Introduction

- Rapidly changing field:
  - vacuum tube -> transistor -> IC -> VLSI
  - memory capacity and processor speed is doubling every 1.5 years:
- Things you’ll be learning:
  - Foundation of computing, design methodologies, issues in design
  - how to analyze their performance (or how not to!)
- Why learn this stuff?
  - You want to design state-of-art system
  - you want to call yourself a “computer scientist or engineer”
  - you want to build software people use (need performance)
  - you need to make a decision or offer “expert” advice

What is computing?

- In 1960, “computer” was still understood to be a person
  - A person who could compute
- By contrast, a recent dictionary begins the definition as
  - “A “computer” is “An electronic machine...”
- But computing has had many abstraction
- We would learn about some of them today

Consider An Example: Example 1

- Let us evaluate an expression
  - \( A=B+C+D+E\times F \)
- It can also be written as
  - \( A=(B+C)+D+E\times F \)
  - \( A=(B+C+D)+E\times F \)
  - \( A=(B+C+D)+(E\times F) \)
  - But are these correct?
    - \( A=(B+C+D+E)\times F \)
    - \( A=B+(C+D)+E\times F \)
- Depends on what are the rules for evaluating expressions
- What are we computing?
- What is the model?

What is a computing abstraction?

- Consider computation a simple expression
  - \( A=B+C \)
- What do we need to do to compute?
  - Need storage for \( B \)
  - Need storage for \( C \)
  - Multiply
  - Need storage for \( A \)
  - How would you do it on your calculator?
- What if you do not have multiplier?
- But you have black boxes that compute, add, log/alog
  - \( \log A = \log B + \log C \)
- It is a functional transformation
- How do we achieve the computation? Put the blocks together

Consider Another Example: Example 2

- Consider the computation \( Y = X^3 - X^2 + X - 1 \)
- How many operations?
  - How many multiply?
  - How many adds/subs?
  - How many storage?
  - Is this the best we can do?
- How do we achieve efficiency is computation?

A Possible Solution: Example 2

- How many operations?
  - How many multiply? 2
  - How many adds/subs? 3
  - How many storage?
  - How much time?
- Is this the best we can do?
  - For multiplication
  - Probably we can argue
  - What about adds/subs?
- This is not very efficient
Another Possible Solution: Example 2

- Simplify the function by factorization?
  - How many multiply? 2
  - How many adds/subs? 2
  - How many storage?
  - How much time?

- Is this the best we can do?
  - For *, probably we can argue
  - What about adds/subs?

- Another factorization does not change number of operations

Consider One More Example: Example 3

- Consider the computation
  - No constraints on values
  - How many operations?
    - How many multiply?
    - How many adds/subs?
    - How much storage?

- Is this the best we can do?

- How do we achieve efficiency in computation?

A Possible Solution: Example 3

- How many operations?
  - How many multiply? 6, but one can be saved easily
  - How many adds/subs? 3
  - How many storage?
  - How much time?

Another Way to Solution: Example 3

- We can view the computation differently
  - Why this form?
    - Provides a building block for computation
    - But has each block big, 4 inputs, 2 outputs

Yet Another Way to Solution: Example 3

- We can look at the expression differently
  - How many operations?
  - Why this form?
    - Provides a way to optimize and provides a building block

Point of Discussion

- A good computation structure require some thinking
- Optimize on hardware design cost
- Optimize on time for computation
- There may be a tradeoff that needs to be explored
- Identify common building blocks that can be implemented and used to realize interesting computations
- Always consider
  - How many operations?
  - How many time steps?
  - What is the tradeoff?
  - Solutions may not be obvious
Computing with a designed Machine

• Consider computation in example 1 (An user may like to directly say this as is)
  – A=B*C
• A given machine has facility to load variables and perform arithmetic and complex functions (who designed it?)
• So how do we compute?
• Here is a conceptual program
  – Load B, mem1
  – Load C, mem2
  – Multiply mem1, mem2, mem3
  – Store A, mem3
• On your simple calculator
  – Key in value of B
  – Press multiply
  – Key in value of C
  – Press = and Read A out

Program Example 2

• Required computation is
  \[ Y = X^2 - X^2 + X - 1 \]
• A complex program may look like
  – X = value
  – \[ Y = X^3 - X^2 + X - 1 \]
• A simple program may look like
  – Load X, mem1
  – Multiply mem1, mem1, mem2
  – Multiply mem1, mem2, mem3
  – Sub mem3, mem2, mem4
  – Add mem4, mem1, mem5
  – Load #1, mem6
  – Sub mem5, mem6, mem7
  – Store Y, mem7
• Do we need all these memory locations?

