

# Assembly Language for x86 Processors

7<sup>th</sup> Edition

Kip Irvine

## Chapter 1: Basic Concepts

# Welcome to Assembly Language

- Some Good Questions to Ask
- Assembly Language Applications

# Questions to Ask

- Why am I learning Assembly Language?
- What background should I have?
- What is an assembler?
- What hardware/software do I need?
- What types of programs will I create?
- What do I get with this book?
- What will I learn?

# Welcome to Assembly Language *(cont)*

- How does assembly language (AL) relate to machine language?
- How do C++ and Java relate to AL?
- Is AL portable?
- Why learn AL?

# Assembly Language Applications

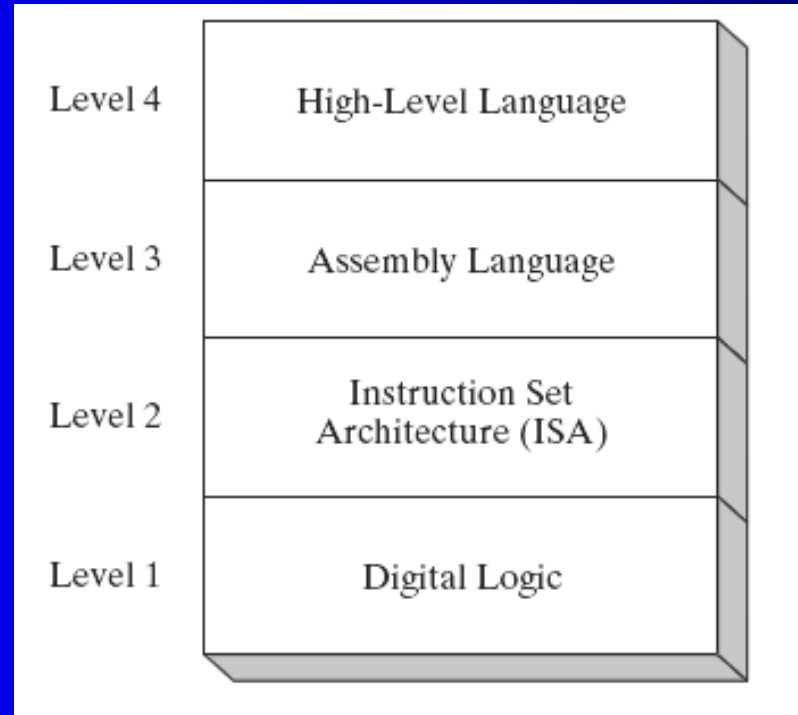
- Some representative types of applications:
  - Business application for single platform
  - Hardware device driver
  - Embedded systems & computer games

(see next panel)

# Comparing ASM to High-Level Languages

Type of Application	High-Level Languages	Assembly Language
Business application software, written for single platform, medium to large size.	Formal structures make it easy to organize and maintain large sections of code.	Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.
Hardware device driver.	Language may not provide for direct hardware access. Even if it does, awkward coding techniques must often be used, resulting in maintenance difficulties.	Hardware access is straightforward and simple. Easy to maintain when programs are short and well documented.
Business application written for multiple platforms (different operating systems).	Usually very portable. The source code can be recompiled on each target operating system with minimal changes.	Must be recoded separately for each platform, often using an assembler with a different syntax. Difficult to maintain.
Embedded systems and computer games requiring direct hardware access.	Produces too much executable code, and may not run efficiently.	Ideal, because the executable code is small and runs quickly.

# Specific Machine Levels



(descriptions of individual levels follow . . . )

# Translating Languages

English: Display the sum of A times B plus C.

C++: `cout << (A * B + C);`

Assembly Language:

```
mov eax,A
mul B
add eax,C
call WriteInt
```

Intel Machine Language:

```
A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000
```



# High-Level Language

- Level 4
- Application-oriented languages
  - C++, Java, Python, C# . . .
- Programs compile into assembly language (Level 4)

# Assembly Language

- Level 3
- Instruction mnemonics that have a one-to-one correspondence to machine language
- Programs are translated into Instruction Set Architecture Level - machine language (Level 2)

# Instruction Set Architecture (ISA)

- Level 2
- Also known as **conventional machine language**
- Executed by Level 1 (Digital Logic)

