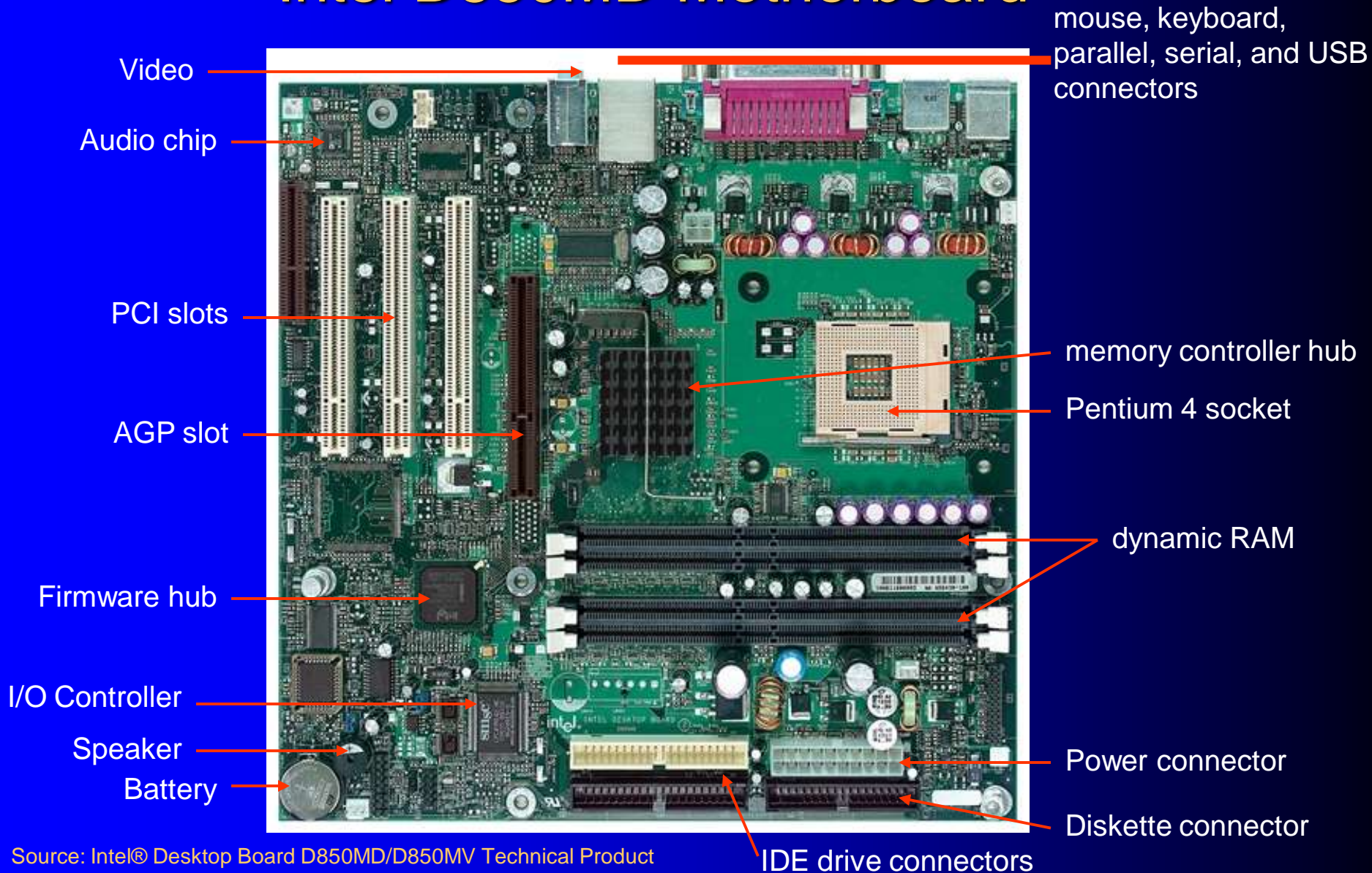


# x86 Processor Architecture

# Microcomputer

- Motherboard contains the main components
  - Microprocessor (the computational core)
  - Memory (DRAM)
  - Support chips (memory controller, audio, IO ...)
  - Slots for peripheral components (video, ...)
  - Input/output ports

