CS 241 Part 1

Introduction and Binary Encoding
With thanks to Brad Lushman, Troy Vasiga and Kevin Lanctot
About the course

- [www.student.cs.uwaterloo.ca/~cs241](http://www.student.cs.uwaterloo.ca/~cs241)
  - Read the Syllabus (policies, due dates, outline, etc.)
  - Read the Announcements
  - Read everything else on the main webpage
  - Make sure you can get on Piazza!

- Assignments
  - Start assignments early!!! Don’t fall behind!!!
  - 10 assignments in total with many subparts.
  - A1 due Monday January 21st at 5:00pm
  - A2-10 due on Fridays at 5:00pm
  - Partial credit possible for late assignments: read the syllabus

- I will try to make a slide deck for this course. If I am able to find the time, these will be on my CS 241 webpage accessible from my personal website: [https://cs.uwaterloo.ca/~cbruni/CS241Resources/index.php](https://cs.uwaterloo.ca/~cbruni/CS241Resources/index.php)

- **Do not** rely on only these throughout the term (I might get busy and not be able to keep up).
Marking

- Assignments: 25%
- Midterm: 25% written on Wednesday, March 6th, 4:30-6:20pm
- Final Exam: 50% written sometime in April
- You must pass the weighted exam average to pass the course otherwise your final average is your exam average.
Marmoset

- Public tests (aka “sanity tests”)

- Release tokens
  - Three tokens for each “part” of each assignment
  - Once one is used, it regenerates after 12 hours

Starting early maximizes your chances of success on your assignments!

Your program must run correctly on the `linux.student.cs` environment.
Personnel

- Instructors:
  - Carmen Bruni (cbruni@uwaterloo.ca)
  - Mark Petrick (mdtpetri@uwaterloo.ca)

- ISAs (See the webpage for more details):
  - Frank Wang (cs241@uwaterloo.ca)
  - Edward Tan (cs241@uwaterloo.ca)
  - Pierre-Louis Guidez (cs241@uwaterloo.ca)

- Instructional Support Coordinator: Gang Lu (glu@uwaterloo.ca)

- IAs/TAs: run tutorials
Other Resources

• Textbooks: optional texts available in DC library. See the webpage for more details.
• Discussion Forum: Piazza
  • Rule 1: Piazza is not Reddit. Be courteous.
  • Rule 2: Post questions in the appropriate folders.
  • Rule 3: Read first, search second, post last
• Clickers: I will use them in class. They won’t count for grades. If you have one, please bring it to class (if not don’t go and buy one just for this class but do still participate).
Purpose of the course

• Assemble a compiler for a ‘watered language’.
• MIPS - Microprocessor without Interlocked Pipelined Stages (Software - not hardware - must deal with data/control/structural hazards).
• Write a program that reads a program and outputs a program.
• Most fundamentally, this course is about *abstraction*. 
What’s in a name?

Foundations of Sequential Programs

- What is a sequential program? (single-threaded; not concurrent or parallel)
- What really happens when I compile and run a program?
- How does a computer take code and turn it into something it can utilize?
- By the end of the course, there should be very little mystery left about computers or computer programs.
Basic Definitions

**Definition**
A **bit** is a **binary digit**, that is a 0 or 1 (on or off).

**Example:** 1001.

**Definition**
A **nibble** is 4 bits.

**Example:** 1001.

**Definition**
A **byte** is 8 bits.

**Example:** 10011101.
### Definition

A **word** is a machine-specific grouping of bytes. For us, a word will be 4 bytes (32-bit architecture) though 8 byte (or 64-bit architectures) words are more common now.

Example: 10011101100111011001110110011101.

It can be hard to read words in binary. Can we make the notation more compact?
Hexadecimal Notation

**Definition**
A **word** is a machine-specific grouping of bytes. For us, a word will be 4 bytes (32-bit architecture) though 8 byte (or 64-bit architectures) words are more common now.

Example: 10011101100111011001110110011101.

It can be hard to read words in binary. Can we make the notation more compact? Yes!
Hexadecimal Notation

Definition

The base-16 representation system is called the hexadecimal system. It consists of the numbers from 0 to 9 and the letters a, b, c, d, e and f (which convert to the numbers from 10 to 15 in decimal notation.

Example: The binary number 10011101 will convert to 9d in hexadecimal.

- Sometimes we denote the base with a subscript like 10011101_2 and 9d_{16}.
- Also, for hexadecimal, you will routinely see the notation 0x9d. (The 0x denotes a hexadecimal representation in computer science).
- Note that each hexadecimal character is a nibble (4 bits).
## Conversion Table

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Hex</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<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>10</td>
<td>a</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
<td>b</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
<td>c</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
<td>d</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
<td>e</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>f</td>
</tr>
</tbody>
</table>

Note: Upper case letters are also used for hexadecimal notation. Context should make things clear.
Binary Numbers - What Is It Good For?

What do bytes represent?
Binary Numbers - What Is It Good For?

What do bytes represent?
- Numbers (but what number?)
- Characters (but what character?)
- Garbage in memory
Binary Numbers - What Is It Good For?

