CS2422 Assembly Language & System Programming

September 19, 2006

Today’s Topics (1/2)

• How the CPU executes a program.
• Where we can store the data.
  – Registers
  – Memory
  – Constants
Today’s Topics (2/2)

• Data representation – What are stored in the main memory?
  – Binary, Decimal, Hexadecimal…etc.
  – Units: Byte, Word, Double-word…etc.
  – Memory Addresses.
  – Characters Strings.
  – Big or Little Endian.

Study Guide

• Textbook: Sections 1.3, 2.1 (except 2.1.2.2), 2.2, 2.3.
• Read (in depth):
  – 2.1.1 and 2.1.2.1
  – 2.2.1 and 2.2.2
  – 2.3.1 and 2.3.2.1
• Browse the rest (like reading news stories).
Program Loading & Execution

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

Instruction Execution Cycle

- Fetch
- Decode
- Fetch operands
- Execute
- Store output
Clock

• Not the wall clock.
• Ever wondering what the CPU clock rate (like 2.0GHz) means?
• Next slide: using the memory read as an example

Figure 2.7 (from Textbook)
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

User
  - sends program name to Operating system
    - gets starting cluster from Directory entry
      - loads and starts Program
    - searches for program in Current directory
      - returns to System path
Multitasking

- OS can run multiple programs at the same time.
- Multiple threads of execution within the same program.
- Scheduler utility assigns a given amount of CPU time to each running program.
- Rapid switching of tasks

Data Representations

- Consider this simple view: The whole program is stored in main memory.
  - Including program instructions (code) and data.
  - CPU loads the instructions and data from memory for execution.
  - Don’t worry about the disk for now.
Where are the Data?

- Registers (inside the CPU)
- Memory
- Constant

Registers

- General-Purpose:
  - AX, BX, CX, DX: 16 bits
  - Splitted into H and L parts, 8 bits each.
  - Extended into E?X to become 32-bit register (i.e., EAX, EBX,…etc.).
Convention

- AX: accumulator
- BX: base register
- CX: count register
- DX: data register

Some instructions use them implicitly. (e.g., LOOP uses CX with no mentioning.)

Other Registers

We will explain their meaning when we encounter them later this semester:
- Segment (CS, DS, SS, ES)
- Pointer (IP, SP, BP)
- Index (SI, DI)
- Flags
Memory

• Organized like mailboxes, numbered 0, 1, 2, 3,…, \(2^n-1\).
  – Each box can hold 8 bits (1 byte)
  – So it is called byte-addressing.

Byte? Word?

• The number has limited range.
• 1 Byte = 8 Bits:
  – Binary: 0000 0000 to 1111 1111
  – Hexadecimal: 00 to FF
  – Decimal: 0 to 255
• Word = 2 or 4 bytes, depending on the machine. In 80x86, it means 2 bytes.
Number Systems

• I assume that now you all know:
  – Binary & hexadecimal numbers.
  – Conversion between them and decimal.
  – How to represent negative numbers (in 2’s compliment).

Memory Address

• Byte Addressing: each memory location has 8 bits.
• If we have only 16 bytes…
  – 4 bits are enough for an address
• What if we have 64K?
• 1M? 1G?
Memory Address

• 16-bit address is enough for up to 64K
• 20-bit for 1M
• 32-bit for 4G
• Most servers need more than 4G!!
  That’s why we need 64-bit CPUs like Alpha (DEC/Compaq/HP) or Merced (Intel).

Confused Now?

• So, is the memory 8-bit or 32-bit?
• It depends on what you ask:
  – Content or Address?
  – Remember addresses can be stored in memory as well. (They are called pointers in PASCAL and C.)
Character String

- So how are strings like “Hello, World!” are stored in memory?
- ASCII Code! (or Unicode…etc.)
- Each character is stored as a byte.
- Review: how is “1234” stored in memory?
Integer

• A byte can hold an integer number:
  – between 0 and 255 (unsigned) or
  – between –128 and 127 (2’s compliment)
• How to store a bigger number?
• Review: how is 1234 stored in memory?

Big or Little Endian?

• Example: 1234 is stored in 2 bytes.
  • = 100-1101-0010 in binary
  • = 04 D2 in hexadecimal
• Do you store 04 or D2 first?
  • Big Endian: 04 first.
  • Little Endian: D2 first. ← Intel’s choice
• Reason for Little-Endian?
  – More consistent for variable length (e.g.,
    2 bytes, 4 bytes, 8 bytes…etc.)

Writing the First Program…

• MOV AX, var1
• MOV BX, 1
• ADD AX, BX
• MOV var1, AX
Constant (or Immediate) Data

• Compare:
  – MOV AX, 25
  – MOV AX, var1 ; assume var1 at address 25
• The constant 25 in the first case is called immediate data.
• The second case may be considered as:
  – MOV AX, [25]

Food for Thought

• Example: MOV AX, 25
• But, where is the number stored? In memory?
• A related question: where is the instruction (MOV AX, 25) stored?
• What do we do if we want to move the data in memory address 25 to AX?
Multi-Stage Pipeline

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

Example of a non-pipelined processor. Many wasted cycles.

<table>
<thead>
<tr>
<th>Cycles</th>
<th>Stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S1</td>
</tr>
<tr>
<td>2</td>
<td>S2</td>
</tr>
<tr>
<td>3</td>
<td>S3</td>
</tr>
<tr>
<td>4</td>
<td>S4</td>
</tr>
<tr>
<td>5</td>
<td>S5</td>
</tr>
<tr>
<td>6</td>
<td>S6</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

Pipelined Execution

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

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

$$k + (n - 1)$$
Instruction Set Architecture

... the attributes of a [computing] system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls, the logic design, and the physical implementation.
– Amdahl, Blaaw, and Brooks, 1964

• interface between hardware and low-level software
• standardizes instructions, machine language bit patterns, etc.
• advantage: different implementations of the same architecture
• disadvantage: sometimes prevents using new innovations

RISC vs. CISC

• RISC: PowerPC, MIPS, DEC Alpha
• CISC: x86/Pentium, AMD, VAX
• Example:
  x86 (CISC):
  MOV AX, a
  ADD AX, b
  
  RISC:
  Load R1, a
  Load R2, b
  Add R2, R1