CS 237 Meeting 3 — 9/12/12

Announcements

1. Notes online.

Basics of Radix Notation

1. Consider the following "solved" addition problems:

$\begin{array}{r} 3\ 3\ 1 \\ +\ 1\ 5\ 2 \\ \hline 5\ 1\ 3 \end{array}$	$\begin{array}{r} 3\ 3\ 1 \\ +\ 1\ 5\ 2 \\ \hline 5\ 0\ 3 \end{array}$	$\begin{array}{r} 3\ 3\ 1 \\ +\ 1\ 5\ 2 \\ \hline 4\ 8\ 3 \end{array}$	$\begin{array}{r} 3\ 3\ 1 \\ +\ 1\ 5\ 2 \\ \hline 5\ 2\ 3 \end{array}$

   Which one is correct?

   - Interpreted according to the rules we usually use for addition, only the third version:
   $$\begin{array}{r} 3\ 3\ 1 \\ +\ 1\ 5\ 2 \\ \hline 4\ 8\ 3 \end{array}$$
   is correct.
   - It is easy to see the "mistakes" in the other versions. For example, in the middle column of the version that look like:
   $$\begin{array}{r} 3\ 3\ 1 \\ +\ 1\ 5\ 2 \\ \hline 5\ 1\ 3 \end{array}$$
   we see that we are adding 3 and 5 and ending up with 1 when we know that 3 plus 5 is 8.
   - In fact, however, this does not prove this version is wrong. To see this, consider the correct addition problem: $\begin{array}{r} 3\ 3\ 1 \\ +\ 1\ 8\ 2 \\ \hline 5\ 1\ 3 \end{array}$
   Here, in the middle column, we add 3 and 8 which clearly sums to 11 but we write 1 instead of 11 for the corresponding column in our answer.
   - The reason we accept this is that we know that if the sum of the numbers in a column is greater than 10, the base we use for our number system, then we just write the ones digit of the sum in the current column and we carry the excess to the next column.
   - Returning, to the problem: $\begin{array}{r} 3\ 3\ 1 \\ +\ 1\ 5\ 2 \\ \hline 5\ 1\ 3 \end{array}$
   note that if we were working in base 7, then since 5 plus 3 is one greater than the base, 7, we would carry 7 and write the reminder 1 in the middle column. That is, the tableau is actually correct, but only for base 7, not for base 10.

2. With this understanding, we can see that each of our original solutions were correct for some base:

$\begin{array}{r} 3\ 3\ 1_7 \\ +\ 1\ 5\ 2_7 \\ \hline 5\ 1\ 3_7 \end{array}$	$\begin{array}{r} 3\ 3\ 1_8 \\ +\ 1\ 5\ 2_8 \\ \hline 5\ 0\ 3_8 \end{array}$	$\begin{array}{r} 3\ 3\ 1_{10} \\ +\ 1\ 5\ 2_{10} \\ \hline 4\ 8\ 3_{10} \end{array}$	$\begin{array}{r} 3\ 3\ 1_6 \\ +\ 1\ 5\ 2_6 \\ \hline 5\ 2\ 3_6 \end{array}$

3. The essential idea here is that although we are very used to working in base 10, numbers can be represented in any base.

   All we need to represent a number in some base b is

   - A set of b symbols (typically called digits).
   - A one-to-one mapping v(d), associating our digits with integer values from 0 to $b - 1$.

