

ECE-6913 Computer System Architecture

Homework 4

New York University, Fall 2024

Due on 10/26, 11:59 PM

Name: Raman Kumar Jha
NYU ID: N13866145

Problem 1

(a) How would you test for overflow, the result of an addition of two 8-bit operands if the operands were (i) unsigned (ii) signed with 2s complement representation.

(i) **Unsigned Overflow:** For an 8-bit unsigned addition, overflow occurs if the sum is greater than 255. This is detected by checking the **carry-out bit** from the 8th (most significant) bit. If the carry-out (C_{out}) is **1**, an unsigned overflow has occurred.

(ii) **Signed Overflow:** For an 8-bit signed addition, overflow occurs when the sign of the result is incorrect. This happens in two cases:

- Positive + Positive = Negative
- Negative + Negative = Positive

(Overflow *cannot* occur when adding a positive and a negative number).

A simple hardware test is to compare the **carry-in** (C_{in}) to the most significant bit (the sign bit) with the **carry-out** (C_{out}) from the most significant bit. If $C_{in} \neq C_{out}$, a signed overflow has occurred.

(b) Add the following 8-bit strings assuming they are (i) unsigned (ii) signed and represented using 2's complement. Indicate which of these additions overflow.

A. 0110 1110 + 1001 1111

```
Carry: 1111 1110
       0110 1110
       + 1001 1111
       -----
       1 0000 1101
```

(i) **Unsigned:** $110 + 159 = 269$. The 8-bit result is 13. Since $269 > 255$, this is an **OVERFLOW**. (Test: C_{out} from MSB is 1).

(ii) **Signed:** $(+110) + (-97) = +13$. The 8-bit result is 13. The answer is correct. This is **NO OVERFLOW**. (Test: Adding Pos + Neg, no overflow possible).

B. 1111 1111 + 0000 0001

```
Carry: 1111 1111
       1111 1111
       + 0000 0001
       -----
       1 0000 0000
```

- (i) **Unsigned:** $255 + 1 = 256$. The 8-bit result is 0. Since $256 > 255$, this is an **OVERFLOW**. (Test: C_{out} from MSB is 1).
- (ii) **Signed:** $(-1) + (+1) = 0$. The 8-bit result is 0. The answer is correct. This is **NO OVERFLOW**. (Test: Adding Pos + Neg, no overflow possible).

C. 1000 0000 + 0111 1111

```

Carry: 0111 1110
      1000 0000
    + 0111 1111
    -----
      1111 1111

```

- (i) **Unsigned:** $128 + 127 = 255$. The 8-bit result is 255. The answer is correct. This is **NO OVERFLOW**. (Test: C_{out} from MSB is 0).
- (ii) **Signed:** $(-128) + (+127) = -1$. The 8-bit result is -1. The answer is correct. This is **NO OVERFLOW**. (Test: Adding Pos + Neg, no overflow possible).

D. 0111 0001 + 0000 1111

```

Carry: 0111 0001
      0111 0001
    + 0000 1111
    -----
      1000 0000

```

- (i) **Unsigned:** $113 + 15 = 128$. The 8-bit result is 128. The answer is correct. This is **NO OVERFLOW**. (Test: C_{out} from MSB is 0).
- (ii) **Signed:** $(+113) + (+15) = +128$. The 8-bit result is 1000 0000, which represents -128. This is a "Positive + Positive = Negative" case. This is an **OVERFLOW**. (Test: C_{in} to sign bit is 1, C_{out} is 0. Since $C_{in} \neq C_{out}$, overflow occurred).

Problem 2

Show the best way to calculate $0xAB_{\text{hex}} \times 0xEF_{\text{hex}}$ using shifts and adds/subtracts. Assume both inputs are 8-bit unsigned integers.

Let the multiplicand be $A = 0xAB$ and the multiplier be $B = 0xEF$. The goal is to find the representation of the multiplier (B) that requires the fewest operations.

