Tutorial 4

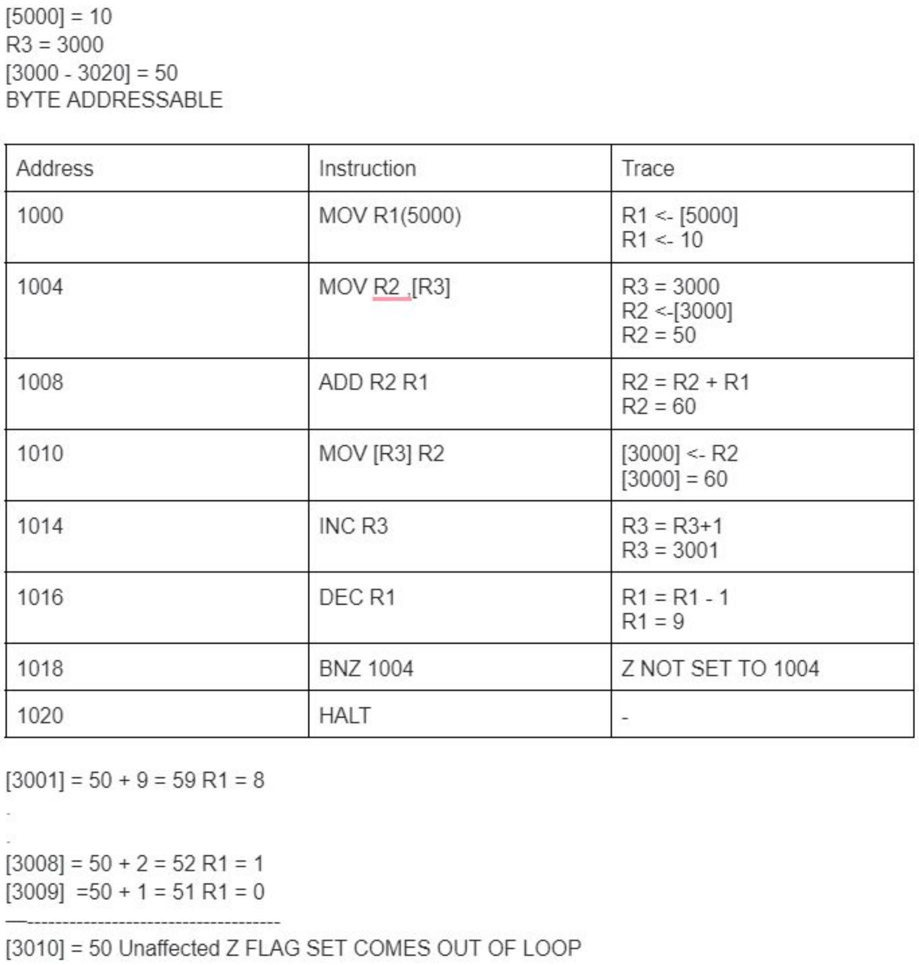
CSN 221

Consider the following instruction sequence where registers R_1, R_2 and R_3 are general purpose and $MEMORY[X]$ denotes the content at the memory location X.

Instruction	**Semantics**	**Instruction Size** (bytes)
MOV $R1$, (5000)	$R1 \leftarrow$ MEMORY[5000]	4
MOV $R2$, ($R3$)	$R2 \leftarrow$ MEMORY[$R3$]	4
ADD$R2$, $R1$	$R2 \leftarrow R1 + R2$	2
MOV ($R3$), $R2$	MEMORY[$R3$] $\leftarrow R2$	4
INC $R3$	$R3 \leftarrow R3 + 1$	2
DEC $R1$	$R1 \leftarrow R1 - 1$	2
BNZ 1004	Branch if not zero to the given absolute address	2
HALT	Stop	1

Assume that the content of the memory location 5000 is 10, and the content of the register R_3 is 3000. The content of each of the memory locations from 3000 to 3020 is 50. The instruction sequence starts from the memory location 1000. All the numbers are in decimal format. Assume that the memory is byte addressable.

