Tutorial 02:

Addition/Subtraction using 2's Complement & Floating-point Numbers

Computer Science Department
CS2208b: Fundamentals of Computer Organization and Architecture
Winter 2018
Instructor: Mahmoud R. El-Sakka
Office: MC-419
Email: elsakka@csd.uwo.ca
Phone: 519-661-2111 x86996

Binary Arithmetic

- These tables cover the fundamental arithmetic operations.

Addition
0 + 0 = 0 (carry 0)
0 + 1 = 1 (carry 0)
1 + 0 = 1 (carry 0)
1 + 1 = 0 (carry 1)

Subtraction
0 – 0 = 0 (borrow 0)
0 – 1 = 1 (borrow 1)
1 – 0 = 1 (borrow 0)
1 – 1 = 0 (borrow 0)

Multiplication
0 × 0 = 0
0 × 1 = 0
1 × 0 = 0
1 × 1 = 1

Addition (three bits)
0 + 0 + 0 = 0 (carry 0)
0 + 0 + 1 = 1 (carry 0)
0 + 1 + 0 = 1 (carry 0)
0 + 1 + 1 = 0 (carry 1)
1 + 0 + 0 = 1 (carry 0)
1 + 0 + 1 = 0 (carry 1)
1 + 1 + 0 = 0 (carry 1)
1 + 1 + 1 = 1 (carry 1)

Subtraction (three bits)
0 – 0 – 0 = 0 (borrow 0)
0 – 0 – 1 = 1 (borrow 1)
0 – 1 – 0 = 1 (borrow 1)
0 – 1 – 1 = 0 (borrow 1)
1 – 0 – 0 = 1 (borrow 0)
1 – 0 – 1 = 0 (borrow 0)
1 – 1 – 0 = 0 (borrow 0)
1 – 1 – 1 = 1 (borrow 1)

Sign and Magnitude Addition/Subtraction

- The operations are carried out similar to normal math calculations
- The resultant sign is arranged separately
  - The sign of A – B depends on the values of A and B
  - If B > A, the answer will be calculated as –(B – A), O.W., it is (A – B)
- The location of the radix points need to be aligned before performing the operation.
- If the provided number of bits are not enough to hold the result, it means an overflow occurred.

2's Complement Addition/Subtraction

- A subtraction operation is converted to an addition operation (after performing the *2's complement* to the operand appearing after the negative sign)

- When adding two positive numbers and finding the result is negative, this means an overflow occurred.
- When adding two negative numbers and finding the result is positive, this means an overflow occurred.

- Overflow will never occur when adding a positive number to a negative number, or vice versa.

- How about
  - subtracting a negative number from a positive number?
  - subtracting a positive number from a negative number?

2's Complement Addition/Subtraction

- *Example 1*:

  Perform $20_{10} - 10_{10}$ using 2's complement 6-bit system

- $20_{10} \rightarrow 10100_2$
- $10_{10} \rightarrow 1010_2$

- $20_{10} - 10_{10} \rightarrow 10100_2 - 1010_2$

  $\rightarrow 010100_2 - 001010_2$

  $\rightarrow 010100_2 + (-001010_2)$

  $\rightarrow 010100_2 + 110110_2$

  $\rightarrow 001010_2$

  $\rightarrow +10_{10}$

Carry out to be ignored

$C_{out} == C_{in}$

```
 11 1
  010100₂
 +110110₂
 1001010₂
```

Overflow can not occur

2's Complement Addition/Subtraction

- ***Example 2***:

  Perform $10_{10} - 20_{10}$ using 2's complement 6-bit system

- $10_{10} \rightarrow 1010_2$
- $20_{10} \rightarrow 10100_2$
- $10_{10} - 20_{10} \rightarrow 1010_2 - 10100_2$