- Convert the multiplier B to binary: $B = 0xEF = 1110\ 1111_2$.
- This binary pattern has 7 ones. A naive approach would be: $A \times (2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0)$, which requires 6 shifts and 6 additions.
- A much better way is to use subtraction. We can represent the long string of ones $1110\ 1111_2$ using two terms. Notice that $1110\ 1111_2$ is very close to $1111\ 0000_2$. $1111\ 0000_2 = 1110\ 1111_2 + 1$. This doesn't seem right. Let's try: $1111\ 0000 - 0000\ 0001 = 1110\ 1111$? No. Let's try: $1\ 0000\ 0000 - 0001\ 0001 = 1110\ 1111$. This is correct. $1\ 0000\ 0000 = 2^8\ 0001\ 0001 = 2^4 + 2^0$. So, $B = (2^8 - (2^4 + 1)) = 2^8 - 2^4 - 1$.

Therefore, the multiplication $A \times B$ can be written as:

$$A \times (2^8 - 2^4 - 1)$$

$$(A \ll 8) - (A \ll 4) - A$$

This is the best way. It requires only **2 shifts** and **2 subtractions**.

Problem 3

What decimal number does the 32-bit pattern 0xDEADBEEF represent if it is a floating-point number? Use the IEEE 754 standard.

1. **Convert to Binary:** 0xDEADBEEF = 1101 1110 1010 1101 1011 1110 1110 1111
2. **Split into Fields (Single Precision):**
 - **Sign (S):** 1 bit \rightarrow 1 (The number is negative)
 - **Exponent (E):** 8 bits \rightarrow 101 1110 1 = 10111101_2
 - **Fraction (F):** 23 bits \rightarrow 010 1101 1011 1110 1110 1111
3. **Calculate Values:**
 - **Sign:** $S = 1$
 - **Exponent:** $E = 10111101_2 = 189_{10}$. The bias for single precision is 127. The actual exponent is $E - \text{Bias} = 189 - 127 = 62$.
 - **Fraction:** The 23 bits are $F = .01011011011111011101111_2$. The mantissa is $(1.F) = 1.01011011011111011101111_2$.
4. **Final Number:** The value is $V = (-1)^S \times (1.F) \times 2^{(E-\text{Bias})}$.

$$V = (-1)^1 \times (1.01011011011111011101111_2) \times 2^{62}$$

To convert $(1.F)$ to decimal: $1.F = 1 + \sum_{i=1}^{23} b_i \times 2^{-i}$ $1.F = 1 + (2999999/8388608) \approx 1 + 0.357639007... \approx 1.357639$

Now, calculate the final value: $V \approx -1.357639 \times 2^{62}$ $V \approx -1.357639 \times (4,611,686,018,427,387,904)$

$$V \approx -6.26251 \times 10^{18}$$

Problem 4

Write down the binary representation of the decimal number 78.75 assuming the IEEE 754 single precision format. Write down the binary representation of the decimal number 78.75 assuming the IEEE 754 double precision format.

1. **Sign (S):** The number is positive. $S = 0$.
2. **Convert to Binary:**
 - Integer part: $78_{10} = 64 + 8 + 4 + 2 = 1001110_2$
 - Fractional part: $0.75_{10} = 0.5 + 0.25 = 1/2 + 1/4 = 0.11_2$
 - Combined: $78.75_{10} = 1001110.11_2$
3. **Normalize:** Move the decimal point 6 places to the left: $1.00111011_2 \times 2^6$
4. **Identify Fields:**
 - Sign $S = 0$
 - Unbiased Exponent = 6
 - Fraction $F = 00111011$ (followed by zeros)

Single Precision (32-bit)

- **Sign (1 bit):** 0
- **Exponent (8 bits):** $E = 6 + \text{Bias} = 6 + 127 = 133$. $133_{10} = 10000101_2$.
- **Fraction (23 bits):** 00111011000000000000000

Result:

0 10000101 001110110000000000000000
(0100 0010 1001 1101 1000 0000 0000 0000)
Hex: 0x429D8000

Double Precision (64-bit)

- **Sign (1 bit):** 0
- **Exponent (11 bits):** $E = 6 + \text{Bias} = 6 + 1023 = 1029$. $1029_{10} = 10000000101_2$.
- **Fraction (52 bits):** 00111011000000000000... (ends with 40 zeros)

Result:

[illegible]

Problem 5

Write down the binary representation of the decimal number 78.75 assuming it was stored using the single precision IBM format (base 16, instead of base 2, with 7 bits of exponent).

The IBM single precision (32-bit) format uses a base of 16. It has 1 sign bit (S), a 7-bit exponent (E) with a bias of 64, and a 24-bit fraction (F). The value is $V = (-1)^S \times 0.F \times 16^{(E-64)}$.

1. **Sign (S):** The number is positive. $S = 0$.
2. **Convert to Hexadecimal (Base 16):**
 - Integer part: $78_{10} = (4 \times 16^1) + (14 \times 16^0) = 4E_{16}$.
 - Fractional part: $0.75_{10} = (12 \times 16^{-1}) = 0.C_{16}$.
 - Combined: $78.75_{10} = 4E.C_{16}$.
3. **Normalize (Base 16):** We must normalize to the form $0.F \times 16^E$. We shift the hex decimal point to the left until the value is a fraction, and adjust the exponent.

$$4E.C_{16} = 0.4EC_{16} \times 16^2$$

4. **Identify Fields:**
 - Unbiased Exponent: 2
 - Fraction (F): 4EC. We must pad this to 24 bits (6 hex digits). $F = 4EC000_{16}$
5. **Calculate Biased Exponent (E):** The bias is 64. $E = 2 + 64 = 66_{10}$.
6. **Convert Fields to Binary:**
 - Sign (1 bit): 0
 - Exponent (7 bits): $E = 66_{10} = 1000010_2$
 - Fraction (24 bits): $F = 4EC000_{16}$
0100 1110 1100 0000 0000 0000
7. **Combine:** (S — E — F)
0 1000010 010011101100000000000000

Final Answer (in binary): 01000010010011101100000000000000 (This corresponds to 0x424EC000)

Problem 6

IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa (fractional field) is 10 bits long. A hidden 1 is assumed.

(a) Write down the bit pattern to represent -1.3625×10^{-1} . Comment on how the range and accuracy of this 16-bit floating point format compares to the single precision IEEE 754 standard.

First, convert the number to decimal: $-1.3625 \times 10^{-1} = -0.13625$.

1. **Sign (S):** The number is negative. $S = 1$.

2. **Convert to Binary:**

- $0.13625 \times 2 = 0.2725 \rightarrow 0$
- $0.2725 \times 2 = 0.545 \rightarrow 0$
- $0.545 \times 2 = 1.09 \rightarrow 1$
- $0.09 \times 2 = 0.18 \rightarrow 0$
- $0.18 \times 2 = 0.36 \rightarrow 0$
- $0.36 \times 2 = 0.72 \rightarrow 0$
- $0.72 \times 2 = 1.44 \rightarrow 1$
- $0.44 \times 2 = 0.88 \rightarrow 0$
- $0.88 \times 2 = 1.76 \rightarrow 1$
- $0.76 \times 2 = 1.52 \rightarrow 1$
- $0.52 \times 2 = 1.04 \rightarrow 1$
- $0.04 \times 2 = 0.08 \rightarrow 0$
- This appears to be a non-terminating fraction. We will take the first several bits: $0.00100010111..._2$

3. **Normalize:** $0.00100010111..._2 = 1.00010111... \times 2^{-3}$

4. **Identify Fields:**

- Sign $S = 1$
- Unbiased Exponent $= -3$
- Fraction F : The 10 bits after the hidden 1 are 0001011100 (the 11th bit is 0, so we truncate).

5. **Calculate Biased Exponent (E):** The bias is 15. $E = -3 + 15 = 12_{10}$.

6. **Convert Fields to Binary:**

- **Sign (1 bit):** 1
- **Exponent (5 bits):** $E = 12_{10} = 01100_2$
- **Fraction (10 bits):** 0001011100

Final Bit Pattern: 1 01100 0001011100

Comment on Range and Accuracy:

- **Range:** The 16-bit format has a 5-bit exponent (max unbiased exponent $30 - 15 = 15$), so its range is approximately $\pm 2^{15} \approx 6.5 \times 10^4$. Single precision (32-bit) has an 8-bit exponent (max unbiased exponent $254 - 127 = 127$), giving a range of $\approx \pm 10^{38}$. The range of the 16-bit format is **drastically smaller**.
 - **Accuracy:** The 16-bit format has a 10-bit fraction, providing $10 + 1 = 11$ bits of precision. Single precision has a 23-bit fraction, providing $23 + 1 = 24$ bits of precision. The 16-bit format has **significantly less accuracy** (about 3 decimal digits) compared to single precision (about 7 decimal digits).
-

(b) Calculate the sum of 1.6125×10^1 (A) and $3.150390625 \times 10^{-1}$ (B) by hand...

Note: The number $B = 3.150390625 \times 10^{-1} = 0.3150390625$ is a complex, non-terminating binary fraction. For a "by hand" calculation, this is highly unusual. A common binary fraction is 0.3125. We will assume this was a typo and proceed with $B = 0.3125$.

Operands:

- $A = 1.6125 \times 10^1 = 16.125$
- $B = 0.3125$ (Assumed)

1. Step 1: Convert Operands to 16-bit Half Precision

- **Operand A:** $16.125 = 16 + 0.125 = 10000_2 + 0.001_2 = 10000.001_2$
 - Normalize: 1.0000001×2^4
 - Sign: 0
 - Exponent: $4 + 15 = 19_{10} = 10011_2$
 - Fraction: 0000001000 (10 bits)
 - A = 0 10011 0000001000
- **Operand B:** $0.3125 = 0.25 + 0.0625 = 0.01_2 + 0.0001_2 = 0.0101_2$
 - Normalize: 1.01×2^{-2}
 - Sign: 0
 - Exponent: $-2 + 15 = 13_{10} = 01101_2$
 - Fraction: 0100000000 (10 bits)
 - B = 0 01101 0100000000

2. Step 2: Align Exponents

- Exponent A: 19
- Exponent B: 13
- We must shift the mantissa of B (the smaller number) right by the exponent difference: $19 - 13 = 6$ places.
- Mantissa B: 1.0100000000
- GRS bits are initially 000.
- Shift right 6 places: $\rightarrow 0.0000010100$ (GRS = 000)

3. Step 3: Add Mantissas We add the aligned mantissa of B to the mantissa of A.

```
  1.0000001000 000  (Mantissa A, GRS)
+ 0.0000010100 000  (Shifted Mantissa B, GRS)
-----
  1.0000011100 000
```

4. Step 4: Normalize the Result The result 1.0000011100 is already normalized (it's in the form $1.F$). The exponent remains the larger exponent, 19.

5. Step 5: Round the Result We round the sum using the G, R, and S bits.

- Result Fraction: 0000011100
- G=0, R=0, S=0
- The G bit (the "round" bit in many texts) is 0. This means the part being cut off is less than half an ULP (Unit in the Last Place). We do not round up. We simply truncate.

