MAR and place the data in MBR.
- IR ← MBR means: copy the instruction now in MBR into the Instruction Register.
- This notation is used throughout the textbook to describe micro-operations (register transfers) at each step of the instruction cycle.

Section 2 · Instruction Execution — The Fetch-Execute Cycle

📖 Stallings Reference

- Chapter 1: Figure 1.3 — Example of Program Execution
- The six-step trace showing PC, IR, AC, MAR, MBR values at each stage is a core exam topic.

Program execution consists of repeatedly performing the two-step instruction cycle: FETCH then EXECUTE. The cycle continues until the processor is turned off, an unrecoverable error occurs, or a HALT instruction is encountered.

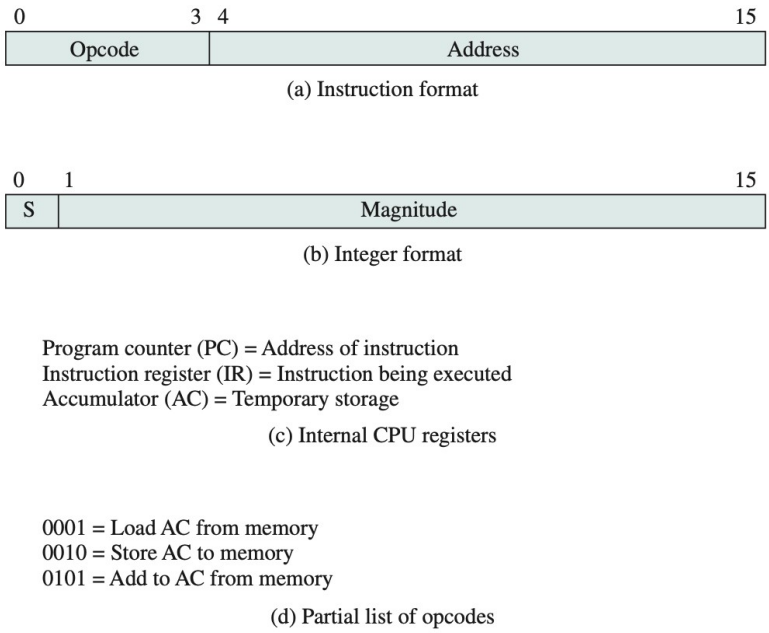


Figure: The basic instruction cycle — Fetch stage followed by Execute stage (Stallings Fig. 1.2)

2.1 The Fetch Stage — Detailed Steps

1. PC → MAR: The processor copies the Program Counter (address of next instruction) into MAR.
2. MAR → Memory: The memory system reads the word at address MAR.
3. Memory → MBR: The retrieved instruction word is placed in the Memory Buffer Register.
4. MBR → IR: The instruction is transferred to the Instruction Register for decoding.
5. PC ← PC + 1: The Program Counter is incremented to point to the next sequential instruction (unless a branch overrides this).