  $\rightarrow 001010_2 - 010100_2$

  $\rightarrow 001010_2 + (-010100_2)$

  $\rightarrow 001010_2 + 101100_2$

  $\rightarrow 110110_2$

  $\rightarrow -001010_2$

  $\rightarrow -10_{10}$

No carry out

$C_{out} == C_{in}$

$$
\begin{array}{r}
1 \\
001010_2 \\
+101100_2 \\
\hline
110110_2
\end{array}
$$

Overflow can not occur

2's Complement Addition/Subtraction

- ***Example 3***:
  Perform $20_{10} + 10_{10}$ using 2's complement 6-bit system
- $20_{10} \rightarrow 10100_2$
- $10_{10} \rightarrow 1010_2$

- $20_{10} + 10_{10} \rightarrow 10100_2 + 1010_2$
  $\rightarrow 010100_2 + 001010_2$
  $\rightarrow 011110_2$
  $\rightarrow +30_{10}$

No carry out

$C_{out} == C_{in}$

$$\begin{array}{r} 010100_2 \\ +001010_2 \\ \hline 011110_2 \end{array}$$

Overflow might occur, but did not in this case

2's Complement Addition/Subtraction

- ***Example 4***:
  Perform $-20_{10} - 10_{10}$ using 2's complement 6-bit system
- $20_{10} \rightarrow 10100_2$
- $10_{10} \rightarrow 1010_2$
- $-20_{10} - 10_{10} \rightarrow -10100_2 - 1010_2$

$\rightarrow -010100_2 - 001010_2$

$\rightarrow (-010100_2) + (-001010_2)$

$\rightarrow 101100_2 + 110110_2$

$\rightarrow 100010_2$

$\rightarrow -011110_2$

$\rightarrow -30_{10}$

Carry out to be ignored

$C_{out} == C_{in}$

```
 1111
  101100₂
 +110110₂
 1100010₂
```

Overflow might occur, but did not in this case

2's Complement Addition/Subtraction

- *Example 5*:

  Perform $20_{10} + 20_{10}$ using 2's complement 6-bit system

- $20_{10} \rightarrow 10100_2$

- $20_{10} + 20_{10} \rightarrow 10100_2 + 10100_2$

  $\rightarrow 010100_2 + 010100_2$

No carry out

$C_{out} \neq C_{in}$

$$
\begin{array}{r}
1\ 1 \\
010100_2 \\
+010100_2 \\
\hline
101000_2
\end{array}
$$

Overflow might occur, and *indeed it did in this case*

2's Complement Addition/Subtraction

- *Example 6*:
  Perform $-20_{10} - 20_{10}$ using 2's complement 6-bit system
- $20_{10} \rightarrow 10100_2$
- $-20_{10} - 20_{10} \rightarrow -10100_2 - \quad 10100_2$

  $\rightarrow -010100_2 - 010100_2$

  $\rightarrow (-010100_2) + (-010100_2)$

  $\rightarrow \quad 101100_2 + \quad 101100_2$

Carry out to be ignored

$C_{out} \neq C_{in}$

```
 1 11
  101100₂
 +101100₂
 1011000₂
```

Overflow might occur, and *indeed it did in this case*

2's Complement Addition/Subtraction

- *Example 7*:
  Perform $20_{10} - 20_{10}$ using 2's complement 6-bit system
- $20_{10} \rightarrow 10100_2$
- $20_{10} - 20_{10} \rightarrow 10100_2 - 10100_2$

$\rightarrow 010100_2 - 010100_2$

$\rightarrow 010100_2 + (-010100_2)$

$\rightarrow 010100_2 + 101100_2$

$\rightarrow 000000_2$

$\rightarrow 0_{10}$

Carry out to be ignored

$C_{out} == C_{in}$

```
 1111
  010100₂
 +101100₂
 1000000₂
```

Overflow can not occur

2's Complement Addition/Subtraction

- *Example 8*:
  Perform 31_{10} + 1_{10} using 2's complement 6-bit system
- $31_{10} \rightarrow 11111_2$
- $1_{10} \rightarrow 1_2$

- $31_{10} + 1_{10} \rightarrow 11111_2 + 1_2$
  $\rightarrow 011111_2 + 000001_2$