Program Example 2 Differently

• Factorized function is
  \[ Y = X^3 - X^2 + X - 1 = (X-1)(X^2 + X + 1) \]
• A simple program may look like

Now Consider Our Complex Example – 3

• Required computation is
  \[ Y = k_0 + k_1 X + k_2 X^2 + k_3 X^3 \]
• How do we approach this
• We need some structure to store variables
  – An array structure k[i], i = 0, 1, 2, 3, ...
  – A variable name X
  – Store powers of X, i.e., X^i in xpower[i], i = 0, 1, 2, 3, ...
  – A result location Y
  – X is given by user
  – K[i] is filled in by user
  – Y is initially zero
• Partial Y computation is, \[ Y = k[0] \]
• Also, xpower[i] = 1
• At each step i = 1, 2, 3, ... we have 3 inputs and 2 outputs
  • we take xpower[i-1], partial result Y, and k[i]
  • And compute xpower[i] and a new partial result Y

Now Program Example – 3 Using Alternate

• What is the big difference?
• Block is simple, but need to start from other end

Differences Between the two Programs

• First approaches computes a 3-inputs, 2-outputs function
• The second one uses a 3-input, 1-output function
• Mathematically that is how we prefer to write functions
• First method can be used for successive addition of term
• The second method requires us to know how many terms
Computing Functions: Difference Engine

- Consider the computation Y = X^3 – X^2 + X – 1
- Consider the table
- What is going on each row
- Can you name each row?
- Can you tell how an entry in a row is computed?

<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>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td></td>
<td>-5</td>
<td>-9</td>
<td>-13</td>
<td>-17</td>
<td>-21</td>
<td>-25</td>
<td>-29</td>
<td>-33</td>
<td>-37</td>
<td>-41</td>
<td>-45</td>
<td>-49</td>
</tr>
<tr>
<td>-1</td>
<td>-4</td>
<td>-9</td>
<td>-14</td>
<td>-19</td>
<td>-24</td>
<td>-29</td>
<td>-34</td>
<td>-39</td>
<td>-44</td>
<td>-49</td>
<td>-54</td>
<td>-59</td>
<td>-64</td>
</tr>
<tr>
<td>-1</td>
<td>-4</td>
<td>-9</td>
<td>-14</td>
<td>-19</td>
<td>-24</td>
<td>-29</td>
<td>-34</td>
<td>-39</td>
<td>-44</td>
<td>-49</td>
<td>-54</td>
<td>-59</td>
<td>-64</td>
</tr>
<tr>
<td>-1</td>
<td>-4</td>
<td>-9</td>
<td>-14</td>
<td>-19</td>
<td>-24</td>
<td>-29</td>
<td>-34</td>
<td>-39</td>
<td>-44</td>
<td>-49</td>
<td>-54</td>
<td>-59</td>
<td>-64</td>
</tr>
<tr>
<td>-1</td>
<td>-4</td>
<td>-9</td>
<td>-14</td>
<td>-19</td>
<td>-24</td>
<td>-29</td>
<td>-34</td>
<td>-39</td>
<td>-44</td>
<td>-49</td>
<td>-54</td>
<td>-59</td>
<td>-64</td>
</tr>
</tbody>
</table>

Difference Engine Abstraction

- Suppose you want to calculate y = sin(x)
- Need a Sin calculator
  - Looks cheap on your calculator, it is expensive computation
  - How would you go about it?
- Consider a Taylor series expansion
  - y = sin(x) = x – x^3/3! + x^5/5! – x^7/7! + ……
- Based on computing differences, a finite n-th order polynomial can be differentiated n times, which can be represented by a difference
- What degree polynomial is sufficient?
  - Depends on accuracy needed (we will visit that many times)
- Let us consider only two terms:
  - y = sin(x) = x – x^3/3!