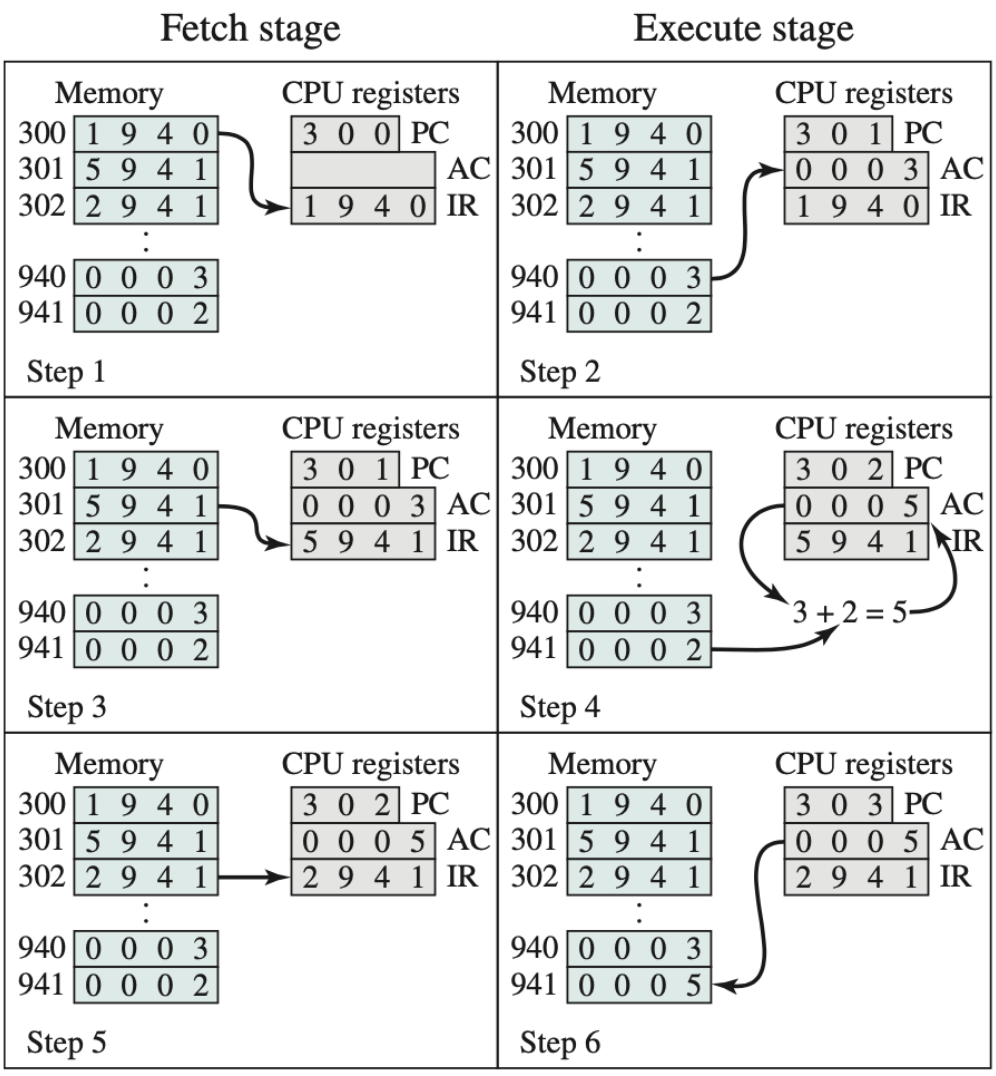


Figure: Instruction format: 4-bit opcode (hex digit) | 12-bit address (three hex digits) (Stallings Fig. 1.4)

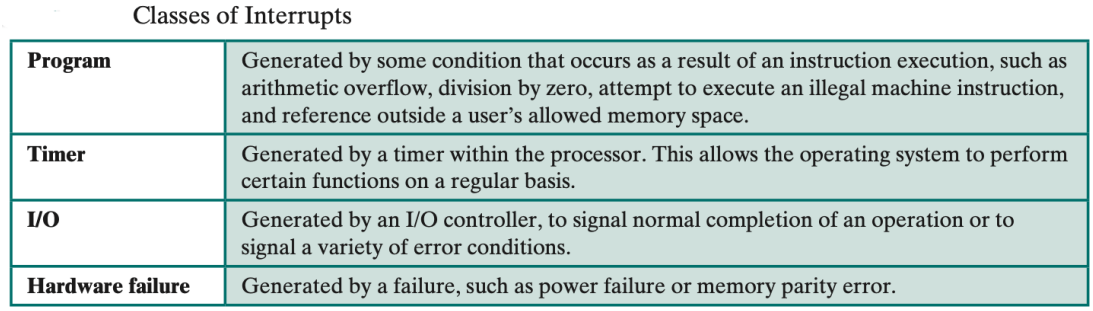
2.2 The Execute Stage — Four Instruction Categories

Category	Description	Example Operation
Processor-Memory	Data transferred between the CPU and main memory. The operand address is decoded from the instruction.	LOAD AC from Memory[940]; STORE AC to Memory[941]
Processor-I/O	Data transferred between the CPU and an I/O module identified by the I/OAR.	READ character from keyboard port 0x60; WRITE byte to serial port
Data Processing	The ALU performs arithmetic (ADD, SUB, MUL) or logic (AND, OR, NOT) on register contents.	ADD AC with Memory[941]; AND AC with bitmask
Control	The instruction alters the Program	PC ← 182 (unconditional jump from address 149 to

Category	Description	Example Operation
	Counter — a branch or jump.	182)

2.3 Complete Worked Trace — Stallings Figure 1.3

The following program adds Memory[940] to Memory[941] and stores the result at Memory[941]. Three instructions are required: LOAD (opcode 1), ADD (opcode 5), STORE (opcode 2). The processor has a single data register — the Accumulator (AC). Instructions and data are both 16-bit words.



