Reviewing (1): Input/Output

• **I/O Device**
  – Each I/O device consists of two parts: 1) controller; 2) I/O device itself
  – Controller is used to control its I/O device and handle bus access for it

• **Direct Memory Access**
  – A controller reads or writes data to or from memory without CPU intervention
  – Controller is used to control its I/O device and handle bus access for it

• **Interrupt**
  – When the transfer is completed, the controller normally causes an interrupt, forcing
    the CPU to immediately suspend running its current program and start running a
    special procedures, called an interrupt handler to check for errors, take any special
    action needed, and inform the operating system that I/O is now finished.
  – When the interrupt handle is finished, the CPU continues with the program that
    was suspended when the interrupt occurred
Reviewing (2): Logical Structure

Logical structure of a simple personal computer.

- **Bus**
  - Used by the I/O controller and the CPU for fetching instructions and data

- **Bus Arbiter**
  - Decide who go first when the CPU and I/O controller want to use the bus at the same time; I/O devices are given preference over the CPU
Reviewing (3): Buses

A typical modern PC with a PCI bus and an ISA bus.

- **ISA vs. PCI Bus**
  - Industry Standard Architecture vs. Peripheral Component Interconnect
  - CPU memory traffic does not go over the PCI bus, high-bandwidth peripherals can connect to the PCI bus directly
Reviewing (4): Terminals

• Keyboards
  – On personal computer, when a key is depressed, an interrupt is generated and the keyboard interrupt handler is started.
  – The interrupt handler reads a hardware register inside the keyboard controller to get the number of the key that was just depressed.
  – When a key is released, a second interrupt is caused.

• Monitors
  – CRT (Cathode Ray Tube) monitors
  – Flat Panel Displays: LCD (Liquid Crystal Display) monitors
  – Video RAM: both displays are refreshed 60-100 times per second from a special memory --- video RAM
Reviewing (5): Video RAM

- **Video RAM**
  - A special memory for displays to refresh 60-100 times per second
  - On the display’s controller card
  - Contain 128 x 128 values for a picture with size of 128 x 128, one for each pixel

- **Problem**
  - A video RAM with 1600 x 1200 pixels at 3 bytes per pixel, what is the size for the video RAM at least to store the image?
    - $1600 \times 1200 \times 3 = 5,760,000$ bytes $= 5.5$ MB
  - If display redraw 60 time a second, how long is the pulse corresponding to one pixel?
    - $1/(1600 \times 1200 \times 60) = 8.68$ ns
Reviewing (6): Mice

- **Design Purpose**
  - Point to the menu items
  - By people with less expertise in computers
  - When it moves, a little pointer on the screen moves too, allow users to point at screen item

- **Three Kinds**
  - Mechanical mice: wheel and ball
  - Optical mice: no wheel and ball but LED (Light Emitting Diode)
  - Optomechanical mice: newer mechanical mice with rolling ball
Reviewing (7): Cable vs. ASDL

Frequency allocation in a typical cable TV system used for Internet access

- **Different from ASDL**
  - Headends are connected to main office by high-bandwidth cable
  - Each headend has more cables that run from it past many homes or offices
  - While ASDL service is constant, the cable service not
Multilevel Machines

Level 5
- Problem-oriented language level
  - Translation (compiler)

Level 4
- Assembly language level
  - Translation (assembler)

Level 3
- Operating system machine level
  - Partial interpretation (operating system)

Level 2
- Instruction set architecture level
  - Interpretation (microprogram) or direct execution

Level 1
- Microarchitecture level
  - Hardware

Level 0
- Digital logic level
ISA

• Instruction Architecture Level
  – Lay between the microarchitecture level and operating system level
• Significance
  – ISA is the interface between the software and the hardware.
  – It is not good to have the hardware directly execute programs written in C, C++, Java, or some other high-level language: 1) the performance advantage of compiling over interpreting would then be lost. 2) most computers have to be able to execute programs written in multiple languages, not just one.

• One Design Principle
  – Let programs in various high-level languages to be translated to a common intermediate form—the ISA level
  – Build hardware to be able to execute ISA-level programs directly.

• Theory vs. Reality
  – Theory: satisfy both compiler writer and hardware engineer
  – Practice: backward compatible
ISA Level

- **Relationship**
  - The ISA level define the interface between the compilers and the hardware.
  - It is the language that both of them have to understand.
  - The relationship among the compilers, the ISA level, and the hardware.
What make a good ISA?

• Implementation
  – A good ISA should define a set of instructions that can be implemented efficiently in current and future technologies, resulting in cost-effective designs over several generations.
  – A poor design is more difficult to implement and may require many more gates to implement a processor and more memory for executing programs. It also may run slower because the ISA obscures opportunities to overlap operations, requiring much more sophisticated designs to achieve equivalent performance.

