15213 Recitation Fall 2014

Section A, 8th September

Vinay Bhat
Welcome to a fun course!

• Yes, it really is fun!
• Recitation will be a place for …
  … a quick overview of the last week’s lectures.
  … re-iteration of some key concepts.
  … some problem solving (if needed).
  … lab related discussion.
• Ask questions if you have – from textbooks or otherwise
• We will get back if we cannot answer immediately (yes, we may not know everything!)
• Let us know if you want anything else – we will try to squeeze it in
Agenda

• Some general stuffs/tips/logistics
• Data Lab
• Bits and bytes
• Unsigned Numbers
• Signed Number
• Arithmetic/Logical Shift – Many questions on this!
• Floating Points – IEEE formats and rounding – Some questions on this!
Getting help: how-to?

• Attend lectures – there is no substitution for this!
• Course website – Go over the FAQ section
• Refer the textbook – it will be useful long after this semester ends
• Always email the staff mailing list – you will get quicker response (15-213-staff@cs.cmu.edu)
• Attend recitations – some problems will/might be solved!
• All recitations are on Monday – different place and time – check course website
• Visit TA office hours – for any specific questions or lab problems
• All office hours are in WEH 5207 - 5:30 PM to 8:30 PM – different days of the week – check course website
Data Lab - General

• Due this Thursday (Sept 11th)
• Try not to use grace days for the initial labs – the value of grace days appreciates as the semester progresses!
• If you have to use grace days – autolab will do it for you, just submit
• Beware - you can use only 2 grace days per lab! (Again, you should really be not using them now!)
• Try to get accustomed to shark machines – really! Future labs will force you to use only shark machines
Data Lab – How to get started?

• Download the handout
  - From Autolab
  - If not registered yet, download from course website (schedule page)
• Un-tar on shark machine (tar xvf <tar-file-name>)
  - Else you might see permission denied error
  - If you see any permission related issues, try “chmod +x <filename>”
• Hope everyone has already done this by now!
Data Lab – Which files to edit and submit?

• bits.c is the file you will be working on
• Read the instructions carefully – both in the bits.c file header and the lab writeup
• Run driver.pl before submitting – runs the same test that autolab uses to grade the lab
• You need to submit to autolab
• You have unlimited submission – but please do not use autolab as your own driver.pl
Data Lab – The last slide!

• You are expected to write C which follows ANSI C standards – Only for this lab!
• This means you cannot mix statements with variable declarations – all variables declared at the beginning of the function
• The closing brace should be in the 1st column!
• Be aware of operator precedence in C – better use braces when you are not sure. Ex: (((a-b) >> 2) & (c | (~a) | 0x1))
• Be clever to use some special values – 0, ~0, Tmin, -1 etc
• Try and finish all puzzles before you set out to optimize your solution
• Any specific question?
Bits and Bytes

• Computers understand everything in bits – in 0’s and 1’s
• So, even you should be able to understand bit level arithmetic – binary representation of numbers
• Might be difficult at first since we are more accustomed to counting in base-10
• But then, if it was easy then there would be no fun in it! (Right?)
• 8 bits make up 1 byte
# Bits and Bytes - Example Data Representation

<table>
<thead>
<tr>
<th>C Data Type</th>
<th>Typical 32-bit</th>
<th>Intel IA32</th>
<th>x86-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>8</td>
<td>10/12</td>
<td>10/16</td>
</tr>
<tr>
<td>pointer</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>
Bits and Bytes - Endianness

• Order in which bits are stored in memory matters – called Endianness
• Not so important for Data Lab
• You will get to see this in Bomb Lab and Buffer Lab
• Big Endian
  - Most significant byte is stored in the smallest address in memory
  - i.e. the ‘big’ part goes first into memory
• Little Endian
  - Least significant byte is stored in the smallest address in memory
  - i.e. the ‘little’ part goes first into memory
Unsigned Numbers