Classes 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, and 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 or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Figure: Partial program execution showing PC, IR, MBR, MAR, AC values at each step (Stallings Fig. 1.3)

Step	PC	MAR	MBR	IR	AC	Action
Fetch 1	300	300	1940	—	—	MAR ← 300; MBR ← Memory[300]=1940; IR ← 1940; PC ← 301
Execute 1	301	940	(Memory[940])	1940	=Memory[940]	Opcode=1: LOAD. MAR ← 940; MBR ← Memory[940]; AC ← MBR
Fetch 2	301	301	5941	—	=Memory[940]	MAR ← 301; MBR ← Memory[301]=5941; IR ← 5941; PC ← 302
Execute 2	302	941	(Memory[941])	5941	AC+Memory[941]	Opcode=5: ADD. MAR ← 941; MBR ← Memory[941]; AC ← AC+MBR
Fetch 3	302	302	2941	—	=sum	MAR ← 302; MBR ← Memory[302]=2941; IR ← 2941; PC ← 303
Execut	303	941	=sum	2941	=sum	Opcode=2: STORE. MAR ← 941; MBR ← AC; Memory[941] ← MBR

Step	PC	MAR	MBR	IR	AC	Action
e 3						

⚠ Common Mistake

- Students often forget that the PC is incremented DURING the fetch stage (step 5 of fetch), not after the execute stage. By the time Execute begins, the PC already points to the next instruction.

Section 3 · Interrupts

📖 Stallings Reference

- Chapter 1: Section 1.4 — Interrupts
- Key figures: 1.6 (program flow without/with interrupts), 1.8 (instruction cycle with interrupt), 1.9 (interrupt processing sequence)

Interrupts are signals sent to the processor from hardware or software that divert normal sequential program execution to an Interrupt Service Routine (ISR). They are the primary mechanism by which the OS regains control of the CPU after handing it to a user process.

3.1 Why Interrupts? — The Motivation

Without interrupts, the CPU must poll I/O devices in a tight loop, burning millions of cycles checking whether a device has finished. Consider:

- A 1 GHz CPU executes ~10^9 instructions per second.
- A hard disk at 7,200 RPM has an average rotational latency of 4 ms.
- Without interrupts: 4 ms × 10^9 cycles/s = 4,000,000 wasted cycles per I/O operation.
- With interrupts: the CPU executes useful work during those 4 ms; only a brief ISR runs when the disk signals completion.

