Signed and Unsigned Numbers

Chapter 3.10

“Binary” means many things

- Binary values are used to store information in a computer
- That value can “encode” many different information types
- The same binary value as different meanings in hex, octal, and ASCII
  - 0100 0001 => 0x41 => 101 (octal) => A (ASCII)
  - 0100 0010 => 0x42 => 102 (octal) => B (ASCII)
- Binary does not always represent [0-2^n] numbers; it can store any format
  - 0(sign) 011111100(exp) 01000000000000(fraction) => 0.15625
  - 010001 => 17
  - 101111 => -17 (2’s complement)

So far “Binary” and “Hex” meant unsigned

- Unsigned 8-bit number => $[0 - (2^8 - 1)]$ => [0 – 127]

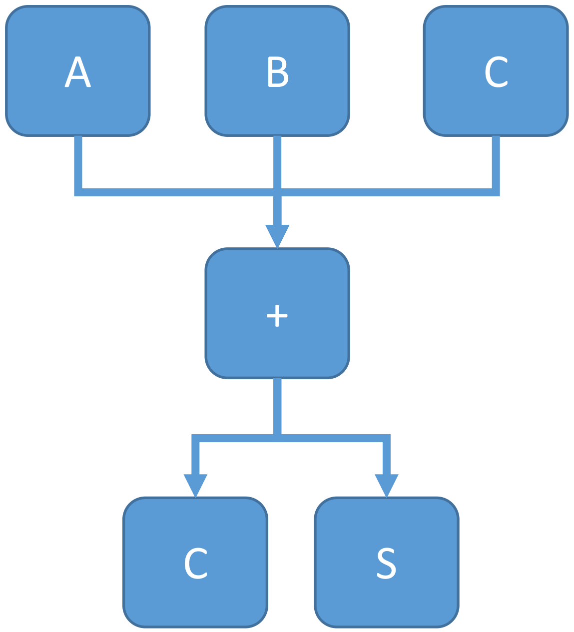
- All values are stored in a register and they “roll over” when the value exceeds 2^n - 1

- Register itself doesn’t roll over but calculations involving unsigned numbers “roll over”
  - Example: using a 3-bit register. 111 + 1 = 000
  - 5-bit register: 11111 + 1 => 00000

Addition Rules

- A + B + Cin = (Sum, Carry Out)

- 0 + 0 = Sum 0 Carry 0
- 1 + 0 = 0 + 1 = Sum 1 Carry 0
- 1 + 1 = Sum 0 Carry 0
- 1 + 1 +1 = Sum 1 Carry 1



Examples

1 + 1 =2

Carry	1	
A	0	1
B	0	1
Sum	**1**	**0**

2 + 1 = 3

Carry	0	
A	1	0
B	0	1
Sum	**1**	**1**

2 + 5 = 7

Carry	0	1	0
A	0	0	1
B	1	0	1
Sum	**1**	**1**	**0**

4 + 9 = 13

Carry	0	0	0	0
A	0	1	0	0
B	1	0	0	1
Sum	**1**	**1**	**0**	**1**

Bit-Width Matters => Overflow

(4bits) 9 + 9 = 18

Carry	0	0	1	0
A	1	0	0	1
B	1	0	0	1
Sum	**0**	**0**	**1**	**0**

Should be: 1 0010

Cannot represent the value 18 in 4-bits, result of addition overflows the registers

(5bits) 20 + 20 = 40

Carry	0	1	0	0	0
A	1	0	1	0	0
B	1	0	1	0	0
Sum	**0**	**1**	**0**	**0**	**0**

Should be: 10 1000

Cannot represent the value 40 in 5-bits, result of addition overflows the registers

How to add numbers…

- Two values: Augend (A), Addend (B)
- Create Sum and Carry

Carry In from previous addition

1

  1 5

\+ 1 7

\----------

$\boxed{3\ 2}$

Sum overflows to cause Carry Out of 1 (5+7=12)

Sum

Carry	1	
Augend (A)	1	5
Addend (B)	0	8
Sum	**2**	**3**

Adding 1-bit Binary Numbers

- Easy for {00, 01, 10}

X	Y	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Addition of {11} causes overflow
- Overflow is not problem on paper
- However, can't create "extra space" in hardware
- Take in to account all sums carry in/out situations

Examples

1 + 1 =2

Carry	1	
A	0	1
B	0	1
Sum	**1**	**0**

2 + 1 = 3

Carry	0	
A	1	0
B	0	1
Sum	**1**	**1**

1 + 5 = 6

Carry	0	1	0
A	0	0	1
B	1	0	1
Sum	**1**	**1**	**0**

4 + 9 = 13

Carry	0	0	0	0
A	0	1	0	0
B	1	0	0	1
Sum	**1**	**1**	**0**	**1**

How to handle negative numbers?

