CMSC 313 Lecture 04

- Homework 2
- IA-32 Basic Execution Environment
- IA-32 General Purpose Registers
- Moore’s “Law”
- Evolution of the Pentium Chip
- “Hello World” in Linux Assembly Language
- Addressing Modes
Due: Tuesday, September 21, 2004

Instructions: For the following questions, show all of your work. It is not sufficient to provide the answers.

Exercise 1. Convert the following decimal numbers to hexadecimal representations of 16-bit two’s complement numbers.
   a. 1293
   b. 31249
   c. -24752
   d. -4096

Exercise 2. Convert the following 16-bit two’s complement numbers in hexadecimal representation to decimal.
   a. FFF5\text{\textsubscript{16}}
   b. 7CD9\text{\textsubscript{16}}
   c. 00BB\text{\textsubscript{16}}
   d. 8000\text{\textsubscript{16}}

Exercise 3. Write the following decimal numbers in IEEE-754 single precision format. Give your answers in binary.
   a. 14.125
   b. 3.14159
   c. $-58.375$
   d. $-4096$

Exercise 4. Write the decimal equivalents for these IEEE-754 single precision floating point numbers given in binary.
   a. 0 10000001 011000000000000000000000
   b. 1 10000001 000100000000000000000000
   c. 1 1000000 000000000000000000000000
   d. 0 00000001 010110000000000000000000
The System Bus Model

- A refinement of the von Neumann model, the system bus model has a CPU (ALU and control), memory, and an input/output unit.
- Communication among components is handled by a shared pathway called the *system bus*, which is made up of the data bus, the address bus, and the control bus. There is also a power bus, and some architectures may also have a separate I/O bus.
The Fetch-Execute Cycle

• The steps that the control unit carries out in executing a program are:
  (1) Fetch the next instruction to be executed from memory.
  (2) Decode the opcode.
  (3) Read operand(s) from main memory, if any.
  (4) Execute the instruction and store results.
  (5) Go to step 1.

This is known as the *fetch-execute cycle*. 
The address space can be flat or segmented. Using the physical address extension mechanism, a physical address space of $2^{36} - 1$ can be addressed.
3.4.2. Segment Registers

The segment registers (CS, DS, SS, ES, FS, and GS) hold 16-bit segment selectors. A segment selector is a special pointer that identifies a segment in memory. To access a particular segment in memory, the segment selector for that segment must be present in the appropriate segment register.

When writing application code, programmers generally create segment selectors with assembler directives and symbols. The assembler and other tools then create the actual segment selector values associated with these directives and symbols. If writing system code, programmers may need to create segment selectors directly. (A detailed description of the segment-selector data structure is given in Chapter 3, Protected-Mode Memory Management, of the Intel Architecture Software Developer’s Manual, Volume 3.)

How segment registers are used depends on the type of memory management model that the operating system or executive is using. When using the flat (unsegmented) memory model, the segment registers are loaded with segment selectors that point to overlapping segments, each of which begins at address 0 of the linear address space (as shown in Figure 3-5). These overlapping segments then comprise the linear address space for the program. (Typically, two overlapping segments are defined: one for code and another for data and stacks. The CS segment register points to the code segment and all the other segment registers point to the data and stack segment.)

When using the segmented memory model, each segment register is ordinarily loaded with a different segment selector so that each segment register points to a different segment within the linear address space (as shown in Figure 3-6). At any time, a program can thus access up to six segments in the linear address space. To access a segment not pointed to by one of the segment registers, a program must first load the segment selector for the segment to be accessed into a segment register.
**3.4.1. General-Purpose Registers**

The 32-bit general-purpose registers EAX, EBX, ECX, EDX, ESI, EDI, EBP, and ESP are provided for holding the following items:

- Operands for logical and arithmetic operations
- Operands for address calculations
- Memory pointers.

Although all of these registers are available for general storage of operands, results, and pointers, caution should be used when referencing the ESP register. The ESP register holds the stack pointer and as a general rule should not be used for any other purpose.

Many instructions assign specific registers to hold operands. For example, string instructions use the contents of the ECX, ESI, and EDI registers as operands. When using a segmented memory model, some instructions assume that pointers in certain registers are relative to specific segments. For instance, some instructions assume that a pointer in the EBX register points to a memory location in the DS segment.