Calculating using Difference Engine

- To compute value of sin(x) at x(0), x(1), x(2), x(3), x(4), x(5), ………
  - ∆x = x(i+1) – x(i)
  - y(x(i)) = sin(x(i)) = x(i) – x^3(i)/3!
- For simplicity, we will drop () and denote the corresponding values of y also as y0, y1, y2, y3, …..
- We can calculate y0, y1, y2, and y3 by hand and also call them ∆y0, ∆y1, ∆y2, and ∆y3, respectively
- Why are we doing it?
  - That forms the basis of difference engine abstraction

Difference Engine (cond.)

- If we differentiate the function, forth differentiation will yield a 0
- What about the third differentiation?
  - A constant (value is -1 in this case)
  - And others can be calculated as well
- First order difference can be written as
  - ∆y0 = y1-y0; ∆y1 = y2-y1; ∆y2 = y3-y2
- Second order difference can be written as
  - ∆^2y0 = ∆y1 - ∆y0 = y2-2y1+y0
  - ∆^2y1 = ∆y2 - ∆y1 = y3-2y2+y1
- Third order difference can be written as
  - ∆^3y0 = ∆^2y1 - ∆^2y0 = y3-3y2+3y1-y0
- And the forth order difference is ∆^4y0 = 0
- Suppose we know ∆^3y0, ∆^2y0, ∆y0, and ∆y0
  - Using this we can recursively compute ∆y1, ∆^2y1, and ∆^3y1, and ∆^4y1
  - And then all y2 and y3, and y4……..

Difference Engine Example

<table>
<thead>
<tr>
<th>IN:</th>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT:</td>
<td>y0</td>
<td>y1</td>
<td>y2</td>
<td>y3</td>
<td>y4</td>
<td>y5</td>
<td>y6</td>
</tr>
<tr>
<td>0th Diff:</td>
<td>∆y0</td>
<td>∆y1</td>
<td>∆y2</td>
<td>∆y3</td>
<td>∆y4</td>
<td>∆y5</td>
<td>∆y6</td>
</tr>
<tr>
<td>1st Diff:</td>
<td>∆^2y0</td>
<td>∆^2y1</td>
<td>∆^2y2</td>
<td>∆^2y3</td>
<td>∆^2y4</td>
<td>∆^2y5</td>
<td>∆^2y6</td>
</tr>
<tr>
<td>2nd Diff:</td>
<td>∆^3y0</td>
<td>∆^3y1</td>
<td>∆^3y2</td>
<td>∆^3y3</td>
<td>∆^3y4</td>
<td>∆^3y5</td>
<td>∆^3y6</td>
</tr>
<tr>
<td>3rd Diff:</td>
<td>∆^4y0</td>
<td>∆^4y1</td>
<td>∆^4y2</td>
<td>∆^4y3</td>
<td>∆^4y4</td>
<td>∆^4y5</td>
<td>∆^4y6</td>
</tr>
</tbody>
</table>

In general
- ∆^(n)(y(i)) = ∆^(n-1)(y(i)) for rth order function and
  - ∆^(n)(y(i)) = ∆(y(n+1) – ∆(y(i)) for j = 0, 1, 2, … n-1, and i = 0, 1, 2, …. Then we can compute second column and so on
- The structure need n+1 memories (to store a column) and n adders
- One can also write a C program to compute a column at a time
  - And the first column is obtained by calculating values by hand
Decimal System

- We are all familiar with decimal numbers
- Consider a number 2375
- What digits representing thousand, hundred, ten and one's place
- How did you get it?
- Give me an algorithm
  - Divide by 1000, result is thousand place value
  - Subtract 1000*thousand place value
  - Divide by 100, result is hundred place value
  - Subtract 100*hundred place value
  - Divide by 10, result is ten place value
  - Subtract 10*ten place value
  - Remainder is one place value
- What is good about this algorithm
- What is bad about it?