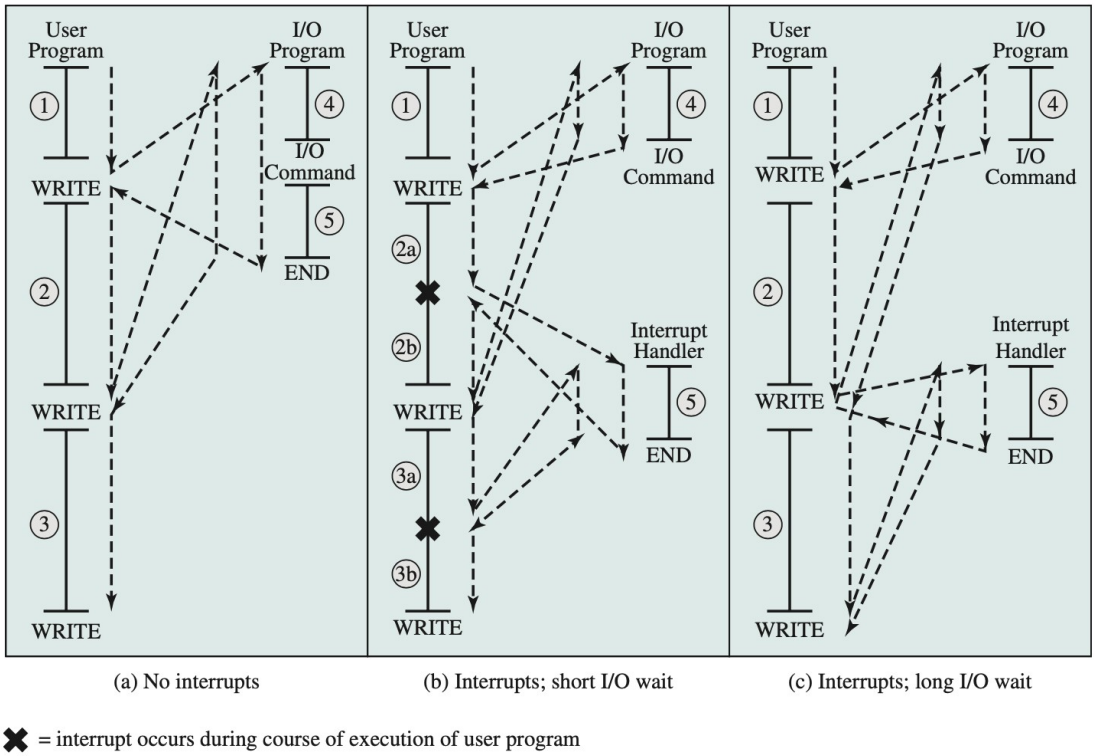


Figure: Program flow: (a) without interrupts — CPU idles during I/O; (b) with interrupts — CPU overlaps I/O with useful computation (Stallings Fig. 1.6)

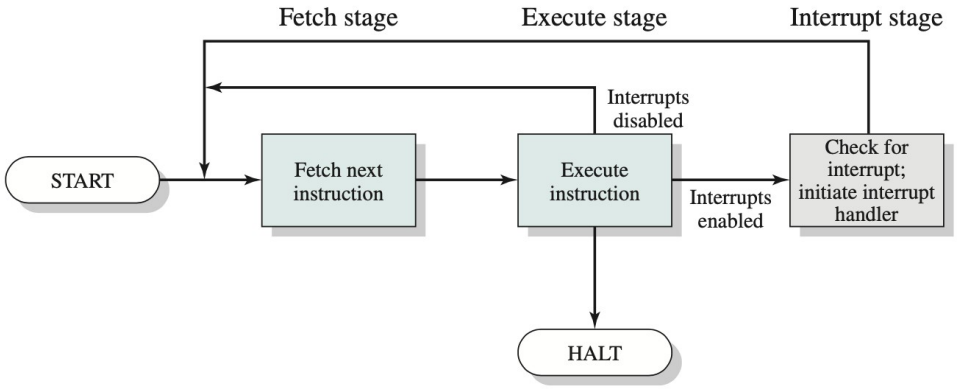


Figure: Extended instruction cycle including the interrupt check phase after each Execute (Stallings Fig. 1.8)

3.2 The 9-Step Interrupt Processing Sequence

When an I/O device completes an operation, the following hardware and software events occur in strict sequence:

Step	Who	Action
1	Device	Issues an interrupt request signal (IRQ) on the interrupt bus to the CPU.

Step	Who	Action
2	CPU	Finishes executing the current instruction completely before responding. Interrupts are only checked between instructions.
3	CPU → Device	CPU asserts an interrupt acknowledge signal. The device removes its IRQ signal.
4	CPU	Saves the current Program Status Word (PSW) and PC onto the kernel control stack. These are the minimum context needed to resume the interrupted program.
5	CPU	Loads PC with the address of the ISR from the Interrupt Vector Table (IVT). Each interrupt type has its own vector entry.
6	ISR	Saves ALL processor registers (EAX, EBX, ECX, EDX, ESI, EDI, EBP) onto the stack — not just PC/PSW — because the ISR will use these registers.
7	ISR	Processes the interrupt: reads device status, transfers remaining data, sends acknowledgments, clears interrupt flags.
8	ISR	Restores all saved general registers from the stack (reverse of step 6).
9	ISR	Restores PSW and PC from the control stack. The processor resumes executing the interrupted program as if nothing happened.

