Intel x86 Architecture

Computer Organization and Assembly Languages
Yung-Yu Chuang
2008/12/8

with slides by Kip Irvine

Early Intel microprocessors

- Intel 8080 (1972)
  - 64K addressable RAM
  - 8-bit registers
  - CP/M operating system
  - 5,6,8,10 MHz
  - 29K transistors

- Intel 8086/8088 (1978)
  - IBM-PC used 8088
  - 1 MB addressable RAM
  - 16-bit registers
  - 16-bit data bus (8-bit for 8088)
  - separate floating-point unit (8087)
  - used in low-cost microcontrollers now

- Intel 80286 (1982)
  - 16 MB addressable RAM
  - Protected memory
  - several times faster than 8086
  - introduced IDE bus architecture
  - 80287 floating point unit
  - Up to 20MHz
  - 134K transistors

The IBM-AT

- Intel 80286 (1982)
  - 16 MB addressable RAM
  - Protected memory
  - several times faster than 8086
  - introduced IDE bus architecture
  - 80287 floating point unit
  - Up to 20MHz
  - 134K transistors
**Intel IA-32 Family**

- Intel386 (1985)
  - 4 GB addressable RAM
  - 32-bit registers
  - paging (virtual memory)
  - Up to 33MHz
- Intel486 (1989)
  - instruction pipelining
  - Integrated FPU
  - 8K cache
- Pentium (1993)
  - Superscalar (two parallel pipelines)

**Intel P6 Family**

- Pentium Pro (1995)
  - advanced optimization techniques in microcode
  - More pipeline stages
  - On-board L2 cache
- Pentium II (1997)
  - MMX (multimedia) instruction set
  - Up to 450MHz
- Pentium III (1999)
  - SIMD (streaming extensions) instructions (SSE)
  - Up to 1+GHz
- Pentium 4 (2000)
  - NetBurst micro-architecture, tuned for multimedia
  - 3.8+GHz
- Pentium D (2005, Dual core)

**IA32 Processors**

- Totally Dominate Computer Market
- Evolutionary Design
  - Starting in 1978 with 8086
  - Added more features as time goes on
  - Still support old features, although obsolete
- Complex Instruction Set Computer (CISC)
  - Many different instructions with many different formats
    - But, only small subset encountered with Linux programs
  - Hard to match performance of Reduced Instruction Set Computers (RISC)
  - But, Intel has done just that!
IA-32 architecture

- Lots of architecture improvements, pipelining, superscalar, branch prediction, hyperthreading and multi-core.
- From programmer’s point of view, IA-32 has not changed substantially except the introduction of a set of high-performance instructions.

Modes of operation

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

Addressable memory

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

General-purpose registers

32-bit General-Purpose Registers

<table>
<thead>
<tr>
<th>EAX</th>
<th>EBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBX</td>
<td>ESP</td>
</tr>
<tr>
<td>ECX</td>
<td>ESI</td>
</tr>
<tr>
<td>EDX</td>
<td>EDI</td>
</tr>
</tbody>
</table>

16-bit Segment Registers

<table>
<thead>
<tr>
<th>EFLAGS</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ES</td>
</tr>
<tr>
<td></td>
<td>FS</td>
</tr>
<tr>
<td></td>
<td>GS</td>
</tr>
</tbody>
</table>
Accessing parts of registers

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

Index and base registers

- Some registers have only a 16-bit name for their lower half (no 8-bit aliases). The 16-bit registers are usually used only in real-address mode.

<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 of 2)

- General-Purpose
  - EAX – accumulator (automatically used by division and multiplication)
  - ECX – loop counter
  - ESP – stack pointer (should never be used for arithmetic or data transfer)
  - ESI, EDI – index registers (used for high-speed memory transfer instructions)
  - EBP – extended frame pointer (stack)

Some specialized register uses (2 of 2)

- Segment
  - CS – code segment
  - DS – data segment
  - SS – stack segment
  - ES, FS, GS - additional segments
- EIP – instruction pointer
- EFLAGS
  - status and control flags
  - each flag is a single binary bit (set or clear)
- Some other system registers such as IDTR, GDTR, LDTR etc.
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

