Chapter Overview

- General Concepts
- IA-32 Processor Architecture
- IA-32 Memory Management
- 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 [1/2]

- The central processor unit (CPU) is where all the calculations and logic operations take place.
- Clock synchronizes CPU operations
- Control unit (CU) coordinates sequence of execution steps
- Arithmetic logic unit (ALU) performs arithmetic and bitwise processing
Basic Microcomputer Design [2/2]

- The memory storage unit is where instructions and data are held while a computer program is running.
- A bus is a group of parallel wires that transfer data from one part of the computer to another.
  - Data bus, address bus and control bus
Clock

- Synchronizes all CPU and BUS operations
- Machine (clock) cycle measures time of a single operation
  - A machine instruction requires at least one clock cycle to execute.
  - A few instructions (e.g., the multiply instruction) require in excess of 50 clocks.
- The duration of a clock cycle is the reciprocal of the clock’s speed
- Clock is used to trigger events
Instruction Execution Cycle

- The execution of a single machine instruction can be divided into a sequence of individual operations.
- Three primary operations: fetch, decode and execute.
- Two more steps are required when the instruction uses a memory operand: fetch operand and store output operand.

- Fetch
- Decode
- Fetch operands
- Execute
- Store output
Multi-Stage Pipeline

- Pipelining makes it possible for a processor to execute instructions in parallel.
- Instruction execution divided into discrete stages.

Example of a non-pipelined processor. Many wasted cycles.
**Pipelined Execution**

- More efficient use of cycles, greater throughput of instructions:

For $k$ states and $n$ instructions, the number of required cycles is:

$$k + (n - 1)$$

<table>
<thead>
<tr>
<th>Cycles</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>I-2</td>
<td>I-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>I-2</td>
<td>I-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>I-2</td>
<td>I-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>I-2</td>
<td>I-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>I-2</td>
<td>I-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I-2</td>
</tr>
</tbody>
</table>
Wasted Cycles (pipelined)

- When one of the stages requires two or more clock cycles, clock cycles are again wasted.

For $k$ states and $n$ instructions, the number of required cycles is:

$$k + (2n - 1)$$
A superscalar processor has multiple execution pipelines. In the following, note that Stage S4 has left and right pipelines (u and v).

For $k$ states and $n$ instructions, the number of required cycles is:

$$k + n$$
Reading from Memory

- Multiple machine cycles are required when reading from memory, because it responds much more slowly than the CPU. The steps are:
  - address placed on address bus
  - Read Line (RD) set low to notify memory that a value is to be read
  - CPU waits one cycle for memory to respond. During this cycle, the memory controller places the operand on the data bus (DATA)
  - Read Line (RD) goes to 1, indicating that the data is on the data bus
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 [1/2]

User

sends program name to

Operating system

searches for program in

Current directory

gets starting cluster from

returns to

System path

loads and starts

Program

Directory entry
How a Program Runs [2/2]

• The user issues a command to run a certain program.
• The OS searches for the program’s filename (in the current directory or predetermined list of directories)
• The OS retrieves basic information about the program’s file from the disk directory.
• The OS loads the program file into memory.
• The CPU begins to execute the program (process).
Multitasking

- OS can run multiple tasks at the same time.
- Scheduler utility assigns a given amount of CPU time to each running program.
- Rapid switching of tasks
  - Gives illusion that all programs are running at once
  - Round-robin scheduling
  - The processor must support task switching.
    - The processor saves the state (e.g., registers, variables, program counter) of each task before switching to a new one.
- OS can assign varying priorities to tasks.
IA-32 Processor Architecture

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

- Protected mode
  - Programs are given separate memory areas (called segments)
  - Windows, Linux
- Real-address mode
  - Implements the programming environment of the Intel 8086 processor
  - Native MS-DOS
  - All Intel processors boot in Real-address mode
- System management mode
  - Provides an operating system with a mechanism for implementing
    - Power management, system security, diagnostics
  - Implemented by computer manufactures
- Virtual-8086 mode
  - Hybrid of Protected and Real-address modes
  - While in Protected mode, the processor can directly execute Real-address mode program in a safe multitasking environment.
  - 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
Registers

• Registers are high-speed storage locations directly inside the CPU
• Optimized for speed (e.g., loop processing)
Accessing Parts of Registers

- Used for arithmetic and data movement
- Use 8-bit name, 16-bit name, or 32-bit name
- Applies to EAX, EBX, ECX, and EDX

<table>
<thead>
<tr>
<th>32-bit</th>
<th>16-bit</th>
<th>8-bit (high)</th>
<th>8-bit (low)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>AX</td>
<td>AH</td>
<td>AL</td>
</tr>
<tr>
<td>EBX</td>
<td>BX</td>
<td>BH</td>
<td>BL</td>
</tr>
<tr>
<td>ECX</td>
<td>CX</td>
<td>CH</td>
<td>CL</td>
</tr>
<tr>
<td>EDX</td>
<td>DX</td>
<td>DH</td>
<td>DL</td>
</tr>
</tbody>
</table>
Index and Base Registers