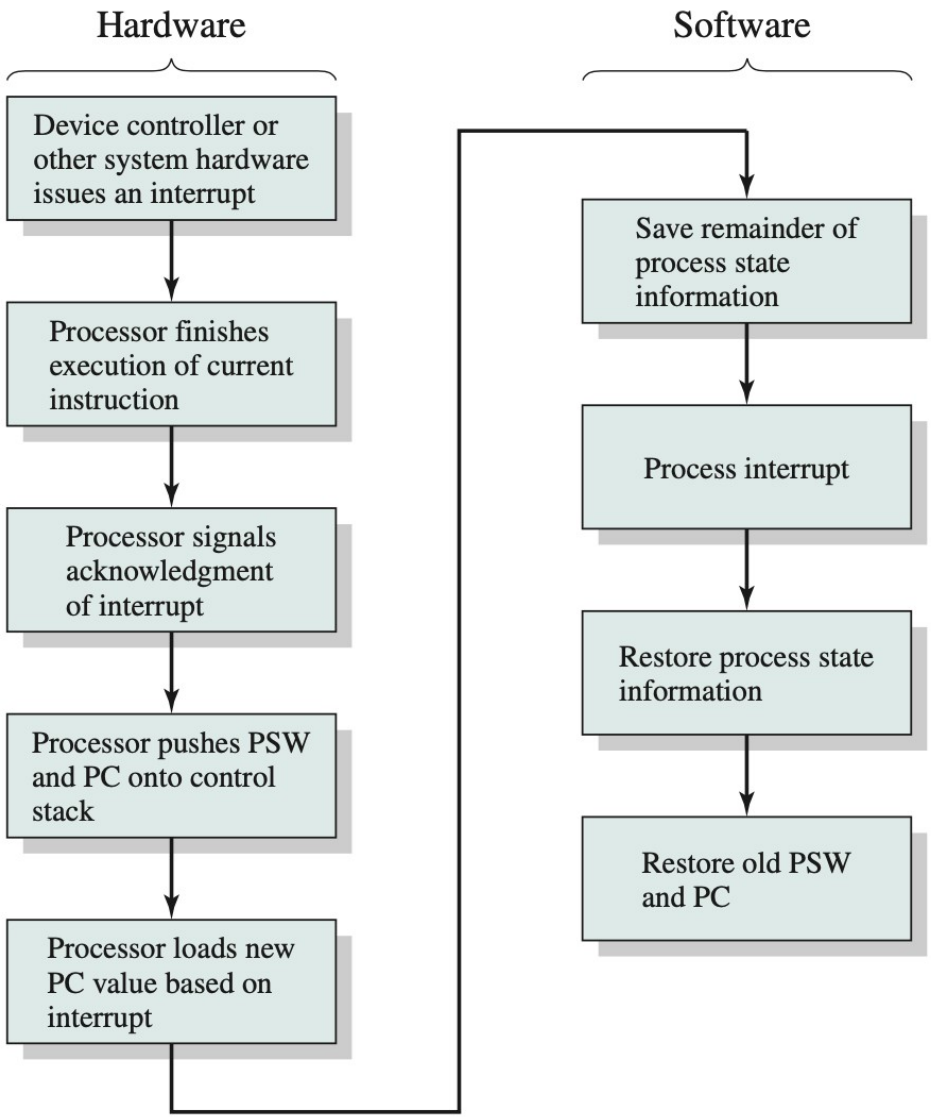


Figure: Context saving and restoring during interrupt processing — control stack at steps 4/6 and 8/9 (Stallings Fig. 1.9)

★ Key Concept — The Interrupt Vector Table (IVT)

- The IVT is a fixed memory region (on x86: at physical address 0x0000) containing 256 entries, each holding the address of an ISR. Entry 0 = divide-by-zero. Entry 32–47 = hardware IRQs 0–15 (timer, keyboard, disk...). When an interrupt N occurs, the CPU automatically fetches the ISR address from IVT[N] and jumps to it.