Programmer's model

Basic Program Execution Registers

<table>
<thead>
<tr>
<th>Eight 32-bit Registers</th>
<th>General-Purpose Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Six 16-bit Registers</td>
<td>EFLAGS Register</td>
</tr>
<tr>
<td>32-bits</td>
<td>EIP (Instruction Pointer Register)</td>
</tr>
<tr>
<td>32-bits</td>
<td></td>
</tr>
</tbody>
</table>

FPU Registers

<table>
<thead>
<tr>
<th>Eight 80-bit Registers</th>
<th>Floating-Point Data Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bits</td>
<td>Control Register</td>
</tr>
<tr>
<td>16 bits</td>
<td>Status Register</td>
</tr>
<tr>
<td>16 bits</td>
<td>Tag Register</td>
</tr>
</tbody>
</table>

Address Space

\[ 2^{31} - 1 \]

*The address space can be flat or segmented. Using the physical address extension mechanism, a physical address space of \( 2^{31} - 1 \) can be addressed.*

Programmer's model

<table>
<thead>
<tr>
<th>Opcode Register (11-bits)</th>
<th>FPU Instruction Pointer Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>46 bits</td>
<td>46 bits</td>
</tr>
</tbody>
</table>

MMX Registers

<table>
<thead>
<tr>
<th>Eight 64-bit Registers</th>
<th>MMX Registers</th>
</tr>
</thead>
</table>

XMM Registers

<table>
<thead>
<tr>
<th>Eight 128-bit Registers</th>
<th>XMM Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bits</td>
<td>MXCSR Register</td>
</tr>
</tbody>
</table>
IA-32 Memory Management

Real-address mode

- 1 MB RAM maximum addressable (20-bit address)
- Application programs can access any area of memory
- Single tasking
- Supported by MS-DOS operating system

Segmented memory

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

\[
\text{Adjusted Segment value: } 0 \ 8 \ F \ 1 \ 0 \\
\text{Add the offset: } \quad 0 \ 1 \ 0 \ 0 \\
\text{Linear address: } \quad 0 \ 9 \ 0 \ 1 \ 0
\]

- A typical program has three segments: code, data and stack. Segment registers CS, DS and SS are used to store them separately.
Example

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

\[ 028F0 + 0030 = 02920 \]

Always use hexadecimal notation for addresses.

Protected mode (1 of 2)

- 4 GB addressable RAM (32-bit address)
  - (00000000 to FFFFFFFFh)
- Each program assigned a memory partition which is protected from other programs
- Designed for multitasking
- Supported by Linux & MS-Windows

Protected mode (2 of 2)

- Segment descriptor tables
- Program structure
  - code, data, and stack areas
  - CS, DS, SS segment descriptors
  - global descriptor table (GDT)
- MASM Programs use the Microsoft flat memory model

Flat segmentation model

- All segments are mapped to the entire 32-bit physical address space, at least two, one for data and one for code
- global descriptor table (GDT)
Multi-segment model

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

---

Translating Addresses

- The IA-32 processor uses a one- or two-step process to convert a variable’s logical address into a unique memory location.
- The first step combines a segment value with a variable’s offset to create a linear address.
- The second optional step, called page translation, converts a linear address to a physical address.

---

Converting Logical to Linear Address

The segment selector points to a segment descriptor, which contains the base address of a memory segment. The 32-bit offset from the logical address is added to the segment’s base address, generating a 32-bit linear address.

---

Indexing into a Descriptor Table

Each segment descriptor indexes into the program’s local descriptor table (LDT). Each table entry is mapped to a linear address:
**Paging (1 of 2)**

- Virtual memory uses disk as part of the memory, thus allowing sum of all programs can be larger than physical memory.
- Only part of a program must be kept in memory, while the remaining parts are kept on disk.
- The memory used by the program is divided into small units called pages (4096-byte).
- As the program runs, the processor selectively unloads inactive pages from memory and loads other pages that are immediately required.

**Paging (2 of 2)**

- OS maintains page directory and page tables.
- Page translation: CPU converts the linear address into a physical address.
- Page fault: occurs when a needed page is not in memory, and the CPU interrupts the program.
- Virtual memory manager (VMM) – OS utility that manages the loading and unloading of pages.
- OS copies the page into memory, program resumes execution.

---

**Page Translation**

A linear address is divided into a page directory field, page table field, and page frame offset. The CPU uses all three to calculate the physical address.