   Then, we simply associate a sequence of digits of the form $d_n d_{n-1} \ldots d_1 d_0$ with the value
   $$v(d_n)b^n + v(d_{n-1})b^{n-1} + \ldots + v(d_1)b^1 + v(d_0)b^0 = \Sigma_{i=0}^{n} v(d_i)b^i$$

4. In the common case (for any base less than 11) where our digits are just the usual digits $0 \ldots 9$, it is common to assume the obvious mapping $v(d) = d$ and just write
   $$d_n b^n + d_{n-1}b^{n-1} + \ldots + d_1 b^1 + d_0 b^0 = \Sigma_{i=0}^{n} d_i b^i$$

5. Thus, we would interpret 152_{10} as $1\times10^2+5\times10^1+2\times10^0 = 100+50+2$ as expected.

   On the other hand, $152_6 = 1\times6^2+5\times6^1+2\times6^0 = 36+30+2 = 68_{10}$

Everyone's Favorite Bases

1. In theory, any number can be used as a base. If you look at computer science textbooks (including our own), you will notice that they seem to have a strong preference for four bases: 2 (binary), 10 (decimal), 16 (hexadecimal), and 8 (octal).

2. The excuse usually given for decimal is that we have 10 fingers. I am willing to accept this, although if you research the history of number systems you will find examples where humans used bases 5, 12, 20 and even 60!

3. I have already hinted that binary is popular simply because using as few digits as possible simplifies the task of building circuits that can represent and process physical representations of these digits as voltages or any other physical property.

4. So the interesting question is why hexadecimal and octal are so popular.

   - The basic idea is that hexadecimal and octal provide a convenient shorthand for describing numbers that ultimately need to be represented in binary.
     – The smaller the base, the more digits it takes to represent a given number.
     – This means that a number expressed in binary will require more symbols than in any other form.
     – Octal or hexadecimal representations will be much shorter.
   - The fact that 8 and 16 are powers of 2 makes it particularly easy to convert between binary and octal or hexadecimal and vice versa.

     To keep things simple, let's focus on how this works with octal. The argument is essentially the same with hexadecimal but you have to get use to thinking of A, B, C, D, E, and F as digits!

     – Suppose we have a nine digit binary number:

$$b_8b_7b_6b_5b_4b_3b_2b_1b_0$$
$$=$$
$$b_82^8+b_72^7+b_62^6+b_52^5+b_42^4+b_32^3+b_22^2+b_12^1+b_02^0$$

     – We can group the terms in the sum so that we can factor out powers of $2^3 = 8$ from each sequence of 3 consecutive terms:

$$b_82^8+b_72^7+b_62^6+b_52^5+b_42^4+b_32^3+b_22^2+b_12^1+b_02^0$$
$$=$$
$$(b_82^2+b_72^2+b_62^2)2^6+(b_52^2+b_42^1+b_32^0)2^3+(b_22^2+b_12^1+b_02^0)2^0$$
$$=$$
$$(b_82^2+b_72^2+b_62^2)8^2+(b_52^2+b_42^1+b_32^0)8^1+(b_22^2+b_12^1+b_02^0)8^0$$

     – This means that we can convert the 9 digit binary number into a 3 digit octal number by converting each sequence of binary digits to a single octal digit independently.
     For example, given 100111010, we can separate out the three subsequences 100, 111, 010, convert them into octal independently (since there are only three binary digits, each sequence can be represented by a single octal digit), and then join them together to get $472_8 = 100111010_2$.
     – Similarly, given an octal value like 317_8 we can separately convert each octal digit into a 3 digit binary number (011, 001, and 111) and then concatenate them to obtain the binary representation of the same number 011001111_2.
     – The same tricks work with base 16 except:
       * You must work with groups of four binary digits rather than 3, and
       * Since there are more than 10 values that can be expressed in 4 binary digits, you must use extra symbols (A, B, C, D, E, F) for the values 10, 11, 12, 13, 14, and 15.