3.3 Multiple Interrupts — Two Strategies

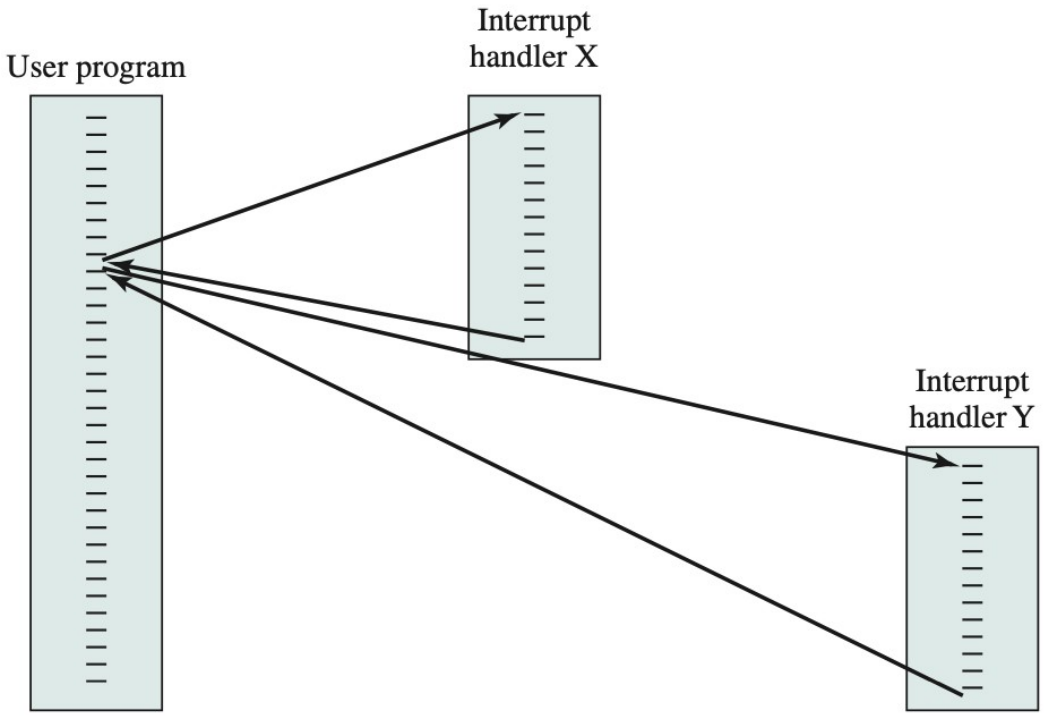


Figure: Sequential interrupt processing (left): no nesting; nested/priority interrupt processing (right): higher-priority ISR can interrupt a lower-priority ISR (Stallings Fig. 1.11)

Strategy	Mechanism	Advantage	Disadvantage
Disable interrupts (Sequential)	CPU sets an interrupt-disable flag during ISR execution. New IRQs are held pending and checked after re-enable.	Simple. ISRs run to completion. No stack overflow risk from nesting.	Cannot respond to time-critical events during ISR. A fast comms line may lose data while a slow printer ISR runs.
Priority-based (Nested)	ISRs run with a certain priority level. An IRQ of HIGHER priority can interrupt the current ISR. Lower-priority IRQs are deferred.	Time-critical events (e.g. network packet arrival) are handled immediately even if a lower-priority ISR is running.	Complex. Requires a nested stack of saved contexts. Risk of stack overflow if too many levels nest.