No carry out

$C_{out} \neq C_{in}$

$$\begin{array}{r} 11111 \\ 011111_2 \\ +000001_2 \\ \hline 100000_2 \end{array}$$

Overflow might occur, and *indeed it did in this case*

2's Complement Addition/Subtraction

- *Example 9*:
  Perform $-31_{10} - 1_{10}$ using 2's complement 6-bit system
- $31_{10} \rightarrow 11111_2$
- $1_{10} \rightarrow 1_2$

Carry out to be ignored

- $-31_{10} - 1_{10} \rightarrow -11111_2 - 1_2$
  $\rightarrow (-011111_2) + (-000001_2)$
  $\rightarrow (100001_2) + (111111_2)$
  $\rightarrow 100000_2$
  $\rightarrow -100000_2$
  $\rightarrow -32_{10}$

$C_{out} == C_{in}$

```
 111111
  100001₂
 +111111₂
 1100000₂
```

Overflow might occur, but did not in this case

2's Complement Addition/Subtraction

- ***Example 10***:

  Encode -3.25_{10} using 2's complement 6-bit system

- $3.25_{10} \rightarrow 11.01_2$
- $-3.25_{10} \rightarrow -0011.01_2$

  $\rightarrow 1100.11_2$

You can also look at it as if it is $-3_{10} - 0.25_{10}$

- $-3_{10} - 0.25_{10} \rightarrow -11_2 - 0.01_2$

  $\rightarrow (-000011_2) + (-0000.01_2)$

  $\rightarrow (111101_2) + (1111.11_2)$

  $\rightarrow 111100.11_2$

  $\rightarrow 1100.11_2$

Carry out to be ignored

$C_{out} == C_{in}$

```
111111
 111101.00₂
+111111.11₂
1111100.11₂
```

Overflow might occur, but did not in this case

Binary points MUST be aligned

Example of Decimal to IEEE-754 Floating-point Conversion

- ***Example 11***:
  Convert 16777216.75_{10} into a *32-bit single-precision IEEE-754 FP* value.

  - o Convert 16777216.75_{10} into a fixed-point binary
    - 16777216_{10} = 1 0000 0000 0000 0000 0000 0000_2 and
    - $0.75_{10} = 0.11_2$.
    - Therefore, 16777216.75_{10} = 1 0000 0000 0000 0000 0000 0000.11_2.
  - o Normalize 1 0000 0000 0000 0000 0000 0000.11_2 to
    1.0000 0000 0000 0000 0000 0000 $11_2 \times 2^{24}$.
  - o The sign bit, S, is 0 because the number is positive
  - o The *biased exponent* is the *true exponent* plus 127; that is,
    $24 + 127 = 151_{10}$ = $1001\ 0111_2$
  - o The significand is 000 0000 0000 0000 0000 0000 011
    - *the leading 1 is stripped* and
    - *the significand to be rounded to 23 bits (rounded to nearest FP number)*.
  - o The final number is 0100 1011 1000 0000 0000 0000 0000 0000,
    or $4B800000_{16}$ ➔ 16777216_{10} *(i.e., there is 0.75 rounding error)*

What is the effect of using rounding toward +∞, rounding toward -∞, or rounding using truncation?

Example of Decimal to IEEE-754 Floating-point Conversion

- ***Example 12***:
  Convert 16777219_{10} into a *32-bit single-precision IEEE-754 FP* value.

  - Convert 16777219_{10} into a fixed-point binary
    - 16777219_{10} = 1 0000 0000 0000 0000 0000 0011_2 and
  - Normalize 1 0000 0000 0000 0000 0000 0011_2 to
    1.0000 0000 0000 0000 0000 $0011_2 \times 2^{24}$.
  - The sign bit, S, is 0 because the number is positive
  - The *biased exponent* is the *true exponent* plus 127; that is,
    $24 + 127 = 151_{10}$ = 1001 0111_2
  - The significand is 000 0000 0000 0000 0000 0001 1
    - *the leading 1 is stripped* and
    - *the significand to be rounded to 23 bits (rounded to nearest FP number)*.
  - The final number is 0100 1011 1000 0000 0000 0000 0000 0010,
    or $4B800002_{16}$ → 16777220_{10} *(i.e., there is 1.0 rounding error)*

What is the effect of using rounding toward +∞, rounding toward -∞, or rounding using truncation?

Mid-way → round to even significand

Example of Decimal to IEEE-754 Floating-point Conversion

- ***Example 13***: *Convert* 3.6_{10} into a *32-bit single-precision IEEE-754 FP* value.

$0.6 \times 2 = 1.2$
$0.2 \times 2 = 0.4$
$0.4 \times 2 = 0.8$
$0.8 \times 2 = 1.6$
$0.6 \times 2 = 1.2$
…

- Convert 3.6_{10} into a fixed-point binary
  - $3_{10} = 11_2$ and
  - $0.6_{10} = 0.1001\ 1001\ \ldots\ _2$.
  - Therefore, $3.6_{10} = 11.1001\ 1001\ \ldots\ _2$
- Normalize $11.1001\ 1001\ \ldots\ _2$ to
  $1.11001\ 1001\ \ldots\ _2 \times 2^1$.
- The sign bit, S, is 0 because the number is positive
- The *biased exponent* is the *true exponent* plus 127; that is,
  $1 + 127 = 128_{10} = 1000\ 0000_2$
- The significand is 110 0110 0110 0110 0110 0110 0110 0110 …
  - *the leading 1 is stripped* and
  - *the significand to be rounded to 23 bits (rounded to nearest FP number)*.
- The final number is 0100 0000 0110 0110 0110 0110 0110 0110,
  or 40666666_{16}. ➔ 3.599999046325684_{10}

Example of Decimal to IEEE-754 Floating-point Conversion

- *Example 14*: *Convert* $5.877472_{10} \times 10^{-39}$ into a *32-bit single-precision IEEE-754 FP* value.

$\mathrm{Log}_2(10) = 1 / \log_{10}(2)$

$10^{-39} = 2^z \rightarrow \log_2(10^{-39}) = z \rightarrow -39 \times \log_2(10) = z \rightarrow z = -129.5551957$
$10^{-39} = 2^{-129.5551957} = 2^{-129} \times 2^{-0.5551957} = = 2^{-129} \times 0.680564734_{10}$
$5.877472_{10} \times 10^{-39} = 5.877472_{10} \times 0.680564734_{10} \times 2^{-129}$
$= 4_{10} \times 2^{-129} = 1_{10} \times 2^{-127}$

- Convert 1_{10} into a fixed-point binary
  - $1_{10} = 1.0_2$ (*already normalized*)
- *True exponent is less than -126 ➔ underflow case*
  - *The exponent needs to be -126:* $-127_{10} = -126 - 1$
  - *Hence, the significant needs to be adjusted to compensate the -1*
  - After moving the radix point backward by 1 position ➔ $0.\mathbf{1}_2$
    i.e., $2^{-127} = 0.1 \times 2^{-126}$
  - After Taking 23 bits ➔ 0. **1**00 0000 0000 0000 0000 0000_2
- The sign bit, S, is 0 because the number is positive
- The final number is 0000 0000 0100 0000 0000 0000 0000 0000 or 00400000_{16}

Example of Decimal to IEEE-754 Floating-point Conversion

- *Example 15*: *Convert* $9.0_{10} \times 10^{-44}$ into a *32-bit single-precision IEEE-754 FP* value.

