Midterm 2 Review

INFO/CSE 100, Spring 2006
Fluency in Information Technology

http://www.cs.washington.edu/100
Readings and References

• Reading
  » *Fluency with Information Technology*
    • Chapters 9, 11 18-21
Overview

• During this quarter, we're looking at the actual workings of computer systems

• Organized as “layers of abstraction”
  » application programs
  » higher level languages: Javascript, SQL, …
  » operating system concepts
  » bits, bytes, assembly language
  » transistors, electrons, photons
Layers of Abstraction

• At any level of abstraction, there are
  » elements at that level
  » the building blocks for those elements

• Abstraction
  » isolates a layer from changes in the layer below
  » improves developer productivity by reducing detail needed to accomplish a task
  » helps define a single architecture that can be implemented with more than one organization
Architecture & Organization

- Architecture (the *logical definition*)
  - defines elements and interfaces between layers
  - Instruction Set Architecture
    - instructions, registers, addressing

- Organization (the *physical implementation*)
  - components and connections
  - how instructions are implemented in hardware
  - many different organizations can implement a single architecture
Computer Architecture

• Specification of how to program a specific computer family
  » what instructions are available?
  » how are the instructions formatted into bits?
  » how many registers and what is their function?
  » how is memory addressed?

• Some examples architectures
  » IBM 360, 370, …
  » PowerPC 601, 603, G5, …
  » Intel x86 286, 386, 486, Pentium, …
  » MIPS R2000, R3000, R4000, R5000, …
Computer Organization

• Processor
  » Data path (ALU) manipulate the bits
  » The control controls the manipulation

• Memory
  » cache memory - smaller, higher speed
  » main memory - larger, slower speed

• Input / Output
  » interface to the rest of the world
A Typical Organization

main memory

processor/memory bus

processor

I/O bus

hard disk
floppy disk
CDROM drive
serial ports
network interface
Anatomy of a Computer

Processor

ALU

Control

Input

Mouse
Keyboard
Scanner

Hard Disk
Floppy Disk

Output

Monitor
Printer
Speakers

Memory
Fetch/Execute Cycle

Computer = instruction execution engine

» The fetch/execute cycle is the process that executes instructions

Instruction Fetch (IF)
Instruction Decode (ID)
Data Fetch (DF)
Instruction Execution (EX)
Result Return (RR)
Memory ...

Programs and the data they operate on must be in the memory while they are running.

Memory locations

```
0  1  2  3  4  5  6  7  8  9  10  11
G o D a w g s ! ! 0 ...
```

- **Memory addresses**
- **Memory contents**

byte=8 bits

0 1 0 0 0 1 0 0
Control

• The Fetch/Execute cycle is hardwired into the computer’s control, i.e. it is the actual “engine”
• Depending on the Instruction Set Architecture, the instructions say things like
  » Put in memory location 20 the contents of memory location 10 + contents of memory location 16
  » The instructions executed have the form ADDB 10, 16, 20
    • Add the bytes from memory address 10 and memory address 16 and store the result in memory address 20
ALU

The Arithmetic/Logic Unit does the actual computation

Depending on the Instruction Set Architecture, each type of data has its own separate instructions:

- ADDB : add bytes
- ADDBU : add bytes unsigned
- ADDH : add half words
- ADDHU : add halves unsigned
- ADD : add words
- ADDU : add words unsigned
- ADDS : add short decimal numbers
- ADDD : add long decimal numbers

Most computers have only about a 100-150 instructions hard wired
Input/Output

- Input units bring data to memory from outside world; output units send data to outside world from memory
  - Most peripheral devices are “dumb”, meaning that the processor assists in their operation
The PC’s PC

• The program counter (PC) tells where the next instruction comes from
  » In some architectures, instructions are always 4 bytes long, so add 4 to the PC to find the next instruction