# Digital Logic

- Level 1
- CPU, constructed from digital logic gates
- System bus
- Memory
- Implemented using bipolar transistors

next: Data Representation

# Assembly Language for x86 Processors

7<sup>th</sup> Edition

Kip Irvine

## Chapter 2: x86 Processor Architecture

# Chapter Overview

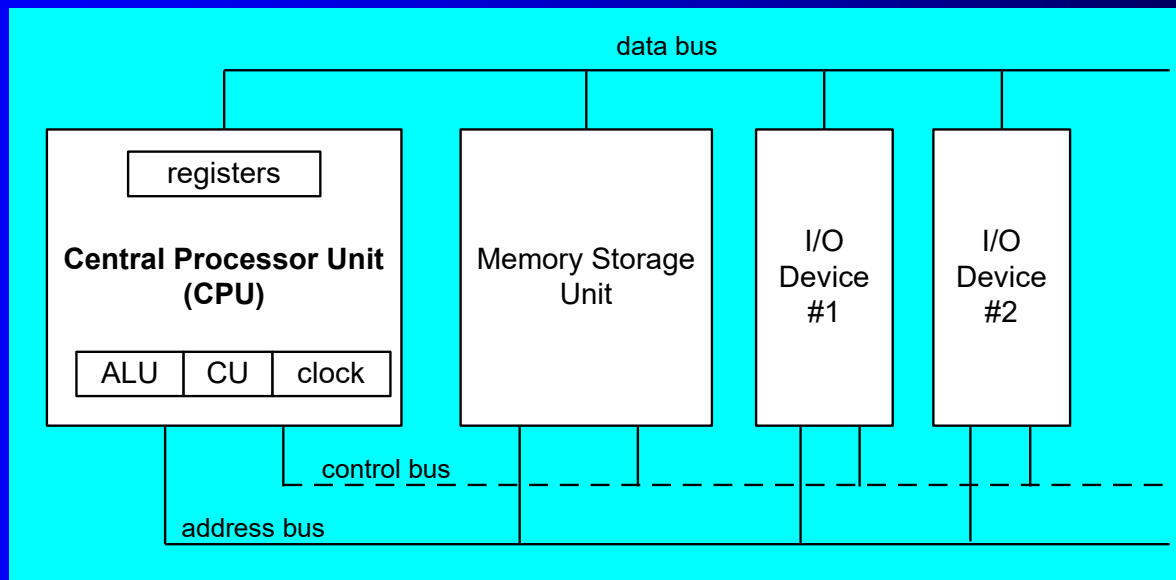
- General Concepts
- IA-32 Processor Architecture
- IA-32 Memory Management
- 64-bit Processors
- Components of an IA-32 Microcomputer
- Input-Output System

# General Concepts

- Basic microcomputer design
- Instruction execution cycle
- Reading from memory
- How programs run

# Basic Microcomputer Design

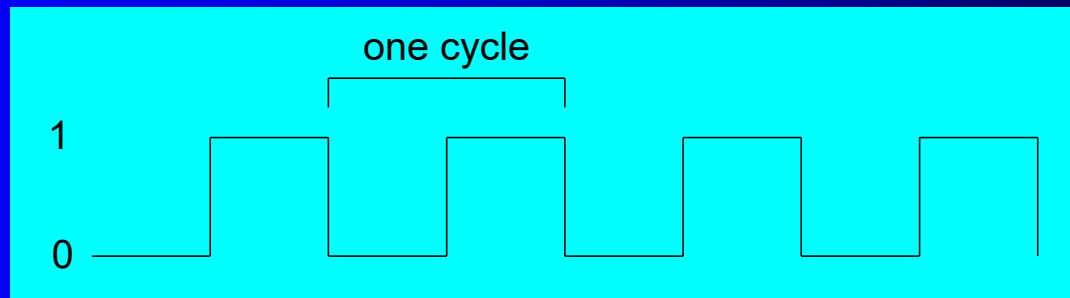
- clock synchronizes CPU operations
- control unit (CU) coordinates sequence of execution steps
- ALU performs arithmetic and bitwise processing





# Clock

- synchronizes all CPU and BUS operations
- machine (clock) cycle measures time of a single operation
- clock is used to trigger events