$\text{Log}_2(10) = 1 / \log_{10}(2)$

$10^{-44}=2^z \rightarrow \log_2(10^{-44})= z \rightarrow -44\times\log_2(10)= z \rightarrow z = -146.164836175$
$10^{-44} = 2^{-146.164836175} = 2^{-146} \times 2^{-0.164836175} = = 2^{-146} \times 0.892029808_{10}$
$9.0_{10} \times 10^{-44} = 9.0_{10} \times 0.892029808_{10} \times 2^{-146} = 8.028268272_{10} \times 2^{-146}$

- Convert 8.028268272_{10} into a fixed-point binary
  - $8_{10} = 1000_2$ and
  - $0.028268272_{10} = 0.000001110011110010 01\ldots_2$.
  - Therefore, $8.028268272_{10} = 1000.00000111001111001001\ldots_2$.
- *Normalization:* $9.0_{10} \times 10^{-44} = 8.028268272_{10} \times 2^{-146} = 1000.00000111001111001001\ldots_2 \times 2^{-146} = 1.00000000111001111001001\ldots_2 \times 2^{-143}$
- *True exponent is less than -126 ➔ underflow case*
  - *The exponent needs to be -126:* $-143_{10} = -126 - 17$
  - *Hence, the significant needs to be adjusted to compensate the -17*
  - After moving the radix point backward by 17 position
    ➔ 0. **0000 0000 0000 0000 1**000 0000 0111001111001001…$_2$

rounded

  - After Taking only 23 bits ➔ 0. **000 0000 0000 0000 01**00 0000 0011…$_2$
- The sign bit, S, is 0 because the number is positive
- The final number is 0000 0000 0000 0000 0000 0000 0100 0000 or 00000040_{16}

Example of IEEE-754 Floating-point to Decimal Conversion

- ***Example 16***: *Convert* $FE600000_{16}$ from *32-bit single-precision IEEE-754 FP* value into a decimal value.
  - Convert the hexadecimal number ($FE600000_{16}$) into binary form

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

  - Unpack the number into *sign bit*, *biased exponent*, and *fractional significand*.
    - S = 1
    - E = 1111 1100
    - F =110 0000 0000 0000 0000 0000
  - As the sign bit is 1, the number is negative.
  - We subtract 127 from the *biased exponent* $1111\ 1100_2$ to get the *true exponent* ➔ $1111\ 1100_2 - 0111\ 1111_2 = 0111\ 1101_2 = 125_{10}$.
  - The fractional significand is $.110\ 0000\ 0000\ 0000\ 0000\ 0000_2$.
  - Reinserting the leading one gives $\mathbf{1}.110\ 0000\ 0000\ 0000\ 0000\ 0000_2$.
  - The number is $-\mathbf{1}.11_2 \times 2^{125} = -1.75 \times 2^{125}$

$2^{125} = 10^{z}$ ➔ $\log_{10}(2^{125}) = z$ ➔ $z = 37.62874946$

$2^{125} = 10^{37.62874946} = 10^{37} \times 10^{0.62874946} = 10^{37} \times 4.253529587$

$-1.75 \times 2^{125} = -1.75 \times 4.253529587 \times 10^{37} = -7.443676776 \times 10^{37}$

Example of IEEE-754 Floating-point to Decimal Conversion

- ***Example 17***: *Convert* 00200000_{16} from *32-bit single-precision IEEE-754 FP* value into a decimal value.
  - Convert the hexadecimal number (00200000_{16}) into binary form

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

  - Unpack the number into *sign bit*, *biased exponent*, and *fractional significand*.
    - S = 0
    - E = 0000 0000
    - F =010 0000 0000 0000 0000 0000
  - As the sign bit is 0, the number is positive.
  - We subtract 126 from the *biased exponent* 0_2 to get the *true exponent* ➔ $0_2 - 0111\ 1110_2 = -126_{10}$.
    As the true exponent is -126, then the F is not normalized
  - The fractional significand is $.010\ 0000\ 0000\ 0000\ 0000\ 0000_2$.
  - The number is $.01_2 \times 2^{-126} = 2^{-2} \times 2^{-126} = 2^{-128}$

We are subtracting 126, not 127, from the biased exponent, because the biased exponent = 0.