What do bytes represent?

- Numbers (but *what* number?)
- Characters (but *what* character?)
- Garbage in memory
- Instructions! (Parts of instructions in our case. Words, or 4 bytes, will correspond to a complete instruction for our computer system).
Bytes as Binary Numbers

We will discuss two types:

- Unsigned (non-negative integers)
- Signed integers

However there are many others (floating point, algebraic, etc.)
Unsigned Integers

This is a positional number system that works like a normal binary system.

The value of a number stored in this system is the binary sum, that is

\[ b_72^7 + b_62^6 + b_52^5 + b_42^4 + b_32^3 + b_22^2 + b_12^1 + b_0 \]

For example,

\[ 01010101_2 = 2^6 + 2^4 + 2^2 + 2^0 = 64 + 16 + 4 + 1 = 85_{10} \]

or

\[ 11111111_2 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 \\
= 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 \\
= 255_{10} \]
Unsigned Integers

Arithmetic is done in the ordinary way:

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
+ & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}
\]

Watch out for overflow errors!
Converting to Binary

• Question: Write 38 in binary.

One way: Take the largest power of 2 less than 38, subtract and repeat.

For example, 32 is the largest power of two less than 38, subtracting gives 6. Next, 4 is the largest power of two less than 6 and subtracting gives 2. This is a power of two hence $38 = 32 + 4 + 2 = (100110)_2$.

Another way is to constantly divide by 2:

<table>
<thead>
<tr>
<th>Number</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

..and in binary (reading bottom to top) this is $(100110)_2$.
Converting to Binary

- **Question:** Write 38 in binary.
- **One way:** Take the largest power of 2 less than 38, subtract and repeat.
- For example, 32 is the largest power of two less than 38, subtracting gives 6. Next, 4 is the largest power of two less than 6 and subtracting gives 2. This is a power of two hence $38 = 32 + 4 + 2 = (100110)_2$.
- **Another way is to constantly divide by 2:**

<table>
<thead>
<tr>
<th>Number</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- ..and in binary (reading bottom to top) this is $(100110)_2$. 
Consider:

\[ N = b_0 + 2b_1 + 2^2b_2 + ... \]

The remainder when dividing \( N \) by 2 gives the \( b_0 \) value. After doing \((N - b_0)/2\), we end up with

\[ \frac{N - b_0}{2} = b_1 + 2b_2 + 2^2b_3 + ... \]

and we can repeat the process. (This is why we have to read bottom up).
Signed Integers

How to we represent negative integers?

Attempt 1: Make the first bit a signed bit. This is called the “sign-magnitude” representation.

Problems:
Signed Integers

How to we represent negative integers?

Attempt 1: Make the first bit a signed bit. This is called the “sign-magnitude” representation.

Problems:

- Two representations of 0 (wasteful and awkward).
- Arithmetic is tricky. Is the sum of a positive and negative number positive or negative? It depends!
Signed Integers

Attempt 2: **Two’s complement form**

- Similar to sign-magnitude in spirit.
- To interpret a binary number in this form, convert the number as a signed integer, if the leading bit is 0 stop. Otherwise, subtract $2^n$ where $n$ is the number of bits.
- To negate a value:
  1. Take the complement of all bits
  2. Add 1
- This will ultimately mean that the first bit is a sign bit (0 if non-negative 1 if negative)
Natural Definition

**Definition**
The **Most Significant Bit (MSB)** is the left-most bit (highest value/sign bit).
The **Least Significant Bit (LSB)** is the right-most bit (highest value/sign bit).
An Example

Let’s compute $-38_{10}$ using this notation in one byte of space.
An Example

Let’s compute $-38_{10}$ using this notation in one byte of space. First, write 38 in binary:

$$38_{10} = 00100110$$

Next, take the complement of all the bits

$$11011001$$

Finally, add 1:

$$11011010$$

This last value is $-38_{10}$. We can convert the binary number by performing

$$2^7 + 2^6 + 2^4 + 2^3 + 2^1 - 2^8 = 128 + 64 + 16 + 8 + 2 - 256 = 218 - 256 = -38$$
Short Cut

• A slightly faster way is to locate the rightmost 1 bit and flip all the bits to the left of it.

• For example:

\[
11011010 \quad \text{Negating} \quad 00100110
\]

Note: Flipping the bits and adding 1 is the same as subtracting 1 and flipping the bits (exercise).
How Signed Integers Work

- Everything works mod $2^n$.
- The negation of a positive integer $k$ is represented in memory as $2^n - k$ where $n$ is the size of the data type.
- Arithmetic works naturally except that any final carry overs are ignored (see the two examples below).
- For a few examples, to add 4 and $-3$ on the left in a 4 bit system or adding $-4$ and $-3$ on the right, we have

  \[
  \begin{array}{c}
  1111 \ 1 \\
  0000 \ 0100 \ (+4) \ \\
  + \ 1111 \ 1101 \ (-3) \ \\
  \hline
  0000 \ 0001
  \end{array}
  \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \qua