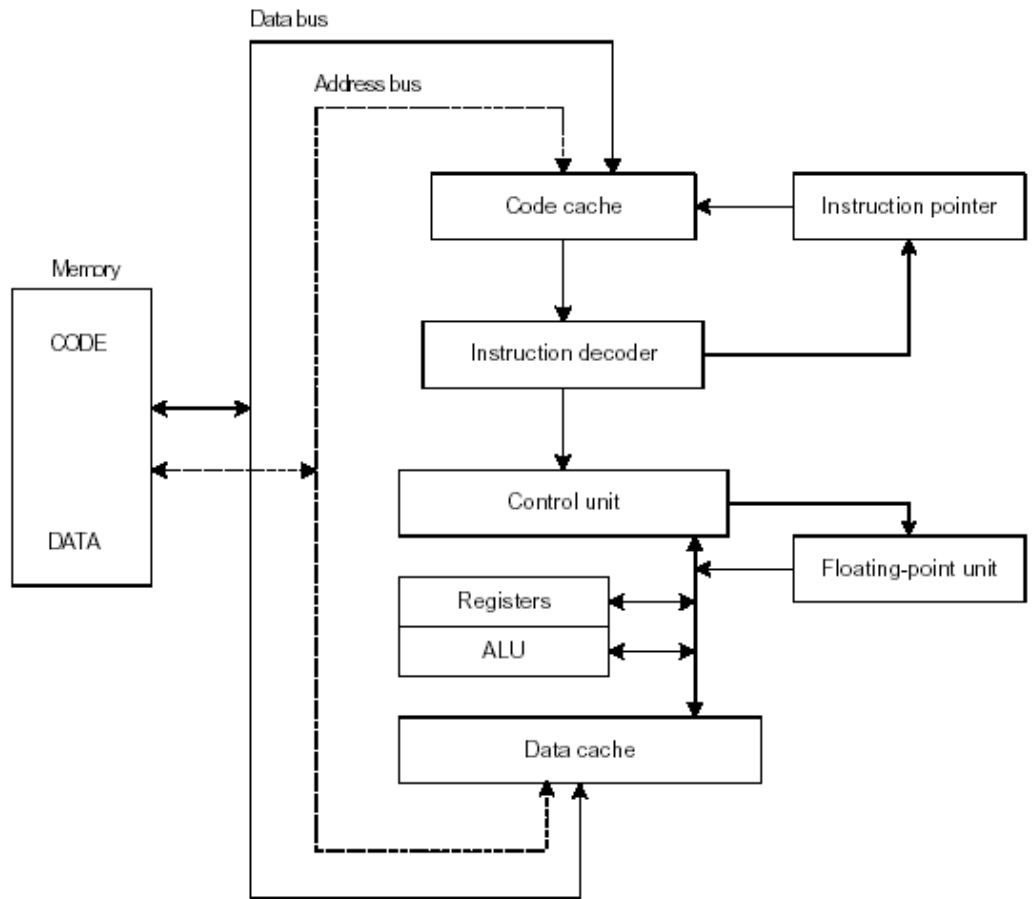
# What's Next

- General Concepts
- **IA-32 Processor Architecture**
- IA-32 Memory Management
- 64-Bit Processors
- Components of an IA-32 Microcomputer
- Input-Output System

# Instruction Execution Cycle

Figure 2-2 Simplified Pentium CPU Block Diagram.

