Australian National University

Negative Numbers and Binary operations

Eric McCreath

Binary Addition

- Adding binary numbers is very similar to how you would add large decimal numbers in primary school
  - It involves positioning the numbers such that the columns line up in their powers,
  - Then adding the single digits of the corresponding powers,
  - If the result is larger than what can be contained in that power then carry one to the next column
- Try the following and check by converting the numbers to decimal (repeating the addition in decimal)

$$11010_{(2)} + 01011_{(2)}$$

Binary Subtraction

- Subtracting binary numbers is also very similar to how you would subtract large decimal numbers in primary school
  - The process is similar to addition
  - The main difference is subtracting the corresponding digits and also the carry
- Try the following and check by converting the numbers to decimal.

$$10101_{(2)} - 01011_{(2)}$$

Negative Integers

- A very simple way of representing negative integers is to reserve one bit as the sign bit
  - The reserved bit is normally the most significant bit where 0 means a positive number and 1 means negative
- If 1 byte words are being used, then:

$$42 \rightarrow 00010101$$

$$-42 \rightarrow 10010101$$

- Using this method, 8-bit numbers range from -127 - 127
- This approach has two main problems:
  - First, results in 2 zeros - this wastes a bit pattern and also makes equality more complex
  - Second, addition or subtraction is not as simple

Negative Integers

- Another approach for representing negative integers is to *offset* the unsigned integer value

- In the example below, a 3 bit unsigned integer is being used with an offset of -4

$$\begin{aligned}
111 &\quad \rightarrow 3 \\
110 &\quad \rightarrow 2 \\
101 &\quad \rightarrow 1 \\
100 &\quad \rightarrow 0 \\
011 &\quad \rightarrow -1 \\
010 &\quad \rightarrow -2 \\
001 &\quad \rightarrow -3 \\
000 &\quad \rightarrow -4
\end{aligned}$$

- With this approach we only have one zero, although, normal addition will not work

Two's Complement

- The two's complement representation of signed integers is the most common approach. Only one zero is represented and normal binary addition can be used.

- The left-most (or most signficant bit) still indicates sign (0 - positive; 1 - negative)

$$00000101 \rightarrow 5$$

$$11111011 \rightarrow -5$$

$$11111000 \rightarrow -8$$

$$00001000 \rightarrow 8$$

$$00000000 \rightarrow 0$$

$$10000000 \rightarrow -128$$

Two's Complement

- A 3 bit two's complement representation would have the following mappings:

$$
\begin{aligned}
011 &\rightarrow 3 \\
010 &\rightarrow 2 \\
001 &\rightarrow 1 \\
000 &\rightarrow 0 \\
111 &\rightarrow -1 \\
110 &\rightarrow -2 \\
101 &\rightarrow -3 \\
100 &\rightarrow -4
\end{aligned}
$$

Two's Complement

- To negate a 2's complement number:

flip all bits and add one

or

flip the bits to the left of the right-most 1

Two's Complement

- When adding 2's complement numbers, use normal unsigned addition and discard any overflow
- With 4-bit 2's complement numbers try the following:

$$4 + (-1)$$

$$-3 + (-2)$$

Overflow

- An arithmetic overflow occurs when the result of an operation is too large (or small) to be represented
  - This can normally be detected by looking at an overflow flag in the status register (or checking the sign of the resulting number is as expected)
- There are different approaches for dealing with the overflow problem. These include:
  - designing the program such that numbers will not be out of range
  - avoiding the situation by checking the numbers before and after an operation
  - setting up an exception
  - propagating the overflow or ignoring the problem.