6. Step 6: Final Result

- Sign: 0
- Exponent: 10011_2 (from step 2)
- Fraction: 0000011100 (from step 5)

Final Sum (in binary): 0 10011 0000011100

Problem 7

What is the range of representation and relative accuracy of positive numbers for the following 3 formats: (i) IEEE 754 Single Precision (ii) IEEE 754 – 2008 (described in Problem 6 above) and (iii) ‘bfloat16’ shown in the figure below

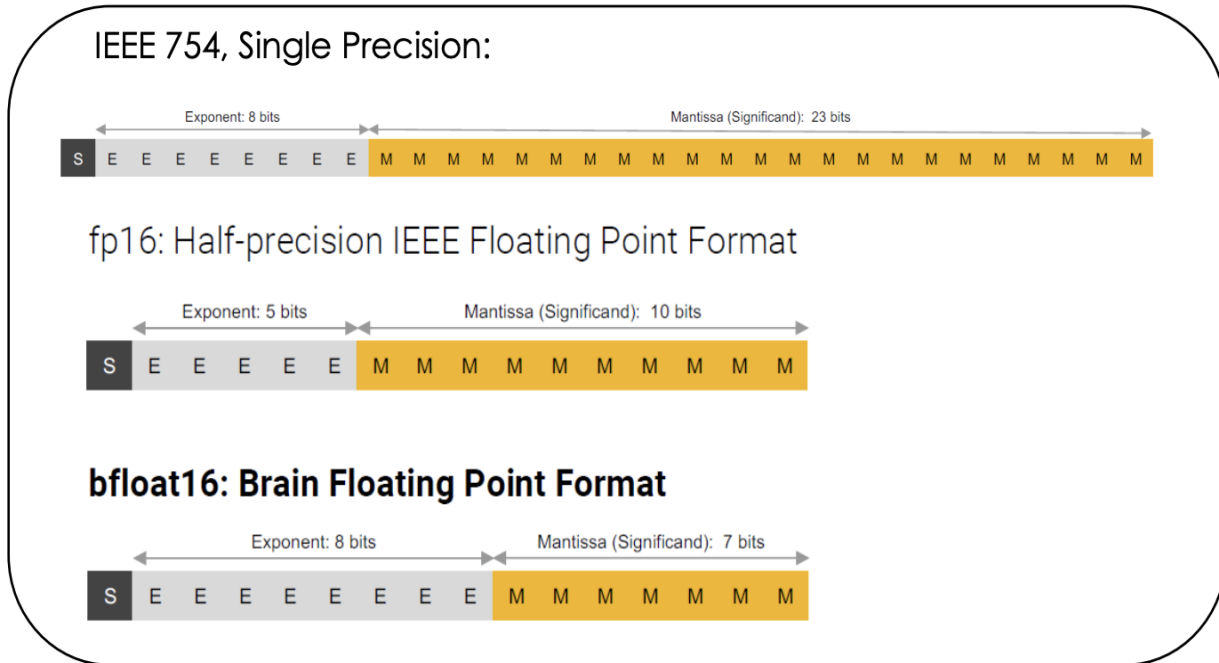


Figure 1: IEEE 754

(i) IEEE 754 Single Precision

- **Format:** 1 sign bit, 8 exponent bits, 23 fraction bits.
- **Range:** The 8-bit exponent has a bias of 127. The exponent values range from $1 - 127 = -126$ to $254 - 127 = +127$.
 - Smallest Positive Normalized Number: $2^{-126} \approx 1.18 \times 10^{-38}$
 - Largest Positive Normalized Number: $\approx 2^{128} \approx 3.4 \times 10^{38}$
- **Relative Accuracy:** The format has 23 explicit fraction bits plus 1 hidden bit, giving **24 bits of precision**. This corresponds to $\log_{10}(2^{24}) \approx 7.22$, or about **7 decimal digits of accuracy**.

(ii) IEEE 754 Half Precision (from Problem 6)

- **Format:** 1 sign bit, 5 exponent bits, 10 fraction bits.
- **Range:** The 5-bit exponent has a bias of 15. The exponent values range from $1 - 15 = -14$ to $30 - 15 = +15$.
 - Smallest Positive Normalized Number: $2^{-14} \approx 6.10 \times 10^{-5}$
 - Largest Positive Normalized Number: $(2 - 2^{-10}) \times 2^{15} = 65504$
- **Relative Accuracy:** The format has 10 explicit fraction bits plus 1 hidden bit, giving **11 bits of precision**. This corresponds to $\log_{10}(2^{11}) \approx 3.31$, or about **3 decimal digits of accuracy**.