The special uses of general-purpose registers by instructions are described in Chapter 5, *Instruction Set Summary*, in this volume and Chapter 3, *Instruction Set Reference*, in the *Intel Architecture Software Developer’s Manual, Volume 2*. The following is a summary of these special uses:

- **EAX**—Accumulator for operands and results data.
- **EBX**—Pointer to data in the DS segment.
- **ECX**—Counter for string and loop operations.
- **EDX**—I/O pointer.
- **ESI**—Pointer to data in the segment pointed to by the DS register; source pointer for string operations.
- **EDI**—Pointer to data (or destination) in the segment pointed to by the ES register; destination pointer for string operations.
- **ESP**—Stack pointer (in the SS segment).
- **EBP**—Pointer to data on the stack (in the SS segment).

As shown in Figure 3-4, the lower 16 bits of the general-purpose registers map directly to the register set found in the 8086 and Intel 286 processors and can be referenced with the names AX, BX, CX, DX, BP, SP, SI, and DI. Each of the lower two bytes of the EAX, EBX, ECX, and EDX registers can be referenced by the names AH, BH, CH, and DH (high bytes) and AL, BL, CL, and DL (low bytes).
Table 2-2. Key Features of Previous Generations of IA-32 Processors

<table>
<thead>
<tr>
<th>Intel Processor</th>
<th>Date Introduced</th>
<th>Max. Clock Frequency at Introduction</th>
<th>Transistors per Die</th>
<th>Register Sizes</th>
<th>Ext. Data Bus Size</th>
<th>Max. External Addr. Space</th>
<th>Caches</th>
</tr>
</thead>
<tbody>
<tr>
<td>8086</td>
<td>1978</td>
<td>8 MHz</td>
<td>29 K</td>
<td>16 GP</td>
<td>16</td>
<td>1 MB</td>
<td>None</td>
</tr>
<tr>
<td>Intel 286</td>
<td>1982</td>
<td>12.5 MHz</td>
<td>134 K</td>
<td>16 GP</td>
<td>16</td>
<td>18 MB</td>
<td>Note 3</td>
</tr>
<tr>
<td>Intel386 DX Processor</td>
<td>1985</td>
<td>20 MHz</td>
<td>275 K</td>
<td>32 GP</td>
<td>32</td>
<td>4 GB</td>
<td></td>
</tr>
<tr>
<td>Intel486 DX Processor</td>
<td>1989</td>
<td>25 MHz</td>
<td>1.2 M</td>
<td>32 GP</td>
<td>32</td>
<td>4 GB</td>
<td>L1: 8 KB</td>
</tr>
<tr>
<td>Pentium Processor</td>
<td>1993</td>
<td>60 MHz</td>
<td>3.1 M</td>
<td>32 GP</td>
<td>64</td>
<td>4 GB</td>
<td></td>
</tr>
<tr>
<td>Pentium Pro Processor</td>
<td>1995</td>
<td>200 MHz</td>
<td>5.5 M</td>
<td>32 GP</td>
<td>64</td>
<td>L1: 16 KB</td>
<td>L1: 16 KB</td>
</tr>
<tr>
<td>Pentium II Processor</td>
<td>1997</td>
<td>266 MHz</td>
<td>7 M</td>
<td>32 GP</td>
<td>64</td>
<td>L1: 32 KB</td>
<td>L1: 32 KB, L2: 256 KB or 512 KB</td>
</tr>
<tr>
<td>Pentium III Processor</td>
<td>1999</td>
<td>500 MHz</td>
<td>8.2 M</td>
<td>32 GP</td>
<td>64</td>
<td>L1: 32 KB</td>
<td>L1: 32 KB, L2: 512 KB</td>
</tr>
<tr>
<td>Pentium III and Pentium III Xeon Processors</td>
<td>1999</td>
<td>700 MHz</td>
<td>28 M</td>
<td>32 GP</td>
<td>64</td>
<td>L1: 32 KB, L2: 256 KB</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:**

1. The register size and external data bus size are given in bits. Note also that each 32-bit general-purpose (GP) registers can be addressed as an 8- or a 16-bit data registers in all of the processors.