Program Counter: 112

```
<table>
<thead>
<tr>
<th>112</th>
<th>113</th>
<th>114</th>
<th>115</th>
<th>116</th>
<th>117</th>
<th>118</th>
<th>119</th>
<th>120</th>
<th>121</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD 210,216,220</td>
<td>AND 414,418,720</td>
<td>OR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

...
Clocks Run The Engine

• The rate that a computer “spins around” the Fetch/Execute cycle is controlled by its clock
  » Current clocks run 2-3 GHz
  » The computer tries do at least one instruction per cycle, depending on the instruction and the availability of memory contents
  » Modern processors often try to do more than one instruction per cycle

Clock rate is not a good indicator of speed anymore, because several things are happening every clock cycle
Algorithm

• Algorithm
  » a precise, systematic method to produce a desired result

• For example, the placeholder technique for deleting a short string except where it occurs in longer strings is an algorithm with an easy specification:

  longStringWithShortStringInIt ← placeholder
  ShortString ← e
  placeholder ← longStringWithShortStringInIt
Programs vs Algorithms

• A program is an algorithm specialized to a particular situation
  » an Algorithm
    
    longStringWithShortStringInIt ← placeholder
    ShortString ← e
    placeholder ← longStringWithShortStringInIt
  
  » a Program that implements the Algorithm
    
    # ← # // replace double <newlines> with <#>
    e ← e // delete all single <newlines>
    # ← # // restore all double <newlines>
What the heck is the DOM?

• Document Object Model
  » Your web browser builds a *model* of the web page (the *document*) that includes all the *objects* in the page (tags, text, etc)
  » All of the properties, methods, and events available to the web developer for manipulating and creating web pages are organized into objects
  » Those objects are accessible via scripting languages in modern web browsers
This is what the browser reads (sampleDOM.html).

```html
<html>
<head>
  <title>Sample DOM Document</title>
</head>
<body>
  <h1>An HTML Document</h1>
  <p>This is a <i>simple</i> document.</p>
</body>
</html>
```

This is what the browser displays on screen.

An HTML Document

This is a simple document.
This is a drawing of the model that the browser is working with for the page.

Figure 17-1. The tree representation of an HTML document
Copied from JavaScript by Flanagan.
Reference to several nodes in the model of the page that the browser constructed

- document
  - The root of the tree is an object of type HTMLDocument
  - Using the global variable `document`, we can access all the nodes in the tree, as well as useful functions and other global information
    - title, referrer, domain, URL, body, images, links, forms, ...
    - open, write, close, `getElementById`, ...
document.getElementById("radioLC").checked

- `getElementById("radioLC")`
  - This is a predefined function that makes use of the `id` that can be defined for any element in the page
  - An `id` must be unique in the page, so only one element is ever returned by this function
  - The argument to `getElementById` specifies which element is being requested
document.getElementById("radioLC").checked

• **checked**

  » This is a particular property of the node we are looking at, in this case, a radio button

  » Each type of node has its own set of properties
    • for radio button: `checked`, `name`, ...
    • refer to the HTML DOM for specifics for each element type

  » Some properties can be both read and set
Representing Data as Symbols

• 24 Greek Letters
• And we decide to use 2 symbols, binary, to represent the data.
• How many bits do we need?!?
  » 24 total possibilities
  » $2 \times 2 \times 2 \times 2 \times 2 = 2^5 = 32$
  • We get 6 extra!
Info Representation

• Adult humans have 32 teeth
  » sometimes a tooth or two is missing!

• How can we represent a set of teeth?
  » How many different items of information?
    • 2 items - *tooth* or *no tooth*
  » How many "digits" or positions to use?
    • 32 positions - one per tooth socket
  » Choose a set of symbols
    *no tooth*: 0  *tooth*: 1
What's your tooth number?

incisors
0 0 0 0 0 0 0 0

canines
0 0 0 0 0 0 0 0

pre-molars
0 0 0 0 0 0 0 0

molars
0 0 0 0 0 0 0 0

no teeth ↔ 0000 0000 0000 0000 0000 0000 0000 0000

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

no molars ↔ 1111 1111 1111 1111 1111 0000 0000 0000

How many possible combinations? $2 \times 2 \times 2 \times \ldots \times 2 = 2^{32} \approx 4 \text{ Billion}$
How many positions should we use?

It depends: how many numbers do we need?

- **One position**
  - Two numbers: 0, 1

- **Two positions**
  - Four numbers:
    - 00
    - 01
    - 10
    - 11

- **Three positions**
  - Eight numbers:
    - 000
    - 001
    - 010
    - 011
    - 100
    - 101
    - 110
    - 111
Converting from binary to decimal

Each position represents one more multiplication by the base value.

For binary numbers, the base value is 2, so each new column represents a multiplication by 2.

\[
\begin{align*}
1 & \cdot 128 + 0 \cdot 64 + 0 \cdot 32 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 138_{10} \\
1 & \cdot 128 + 1 \cdot 8 + 1 \cdot 2 = 138_{10}
\end{align*}
\]
Base 16 Hexadecimal

- The base value can be 16 - hexadecimal numbers
  - Sixteen symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - Each column represents a multiplication by sixteen
  - Hex is easier to use than binary because the numbers are shorter even though they represent the same value

\[
\begin{array}{c|c|c|c|c}
16^3 & 16^2 & 16^1 & 16^0 \\
16^3 = 4096 & 16^2 = 256 & 16^1 = 16 & 16^0 = 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 8 & A \\
\end{array}
\]

\[
8 \cdot 16 + 10 \cdot 1 = 138_{10}
\]
## Four binary bits ⇔ One hex digit

<table>
<thead>
<tr>
<th>binary (base 2)</th>
<th>hexdecimal (base 16)</th>
<th>decimal (base 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
<td>13</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td>14</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td>15</td>
</tr>
</tbody>
</table>
Binary to Hex examples

1000 0010 0000 0111 1010 0001 0000 1111

8 2 0 7 A 1 0 F

100000100000011110100001000011112 = 8207A10F16

10000011010001011010100110111110

10000110100010110101001101111102 = 16

base 2

base 16
Represent Text - ASCII

• Assign a unique number to each character
  » 7-bit ASCII
    • Range is 0 to 127 giving 128 possible values
    • There are 95 printable characters
    • There are 33 control codes like tab and carriage return

image is from Wikipedia
Represent Text - Unicode

• The goal of Unicode is to provide the means to encode the text of every document people want to store in computers
• Unicode aims to provide a unique number for each letter, without regard to typographic variations used by printers
• Unicode encodes each character in a number
  » the number can be 7, 8, 16, or 32 bits long
  » 16-bit encoding is common today
This page was created in order to test support for different encodings that allows to present Cyrillic correctly.

Эта страничка создана для тестирования поддержки различных кодировок позволяющих корректно показывать кириллицу.
Represent Text - Postscript

- Postscript is a page description language somewhat like HTML
  - The file is mostly text and can be looked at with a regular text editor
  - Programs that know what it is can interpret the embedded commands
  - Programs *and printers* that understand Postscript format can display complex text and graphical images in a standard fashion
This page was created in order to test support for different encoding, allows to present Cyrillic correctly.

Эта страница создана для тестирования поддержки различных кодировок, корректно показывая кириллицу.
Represent Text - PDF

- PDF is another page description language based on Postscript
- The file is mostly text
  - can be looked at with a regular text editor
  - programs that know what it is can interpret the embedded commands
  - just like Postscript and HTML in that respect
Represent Color - Bit Map

- Numbers can represent anything we want
- Recall that we can represent colors with three values
  » Red, Green, Blue brightness values
- There are *numerous* formats for image files
  » All of them store some sort of numeric representation of the brightness of each color at each pixel of the image
  » commonly use 0 to 255 range (or 0 to FF₁₆)
What about "continuous" signals?

- Color and sound are natural quantities that don't come in nice discrete numeric quantities
- But we can “make it so!”
Digitized image contains color data
And much, much more!
Summary

• Bits can represent any information
  » Discrete information is directly encoded using binary
  » Continuous information is made discrete

• We can look at the bits in different ways
  » The format guides us in how to interpret it
  » Different interpretations let us work with the data in different ways