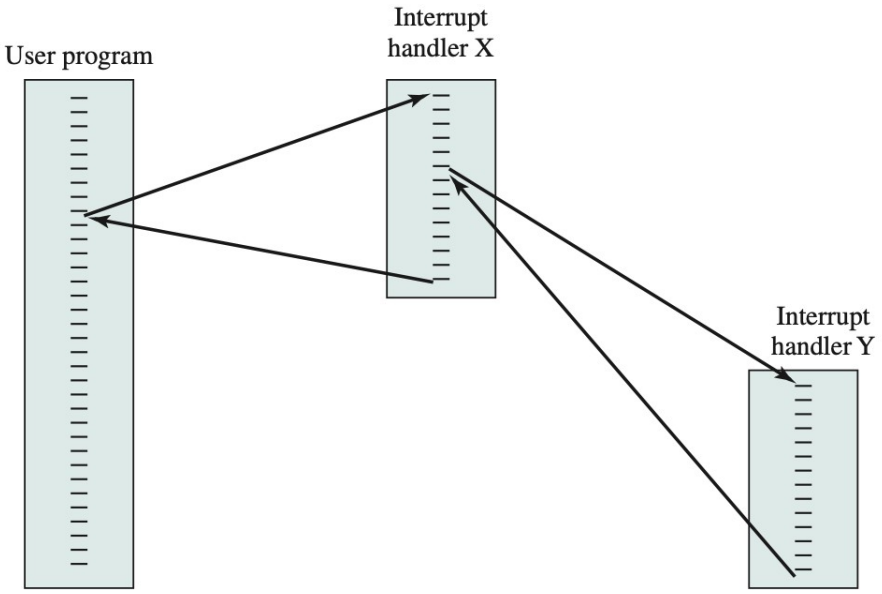


Figure: Nested interrupt processing example — three interrupt sources with different priorities (Stallings Fig. 1.12)

Section 4 · Practice Exercises — Stallings Style

✎ Exercise 1.1 [10 marks]

- The processor in Figure 1.3 also has two I/O instructions: opcode 0011 = Load AC from I/O, opcode 0111 = Store AC to I/O. The 12-bit address field identifies the device.
- Program: (1) Load AC from device 5; (2) Add contents of Memory[940]; (3) Store AC to device 6. Device 5 returns value 3; Memory[940] = 2.
- (a) [4] Show the complete execution trace (Fetch + Execute for each of the 3 instructions) in the format of Stallings Fig. 1.3, showing PC, MAR, MBR, IR, and AC at each step.
- (b) [3] Write the 6 hex instruction words that would appear in memory starting at location 300.
- (c) [3] Explain why the I/OAR and I/OBR are used in the execute stages of instructions 1 and 3, but not instruction 2.

✎ Exercise 1.2 [8 marks]

- The program execution in Figure 1.3 (three instructions) is described in six steps. Expand this description to show the use of MAR and MBR at every micro-operation.
- (a) [4] For Fetch 1, write out all five register-transfer micro-operations in order.
- (b) [2] For Execute 1 (LOAD), write out all micro-operations showing how data moves from Memory[940] to AC via MAR and MBR.
- (c) [2] For Execute 3 (STORE), write out the micro-operations showing how AC is stored to Memory[941].

✎ Exercise 1.3 [12 marks]

- A hypothetical 32-bit microprocessor has 32-bit instructions structured as: [8-bit opcode | 24-bit

operand/address].

- (a) [2] What is the maximum directly addressable memory capacity in bytes? Justify.
- (b) [3] Describe the impact on throughput if the bus has a 32-bit address bus and 16-bit data bus vs. a 16-bit address bus and 16-bit data bus. Which creates a bottleneck and why?
- (c) [2] How many bits are required for the PC and for the IR? Justify both.
- (d) [5] Assume an instruction accessing a 32-bit memory operand takes 3 bus cycles (one for instruction fetch, two for data). If the clock is 200 MHz and each bus cycle takes 2 clock ticks, calculate the instruction throughput in MIPS.

✎ Exercise 1.4 [10 marks]

- A microprocessor generates 16-bit addresses and has a 16-bit data bus.
- (a) [2] Maximum memory address space if connected to a '16-bit memory' (16-bit words)? Express in bytes.
- (b) [2] Maximum space if connected to an '8-bit memory' (byte-addressable)? Explain the difference from (a).
- (c) [3] What architectural feature allows this processor to access a separate I/O address space, distinct from memory? How does the CPU signal which space is being accessed?
- (d) [3] If I/O instructions specify an 8-bit port number, how many 8-bit I/O ports are supported? If the bus is 16 bits wide, how many 16-bit I/O ports? Explain the difference.

