Digital Information

INFO/CSE 100, Spring 2005
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 \textit{logical definition})
  » defines elements and interfaces between layers
  » Instruction Set Architecture
    • instructions, registers, addressing

• Organization (the \textit{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
- processor/memory bus
- I/O bus
- hard disk
- floppy disk
- CDROM drive
- serial ports
- network interface
Anatomy of a Computer

Processor

ALU
Control
Memory

Input
Output

Mouse
Keyboard
Scanner
Hard Disk
Floppy Disk
Monitor
Printer
Speakers
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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>o</td>
<td>D</td>
<td>a</td>
<td>w</td>
<td>g</td>
<td>s</td>
<td>!</td>
<td>!</td>
<td>0</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

byte=8 bits

memory addresses

memory contents
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

```
  10  11  12  13  14  15  16  17  18  19  20  21
  6   12   18 ...
```
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

```
ADD 210,216,220  AND 414,418,720  OR
```

...
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 to 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:

\[
\text{longStringWithShortStringInIt} \leftarrow \text{placeholder} \\
\text{ShortString} \leftarrow e \\
\text{placeholder} \leftarrow \text{longStringWithShortStringInIt}
\]
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 // delete all single < newlines>
    # ← ‹|‹| // restore all double <newlines>
Variables In Real Life

• A variable is a "container" for information you want to store
  » The name of the variable stays the same, but the value associated with that name can change

That’s why it’s called a “variable”!

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Current Value</th>
<th>Previous Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1 Single</td>
<td>My Boo, Usher And Alicia Keys</td>
<td>Goodies, Ciara</td>
</tr>
<tr>
<td>AL Champion</td>
<td>Boston Red Sox</td>
<td>New York Yankees</td>
</tr>
<tr>
<td>#1 Box Office</td>
<td>Shark Tale</td>
<td>Shark Tale</td>
</tr>
<tr>
<td>Day Of The Week</td>
<td>Monday</td>
<td>Sunday</td>
</tr>
<tr>
<td>Husky Card Balance</td>
<td>$52</td>
<td>$60</td>
</tr>
</tbody>
</table>
Variables In Programming

- Program variables have names and values
  - Names (also called identifiers)
    - generally start with a letter and can contain letters, numbers, and underscore characters “_”
    - Names are case sensitive
  - Values
    - can be numbers, strings, boolean, etc
    - change as the program executes

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Current Value</th>
<th>Previous Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No_1_Single</td>
<td>My Boo, Usher And Alicia Keys</td>
<td>Goodies, Ciara</td>
</tr>
<tr>
<td>ALChampion</td>
<td>Boston Red Sox</td>
<td>New York Yankees</td>
</tr>
<tr>
<td>No_1_Box_Office</td>
<td>Shark Tale</td>
<td>Shark Tale</td>
</tr>
<tr>
<td>dayOfTheWeek</td>
<td>Monday</td>
<td>Sunday</td>
</tr>
<tr>
<td>huskyCardBalance</td>
<td>$52</td>
<td>$60</td>
</tr>
</tbody>
</table>
Variable Declarations

<script type="text/javascript">

var eyeColor;  <<< undefined!

var eyeColor = "green";  <<< initialized

var eyeColor = "";  <<< initialized, empty

var eyeColor = "green", hairColor="blonde";

hairColor = "carmel";
</script>
Basic Data Types in Javascript

Numbers:
var gasPrice = 2.55;

Strings
var eyeColor = "hazel green";

Boolean
var isFriday = true;
var isWeekend = 0;
Expressions

• The right-hand side of an assignment statement can be any valid *expression*
• Expressions are “formulas” saying how to manipulate existing values to compute new values

\[
\begin{align*}
\text{balance} &= \text{balance} - \text{transaction}; \\
\text{seconds} &= 60\times\text{minutes}; \\
\text{message} &= "\text{Status code is } \text{codeValue}"; \\
\text{isFreezing} &= (\text{temp} < 32);
\end{align*}
\]
Operators

Use operators to build expressions

» Numeric operators
   + - * / mean add, subtract, multiply, divide
   \[ 3 + 3 = 6 \]

» String operator
   + means concatenate strings
   \[ "3" + "3" = "33" \]

» Relational operators
   < <= == != => mean less than, less than or equal to, equal to, not equal to, greater than or equal to, greater than

» Boolean operators
   && || ! mean and, or, not
Functions

A *function* is a way to bundle a set of instructions and give them a name so that you can reuse them easily.