Practice, Practice, Practice

1. I assume you can figure out how to convert numbers from one base to another. If not, practice on your own.
2. What I would like to practice a bit now is doing addition and subtraction in various bases. You may have to do this occasionally while programming in assembly language. In any event, it is a good way to force yourself to think about the correct interpretation of radix notation.
3. Let's start with:

$$\begin{array}{cccc} & 3 & 6 & 1_{10} \\ + & 3 & 5 & 7_{10} \\ \hline \end{array}$$

   - 1 + 7 in the ones column gives 8 which is less than our base, 10, so we just write 8 in the ones position of the answer.

$$\begin{array}{cccc} & 3 & 6 & 1_{10} \\ + & 3 & 5 & 7_{10} \\ \hline & & & 8_{10} \end{array}$$

   - 6 + 5 in the 10s column (which is therefore really 60 + 50) yields 11 which is bigger than our base. So we divide 11 by 10, place the remainder, 1 (which is really 10), in the 10s place of the answer and carry the quotient, 1 (which is really 100), to the 100s column.

$$\begin{array}{cccc} & 3^1 & 6 & 1_{10} \\ + & 3 & 5 & 7_{10} \\ \hline & & 1 & 8_{10} \end{array}$$

   - Finally, 3 plus the carry of 1 is 4. This plus the 3 in the hundreds place of 357 yields 7. 7 is less than our base so we just write 7 in the hundreds place of the answer. DONE!

$$\begin{array}{cccc} & 3^1 & 6 & 1_{10} \\ + & 3 & 5 & 7_{10} \\ \hline & 7 & 1 & 8_{10} \end{array}$$

4. Of course, it would be much more interesting to work the problem in a different base. Since one of the digits in our operands is 7, the smallest base that could be used here is base 8.

$$\begin{array}{cccc} & 3 & 6 & 1_{8} \\ + & 3 & 5 & 7_{8} \\ \hline \end{array}$$

   - 1 + 7 in the ones column gives 8 which is equal to our base. So, we divide by 8 and place the remainder, 0, in the 1 place while carrying the quotient,1, to the next column which is the 8s column.

$$\begin{array}{cccc} & 3 & 6^1 & 1_{8} \\ + & 3 & 5 & 7_{8} \\ \hline & & & 0_{8} \end{array}$$

   - 6 + 5 plus the carry of 1 in the s column (which is therefore really 6x8 + 5x8 + 1x8) yields 12 which is bigger than our base. So we divide 12 by 8, place the remainder, 4 (which is really 4x8 = 32), in the 8 place of the answer and carry the quotient, 1 (which is really 64), to the 64s column.

$$\begin{array}{cccc} & 3^1 & 6^1 & 1_{8} \\ + & 3 & 5 & 7_{8} \\ \hline & & 4 & 0_{8} \end{array}$$

   - Finally, 3 plus the carry of 1 is 4. This plus the 3 in the 64s place of 357 yields 7. 7 is less than our base so we just write 7 in the 64s place of the answer. DONE!

$$\begin{array}{cccc} & 3^1 & 6^1 & 1_{8} \\ + & 3 & 5 & 7_{8} \\ \hline & 7 & 4 & 0_{8} \end{array}$$

5. Let's do this one last time, but this time in hexadecimal (base 16).

$$\begin{array}{cccc} & 3 & 6 & 1_{16} \\ + & 3 & 5 & 7_{16} \\ \hline \end{array}$$

   - 1 + 7 in the ones column gives 8 which is less than our base. So, we just write 8 in the ones position of the answer.

$$\begin{array}{cccc} & 3 & 6 & 1_{16} \\ + & 3 & 5 & 7_{16} \\ \hline & & & 8_{16} \end{array}$$

- 6 + 5 in the 16s column (which is therefore really 6x16 + 5x16) yields 11 which is also smaller than our base. It is however, bigger than the decimal digits we are familiar with. So, we have to find the letter that corresponds to 11 in hexadecimal. Until you commit the letters to memory, you just count starting with A and 10. A=10, B=11, C=12, ... Therefore, we just write B in the 16s place of the answer.

$$\begin{array}{rrrr} & 3 & 6 & 1_{16} \\ + & 3 & 5 & 7_{16} \\ \hline & & B & 8_{16} \end{array}$$

- Finally, 3 plus the 3 in the 256s place of 357 yields 6. 6 is less than our base so we just write 6 in the 256s place of the answer. DONE!