• Unsigned are non-negative numbers (including zero)
• Represented using direct binary notation
• Using k bits, we can represent $2^k$ distinct numbers
• So, the range becomes 0 to $2^k-1$
• So, $U_{\text{min}} = 0$ and $U_{\text{max}} = 2^k-1$
Signed Numbers (C ‘int’s)

• Most significant bit used to represent the sign
  - MSB = 0 for +ve integers
  - MSB = 1 for –ve integers
• So, with k bits, we get only k-1 bits to encode the value
• Using k bits, we can represent $2^{k-1}$ distinct ‘int’s
• The range becomes $-2^{k-1}$ to $(2^{k-1}-1)$
• Note how we can represent one more negative integer than the number of positive integer
• This is where the expression “$|T_{\text{min}}| = T_{\text{max}} + 1$” comes from
Operators of interest

• Bitwise
  - AND -> &
  - OR -> |
  - NOT -> ~
  - XOR -> ^
• Logical
  - AND -> &&
  - OR -> ||
  - NOT -> !
• Mixing logical AND/OR with bitwise AND/OR is a common source of bug (and frustration)
• Ex: x = (2 & 1) makes x to be equal to 0
• Ex: x = (2 && 1) makes x to be equal to 1
Operators of interest

• Bitwise Shift
  - Right Shift -> ‘>>’
  - Left Shift -> ‘<<’
• Mixing bitwise shift with relational operators is a common source of bug (and frustration)
• Ex: \( x = (4 >> 1) \) makes \( x \) to be equal to 2
• Ex: \( x = (4 > 1) \) makes \( x \) to be equal to 1
Some insights on right shifts

- Most modern machine performs arithmetic right shift on signed numbers i.e int’s
- This means that the sign bit is extended on right shift of signed numbers
- Wrong assumption - shark machines always perform arithmetic right shift
- Right assumption - shark machines always perform arithmetic right shift on signed numbers
- Int’s are signed in C. Unsigned are, well, not-signed numbers in C
- So, what actually determines the arithmetic or logical right shifts are not the machines. It is the signed-ness of the number
Some insights on right shifts

• Long story short
  - Arithmetic right shift on int’s: MSB gets filled with the sign bit
  - Logical right shift on unsigned: MSB always gets filled with a 0

• Remember, in C, numbers are ‘typed’ to be signed (i.e int) only after specifying it explicitly.

• By default everything is unsigned

• Try this out to get a clear picture:

```c
int x = 0x80000000;  int x = 0x80000000 >> 1;
x = x >> 1;            printf(“%x\n”, x);
printf(“%x\n”, x);
```
Floating Point – Binary Fractions

• Bits to the right of ‘binary point’ indicate fractional powers of 2
• How to calculate the value:
\[ \sum_{k=-j}^{i} b_k \times 2^k \]
• Ex: 1101.1011
Floating Point – IEEE Standards

• Single Precision: 32 bits
  - 1 bit sign
  - 8-bit exponent
  - 23-bit fraction

• Double Precision: 64 bits
  - 1 bit sign
  - 11-bit exponent
  - 52-bit fraction

• Extended Precision: 80 bits (Intel only)
  - 1 bit sign
  - 15-bit exponent
  - 63 or 64-bit fraction
Floating Point – IEEE Standards

• Can be thought of as: \((-1^{\text{sign}}) \ M \ 2^E\)
• MSB represents the sign: 1 for negative, 0 for positive
• The exp field encodes E (but is not equal to E)
• The frac field encodes M (but is not equal to M)
Floating Point – Normalized Values

- When exp ≠ 000..0 and exp ≠ 111..1
- E = Exp – bias
  - Exp = unsigned value of exp
  - bias = $2^{k-1} − 1$, where $k$ = number of exp bits
- Significand coded with implied leading 1: $M = 1.xxx...xxx$ (base 2)
  - xxx...xxx: bits of frac
- For single precision: $k=8$, bias=127
- Ex: $15213 = 11\ 1011\ 0110\ 1101 = 1.1101101101101 * 2^{13}$
  - Exp = E + bias = 13 + 127 = 140 = 1000 1100 (base 2)
  - frac = 11011011011000000000000
- Ex: $15213.0 = 0\ 10001100\ 11011011011010000000000000$
Floating Point – Denormalized Values