Functions have a specific layout:

- `<name>` ← the function name is an identifier
- `<parameter list>` ← list of input variables for the function
- `<statements>` ← the statements do the work

```plaintext
function <name> ( <parameter list> ) {
  <statements>
}
```
Example Function

Write a simple function to compute the Body Mass Index when the inputs are in English units (ie, US units)

```javascript
function bmiE(weightLBS, heightIN) {
    var heightFt = heightIN / 12;  // convert to feet
    return 4.89 * weightLBS / (heightFt * heightFt);
}
```
Global or Local?!?

- Scope of a variable describes where and when it can be referenced
  - Local variables are only known inside of a function (curly braces)
  - Global variables are known by all the Javascript inside of `<script>` `</script>` pairs

```javascript
// Calculate Percentage of Study Hours/Week
// time in hours
// returns hours
var days = 7;
function calculateStudyHrs(time) {
    var totalHrs = 24 * days;
    return time/totalHrs;
}
```
Layout of the GUI

• The layout of the page is controlled with HTML in the body of the page

```html
<body>
  HTML form layout and specification
</body>
</html>
```

• The layout and controls are provided using new tags

  » <form id="buttonForm">
  » <button type="button" ...
  » <input type="text" …
  » <input type="radio" …
  » <button type="reset" …
```
A simple example

This GUI has several simple controls.

- Two buttons to control the results
- One text field to display the results
- One pair of radio buttons to control the display
- One button to reinitialize

• a `<button>` can have one of three types
  » type “button” is used locally
  » type “submit” sends data back to the server
  » type “reset” re-initializes the form
• the value of the “onclick” attribute is some JavaScript code, in this case a call to the function `setResults(string)`
• an `<input>` with `type="text"` is used for user input and program output
• `value="nada"` sets the initial (and reset) value
• `readonly` means that the user cannot set the value, only the script can set the value
• `id="resultField"` gives us a way to identify this particular control in our JavaScript
Events Cause Processing

• After drawing a page, the browser sits idle waiting for something to happen … when we give input, we cause *events*

• Processing events is the task of a block of code called an *event handler*  
  » The code to execute is identified in the tag using the appropriate attribute  
  » There are many event types  
    • `onClick`, `onChange`, `onMouseOver` ...

5/11/05 fit100-19-review © 2005 University of Washington

© 2005 University of Washington
setResults(resultString)

```javascript
function setResults(resultString) {
    var tempString = resultString;
    if (document.getElementById("radioLC").checked) {
        tempString = tempString.toLowerCase();
    } else if (document.getElementById("radioUC").checked) {
        tempString = tempString.toUpperCase();
    }
    document.getElementById("resultField").value = tempString;
}
</script>
```

parameter variable, local variable, if/else statement, field reference, call to toLowerCase() function
The \texttt{if/else} statement

The \texttt{if} statement is a \textit{conditional statement}

- a conditional expression is evaluated as being \texttt{true} or \texttt{false}
  - the expression is a \textit{boolean expression} (ie, returns \texttt{true} or \texttt{false})
- if the condition is \texttt{true}, then one set of statements is executed
- if the statement is \texttt{false}, then a different set of statements is executed

\begin{verbatim}
if (<boolean expression>) {
  <statements>
} else {
  <statements>
}
\end{verbatim}
Examples

```java
if (count == 0) {
    ready = false;
} else {
    ready = true;
    count = count - 1;
}
```

What is the conditional expression?
What statements are part of the true block?
Which statements are part of the false block?
What happens when count is 21? 0? -1?

```java
if (pageCount >= 100) {
    alert("This may take a few minutes.");
}
```

What is the conditional expression?
Which statements are part of the false block?
What statements are part of the true block?
What happens when pageCount is 21? 100? 200?
More if/else Statements

```javascript
if (temp < 32) {
    if (sky == "cloudy") {
        alert("Snow is forecast!"antha);
    }
}

if (temp < 32 && sky == "cloudy") {
    alert("Snow is forecast!"antha);
}
```
The **for** loop

A counting loop is usually implemented with **for**

```javascript
var count = 10;

for (var i=0; i < count; i++) {
    document.writeln("<br>index value is : "+i);
}
```

- initialize
- check for limit
- update loop control index shorthand for `i=i+1`
- one or more statements in the loop body
**i++ is a shortcut**

- for \((i=0; i < \text{count}; i++)\)
- at the end of every pass through the **for** loop body, do the following:
  - get the value of \(i\)
  - increment \(i\)
  - store the incremented value
- Used as it is here, this is the same as writing
  \(i = i + 1\)
body of loop may not execute at all

- Notice that depending on the values of the control variables, it is quite possible that the body of the loop will not execute at all.