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

<table>
<thead>
<tr>
<th>32-bit</th>
<th>16-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESI</td>
<td>SI</td>
</tr>
<tr>
<td>EDI</td>
<td>DI</td>
</tr>
<tr>
<td>EBP</td>
<td>BP</td>
</tr>
<tr>
<td>ESP</td>
<td>SP</td>
</tr>
</tbody>
</table>
Some Specialized Register Uses [1/2]

- General-Purpose
  - EAX – accumulator (used by multiplication and division instructions)
  - ECX – loop counter
  - ESP – stack pointer addresses data on the stack
  - ESI, EDI – source/destination index registers (for high-speed memory transfer)
  - EBP – extended frame pointer (used by high-level languages to reference function parameters and local variables)
Some Specialized Register [2/2]

- Segment (as base locations for pre-assigned memory areas)
  - CS – code segment
  - DS – data segment
  - SS – stack segment
  - ES, FS, GS - additional segments
- EIP – instruction pointer
  - Containing the address of the next instruction to be executed
- EFLAGS
  - Status and control flags
  - Control the operation of the CPU or reflect the outcome of some CPU operation
  - each flag is a single binary bit
Flags

• Control Flags
  • Direction
  • Interrupt

• 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 in an 8-bit operand
  • 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)
  • used for high-speed floating-point arithmetic

• Eight 64-bit MMX registers

• Eight 128-bit XMM registers for single-instruction multiple-data (SIMD) operations
Intel Microprocessor History

- Intel 8086
- Intel 80286
- IA-32 processor family
- P6 processor family
- CISC and RISC
Early Intel Microprocessors

- Intel 8086/8088
  - Mark the beginning of the modern Intel Architecture family
  - IBM-PC used 8088
  - 1 MB addressable RAM
  - 16-bit registers
  - 16-bit data bus (8-bit for 8088 – low-cost microcontroller)
  - separate floating-point unit (8087)
The IBM-PC/AT Computer

- Intel 80286
  - 16 MB addressable RAM
  - Protected memory
  - several times faster than 8086
  - 80287 floating point unit
- Downward Compatibility
Intel IA-32 Family

- Intel 386
  - 4 GB addressable RAM, 32-bit registers, paging (virtual memory)
- Intel 486
  - instruction pipelining
- Pentium
  - superscalar, 32-bit address bus, 64-bit internal data path
Intel P6 Family

• Based on a new micro-architecture design that improves execution speed (extension of the basic IA-32 architecture)
• Pentium Pro
  • Advanced optimization techniques in microcode
• Pentium II
  • MMX (multimedia) instruction set
• Pentium III
  • SIMD (streaming extensions) instructions with special 128-bit registers designed to move large amounts of data quickly
• Pentium 4
  • NetBurst micro-architecture, tuned for multimedia applications
CISC and RISC

- CISC – complex instruction set
  - large instruction set
  - high-level operations
  - High-level language compilers would have less work
  - requires microcode interpreter
  - Complex instructions require a long time for the processor to decode and execute
  - examples: Intel 80x86 family
- RISC – reduced instruction set
  - simple, atomic instructions (for pipelining)
  - small instruction set
  - directly executed by hardware
  - High-speed engineering and graphics workstation use RISC processors
  - examples:
    - ARM (Advanced RISC Machines)
    - DEC Alpha (now Compaq)
IA-32 Memory Management

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

- 1 MB RAM maximum addressable (00000~FFFFFh)
- Application programs can access any area of memory
- Single tasking
- Supported by MS-DOS operating system
Segmented Memory

- The original 8086 processor had only 16-bit registers, which cannot directly represent a 20-bit address.
- Segmented memory addressing: absolute (linear) address is a combination of a 16-bit segment value added to a 16-bit offset.
Calculating Linear Addresses

• Given a segment address, multiply it by 16 (add a hexadecimal zero), and add it to the offset
• Example: convert 08F1:0100 to a linear address

<table>
<thead>
<tr>
<th>Adjusted Segment value:</th>
<th>0 8 F 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add the offset:</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>Linear address:</td>
<td>0 9 0 1 0</td>
</tr>
</tbody>
</table>
Your turn . . .

What linear address corresponds to the segment/offset address 028F:0030?

028F0 + 0030 = 02920

Always use hexadecimal notation for addresses.
Your turn . . .

What segment addresses correspond to the linear address 28F30h?