- Fetch
- Decode
- Fetch operands
- Execute
- Store output



# Reading from Memory

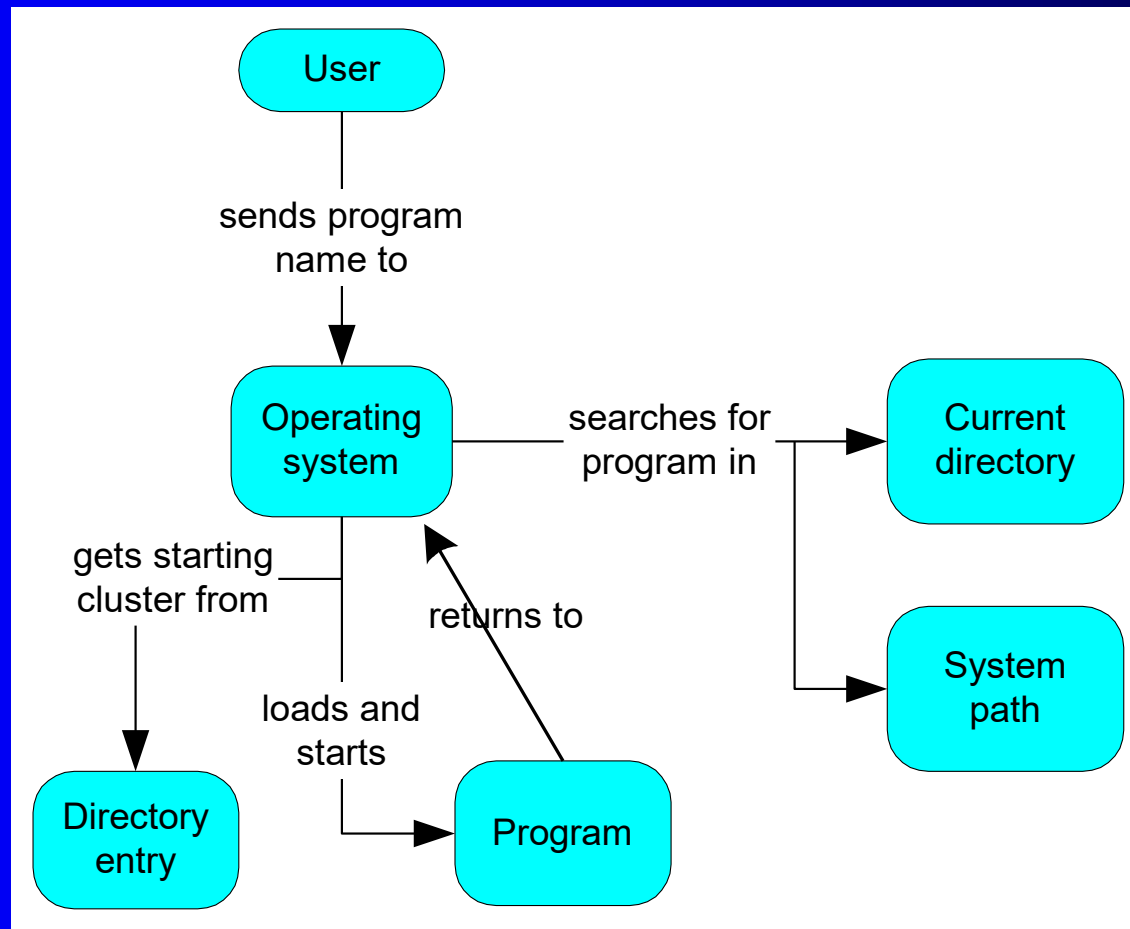
Multiple machine cycles are required when reading from memory, because it responds much more slowly than the CPU. The steps are:

1. Place the address of the value you want to read on the address bus.
2. Assert (changing the value of) the processor's RD (read) pin.
3. Wait one clock cycle for the memory chips to respond.
4. Copy the data from the data bus into the destination operand

# Cache Memory

- High-speed expensive static RAM both inside and outside the CPU.
  - Level-1 cache: inside the CPU
  - Level-2 cache: outside the CPU
- Cache hit: when data to be read is already in cache memory
- Cache miss: when data to be read is not in cache memory.

# How a Program Runs



# IA-32 Processor Architecture

- Modes of operation
- Basic execution environment
- Floating-point unit
- Intel Microprocessor history

# Modes of Operation

- Protected mode
    - native mode (Windows, Linux)
  - Real-address mode
    - native MS-DOS
  - System management mode
    - power management, system security, diagnostics
- Virtual-8086 mode
    - hybrid of Protected
    - each program has its own 8086 computer



# Basic Execution Environment

- Addressable memory
- General-purpose registers
- Index and base registers
- Specialized register uses
- Status flags
- Floating-point, MMX, XMM registers

# Addressable Memory

- Protected mode
  - 4 GB
  - 32-bit address
- Real-address and Virtual-8086 modes
  - 1 MB space
  - 20-bit address

# General-Purpose Registers

Named storage locations inside the CPU, optimized for speed.