2. Internal data paths are 2 to 4 times wider than the external data bus for each processor.
The key features of the Intel Pentium 4 processor, Intel Xeon processor, Intel Xeon processor MP, Pentium III processor, and Pentium III Xeon processor with advanced transfer cache are shown in Table 2-1. Older generation IA-32 processors, which do not employ on-die Level 2 cache, are shown in Table 2-2.

<table>
<thead>
<tr>
<th>Intel Processor</th>
<th>Date Introduced</th>
<th>Microarchitecture</th>
<th>Clock Frequency at Introduction</th>
<th>Transistors Per Die</th>
<th>Register Sizes</th>
<th>System Bus Bandwidth</th>
<th>Max. Extern. Addr. Space</th>
<th>On-Die Caches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium 4</td>
<td>2000</td>
<td>NetBurst Microarchitecture</td>
<td>1.50 GHz</td>
<td>42 M</td>
<td>GP: 32 FPU: 80 MMX: 64 XMM: 128</td>
<td>3.2 GB/s</td>
<td>64 GB</td>
<td>12K L0 pop Execution Cache; 8KB L1; 256-KB L2</td>
</tr>
<tr>
<td>Intel Xeon</td>
<td>2001</td>
<td>NetBurst Microarchitecture</td>
<td>1.70 GHz</td>
<td>42 M</td>
<td>GP: 32 FPU: 80 MMX: 64 XMM: 128</td>
<td>3.2 GB/s</td>
<td>64 GB</td>
<td>12K L0 pop Trace Cache; 8-KB L1; 256-KB L2</td>
</tr>
<tr>
<td>Processor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel Xeon</td>
<td>2002</td>
<td>NetBurst Microarchitecture; Hyper-Threading Technology</td>
<td>2.20 GHz</td>
<td>55 M</td>
<td>GP: 32 FPU: 80 MMX: 64 XMM: 128</td>
<td>3.2 GB/s</td>
<td>64 GB</td>
<td>12K L0 pop Trace Cache; 8-KB L1; 512-KB L2</td>
</tr>
<tr>
<td>Processor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel Xeon</td>
<td>2002</td>
<td>NetBurst Microarchitecture; Hyper-Threading Technology</td>
<td>1.60 GHz</td>
<td>108 M</td>
<td>GP: 32 FPU: 80 MMX: 64 XMM: 128</td>
<td>3.2 GB/s</td>
<td>64 GB</td>
<td>12K L0 pop Trace Cache; 8-KB L1; 256-KB L2, 1-MB L3</td>
</tr>
<tr>
<td>Processor MP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel Pentium 4</td>
<td>2002</td>
<td>NetBurst Microarchitecture; Hyper-Threading Technology</td>
<td>3.06 GHz</td>
<td>55 M</td>
<td>GP: 32 FPU: 80 MMX: 64 XMM: 128</td>
<td>4.2 GB/s</td>
<td>64 GB</td>
<td>12K L0 pop Execution Cache; 8-KB L1; 512-KB L2</td>
</tr>
<tr>
<td>Processor with Hyper-Threading Technology</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel Pentium M</td>
<td>2003</td>
<td>Pentium M Processor</td>
<td>1.60 GHz</td>
<td>77 M</td>
<td>GP: 32 FPU: 80 MMX: 64 XMM: 128</td>
<td>3.2 GB/s</td>
<td>4 GB</td>
<td>L1: 64 KB L2: 1 MB</td>
</tr>
<tr>
<td>Processor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2-16  Vol. 1
### Table 2-1. Key Features of Most Recent IA-32 Processors (Contd.)