• Target for Compiler Code
  – A good ISA should provide a clean target for compiled code.
  – Regularity and completeness of a range of choices are important traits that are not always present in an ISA.
  – Since the ISA is the interface between the hardware and the software, it should make the hardware designers happy (easy to implement efficiently) and make the software designers happy (easy to generate good code for).
Properties of the ISA

• **Definition**
  – The ISA level is defined by how the machine appears to a machine language programmer.
  – Also defined as: ISA level code is what a compiler outputs

• **To produce ISA level code, the compiler writer has to know**
  – What the memory model is
  – What registers there are
  – What data types and instructions are available
  – ……

• **Two Mode**
  – Kernel mode is intended to run the OS and allows all instructions to be executed
  – User mode is intended to run application programs and does not permit certain sensitive instructions (manipulate cache directly)
Memory Models

An 8-byte word in a little-endian memory. (a) Aligned. (b) Not aligned. Some machines require that words in memory be aligned.

- **Alignment**
  - Necessary because memories operate more efficiently that way
- **Memory Semantics**
  - The ISA memory model is the memory semantics
Memory and Register

- **Memory Models**
  - All memory requests are serialized. Each one is completed before the next one is issued.
  - Execute a SYNC instruction to block issuing new memory operations until all previous ones have completed.
  - Intermediate: hardware automatically blocks the issuing of certain memory references but not others.

- **Registers**
  - Some of them visible in the ISA level.
  - If visible in the ISA level, also visible in the microarchitecture level.
  - Control execution of the program.
  - Hold temporary results.

- **Program Status Word (PSW)**
  - Holds bits that are needed by the CPU.
  - Condition Codes: N (negative), Z (zeros), V (overflow), P (even parity).
Pentium 4 ISA

- **Operation Mode**
  - Real Mode: like a simple 8088
  - Virtual mode: run old 8088 programs in a protected mode
  - Protected Mode

- **Protected Mode**
  - Level 0: corresponds to kernel mode, used by OS
  - Level 3: used by user program
  - Level 1 and 2, rarely used
Pentium 4: Primary Registers

- **Registers**
  - EAX: main arithmetic
  - EBX: points, memory addresses
  - ECX: looping
  - EDX: multiplication and division
  - ESI and EDI: hold pointers into memory
  - ESI: points to source string
  - EDI: point to destination string
  - EBP: point to the base of the current stack frame
  - ESP: stack pointer
  - CS ~ GS: segment registers
  - EIP: program counter
  - EFLAGS: Program Status Word
Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>1 Bit</th>
<th>8 Bits</th>
<th>16 Bits</th>
<th>32 Bits</th>
<th>64 Bits</th>
<th>128 Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signed integer</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unsigned integer</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary coded decimal integer</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Floating point</td>
<td></td>
<td></td>
<td></td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
</tbody>
</table>

The Pentium 4 numeric data types. Supported types are marked with ×.

• **Hardware Support**
  – Key issue: the data type is supported by the hardware?
  – It means that one or more instructions expect data in a particular format and the user is not free to pick a different format

• **Two Categories**
  – numeric
  – Nonnumeric (ASCII, UNICODE; Boolean Values; Pointer)
Instruction Formats (1)

Four common instruction formats:
(a) Zero-address instruction. (b) One-address instruction
(c) Two-address instruction. (d) Three-address instruction.

• An Instruction
  – Consists of an opcode, usually along with some additional information such as where operands come from, and where results go to
  – Instruction always have an opcode to tell what the instruction does. There can be zeros, one, two, or three addresses present
Instruction Formats (2)

- **Length**
  - On some computer, all instructions have the same length
  - On others, many different length
  - Have all instruction be the same length is simpler and makes decoding easier but often wastes space, since all instructions then have to be as long as the longest one
Design Principles for Instruction Formats

• **Shorter is better**
  – All things being equal, shorter instructions are better
  – Shorter instruction is important for memory bandwidth

  Suppose the bandwidth of an instruction cache is $t$ bps and the average instruction length is $r$ bits, the cache can deliver at most $t/r$ instructions

• **Sufficient Room**
  – To express all the operations desired;
  – $2^n$ operations at most with $n$ bit instructions

• **Number bit in an address field**
  – To gain a finer memory resolution, one must pay the price of longer addresses and thus longer instructions
Expanding Opcodes (1)

An instruction with a 4-bit opcode and three 4-bit address fields.

- **Tradeoff between opcode and address**
  - \((n+k)\) bit instruction with a \(k\) bit opcode and \(n\) bit address
  - \(2^k\) different operations and \(2^n\) addressable memory cells
  - \(2^{k-1}\) different operations and \(2^{n+1}\) addressable memory cells
  - \(2^{k+1}\) different operations and \(2^{n-1}\) addressable memory cells
Expanding Opcodes (2)

An expanding opcode allowing 15 three-address instructions, 14 two-address instructions, 31 one-address instructions, and 16 zero-address instructions. The fields marked \textit{xxxx}, \textit{yyyy}, and \textit{zzzz} are 4-bit address fields.
The Pentium 4 Instruction Formats