## 32-bit General-Purpose Registers

EAX
EBX
ECX
EDX

EBP
ESP
ESI
EDI

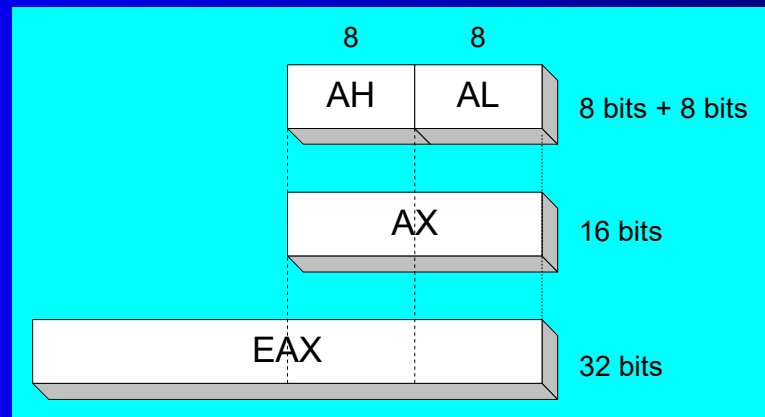
## 16-bit Segment Registers

EFLAGS
EIP

CS	ES
SS	FS
DS	GS

# Accessing Parts of Registers

- Use 8-bit name, 16-bit name, or 32-bit name
- Applies to EAX, EBX, ECX, and EDX



32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

# Index and Base Registers

- Some registers have only a 16-bit name for their lower half:

32-bit	16-bit
ESI	SI
EDI	DI
EBP	BP
ESP	SP

# Some Specialized Register Uses (1 of 2)

- General-Purpose
  - EAX – accumulator
  - ECX – loop counter
  - ESP – stack pointer
  - ESI, EDI – index registers
  - EBP – extended frame pointer (stack)
- Segment
  - CS – code segment
  - DS – data segment
  - SS – stack segment
  - ES, FS, GS - additional segments

# Some Specialized Register Uses (2 of 2)

- EIP – instruction pointer
- EFLAGS
  - status and control flags
  - each flag is a single binary bit

# Status Flags

- Carry
  - unsigned arithmetic out of range
- Overflow
  - signed arithmetic out of range
- Sign
  - result is negative
- Zero
  - result is zero
- Auxiliary Carry
  - carry from bit 3 to bit 4
- Parity
  - sum of 1 bits is an even number



# Floating-Point, MMX, XMM Registers

- Eight 80-bit floating-point data registers
  - ST(0), ST(1), . . . , ST(7)
  - arranged in a stack
  - used for all floating-point arithmetic
- Eight 64-bit MMX registers
- Eight 128-bit XMM registers for single-instruction multiple-data (SIMD) operations
- MMX(Multi Media Extension(多媒体扩展))

ST(0)
ST(1)
ST(2)
ST(3)
ST(4)
ST(5)
ST(6)
ST(7)

# What's Next

- General Concepts
- IA-32 Processor Architecture
- **IA-32 Memory Management**
- 64-Bit Processors
- Components of an IA-32 Microcomputer
- Input-Output System

# IA-32 Memory Management

- Real-address mode
- Calculating linear addresses
- Protected mode
- Multi-segment model
- Paging

# Protected Mode (1 of 2)

- 4 GB addressable RAM
  - (00000000 to FFFFFFFFh)
- Each program assigned a memory partition which is protected from other programs
- Designed for multitasking
- Supported by Linux & MS-Windows

# What's Next

- General Concepts
- IA-32 Processor Architecture
- IA-32 Memory Management
- **64-Bit Processors**
- Components of an IA-32 Microcomputer
- Input-Output System

# 64-Bit Processors

- 64-Bit Operation Modes
  - Compatibility mode – can run existing 16-bit and 32-bit applications (Windows supports only 32-bit apps in this mode)
  - 64-bit mode – Windows 64 uses this
- Basic Execution Environment
  - addresses can be 64 bits (48 bits, in practice)
  - 16 64-bit general purpose registers
  - 64-bit instruction pointer named RIP

# 64-Bit General Purpose Registers

- 32-bit general purpose registers:
  - EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D, R9D, R10D, R11D, R12D, R13D, R14D, R15D
- 64-bit general purpose registers:
  - RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8, R9, R10, R11, R12, R13, R14, R15

# What's Next

- General Concepts
- IA-32 Processor Architecture
- IA-32 Memory Management
- **Input-Output System**