An Easier Algorithm

- Divide by 10
- Remainder is one place value
- Divide the result by 10
- Remainder is ten place value
- Divide the result by 10
- Remainder is hundred place value
- Divide the result by 10
- Remainder is thousand place value
- Any time result is zero, that means no more value
- Division is always by 10
- We always need result and remainder

Any Base b Algorithm

- Divide by b
- Remainder is one place value
- Divide the result by b
- Remainder is ten place value
- Divide the result by b
- Remainder is hundred place value
- Divide the result by b
- Remainder is thousand place value
- Any time result is zero, that means no more value
- Division is always by b
- Remainder is always between 0 and b-1

Information Representation

- Information theory: discusses how to deal with information
- We only deal with some aspects of it
- Virtually all computers now store information in binary form
- A binary number system has two digits, 0 and 1
- Combination of binary digits represent various kind of information
- Examples
  - 01001011
  - It can be interpreted as an integer value, a character code, a floating point number....
- Non binary numbers are also possible
- How do we represent negative numbers?
  - i.e., which bit patterns will represent which numbers?

Abstraction

- Delving into the depths reveals more information
- An abstraction omits unneeded detail, helps us cope with complexity
- What are some of the details that appear in these familiar abstractions?

Historical Perspective

- 1642 Pascal: Mechanical Computer
- 1671: Gottfried Leibniz ADD/SUB/MUL/DIV
- 1801: Automatic Control of Weaving Process
- 1827 The Difference Engine by Charles Babbage
- 1936: Zuse Z1: electromechanical computers
- 1941: Zuse Z2
- 1943: Zuse Z3
- 1944: Aiken: Ark 1 at Harvard
- 1942-45: ABC at Iowa State (Atanasoff-Berry Computer)
- 1946: ENIAC: Eckert and Mauchley: Vacuum Tube
- 1945 EDVAC by von-Neumann machine, father of modern computing
Why Binary?

- Easy to represent
  - Off and On
  - Open and close switch
  - Head and tail on a coin
  - Polarity of magnetization
  - 0 and nonzero voltage levels
- How to represent information in binary?
  - Say we want to represent positive number 0 and 1
    - 0 is 0 and 1 is 1
  - Say we want to represent red and green colors
    - 0 is red and 1 is green (or vice versa)
  - Say we want to represent fall and spring semesters
    - 0 is fall and 1 is spring (or vice versa)

More Complicated Examples

- Numbers 0 to 7
  - We use combination of digits
    - 1 digits gives us two combination
    - 2 will yield four
    - 3 will yield 8
  - Need three bits (binary digits)
- What if we want to represent 16 alphabets - Need four bits
- What if we want to represents numbers from 11 to 25?
- Homework Problem:
  - For each part below devise a scheme to represent, in binary, each set of symbols
    - (A) Numbers: 0, 1, 2, 3, 4, 5, 6, 7
    - (B) Alphabets: A, B, C, D, E, F
    - (C) Integers from 21 to 36

Bits and Combinations

<table>
<thead>
<tr>
<th># of Bits</th>
<th># of quantities</th>
<th>• What happens in other number systems?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>In base b, n digits give b^n combinations</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Base 10: decimal</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>Base 8: Octal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Base 16: Hexadecimal</td>
</tr>
<tr>
<td>n</td>
<td>2^n</td>
<td></td>
</tr>
</tbody>
</table>

Representation of Positive Numbers

- Positional value
- Binary digits are numbered
- Right most digit is 0
- Next to that is a 1
- And so on up to n-1 in n-bit representation
- Decimal point is implied at the right of bit 0
- Each bit is assigned a weight
- The weight of i-th bit is 2^i
- Using this notation
  - The value of an n-bit sequence is
  - \[ x_n 2^n + x_{n-1} 2^{n-1} + \ldots + x_1 2^1 + x_0 2^0 \]
  - \[ = \sum_{i=0}^{n} 2^i x_i \]

Some Examples

- Convert 0101 into decimal
  - Position: 3 2 1 0
  - Weight: 8 4 2 1
  - Digits: 0 1 0 1
  - Decimal value: \[ 8 \cdot 0 + 4 \cdot 1 + 2 \cdot 0 + 1 \cdot 1 = 5 \]
- Convert 1010101 into decimal
  - Position: 7 6 5 4 3 2 1 0
  - Weight: 128 64 32 16 8 4 2 1
  - Digits: 1 0 1 1 0 1 0 1
  - Decimal value: \[ 128 + 64 + 32 + 16 + 8 + 0 + 2 + 0 + 1 + 1 + 0 + 1 + 0 + 1 = 251 \]
  - Now try 1000000
  - And try 01111111

And What About the Reverse Operation?

- First see the largest weight of a binary positional digit contained in the number
- Put that binary digit = 1 and subtract weight
- Then try subtracting the next bit’s weight
- If successful
  - next bit is 1, else next bit is 0 (and restore the value)
- Repeat the last two steps until done
- Convert decimal number 181 into binary
- Largest weight is 128, subtract 128 and set bit 7 = 1
- Try subtracting 64 out of remainder 53 (181-128)
- No successful, so the next digit is 0
- Try weight 32, 16, 8, 4, 2, and 1 successively
- Number is 1 0 1 1 0 1 0 1
A Simpler Method

- Convert decimal number 181 into binary
- Start dividing by 2
- Successive remainders are digits from right
- \(181/2 = 90\) remainder 1
- \(90/2 = 45\) remainder 0
- \(45/2 = 22\) remainder 1
- \(22/2 = 11\) remainder 0
- \(11/2 = 5\) remainder 1
- \(5/2 = 2\) remainder 1
- \(2/2 = 1\) remainder 0
- \(1/2 = 0\) remainder 1
- Number is 1 0 1 1 0 1 0 1

And Now Try Some Problems

- Suppose you want to represent positive integers in binary.
- Indicate how many bits are required to represent each of the following sets of integers:
  - (1) The integers from 0 to 127 inclusive
  - (2) The integers from 0 to 2,048 inclusive
  - (3) The integers from 0 to 32,500 inclusive
  - (4) The integers from 0 to 1,500,345 inclusive
- Indicate how large a value can be represented by each of the binary quantities: A (1) 4-bit, (2) 12-bit, and (3) 24-bit quantity.
- Convert each of the following binary digits into decimal. Assume these quantities represent unsigned integers.
  - (1) 1010;  (2) 10010; (3) 0111110; (4) 10000000; (5) 0111111
- Convert each of the following decimal numbers into binary.
  - (1) 6; (2) 13; (3) 111; (4) 147; (5) 511

Base ‘b’ number

- In general a number system can have any base \(b\)
- the digit used are 0, 1, ..., \(b\)-1
- The weight of \(i^{th}\) place is \(b^i\)
- The conversion formula from base \(b\) into decimal number is
  \[\sum_{i=0}^{n-1} b^i x_i\] for \(i = 0\) to \(n - 1\) for an \(n\) digit quantity
- Commonly used base are 2, 3, 8, 10, 16, ...

Bases 2, 8, and 16 are related

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Octal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
<td>12</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
<td>13</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
<td>14</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
<td>15</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
<td>16</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>17</td>
<td>F</td>
</tr>
</tbody>
</table>

Conversion

- From binary to octal
  - make groups of 3 bits from right to left
    \(01 110 110_2 = 166_8\)
- From octal to binary
  - make each digit as 3 bits sequence
    \(276_8 = 010 111 110_2\)
- From binary to hexadecimal
  - make groups of 4 bits from right to left
    \(0111 0110_2 = 76_{16}\)
- From hexadecimal to binary
  - make each digit as 4 bits sequence
    \(37_{16} = 0011 0111_2\)

Signed numbers

- Positive numbers are well understood
- An \(n\)-bit number represents numbers from 0 to \(2^n-1\)
- \(n+m\) bits can be used to represent \(n\)-bit integer and \(m\)-bit fraction of a number
- However negative numbers cause another problem
- In all solutions, one bit is needed to represent the sign, + or -
- MSB can be used for that purpose, i.e., represent sign
- Remaining bits can be interpreted differently
  - They can represent magnitude as a positive number
  - They can be complemented (represent 0 by 1 and 1 by 0)
  - Or manipulate in some other way