# Intel D850MD Motherboard



Source: Intel® Desktop Board D850MD/D850MV Technical Product Specification

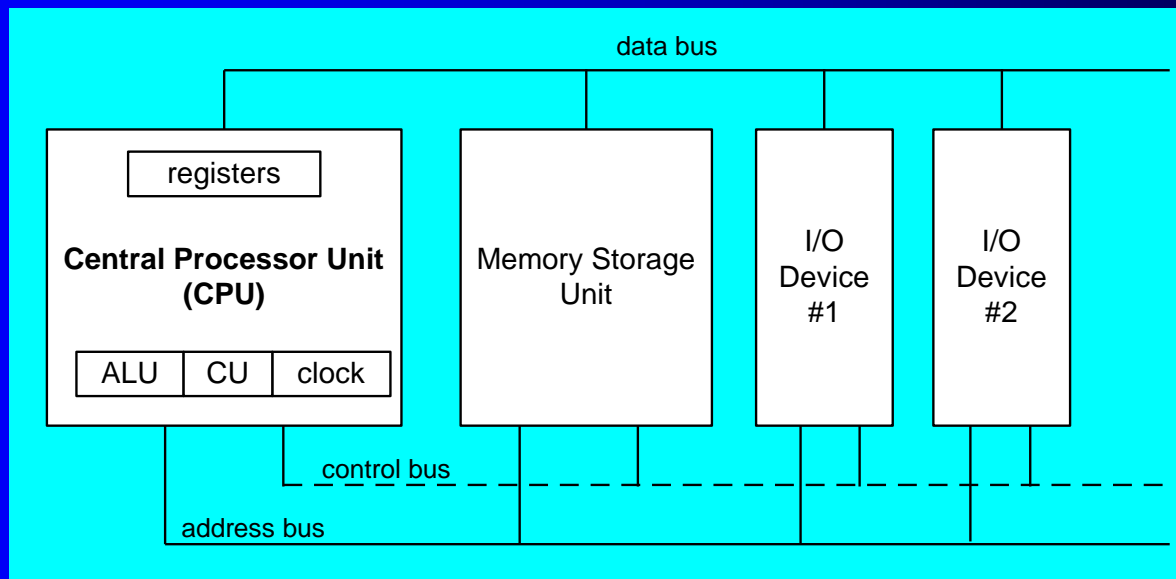
# Microprocessor

## Central Processing Unit (CPU)

- Arithmetic Logic Unit (ALU)
  - Performs the actual computations
- Control Unit (CU)
  - Controls the sequence of instruction execution
- Clock
  - Manages the timing of instruction execution
- Cache Memory
  - Helps speed program execution
- Registers
  - Localized storage of operands and data
  - Speeds execution by avoiding (much slower) memory access

# Basic Microcomputer Design

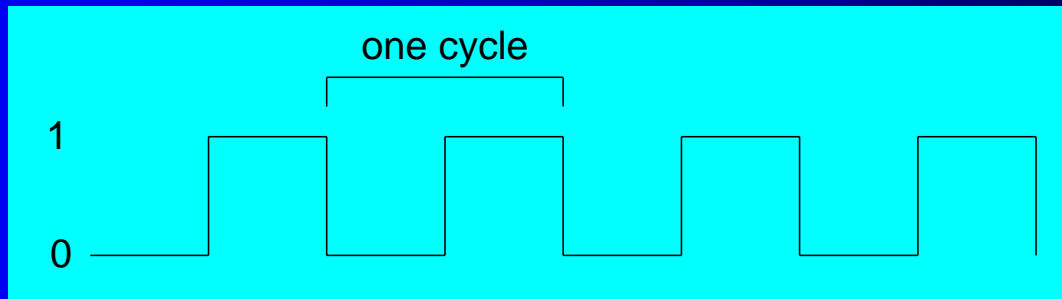
- clock synchronizes CPU operations
- control unit (CU) coordinates sequence of execution steps
- ALU performs arithmetic and bitwise processing