```javascript
var itemCount = 0;
...
for (var i=0; i < itemCount; i++) {
    document.writeln("<br>..processing item "+i);
}
```

check for limit condition

itemCount is 0 when we get here, so i < itemCount is immediately false and the loop body is skipped completely.
Arrays

• JavaScript (and most other languages) includes *arrays* as the most basic kind of collection.
  » Simple, ordered collections
  » Special syntax for accessing elements by position

• JavaScript arrays can be created
  » by the programmer in the script
  » by the system and provided to the script
    • for example, the elements array in the iCCC program
Array Example

variable
petNames

Array
length: 5
index 0
index 1
index 2
index 3
index 4

String
"Jaba"

String
"Bingo"

String
"Jessica"
JavaScript Indexed Arrays

• An indexed array is a data type that stores a collection of values, accessible by number
  » the values in the array are called the elements of the array
  » the elements (or values) are accessed by index
    • the index of the first value is 0
  » the values in the array can be any type
    • usually all the values are the same type
    • but they can be different from one another if necessary
Array Declaration and Creation

• Arrays can be created several different ways
  » var petNames = new Array();  
    • 0-length array with no elements in it yet
  » var studentNames = new Array(102);  
    • 102-element array, all of which have the value undefined
  » var myList = ["Sally", "Splat", "Google"];  
    • 3-element array initialized with an array literal

• Arrays have a property that stores the length
  <array name>.length
  » you can lengthen or shorten an array by setting the length to a new value
Array Element Access

- Access an array element using the array name and position:
  \( <\text{array name}>[<\text{position}>] \)

- Details:
  » \(<\text{position}>\) is an integer expression.
  » Positions count from zero

- Update an array element by assigning to it:
  \( <\text{array name}>[<\text{position}>] = <\text{new element value}>;\)

  ```
  myCurrentCarNo = carList.length-1;
  myCurrentCar = carList[myCurrentCarNo];
  ```
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.
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.getElementById("radioLC").checked`

- **`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
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 canines pre-molars molars

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 0 0 0 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? $2 \times 2 \times 2 \times \ldots \times 2 = 2^{32} \approx 4$ Billion
How many positions should we use?

It depends: how many numbers do we need?

<table>
<thead>
<tr>
<th>one position</th>
<th>two positions</th>
<th>three positions</th>
</tr>
</thead>
</table>
| \[
\begin{array}{c}
0 \\
1 \\
\end{array}
\]
| \[
\begin{array}{c}
0 \\
0 \\
0 \\
1 \\
1 \\
1 \\
1 \\
1 \\
\end{array}
\]
| \[
\begin{array}{c}
0 \\
0 \\
0 \\
0 \\
0 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
\end{array}
\]  |

two numbers

four numbers

eight numbers
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.

\[
\begin{array}{cccccccc}
2^7 & = & 128 & \quad & 2^6 & = & 64 & \quad & 2^5 & = & 32 \\
2^4 & = & 16 & \quad & 2^3 & = & 8 & \quad & 2^2 & = & 4 \\
2^1 & = & 2 & \quad & 2^0 & = & 1 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
1 & 0 & 0 & 0 & \quad & 1 & 0 & 1 & 0
\end{array}
\]

\[
1 \cdot 128 + 0 \cdot 64 + 0 \cdot 32 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 138_{10}
\]

\[
1 \cdot 128 + 1 \cdot 8 + 1 \cdot 2 = 138_{10}
\]
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*

\[
\begin{align*}
16 \times 16 \times 16 & = 4096 \\
16 \times 16 & = 256 \\
16 & = 16 \\
1 & = 1 \\
\hline
0 & 0 & 8 & A \\
\hline
\end{align*}
\]

- base 10
- base 16

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

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hexadecimal</th>
<th>Decimal</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

\begin{align*}
  100000100000010000111101000100001111_2 &= 8207A10F_{16} \\
  10000011010001010110100110111110_2 &= \underline{\text{15217C5F}}_{16}
\end{align*}