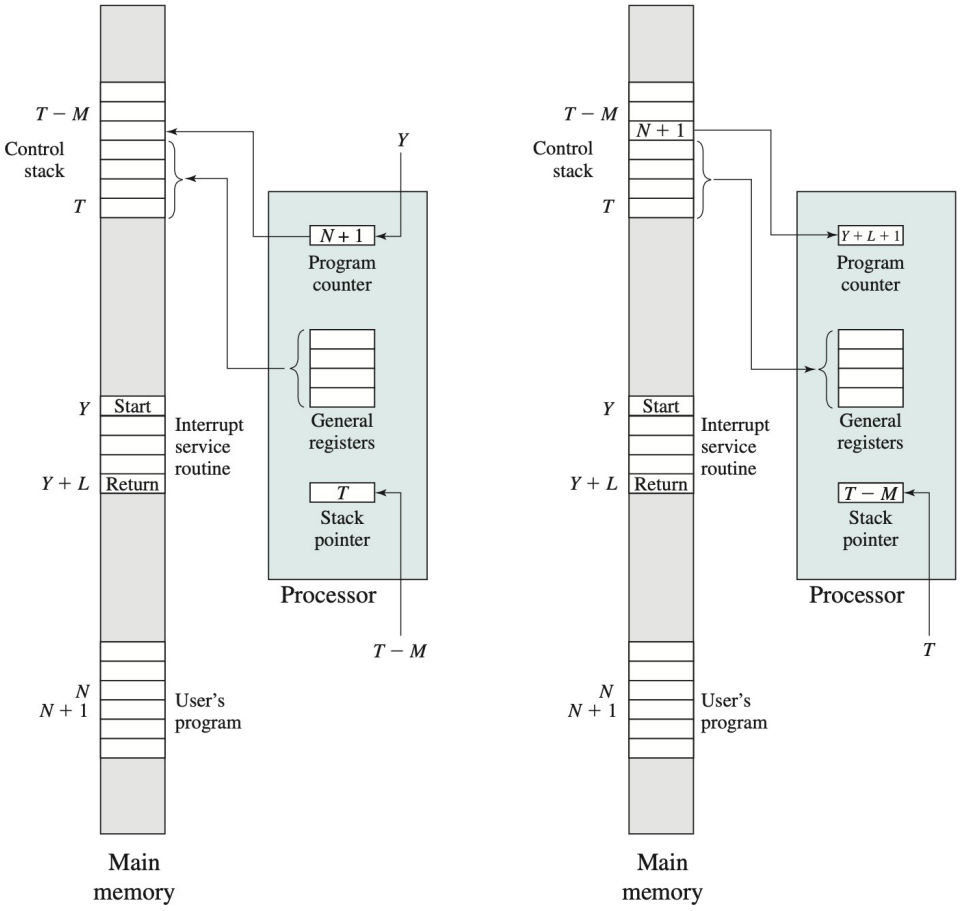


Figure: Reference figure for Exercises 1.1 and 1.2 — Stallings Figure 1.3 region

Interrupt Handling

For the user program, an interrupt suspends the normal sequence of execution. When the interrupt processing is completed, execution resumes . Thus, the user program does not have to contain any special code to accommodate interrupts; the processor and the OS are responsible for suspending the user program and then resuming it at the same point.

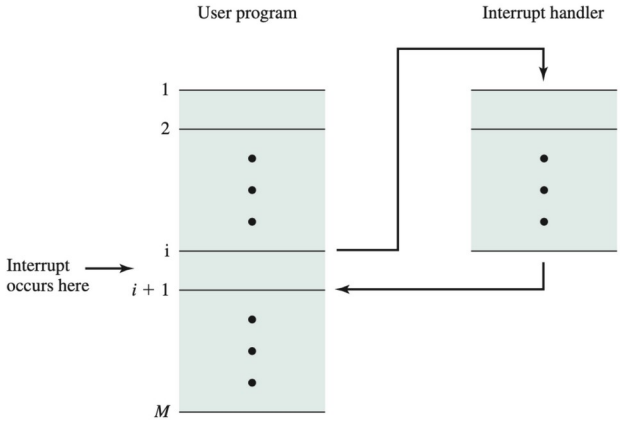


Figure: Reference figure for Exercise 1.4 — I/O port addressing