- buses carry multi-bit words representing memory addresses, data, or control information
  - typically 64 bits wide, 32 bits in older machines

# Clock

- Synchronizes all CPU and BUS operations
- Machine cycle (or clock cycle) measures time of a single operation
- Clock is used to trigger events
- Continuously running (normally)
  - Generated with a motherboard-mounted quartz crystal
  - Typically over 1Ghz (1 billion cycles per second)



# Instruction Execution Cycle

## 1. Fetch

- Load the next program instruction from the code cache
  - Code Cache: section of memory storing the program
  - Location of the next instruction is stored in the “instruction pointer”

## 2. Decode

- Interpret the instruction bit pattern (opcode) to determine the operation to perform (this is “machine Language”)

## 3. Fetch operands (if any)

- Retrieve any needed data from registers or memory

## 4. Execute

## 5. Store the output (if necessary)

## 6. Update the “instruction pointer” to the next instruction and go to step 1.



# Execution Cycle vs. Machine Cycles

Some instructions take multiple clock cycles (or machine cycles) to execute

- Actual number depends on source and destination of the data (register or memory), computer components and architecture ...
- Register based operations are faster than operations requiring memory access
  - Because registers are located right in the CPU
  - No need to go to external memory to get data



# x86 Family of Processors

## CISC architecture

- Complex Instruction Set Computer
- Supports both 32 and 64 bit operation\*

## Register Memory Machines

- Operands can be in either registers or memory
  - But operations using registers are faster

\*This course will concentrate on 32 Bit Operation

# x86 Overview

Supports multiple modes of operation:

- Protected mode\*
  - native mode (Windows, Linux)
- Others:
  - Real-address mode (native MS-DOS)
  - System management mode (power management, system security, diagnostics)
  - Virtual-8086 mode (variant of Protected)

\*This course will concentrate on Protected Mode

# 32 Bit Operation

The basic (default) sizes of the following are all 32 bits:

- CPU registers
  - Data
  - Memory addresses
- 
- 32 bit memory addresses implies up to 4GB of addressable memory ( $2^{32}$  is approximately  $4 \times 10^9$ )

# Registers, an overview

- Located in the CPU
- Registers provide rapid, direct access to operands in an instruction (like adding, comparing, ...)
  - Avoid accessing system bus and memory
- Some registers are general purpose
- ... but others are dedicated to particular tasks
- Assembly language programs make extensive use of registers
  - Registers are accessed by name in assembly language programs

# 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

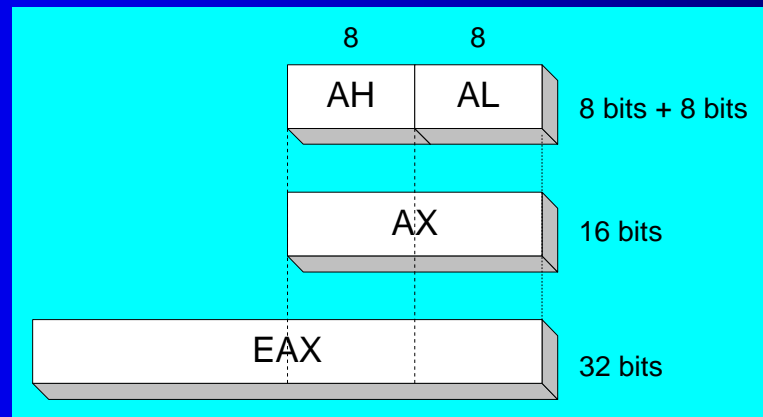
## 16-bit Segment Registers

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

\* can't access at the byte level



# Some Specialized Register Uses (1 of 2)

- General-Purpose Registers
  - EAX – accumulator (multiplication/division)
  - ECX – loop counter
  - ESP – stack pointer
  - ESI, EDI – index registers (high speed memory transfer)
  - EBP – extended frame pointer (stack)
- Segment Registers (used for pointers)
  - CS – code segment
  - DS – data segment
  - SS – stack segment
  - ES, FS, GS - additional segments