- How to represent -35? -110?

- Naïve solution is [sign bit] [unsigned valued]
  - -16 => [1] [0001 0000]
  - Leads to excessive bits and complicated hardware

- Use 2’s complement: (2)’ + 1

2’s Complement

- Use n-bits to represent values between $[-2^{n-1}\colon\ (2^{n-1}-1)]$
  - E.g. 5-bits: -15 to 14
  - E.g. 7 bits: -64 to 63

- Create two’s complement via the following:
  - Select the appropriate number of bits for the value you wish to store
    - Ex: -23 requires 6 bits
  - Represent the original unsigned value in binary
    - Ex: 23 => 01 0111
  - Invert the value and add 1
    - Ex: 01 0111 => 10 1000
    - 10 1000 + 1 => 10 1001 (-23)

2's Complement

- [Sign Bit][value]
- Do not read 1111 as -7. Only first bit give sign, need to calc others
- Can read 0111 as 7.

Convert 3 to -3

Value (3)	0	0	1	1
Invert	1	1	0	0
+1	0	0	0	1
2's	**1**	**1**	**0**	**1**

Convert 5 to -5

Value (5)	0	1	0	1
Invert	1	0	1	0
+1	0	0	0	1
2's	**1**	**0**	**1**	**1**

Range for two's complement: -2^{n-1} : +2^{n-1} -1

Need this extra bit to store -15. 4-bits gives -8 to 15

Convert 15 to -15

Value (3)	0	1	1	1	1
Invert	1	0	0	0	0
+1	0	0	0	0	1
2's	**1**	**0**	**0**	**0**	**0**

Two's complement	Decimal
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

2’s Complement

- Using two’s complement representation you can freely add signed and unsigned numbers

- However, ensure there are sufficient bits to store the result

- Expand unsigned numbers by adding 0 beyond MSB

- Expand signed numbers by adding 1 beyond MSB

Subtracting by Adding Signed and Unsigned

5 + -2 = 3

Carry	1	0	0	0
A	0	1	0	1
B	1	1	1	0
Sum	**0**	**0**	**1**	**1**

Discard carry 0

-5 + -2 = - 7

Carry	1	1		
A	1	0	1	1
B	1	1	1	0
Sum	**1**	**0**	**0**	**1**

Assuming there is no underflow or overflow, you can discard the carry out from the addition

Signed Overflow / Underflow

- Overflow: exceed upper bounds of number system
- Underflow: exceed lower bounds of number system

- Can’t simply look at carry-out bit; may be valid as extended sign

Signed Examples to Detect Under/over flow

4-bit Addition: 2 + 2 = 4

Carry	0	1	0	0
A	0	0	1	0
B	0	0	1	0
Sum	**0**	**1**	**0**	**0**

Carry out = 0

4-bit Addition: -3 + -3 = -6

Carry	1	0	1	
A	1	1	0	1
B	1	1	0	1
Sum	**1**	**0**	**1**	**0**

Carry out = 1

4-bit Overflow: 4 + 5 = 9

Carry	1	0	0	0
A	0	1	0	0
B	0	1	0	1
Sum	**1**	**0**	**0**	**1**

Carry out = 0

4-bit Underflow:-5 + -5 = -10

Carry	0	1	1	0
A	1	0	1	1
B	1	0	1	1
Sum	**0**	**1**	**1**	**0**

Carry out = 1

Signed Examples to Detect Under/over flow

4-bit Addition: 2 + 2 = 4

Carry	0	1	0	0
A	0	0	1	0
B	0	0	1	0
Sum	**0**	**1**	**0**	**0**

Carry out = 0

4-bit Addition: -3 + -3 = -6

Carry	1	0	1	
A	1	1	0	1
B	1	1	0	1
Sum	**1**	**0**	**1**	**0**

Carry out = 1

4-bit Overflow: 4 + 5 = 9

Carry	1	0	0	0
A	0	1	0	0
B	0	1	0	1
Sum	**1**	**0**	**0**	**1**

Carry out = 0

4-bit Underflow:-5 + -5 = -10

Carry	0	1	1	0
A	1	0	1	1
B	1	0	1	1
Sum	**0**	**1**	**1**	**0**

Carry out = 1

Overflow

- In unsigned addition, overflow occurs if final C is 1

- In signed addition, overflow occurs if the final carry out differs from the final carry in.