Digital Systems

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Why Do You Need To Take This Course

- Abstraction is good, but don’t forget reality
  - Most CSE courses emphasize abstraction
    - Abstract data types
    - Asymptotic analysis
  - These abstractions have limits
    - In performance optimization
    - In detection and elimination of bugs
Great Reality #1

• Ints are not Integers, Floats are not Reals
  
  – Example 1: Is $x^2 \geq 0$?
    
    • Float’s: Yes!
    
    • Int’s:
      
      – $40000 \times 40000 \rightarrow 1600000000$
      
      – $50000 \times 50000 \rightarrow ?$
  
  – Example 2: Is $(x+y)+z = x+(y+z)$?
    
    • Unsigned & Signed Int’s: Yes!
    
    • Float’s:
      
      – $(1e20 + -1e20) + 3.14 \rightarrow 3.14$
      
      – $1e20 + (-1e20 + 3.14) \rightarrow ??$
Computer Arithmetic

• Does not generate random values
  – Arithmetic operations have important mathematical properties

• Cannot assume all “usual” mathematical properties
  – Due to finiteness of representations
  – Integer operations satisfy “ring” properties
    • Commutativity, associativity, distributivity
  – Floating point operations satisfy “ordering” properties
    • Monotonicity, values of signs

• Observation
  – Need to understand which abstractions apply in which contexts
  – Important issues for compiler writers and serious application programmers
Great Reality #2

• You’ve Got to Know Assembly
  – Chances are, you’ll never write programs in assembly
    • Compilers are much better & more patient than you are
  – But: Understanding assembly is key to machine-level execution model
    • Behavior of programs in presence of bugs
      – High-level language models break down
    • Tuning program performance
      – Understand optimizations done / not done by the compiler
      – Understanding sources of program inefficiency
    • Implementing system software
      – Compiler has machine code as target
      – Operating systems must manage process state
    • Creating / fighting malware
      – x86 assembly is the language of choice!
Great Reality #3

• **Random Access Memory Is an Unphysical Abstraction**
  
  – Memory is not unbounded
    • It must be allocated and managed
    • Many applications are memory dominated
  
  – Memory referencing bugs especially pernicious
    • Effects are distant in both time and space
  
  – Memory performance is not uniform
    • Cache and virtual memory effects can greatly affect program performance
    • Adapting program to characteristics of memory system can lead to major speed improvements
Memory Referencing Bug Example

```c
typedef struct {
    int a[2];
    double d;
} struct_t;

double fun(int i) {
    volatile struct_t s;
    s.d = 3.14;
    s.a[i] = 1073741824; /* Possibly out of bounds */
    return s.d;
}
```

- fun(0) --> 3.14
- fun(1) --> 3.14
- fun(2) --> 3.1399998664856
- fun(3) --> 2.00000061035156
- fun(4) --> 3.14
- fun(6) --> Segmentation fault

– Result is system specific
Memory Referencing Errors

• C and C++ do not provide any memory protection
  – Out of bounds array references
  – Invalid pointer values
  – Abuses of malloc/free

• Can lead to nasty bugs
  – Whether or not bug has any effect depends on system and compiler
  – Action at a distance
    • Corrupted object logically unrelated to one being accessed
    • Effect of bug may be first observed long after it is generated

• How can I deal with this?
  – Program in Java, Ruby, Python, ML, …
  – Understand what possible interactions may occur
  – Use or develop tools to detect referencing errors (e.g. Valgrind)
Great Reality #4

• There’s more to performance than asymptotic complexity

```c
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (i = 0; i < 2048; i++)
        for (j = 0; j < 2048; j++)
            dst[i][j] = src[i][j];
}
```

```c
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (j = 0; j < 2048; j++)
        for (i = 0; i < 2048; i++)
            dst[i][j] = src[i][j];
}
```

4.3ms

81.8ms

2.0 GHz Intel Core i7 Haswell

• Hierarchical memory organization

• Performance depends on access patterns
  – Including how step through multi-dimensional array
Great Reality #5

• Computers do more than execute programs
  – They need to get data in and out
    • I/O system critical to program reliability and performance

  – They communicate with each other over networks
    • Many system-level issues arise in presence of network
      – Concurrent operations by autonomous processes
      – Coping with unreliable media
      – Cross platform compatibility
      – Complex performance issues
Digital Systems
Introduction

• The advent of the digital age
  – Analog vs. digital?

– Compact Disc (CD)
  • 44.1 KHz, 16-bit, 2-channel

– MP3
  • A digital audio encoding with lossy data compression
Representing Information

• Information = Bits + Context
  – Computers manipulate representations of things
  – Things are represented as binary digits
  – What can you represent with $N$ bits?
    • $2^N$ things
    • Numbers, characters, pixels, positions, source code, executable files, machine instructions, …
  • Depends on what operations you do on them

<table>
<thead>
<tr>
<th>(char)</th>
<th>01110011 01101011 01101011 01110101 01110011 01100101 01101101 01101001</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>‘s’</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(int)</th>
<th>1969974131</th>
</tr>
</thead>
<tbody>
<tr>
<td>(double)</td>
<td>7.03168990329170808178... x 10^{99}</td>
</tr>
</tbody>
</table>
Binary Representations

• Why not base 10 representation?
  – Easy to store with bistable elements
  – Straightforward implementation of arithmetic functions
  – Reliably transmitted on noisy and inaccurate wires

• Electronic implementation
Encoding Byte Values

- **Binary:** $00000000_2$ to $11111111_2$
- **Octal:** $000_8$ to $377_8$
  - An integer constant that begins with 0 is an octal number in C
- **Decimal:** $0_{10}$ to $255_{10}$
  - First digit must not be 0 in C
- **Hexadecimal:** $00_{16}$ to $FF_{16}$
  - Base 16 number representation
  - Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
  - Write FA1D37B$_{16}$ in C as $0xFA1D37B$ or $0xFA1D37B$
Boolean Algebra

• Developed by George Boole in 1849
  – Algebraic representation of logic
  – Encode “True” as 1 and “False” as 0

▪ AND
  • A&B = 1 when both A=1 and B=1

\[
\begin{array}{c|cc}
& 0 & 1 \\
\hline
0 & 0 & 0 \\
1 & 0 & 1 \\
\end{array}
\]

▪ OR
  • A|B = 1 when either A=1 and B=1

\[
\begin{array}{c|ccc}
& 1 & 0 & 1 \\
\hline
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

▪ NOT
  • \sim A = 1 when A=0

\[
\begin{array}{c|c}
& \\
\hline
0 & 1 \\
1 & 0 \\
\end{array}
\]

▪ XOR (Exclusive-OR)
  • A^B = 1 when either A=1 or B=1, but not both

\[
\begin{array}{c|ccc}
^ & 0 & 1 \\
\hline
0 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]
General Boolean Algebra

• Operate on bit vectors
  – Operations applied bitwise

01101001 & 01010101 = 01000001
01101001 | 01010101 = 01111101
01101001 ^ 01010101 = 00111100
~ 01010101 = 10101010

• All of the properties of Boolean Algebra apply
Digital Logic

Basic operations: AND(&), OR(|), NOT(~)

\[
X \land Y = (X \land \neg Y) \lor (\neg X \land Y)
\]

\[
X \implies Y = \neg X \lor Y
\]

A complete set: NAND = \(\neg (X \land Y)\)
Combinational Logic

• Adder

Full Adder

4-bit Ripple Carry Adder
Sequential Logic

• Flip-flops

Edge triggered D flip-flop

clock

4-bit register

Shifter
Transistors

- Transistor = Electronic switch
  - Controlled by voltages
    - e.g. Logic 1 = 5V, Logic 0 = 0V
  - NMOS transistor
NOT

• NOT logic built with NMOS technology
NAND

- NAND logic built with NMOS technology
CMOS NOT

• NOT logic built with CMOS technology
CMOS NAND

• NAND logic built with CMOS technology
Summary

• Boolean algebra is a mathematical foundation for modern digital systems

• Boolean algebra provides an effective means of describing circuits built with switches
  – Claude Shannon’s master thesis in 1939

• You can build any digital systems with NAND gates

• A NAND gate can be easily built with CMOS transistors

• The transistor is the basic building block for digital systems