# Some Specialized Register Uses (2 of 2)

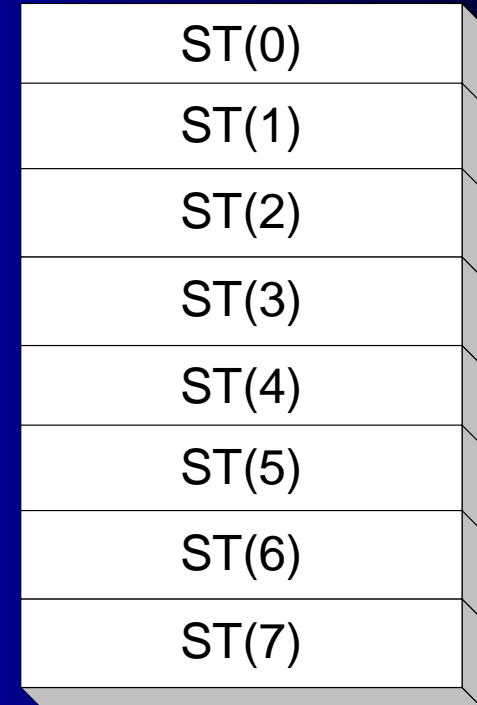
- EIP – instruction pointer
  - Contains the address of the next instruction to be executed
- EFLAGS
  - status and control flags
  - each flag is a single binary bit

# 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



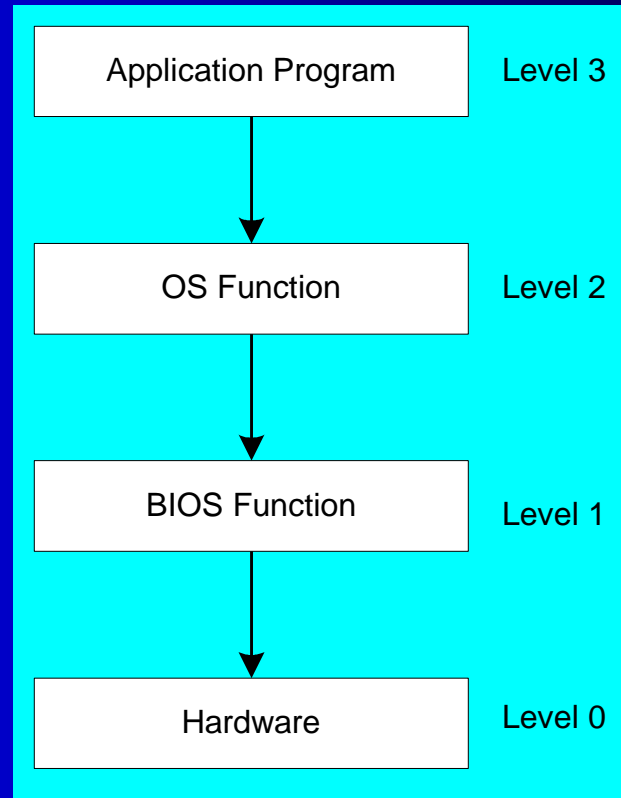
ST(0)
ST(1)
ST(2)
ST(3)
ST(4)
ST(5)
ST(6)
ST(7)

# 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
- Programs in this course will use Level-3 I/O functions provided in a library that come with the textbook

# 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:

