

Binary Arithmetic

- In adder circuits, two binary numbers are added together at a time (typically 32 bits long).
 - integers: 2^{32} possible values
 - floats: 1 sign bit, 8 exponent bits and 23 mantissa bits
- 8 bits combine to form a **byte**.
- Half a byte (4 bits) is called a **nybble**.
- All numbers are assumed to be **signed** values (i.e. both positive and negative values)

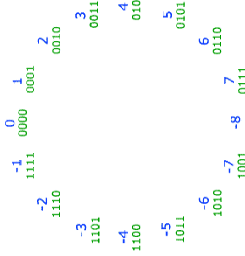


Binary Addition

- Addition in binary follows the same principles as addition in decimal.
 - Add the least significant columns first.
 - When the sum of a column cannot be expressed as a single digit, carry the extra digits over into the next column.
- If the overall sum of two numbers has more digits than the register can hold (i.e. greater than 32), a special carry-out signal is used to alert the user to the **overflow**.
- Binary addition is performed on two numbers at a time. Adding additional number will require additional operations.

Binary Subtraction

- Subtraction is treated as an addition operation on a negative value.
 - Multiple representations exist for negative numbers.
 - Sign-and-magnitude**: first bit of an integer indicates the sign (0 is positive, 1 is negative), and the remaining bits store the value.
 - One's complement**: the negative value for an integer is the inverse of the positive value. Results in two values for zero (0000...000 and 1111...111).
 - Two's complement**: an integer's negative value and its positive value would sum to zero if added.



Two's Complement

- 2's complement is the most commonly used representation of negative numbers, although references are made to 1's complement occasionally.
 - 1's complement requires a carry bit to be added whenever negative numbers are involved in the operation.
- To calculate 2's complement of a number:
 - calculate the 1's complement value of that number (easy)
 - add 1 to the resulting value (also easy)
- Example:

42 : 00101010
 1's complement : 11010101
 2's complement : 11010110

 - see what happens when you add 42 and -42 together.

Booth's Algorithm

- Andrew D. Booth had hardware that shifted faster than it added, and created this multiplication algorithm for it.
- Algorithm details:
 - Given a multiplicand with x bits and a multiplier with y bits, create three binary numbers A , S & P , each with $(x+y+1)$ bits.
 - Initialize each of A , S & P with the following values:
 - A : the multiplicand bits, followed by $y+1$ zeros.
 - S : the negative multiplicand bits, followed by $y+1$ zeros.
 - P : x zeros, followed by the multiplier, followed by a zero.
 - Example:

$13 * -5$
A: 01101 0000 0
S: 10011 0000 0
P: 00000 1011 0

Booth's Algorithm

- (continued from previous slide)
- Repeat the following steps y times:
 - Check the last two bits of P .
 - If the last two bits of P are 00 or 11, do nothing
 - If the last two bits of P are 01, let $P = P+A$
 - If the last two bits of P are 10, let $P = P+S$

- Arithmetically shift P one bit to the right.

- Discard the rightmost bit from P to obtain the final result.

- Algorithm results in fewer overall additions

- Example:

$13 * -5$	$P = 00000$	1011	0	
Step #1 (shift)	→	$P = 10011$	1011	0
Step #2 (shift)	→	$P = 11001$	1101	1
Step #3 (shift)	→	$P = 11100$	1110	1
Step #4 (shift)	→	$P = 01001$	1110	1
Step #4 (shift)	→	$P = 00100$	1111	0
Step #4 (shift)	→	$P = 10111$	1111	0
Step #4 (shift)	→	$P = 11011$	1111	1
		$P = -65$		