Binary Arithmetic

- Subtrahend: what is being subtracted
- Minuend: What it is being subtracted from
- Minuend – Subtrahend = Result
- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0 carry over 1
- 1 + 1 + 1 = 1 carry over 1

Example: Perform the binary subtraction 1010101 - 11100

A''		0	1	10				
A'		1	0	0	10			
A		1	0	1	0	1	0	1
B			–	1	1	1	0	0
Diff		0	1	1	1	0	0	1
Step		7	6	5	4	3	2	1

A				1	1	1	0	1
B					x	1	1	1
				1	1	1	0	1
			1	1	1	0	1	
		1	1	1	0	1		
Answer	1	1	0	0	1	0	1	1
Carry	1	10	10	1	1			

- Division = Dividend / Divisor = Quotient + (remainder / divisor)
- Use long division.

Overflow

- When the result of an arithmetic operation is too large or too small to fit into the resultant bit-group (e.g. 9 cannot fit into 4-bits in 2's C)
- Normally left to programmer to deal with this

Two's Complement - Addition

- Add the values
- Discard all carry out bits

Two’s Complement – Overflow through Addition

- Occurs if and only if 2 Two’s Complement numbers are added and they both have the same sign and the result has the opposite sign
- E.g (-7) + (-6) using 4 bits
- +7 = 0111
- -7 = 1001
- +6 = 0110
- -6 = 1010
- (-7) + (-6) = 1 0011 = Overflow = +3

Two’s Complement – Subtraction

- Negate the subtrahend and add to the minuend
- Discard all carry out bits
- e.g. 8 – 5 = 8 + (-5) = using 8 bits
- +8 = 00001000
- +5 = 00000101
- -5 = 11111011
- 1 00000011 = 00000011

Two’s Complement – Overflow through Subtraction

- Occurs if and only if 2 Two’s Complement numbers are subtracted, and their signs are different, and the result has the same sign as the subtrahend
- e.g. using 4 bits = 7 – (-6)
- 7 = 0111
- 6 = 0110
- -6 = 1010

- --6 = 0110
- 7 + (--6) = 1101 = Overflow = -3

Two's Complement – Multiplication and Division

Cannot be accomplished using the standard technique

- Example: consider X * (–Y)
  - Two's complement of –Y is 2^n–Y → X * (-Y) = X * (2^n–Y) = 2^nX – XY
  - Expected result should be 2^{2n} – XY

Sign extension to 2^{2n} and omission of the leading n bits works but is extremely inefficient

+3 * -2 (4-bit)

```
     0000 0011 x 1111 1110
      0000 0011
     0000 0011
    0000 0011
   0000 0011
  0000 0011
 0000 0011
0000 0011
0000 0011
----------------
0000 1011 1111 1010
```