<table>
<thead>
<tr>
<th>Intel Processor</th>
<th>Date Introduced</th>
<th>Microarchitecture</th>
<th>Clock Frequency at Introduction</th>
<th>Transistors Per Die</th>
<th>Register Sizes&lt;sup&gt;1&lt;/sup&gt;</th>
<th>System Bus Bandwidth</th>
<th>Max. External Addr. Space</th>
<th>On-Die Caches&lt;sup&gt;2&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Pentium 4 Processor Supporting Hyper-Threading Technology at 90 nanometer process</td>
<td>2004</td>
<td>Intel NetBurst Microarchitecture; Hyper-Threading Technology</td>
<td>3.40 GHz</td>
<td>125 M</td>
<td>GP: 32 FPU: 60 MMX: 64 XMM: 128</td>
<td>6.4 GB/s</td>
<td>64 GB</td>
<td>12K μop Execution Trace Cache; 16 KB L1; 1 MB L2</td>
</tr>
<tr>
<td>Intel Pentium M Processor 755&lt;sup&gt;3&lt;/sup&gt;</td>
<td>2004</td>
<td>Intel Pentium M Processor</td>
<td>2.00 GHz</td>
<td>140 M</td>
<td>GP: 32 FPU: 60 MMX: 64 XMM: 128</td>
<td>3.2 GB/s</td>
<td>4 GB</td>
<td>L1: 64 KB L2: 2MB</td>
</tr>
</tbody>
</table>

### NOTES

1. The register size and external data bus size are given in bits.
2. First level cache is denoted using the abbreviation L1. 2nd level cache is denoted as L2. The size of L1 includes the first-level data cache and the instruction cache where applicable, but does not include the trace cache.
Moore’s “Law”

• In the mid-1960’s, Intel Chairman of the Board Gordon Moore observed that “the number of transistors that would be incorporated on a silicon die would double every 18 months for the next several years.”

• His prediction has continued to hold true.

• Perhaps a self-fulfilling prophecy?
“Hello World” in Linux Assembly

- Use your favorite UNIX editor (vi, emacs, pico, ...)
- Assemble using NASM on gl.umbc.edu
  
  nasm -f elf hello.asm

- NASM documentation is on-line.

- Need to “load” the object file
  
  ld hello.o

- Execute
  
  a.out

- CMSC 121 Introduction to UNIX
; Linux style "fast call". Assemble using NASM

SECTION .data ; Data section

msg:    db "Hello, world", 10 ; The string to print.
len:    equ $-msg

SECTION .text ; Code section.
global _start

_start: nop ; Entry point.

mov    edx, len ; Arg 3: length of string.
mov    ecx, msg ; Arg 2: pointer to string.
mov    ebx, 1 ; Arg 1: file descriptor.
mov    eax, 4 ; Write.
int    080H

mov    ebx, 0 ; exit code, 0=normal
mov    eax, 1 ; Exit.
int    080H ; Call kernel.
80x86 Addressing Modes

• We want to store the value 1734h.
• The value 1734h may be located in a register or in memory.
• The location in memory might be specified by the code, by a register, …
• Assembly language syntax for MOV

    MOV      DEST, SOURCE
### Addressing Modes

<table>
<thead>
<tr>
<th>Register</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td></td>
</tr>
<tr>
<td>EBX</td>
<td></td>
</tr>
<tr>
<td>ECX</td>
<td>1734</td>
</tr>
<tr>
<td>EDX</td>
<td></td>
</tr>
<tr>
<td>EBP</td>
<td></td>
</tr>
<tr>
<td>ESI</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td></td>
</tr>
<tr>
<td>ESP</td>
<td></td>
</tr>
</tbody>
</table>

### Register from Register

**MOV EAX, ECX**
Register from Register Indirect

MOV EAX, [ECX]
Addressing Modes

Register from Memory

MOV EAX, [08A94068]
MOV EAX, [x]
Register from Immediate

MOV EAX, 1734
Register Indirect from Immediate

MOV [EAX], DWORD 1734
Addressing Modes

| EAX | EBX | ECX | EDX | EBP | ESI | EDI | ESP |

Memory from Immediate

MOV [08A94068], DWORD 1734
MOV [x], DWORD 1734
Notes on Addressing Modes

• More complicated addressing modes later:

   MOV EAX, [ESI+4*ECX+12]

• Figures not drawn to scale. Constants 1734h and 08A94068h take 4 bytes (little endian).

• Some addressing modes are not supported by some operations.

• Labels represent addresses not contents of memory.
Next Time

• Overview of i386 instruction set.
• Arithmetic instructions, logical instructions.
• EFLAGS register