Many different segment-offset addresses can produce the linear address 28F30h. For example:

28F0:0030, 28F3:0000, 28B0:0430, . . .
Protected Mode

- 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
Flat Segmentation Model

- All segments are mapped to the entire 32-bit physical address space of the computer.
  - At least two segments: one for program code and one for data
- Each segment is defined by a segment descriptor, a 64-bit value stored in a table known as the global descriptor table (GDT)
Multi-Segment Model

- Each program has a local descriptor table (LDT)
  - holds descriptor for each segment used by the program

```
Local Descriptor Table

<table>
<thead>
<tr>
<th>base</th>
<th>limit</th>
<th>access</th>
</tr>
</thead>
<tbody>
<tr>
<td>00026000</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>00008000</td>
<td>000A</td>
<td></td>
</tr>
<tr>
<td>00003000</td>
<td>0002</td>
<td></td>
</tr>
</tbody>
</table>
```

Paging

- Supported directly by the CPU
- Divides each segment into 4096-byte blocks called pages
- Sum of all programs can be larger than physical memory
- Part of running program is in memory, part is on disk
- Virtual memory manager (VMM) – OS utility that manages the loading and unloading of pages
- Page fault – issued by CPU when a page must be loaded from disk
Components of an IA-32 Microcomputer

- Motherboard
- Video output
- Memory
- Input-output ports
Motherboard

- The heart of a microcomputer
- CPU socket
- External cache memory slots
- Main memory slots
- BIOS (a collection of functions that communicate directly with hardware devices) chips
- Sound synthesizer chip (optional)
- Video controller chip (optional)
- IDE (hard and CD-ROM drives), parallel, serial, USB
- PCI bus connectors (expansion cards)
Intel D850MD Motherboard

- Video chip
- Audio chip
- PCI slots
- Firmware hub
- I/O Controller
- Speaker
- Battery
- IDE drive connectors
- Mouse, keyboard, parallel, serial, and USB connectors
- Memory controller hub
- Intel 486 socket
- Dynamic RAM
- Power connector
- Diskette connector

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

Video Output

- Video controller
  - on motherboard, or on expansion card
  - A special-purpose microcomputer, relieving the primary CPU of the job of controlling video hardware
- Video memory (VRAM)
- Video CRT Display
  - uses raster scanning
  - horizontal retrace
  - vertical retrace
- Direct digital LCD monitors
  - no raster scanning required
Memory

- **ROM**
  - read-only memory (permanently burned into a chip)
- **EPROM**
  - erasable programmable read-only memory (be erased slowly with ultraviolet light)
- **Dynamic RAM (DRAM)**
  - inexpensive; must be refreshed constantly
- **Static RAM (SRAM)**
  - expensive; used for cache memory; no refresh required
- **Video RAM (VRAM)**
  - dual ported (refreshing+writing); optimized for constant video refresh
  - Allocated on a video controller
- **CMOS RAM**
  - system setup information
  - On the system motherboard
Input-Output Ports [1/2]

• USB (universal serial bus)
  • intelligent high-speed connection to devices
  • up to 12 megabits/second
  • USB hub connects multiple devices
  • *enumeration*: computer queries devices

• Parallel
  • short cable (less than 10 feet), high speed
  • common for printers
  • bidirectional, parallel data transfer (8 or 16 bits simultaneously)
**Input-Output Ports [2/2]**

- **Serial**
  - RS-232 serial port
  - one bit at a time
  - uses long cables and modems
  - 16550 UART (Universal Asynchronous Receiver Transmitter) controls the serial ports.
    - Located either on the motherboard or on an adapter card
Levels of Input-Output

- Level 3: Call a library function (C++, Java)
  - easy to do; abstracted from hardware; details hidden
  - slowest performance
- Level 2: Call an operating system function
  - specific to one OS; device-independent
    - E.g., writing entire strings to files, reading string from the keyboard, allocating blocks of memory for application programs
  - medium performance
- Level 1: Call a BIOS (basic input-output system) function
  - may produce different results on different systems
  - knowledge of hardware required
  - usually good performance
Displaying a String of Characters

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

1. Application program writes the string to standard output.
2. The library function calls OS, passing a string pointer.
3. OS passes the ASCII code and color of each character to BIOS. OS also calls BIOS function to control the cursor.
4. BIOS maps each character to a particular system font and sends it to a hardware port attached to the video controller card.
5. The video controller card generates hardware signals to the video display.
ASM Programming levels

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

- Level 0: Hardware
- Level 1: BIOS Function
- Level 2: OS Function
Playing a WAV file

• At the OS level, you do not have to know what type of device was installed and the card’s features.
• At the BIOS level, you would query the sound card and find out whether it belongs to a certain class of sound cards.
• At the hardware level, you would fine-tune the program for certain brands of audio cards, to take advantage of each card’s special features.
  • Not all operating system permit user programs to directly access system hardware.