CS 520
Signed Integer Representation

Sign & Magnitude

- Consider a byte of information
- Use the MSB as a sign bit

Sign	Magnitude
0 (positive) 1 (negative)	1000101 0101001

- 37_{10} would be: 00100101
- -15_{10} would be: 10001111
- Adding 37+15 is easy
- How about doing 37 + (-15)?

$$\begin{array}{r} 00100101 \\ \underline{-10001111} \\ 00010110 \end{array}$$

- Negate is easy
- Add:
  - Test sign bits, if same sign keep sign
  - If different signs, subtract smaller magnitude from larger and keep sign of larger

Sign & Magnitude

- Negating is easy
- Addition is complicated with the need to check for larger number when numbers are not the same sign
- Two representations for zero!

One’s Complement

- Express absolute value of a number
- If negative complement all bits i.e. reverse the bit value
- Perform addition just like unsigned numbers
  - If the last (MSB) bit addition results in carry over add 1 to the rightmost bit position
- 37_{10} : 00100101 and 15_{10} : 00001111
- -15_{10} would need 1’s complement representation --> 11110000
- Now add 37 + (-15):

$$
\begin{array}{lr}
 & 00100101 \\
 & \underline{+11110000} \\
\text{(Carry 1)} & 00010101 \\
 & \underline{+1} \\
 & 00010110 \text{ --> } 22_{10}
\end{array}
$$

One’s Complement

- Another example:
- 10_{10} : 00001010 and 5_{10} : 00000101
- Now add (-10) + 5: -10_{10} --> 11110101

```
  11110101
 +00000101
 ---------
  11111010
```

- Since the result is a negative number we negate the result

  --> 00000101 --> 5_{10}
- How do we know the result 11111010 is a negative number?

One's Complement

- Another example:
- 102_{10}: 01100110 and 38_{10}: 00100110
- Now add 102 + (-38): -38_{10} --> 11011001

$$
\begin{array}{lr}
 & 01100110 \\
 & \underline{+11011001} \\
\text{(Carry 1)} & 00111111 \\
 & \underline{+1} \\
 & 01000000
\end{array}
$$

--> 64_{10}

- So the second add has a ripple effect and can take as much time as the first add!

One’s Complement

- Still two representations for zero 00000000 (+0) and 11111111 (-0)
- Sign bit is still the leftmost bit
- 15_{10} : 00001111 and -15_{10} --> 11110000
- So what if it’s a positive number like 128_{10} --> 1000000?
  - The MSB is part of the magnitude of the number
  - The 1 in MSB is not meant to indicate a negative number!
  - The number is too big to handle addition using this technique for signed integers
  - If you try to complement it --> 00000001
- 1s Complement range for 8 bits: -127_{10} to $+127_{10}$
- For N bits: Range is $-(2^{N-1}-1)$ to $2^{N-1}-1$

Two's Complement

- Express absolute value of a number
- If negative complement all bits i.e. reverse the bit value and always add 1
- Perform addition just like unsigned numbers
  - If the last (MSB) bit addition results in carry, simply ignore the carry
- 37_{10} : 00100101 and 15_{10} : 00001111
- -15_{10} would need 2's complement representation --> 11110001
- Now add 37 + (-15):

$$\begin{array}{r} 00100101 \\ \underline{+11110001} \\ 00010110 \end{array}$$

(Ignore carry) 00010110 --> 22_{10}

Two's Complement

- Negation is more complicated because it involves adding a 1 after complementing all bits
- Addition is easy
- Sign bit still in use
- There is only one representation of zero!
  - Negative 00000000 --> 11111111 + 1 --> 00000000 !
- But we have one more negative number than positive (easy to forget!)
- Take 11111111:
  - Sign bit is on so it's a negative number
  - Take 2s Complement and add 1: 00000000 + 1 = 00000001 which is -1_{10}
- Take 10000000:
  - Sign bit is on so it's a negative number
  - Take 2s Complement and add 1: 01111111 + 1 = 10000000 (original number) which is: -128_{10}
  - So rather than getting a positive value back we get a negative number again which is an indication of Overflow
- 2s Complement range for 8 bits: -128_{10} to $+127_{10}$
- For N bits: Range is -2^{N-1} to $2^{N-1}-1$

Two's Complement

- Leftmost bit still indicates sign
- Extra negative number instead of two representations for zero
- 2's Complement range for 8 bits: -128_{10} to $+127_{10}$
- For N bits: Range is -2^{N-1} to $2^{N-1}-1$
- All modern machines are 2's Complement

Two's Complement moving values between containers

- Promotion from small to large container
  - Say you want to move a value from 8 bits to 16 bits
  - Extend the sign bit value to all the upper bits of the target container
  - 15_{10} : 00001111 --> 0000 0000 0000 1111
  - Same with negative numbers also, replicate the top bits with the sign bit value:
    - -15_{10} : 11110001 --> 1111 1111 1111 0001
    - Let's verify this: If we take the 2's complement of this:

    1111 1111 1111 1111 0001 --> negate it --> 0000 0000 0000 1110 --> Add 1 --> 0000 0000 0000 1111
- Demotion from large to small container
  - Will work by chopping off upper bits as long as the value being moved does not cause an overflow
  - Try moving 15_{10} , -15_{10} , 257_{10} , -257_{10} from 16 bits to 8 bits
  - Scheme will work as long as all the chopped bit values are the same