### Opcode
- In general, the opcode must be fully decoded to determine what class of operation is to be performed and thus how long the instruction is.
- This makes high performance implementation difficult.
- Extensive decoding is necessary before it can even be determined where the next instruction starts.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>0 - 5</th>
<th>1 - 2</th>
<th>0 - 1</th>
<th>0 - 1</th>
<th>0 - 4</th>
<th>0 - 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PREFIX</td>
<td>OPCODE</td>
<td>MODE</td>
<td>SIB</td>
<td>DISPLACEMENT</td>
<td>IMMEDIATE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits</th>
<th>6</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INSTRUCTION</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Which operand is source?**

**Byte/word**

<table>
<thead>
<tr>
<th>Bits</th>
<th>2</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MOD</td>
<td>REG</td>
<td>R/M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits</th>
<th>2</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SCALE</td>
<td>INDEX</td>
<td>BASE</td>
</tr>
</tbody>
</table>
Problem

Problem 4 in Chapter 1

Consider a multilevel computer in which all the levels are different. Each level has instructions that are \( m \) times as powerful as those of the level below it; that is, one level \( r \) instruction can do the work of \( m \) level \( r - 1 \) instructions. If a level 1 program requires \( k \) seconds to run, how long would equivalent programs take at levels 2, 3, and 4, assuming \( n \) level \( r \) instructions are required to interpret a single \( r + 1 \) instruction?

Solution

- One level-\( r \) instruction can do the work of \( m \) level-(\( r-1 \)) instructions
- \( n \) level-(\( r-1 \)) instructions are required to interpret a single \( r \)-level instruction
- So, \( n \) level-(\( r-1 \)) instruction with interpretation can do the same work of \( m \) level-(\( r-1 \)) instruction
- The time ratio is \( n/m \) between level \( r \) and level \( r-1 \)
- Thus, level 2 requires \( k\times(n/m)^1 \)
- Thus, level 3 requires \( k\times(n/m)^2 \)
- Thus, level 4 requires \( k\times(n/m)^3 \)
Problem: Parity Bits

- **Hamming Algorithm**
  - Construct error-correcting codes for any size memory word
  - In a Hamming code, \( r \)-bit parity bits are added to an \( m \)-bit word, forming a new word with \( m+r \) bits
  - The bits are numbered starting at 1 with bit 1 the leftmost bit
  - All bits with bit number a power of 2 are parity bits, all the rest are data bits
  - Each parity bit will check specific bit positions
  - The parity bit is set so that the total number of 1s in the checked position is even (odd)
  - Bit \( b \) is checked by those bits \( b_1, b_2, \ldots, b_j \) such that \( b_1 + b_2 + \ldots + b_j = b \)

- **Exercise**
  - Which bits will be parity bits? Answer: 1, 2, 4, 8, 16, 32, 64, …
  - What bits will check bit 5? Answer: 1 and 4
  - What bits will check bit 6? Answer: 2 and 4
  - What bits will check bit 7? Answer: 1, 2 and 4
Problem: Parity Bits

\[ \begin{array}{cccccccccc}
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
\end{array} \]

1 1 1 0 1 0 0 1 0 1 1 1 1

Bit 1 check bits: 1, 3, 5, 7, 9, 11

Bit 2 check bits: 2, 3, 6, 7, 10, 11

Bit 4 check bits: 4, 5, 6, 7, 12

Bit 8 check bits: 8, 9, 10, 11, 12
Exercise

Ex 1: Suppose the memory word is 10010100.

- To be able to correct single bit error, how many number of check bits we need?
- Please give the construction of the even-parity Hamming code for this memory word according to the Hamming algorithm

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 1 & 0 & 0 & 1 & 1
\end{array}
\]

- Answer:
  - Bit 1: 1, 3, 5, 7, 9, 11
  - Bit 2: 2, 3, 6, 7, 10, 11
  - Bit 4: 4, 5, 6, 7, 12
  - Bit 8: 8, 9, 10, 11, 12
Exercise

Ex 2: Suppose the memory word is 00010100.

- To be able to correct single bit error, how many number of check bits we need?
- Please give the construction of the even-parity Hamming code for this memory word according to the Hamming algorithm

<table>
<thead>
<tr>
<th>Bit 1: 1, 3, 5, 7, 9, 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 2: 2, 3, 6, 7, 10, 11</td>
</tr>
<tr>
<td>Bit 4: 4, 5, 6, 7, 12</td>
</tr>
<tr>
<td>Bit 8: 8, 9, 10, 11, 12</td>
</tr>
</tbody>
</table>

- Answer: 4.
Exercise

Ex 3: Suppose the memory word is 00010101.
- To be able to correct single bit error, how many number of check bits we need?
- Please give the construction of the even-parity Hamming code for this memory word according to the Hamming algorithm

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

- Answer: 
- 4.
- 100000100101

Bit 1: 1, 3, 5, 7, 9, 11
Bit 2: 2, 3, 6, 7, 10, 11
Bit 4: 4, 5, 6, 7, 12
Bit 8: 8, 9, 10, 11, 12