• When exp = 000…0
• E = -bias + 1
• Significand coded with implied leading 0: M = 0.xxx…xxx (base 2)
  - xxx…xxx: bits of frac
• Exp = 000…0 and frac = 000…0 represents 0
  - +0 and -0 have different representation!
• Exp = 000…0 and frac != 000…0 represents numbers closest to 0.0
Floating Point – Special Values

• When \( \exp = 111\ldots1 \)
• \( \Exp = 111\ldots1 \) and \( \frac{\text{frac}}{} = 000\ldots0 \)
  - Represents infinity (\( \infty \))
  - Both positive and negative (depending on sign bit)
• \( \Exp = 111\ldots1 \) and \( \frac{\text{frac}}{} \neq 000\ldots0 \)
  - Represents Not-a-Number (NaN)
  - Ex: \( \sqrt{-1}, \infty - \infty, \infty \times 0 \)
Floating Point - Rounding

• Cannot accurately represent all fractional ranges with limited number of bits
• Fractions need to be rounded most of the time
• Rounding scheme – round to even
• Ex:
  10.1011 → More than ½, round up → 10.11
  10.1010 → Equal to ½, round down to even → 10.10
  10.0101 → Less than ½, round down → 10.01
  10.0110 → Equal to ½, round up to even → 10.10
• Round up/down to even used only in interesting cases (Equal to ½ cases above)
• All other cases involve rounding up or rounding down
Floating Point - Examples

- Consider the following 5 bit floating point representation
  - k=3 exponent bits
  - n=2 frac bits
  - no sign bit (only positive floats)
- Bias = ?
- Largest normalized number = ?
- Smallest normalized number = ?
- Largest denormalized number = ?
- Smallest denormalized number = ?
Floating Point - Examples

• Consider the following 5 bit floating point representation
  - k=3 exponent bits
  - n=2 frac bits
  - no sign bit (only positive floats)
• Bias = 3
• Largest normalized number = 110 11 = 1110.0₂ = 14
• Smallest normalized number = 001 00 = 0.0100₂ = 1/4
• Largest denormalized number = 000 11 = 0.0011₂ = 3/16
• Smallest denormalized number = 000 01 = 0.0001₂ = 1/16
Floating Point - Examples

• When converting to IEEE floating points, assume that the IEEE floating point would be normalized
• It will be - most of the time
• If not, you can easily tell – exponent would not make any sense
Floating Point - Examples

- Consider the following 5 bit floating point representation
  - k=3 exponent bits
  - n=2 frac bits
  - no sign bit (only positive floats)

<table>
<thead>
<tr>
<th>Value in Decimal</th>
<th>IEEE floating point representation</th>
<th>Rounded Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3/16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15/2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Floating Point - Examples

- Consider the following 5 bit floating point representation
  - \( k=3 \) exponent bits
  - \( n=2 \) frac bits
  - no sign bit (only positive floats)

<table>
<thead>
<tr>
<th>Value in Decimal</th>
<th>IEEE Floating point representation</th>
<th>Rounded Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/32</td>
<td>001 00</td>
<td>1/4</td>
</tr>
<tr>
<td>3</td>
<td>100 10</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>110 00</td>
<td>8</td>
</tr>
<tr>
<td>3/16</td>
<td>000 11</td>
<td>3/16</td>
</tr>
<tr>
<td>15/2</td>
<td>110 00</td>
<td>8</td>
</tr>
</tbody>
</table>
Credits

- [http://www.cs.cmu.edu/~213/](http://www.cs.cmu.edu/~213/)
- The course text book
- Lecture slides
- Previous recitation slides