$2^{-128} = 10^{z}$ ➔ $\log_{10}(2^{-128}) = z$ ➔ $z = -38.53183944$

$2^{-128} = 10^{-38.53183944} = 10^{-38} \times 10^{-0.53183944} = 10^{-38} \times 0.293873587$

$2^{-128} = 0.293873587 \times 10^{-38} = 2.9387358 \times 10^{-39}$

Example of IEEE-754 FP to Decimal to IEEE-754 FP Conversion

- ***Example 18***:
  Convert $4B800002_{16}$ from the *32-bit single-precision IEEE-754 FP* representation into decimal representation. ***Then*** add 1.0_{10} to the result. And ***finally*** convert it back to the *32-bit single-precision IEEE-754 FP* representation.

  - Convert the hexadecimal number ($4B800002_{16}$) into binary form

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	1	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

  - Unpack the number into *sign bit*, *biased exponent*, and *fractional significand*.
    - S = 0
    - E = 1001 0111
    - F =000 0000 0000 0000 0000 0010
  - As the sign bit is 0, the number is positive.
  - We subtract 127 from the *biased exponent* $1001\ 0111_2$ to get the *true exponent* ➔ $1001\ 0111_2 - 0111\ 1111_2 = 0001\ 1000_2 = 24_{10}$.
  - The fractional significand is $.000\ 0000\ 0000\ 0000\ 0000\ 0010_2$.
  - Reinserting the leading one gives $\mathbf{1}.000\ 0000\ 0000\ 0000\ 0000\ 0010_2$.
  - The number is $+(\mathbf{1} + 2^{-22})\times 2^{24} = 2^{24} + 2^{2} = 1024_{10} \times 1024_{10} \times 16_{10} + 4_{10}$
    $= 16777216_{10} + 4_{10} = 16777220_{10}$

Example of IEEE-754 FP to Decimal to IEEE-754 FP Conversion

- ***Example 18 (continution)***:
  Adding 1.0_{10} to the result ➔ $16777220_{10} + 1.0_{10} = 16777221_{10}$

  Converting the result back to the *32-bit single-precision IEEE-754 FP* format

  - Convert 16777221_{10} into a fixed-point binary
    - $16777221_{10} = 1\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101_2$ and
  - Normalize $1\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101_2$ to
    $1.0000\ 0000\ 0000\ 0000\ 0000\ 0101_2 \times 2^{24}$.
  - The sign bit, S, is 0 because the number is positive
  - The *biased exponent* is the *true exponent* plus 127; that is,
    $24 + 127 = 151_{10} = 1001\ 0111_2$
  - The significand is 000 0000 0000 0000 0000 0010 1

    Mid-way ➔ round to even significand
    - *the leading 1 is stripped* and
    - *the significand to be rounded to 23 bits (rounded to nearest FP number)*.
  - The final number is 0100 1011 1000 0000 0000 0000 0000 0010,
    or $4B800002_{16}$ ➔ 16777220_{10}

16777220_{10} + 1.0_{10} = 16777220_{10}!!!

(This is due to the rounding error)

Example of IEEE-754 FP to Decimal to IEEE-754 FP Conversion

- ***Example 18 (continution)***:

Note that:

- $16777220_{10} = 1.0000\ 0000\ 0000\ 0000\ 0000\ 0101_2 \times 2^{24}$
- $1.0_{10} = 1_2 \times 2^0$
- The absolute difference between the exponents of the two FP normalized numbers = 24
- The significand is expressed in 23 bits
- As the absolute *difference* between the exponents of the two FP normalized numbers is ≥ the *number of significand bits + 1* ➔ the result is *the larger number of the two*, which is 16777220_{10}
- Run the following program to verify Example 18:

```
#include <stdio.h>
int main()
{
  float f = 16777220, ff;
  ff = f + 1;
  printf("%f %f \n", f, ff);
}
```

Change the "`float`" to "`int`" and the "`%f`" to "%d" and repeat executing the program again.

Final Word!!

- **How can I verify my results?**
- There are many online converters between IEEE FP format to float.
  - For example, https://www.h-schmidt.net/FloatConverter/IEEE754.html