$$\begin{array}{rrrr} & 3 & 6 & 1_{16} \\ + & 3 & 5 & 7_{16} \\ \hline & 6 & B & 8_{16} \end{array}$$

6. Now that we have addition down pat. Let's do some subtraction. To keep things simple, I will even stick with the same numbers:

$$\begin{array}{rrrr} & 3 & 6 & 1_{10} \\ - & 3 & 5 & 7_{10} \\ \hline & & & \end{array}$$

   - To handle the rightmost column, we have to take 7 from 1. This is hard because 7 is bigger than 1. So, we borrow 1 from the column to the left. Since the column to the left is really the 10s place, the 1 from that column becomes a 10 in the rightmost column. So we add this 10 to our 1 giving 11 and then take 7 away leaving 4 in the units position of the answer

$$\begin{array}{rrrr} & 3 & 6 & 1_{10} \\ - & 3 & 5_1 & 7_{10} \\ \hline & & & 4_{10} \end{array}$$

   - Because of the 1 we borrowed from the 10s column while handling the units, we now have to take 6 from 6. This leaves 0. EASY!

$$\begin{array}{rrrr} & 3 & 6 & 1_{10} \\ - & 3 & 5_1 & 7_{10} \\ \hline & & 0 & 4_{10} \end{array}$$

   - Finally, 3 minus 3 leaves 0 again. DONE!

$$\begin{array}{rrrr} & 3 & 6 & 1_{10} \\ - & 3 & 5_1 & 7_{10} \\ \hline & 0 & 0 & 4_{10} \end{array}$$

   - Obviously, I picked this as our example of subtraction to save us some time. All the interesting stuff happens in the first column.

7. Of course we also have to see how this works in some other base:

$$\begin{array}{rrrr} & 3 & 6 & 1_{8} \\ - & 3 & 5 & 7_{8} \\ \hline & & & \end{array}$$

   - To handle the rightmost column, we have to take 7 from 1. This is hard because 7 is bigger than 1. So, we borrow 1 from the column to the left. Since the column to the left is really the 8 place, the 1 from that column becomes an 8 in the rightmost column. So we add this 8 to our 1 giving 9 and then take 7 away leaving 2 in the units position of the answer

$$\begin{array}{rrrr} & 3 & 6 & 1_{8} \\ - & 3 & 5_1 & 7_{8} \\ \hline & & & 2_{8} \end{array}$$

   - Then, just as in the base 10 case, all the other positions in the answer become 0s.

$$\begin{array}{rrrr} & 3 & 6 & 1_{8} \\ - & 3 & 5_1 & 7_{8} \\ \hline & 0 & 0 & 2_{8} \end{array}$$

8. Before we move on, we should also experience the agony of binary addition:

- $$\begin{array}{rrrrrrr} & 1 & 0 & 1 & 1 & 0 & 1_2 \\ + & 0 & 0 & 1 & 1 & 1 & 0_2 \\ \hline \end{array}$$
- The rightmost two columns are simple since the sum of 0 and 1 is less than our base, 2, so we just write 1s in the matching positions in the answer.
$$\begin{array}{rrrrrrr} & 1 & 0 & 1 & 1 & 0 & 1_2 \\ + & 0 & 0 & 1 & 1 & 1 & 0_2 \\ \hline & & & & & 1 & 1_2 \end{array}$$
- The third column sums to 2, our base, so we divide by 2, place the remainder of 0 in the third position of the answer and carry the 1.
$$\begin{array}{rrrrrrr} & 1 & 0 & 1^1 & 1 & 0 & 1_2 \\ + & 0 & 0 & 1 & 1 & 1 & 0_2 \\ \hline & & & & 0 & 1 & 1_2 \end{array}$$
- Including the carry, the fourth column sums to 3, bigger than our base, so we divide by 2, place the remainder of 1 in the third position of the answer and carry the 1.
$$\begin{array}{rrrrrrr} & 1 & 0^1 & 1^1 & 1 & 0 & 1_2 \\ + & 0 & 0 & 1 & 1 & 1 & 0_2 \\ \hline & & & 1 & 0 & 1 & 1_2 \end{array}$$
- The last two columns are simple because they involve no carries.
$$\begin{array}{rrrrrrr} & 1 & 0^1 & 1^1 & 1 & 0 & 1_2 \\ + & 0 & 0 & 1 & 1 & 1 & 0_2 \\ \hline & 1 & 1 & 1 & 0 & 1 & 1_2 \end{array}$$