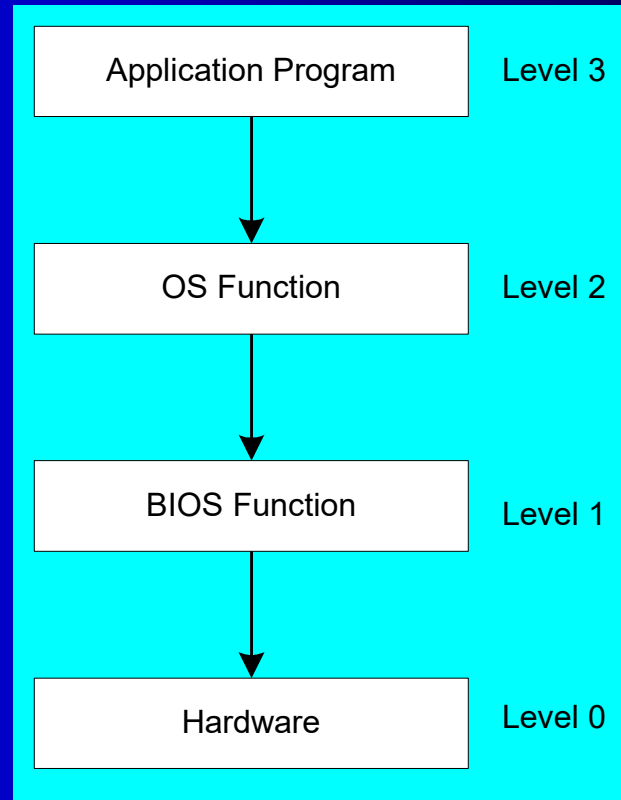


# Levels of Input-Output

- Level 3: High-level language function
  - examples: C++, Java
  - portable, convenient, not always the fastest
- Level 2: Operating system
  - Application Programming Interface (API)
  - extended capabilities, lots of details to master
- Level 1: BIOS
  - drivers that communicate directly with devices
  - OS security may prevent application-level code from working at this level

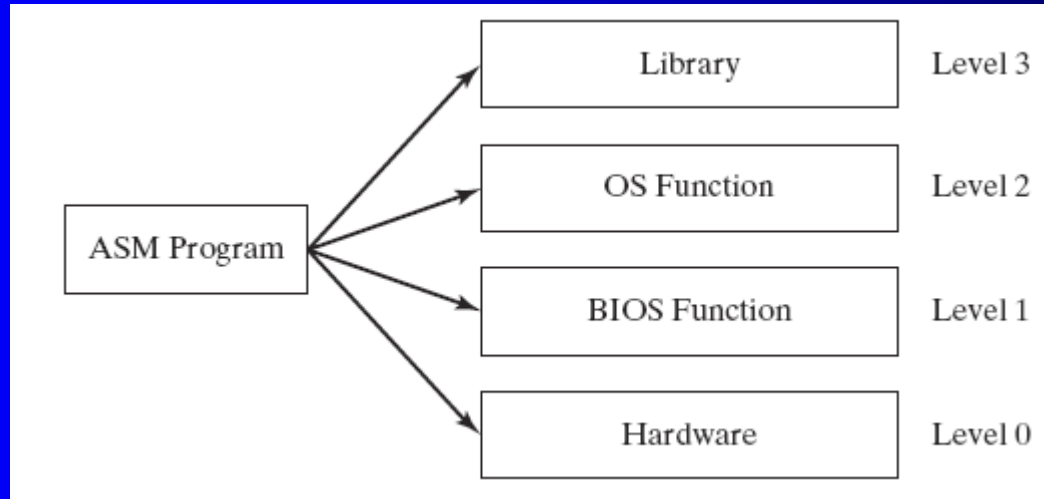
# Displaying a String of Characters

When a HLL program displays a string of characters, the following steps take place:



# Programming levels

Assembly language programs can perform input-output at each of the following levels:



# Summary

- Central Processing Unit (CPU)
- Arithmetic Logic Unit (ALU)
- Instruction execution cycle
- Multitasking
- Floating Point Unit (FPU)
- Complex Instruction Set
- Real mode and Protected mode
- Motherboard components
- Memory types
- Input/Output and access levels