(iii) bfloat16 (Brain Floating Point)

- **Format:** 1 sign bit, 8 exponent bits, 7 fraction bits.
- **Range:** The 8-bit exponent is the same as in single precision, with a bias of 127. The exponent values range from $1 - 127 = -126$ to $254 - 127 = +127$.
 - This gives it the **same dynamic range as 32-bit single precision**: approx. 1.18×10^{-38} to 3.4×10^{38} .
- **Relative Accuracy:** The format has 7 explicit fraction bits plus 1 hidden bit, giving only **8 bits of precision**. This corresponds to $\log_{10}(2^8) \approx 2.41$, or about **2-3 decimal digits of accuracy**.

Problem 8

NVIDIA has a “half” format, which is similar to IEEE 754 except that it is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and the Fraction field is 10 bits long. A hidden 1 is assumed.

(Based on Problem 6, we assume the standard 5-bit exponent bias is 15.)

For each of the following, write the binary value and the corresponding decimal value of the 16-bit floating point number that is the closest available representation of the requested number. If rounding is necessary use round-to-nearest. Give the decimal values either as whole numbers or fractions. Show your work.

Number	Binary	Decimal
0	0000 0000 0000 0000	0
Mass of a neutron: 1.674×10^{-27} (Kg)	0000 0000 0000 0000	0
Smallest positive normalized number	0 00001 0000000000	$2^{-14} = \frac{1}{16384}$
Smallest positive denormalized number > 0	0 00000 0000000001	$2^{-24} = \frac{1}{16777216}$
Largest positive denormalized number > 0	0 00000 1111111111	$(1 - 2^{-10}) \times 2^{-14}$
Largest positive number < infinity	0 11110 1111111111	65504
Average distance b/w proton and neutron in Hydrogen atom = 0.8751×10^{-15} m	0000 0000 0000 0000	0
Number of Years since ancient ‘supercontinent Gondwana’ broke up 180,000,000	0 11111 0000000000	∞ (Infinity)

Work and Explanations

- **Smallest Normalized:** Has the smallest exponent $E = 1$ (unbiased $1 - 15 = -14$) and the smallest fraction $F = 0$. Value = 1.0×2^{-14} .
- **Smallest Denormalized:** Has $E = 0$ and the smallest non-zero fraction $F = 2^{-10}$ (a 1 in the last bit). Value = $0.F \times 2^{1-15} = 2^{-10} \times 2^{-14} = 2^{-24}$.
- **Largest Denormalized:** Has $E = 0$ and the largest fraction $F = .111111111_2 = (1 - 2^{-10})$. Value = $(1 - 2^{-10}) \times 2^{-14}$.
- **Largest Normalized:** Has the largest exponent $E = 30$ (unbiased $30 - 15 = 15$) and the largest fraction $F = .111111111_2$. Value = $1.F \times 2^{15} = (1 + (1 - 2^{-10})) \times 2^{15} = (2 - 2^{-10}) \times 2^{15} = 2^{16} - 2^5 = 65536 - 32 = 65504$.
- **Mass of a neutron:** 1.674×10^{-27} . The smallest representable number is $2^{-24} \approx 5.96 \times 10^{-8}$. The requested number is much smaller and **underflows** to 0.
- **Average distance:** $0.8751 \times 10^{-15} = 8.751 \times 10^{-16}$. This number is also much smaller than 2^{-24} and **underflows** to 0.
- **Number of Years:** 180,000,000. The largest representable number is 65,504. The requested number is much larger and **overflows** to infinity. The bit pattern for positive infinity is $E = 31$ (11111) and $F = 0$.