9. and the agony of binary subtraction:

- $$\begin{array}{rrrrrrr} & 1 & 0 & 1 & 1 & 0 & 1_2 \\ - & 0 & 0 & 1 & 1 & 1 & 0_2 \\ \hline \end{array}$$
- The first column is easy:
$$\begin{array}{rrrrrrr} & 1 & 0 & 1 & 1 & 0 & 1_2 \\ - & 0 & 0 & 1 & 1 & 1 & 0_2 \\ \hline & & & & & & 1_2 \end{array}$$
- In the second column, since we cannot take 1 away from 0, we have to borrow 2 (our base) from the third column:
$$\begin{array}{rrrrrrr} & 1 & 0 & 1 & 1 & 0 & 1_2 \\ - & 0 & 0 & 1 & 1_1 & 1 & 0_2 \\ \hline & & & & & 1 & 1_2 \end{array}$$
- Similarly, in the third column since we are now trying to take 2 from 1, we have to borrow 2 from the fourth column:
$$\begin{array}{rrrrrrr} & 1 & 0 & 1 & 1 & 0 & 1_2 \\ - & 0 & 0 & 1_1 & 1_1 & 1 & 0_2 \\ \hline & & & & 1 & 1 & 1_2 \end{array}$$
- Ditto for the fourth column:
$$\begin{array}{rrrrrrr} & 1 & 0 & 1 & 1 & 0 & 1_2 \\ - & 0 & 0_1 & 1_1 & 1_1 & 1 & 0_2 \\ \hline & & & 1 & 1 & 1 & 1_2 \end{array}$$
- And then in two more steps:
$$\begin{array}{rrrrrrr} & 1 & 0 & 1 & 1 & 0 & 1_2 \\ - & 0_1 & 0_1 & 1_1 & 1_1 & 1 & 0_2 \\ \hline & 0 & 1 & 1 & 1 & 1 & 1_2 \end{array}$$

Reality Sets In

1. Consider what happens if we change just one digit of the binary addition problem we considered above.
$$\begin{array}{rrrrrrr} & 1 & 0 & 1 & 1 & 0 & 1_2 \\ + & 0 & 1 & 1 & 1 & 1 & 0_2 \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & 1_2 \end{array}$$
2. This is an example where the sum of two 6 bit numbers is a 7 bit number!
3. In a computer (or any physical device that represents numbers using radix notation), there will be some fixed number of digits represented. If the result of an addition (or any other operation) requires more bits than the device supports, data will be lost.
    - If a machine could only store six bits for any number, the answer returned when asked to add the numbers above would be WRONG!

$$\begin{array}{rrrrrr} 1 & 0 & 1 & 1 & 0 & 1_2 \\ +\,0 & 1 & 1 & 1 & 1 & 0_2 \\ \hline 0 & 0 & 1 & 0 & 1 & 1_2 \end{array}$$

4. This is called *overflow*.

Negative Thoughts

1. We also need to figure out a way to represent negative numbers!

   (a) The subtraction example we considered earlier:
   $$\begin{array}{rrrrrrcr} & 1 & 0 & 1 & 1 & 0 & 1_2 & = & 45_{10} \\ - & 0 & 0 & 1 & 1 & 1 & 0_2 & = & 14_{10} \\ \hline & 0 & 1 & 1 & 1 & 1 & 1_2 & = & 31_{10} \end{array}$$
   yields a positive result.

   (b) If we reverse the operands, the sign of the result is reversed. How should this number be represented in binary?
   $$\begin{array}{rrrrrrcr} & 0 & 0 & 1 & 1 & 1 & 0_2 & = & 14_{10} \\ - & 1 & 0 & 1 & 1 & 0 & 1_2 & = & 45_{10} \\ \hline & ? & ? & ? & ? & ? & ?_2 & = & -31_{10} \end{array}$$

   (c) Recall that the reason we use binary is that we really want to limit ourselves to two symbols. So adding "+" and "−" signs is not an option.

   (d) One obvious option is to interpret one of the binary digits in the number's representation as the sign. Typically, the first/most significant bit is chosen with 0 representing + and 1 representing −.

   With this approach:
   $$\begin{array}{rrrrrrcr} 0 & 1 & 1 & 1 & 1 & 1_2 & = & 31_{10} \end{array}$$
   and
   $$\begin{array}{rrrrrrcr} 1 & 1 & 1 & 1 & 1 & 1_2 & = & -31_{10} \end{array}$$
   Note: This approach depends on having a known, fixed number of bits used for each number.

   (e) This approach is known as sign-magnitude notation. It is used in some systems, but there is a much better approach.

   (f) It would be nice if the approach we used to represent negative numbers allowed us to add/subtract without worrying about the sign of the numbers involved. That is, we would just process columns passing carries along as we did in our earlier examples.

   (g) If we just perform the subtraction with a negative result shown above, we would obtain the result:
   $$\begin{array}{rrrrrrrcr} & & 0 & 0 & 1 & 1 & 1 & 0_2 & = & 14_{10} \\ & - & 1 & 0 & 1 & 1 & 0 & 1_2 & = & 45_{10} \\ \hline & 1 & 1 & 0 & 0 & 0 & 0 & 1_2 & = & -31_{10} \end{array}$$

   (h) Of course, on a machine using 6-bits to represent numbers, we would only keep the last 6 bits:
   $$\begin{array}{rrrrrrrcr} & 0 & 0 & 1 & 1 & 1 & 0_2 & = & 14_{10} \\ - & 1 & 0 & 1 & 1 & 0 & 1_2 & = & 45_{10} \\ \hline & 1 & 0 & 0 & 0 & 0 & 1_2 & = & -31_{10} \end{array}$$

   (i) The number 100001_2 has a 1 in the 2^5 position and a 1 in the 2^0 position so we would normally interpret it as $100001_2 = 2^5 + 1 = 32 + 1 = 33$.

   (j) If, however, as a way to make the high-order digit function more like a sign we interpret the high order bit as the -2^5 position rather than the 2^5 position, we can now interpret $100001_2 = -2^5 + 1 = -32 + 1 = -31$ as desired.

   (k) Luckily, this works in general. That is given a sequence of binary digits of the form $d_n d_{n-1} \ldots d_1 d_0$ we will interpret their value as
   $$-d_n 2^n + d_{n-1} 2^{n-1} + \ldots + d_1 2^1 + d_0 2^0$$
   This approach is called twos-complement notation.

   (l) Twos complement notation has several advantages:
   - Addition performed using the "normal" rules yields the correct result if the result is in range.
   - Each number has a unique representation (in sign-magnitude notation, there is a positive 0 and a negative 0).
   - There is an easy way to determine the representation of a number's complement. You first flip all the bits (replace 0s with 1s and vice versa) and then add 1. This is called taking the twos complement.

- Subtraction can be performed by complementing one number and then doing addition.

(m) Twos complement notation also has a slight glitch: The absolute value of the largest negative value that can be represented is bigger than the largest positive value.

Floating About

1. There is one important aspect of schemes for representing numbers in binary that we have not addressed here. How do we approximate real numbers in binary. This is needed for types like float and double in Java and C. With a bit of luck we will return to this later in the semester.