After the execution of the program, the content of memory location 3010 is ________



[5000] = 10
R3 = 3000
[3000 - 3020] = 50
BYTE ADDRESSABLE

Address	Instruction	Trace

1000	MOV R1(5000)	R1 <- [5000]
		R1 <- 10

1004	MOV R2 ,[R3]	R3 = 3000
		R2 <-[3000]
		R2 = 50

1008	ADD R2 R1	R2 = R2 + R1
		R2 = 60

1010	MOV [R3] R2	[3000] <- R2 [3000] = 60

1014	INC R3	R3 = R3+1
		R3 = 3001

1016	DEC R1	R1 = R1 - 1
		R1 = 9

1018	BNZ 1004	Z NOT SET TO 1004
1020	HALT	-

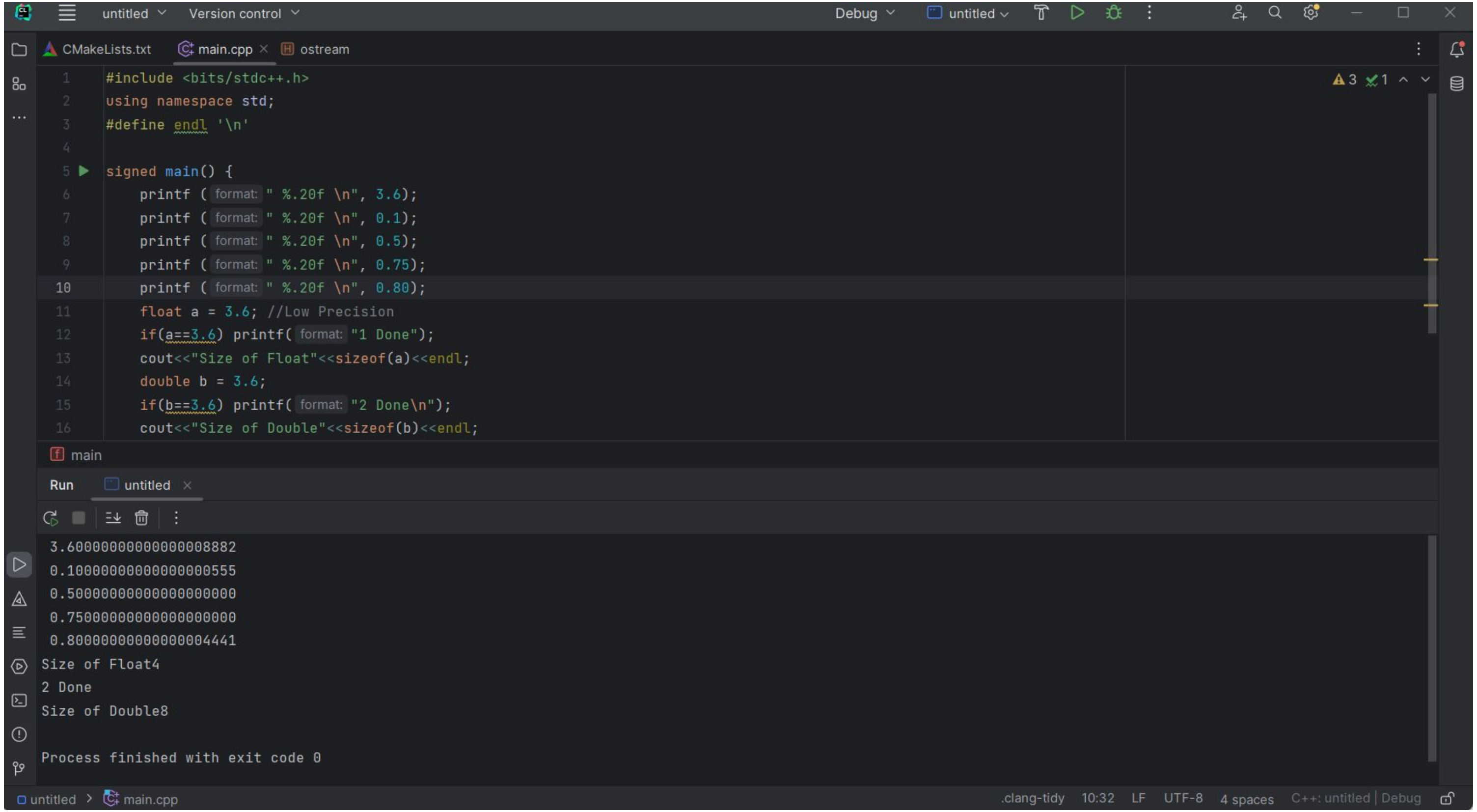
[3001] = 50 + 9 = 59 R1 = 8

.

.

[3008] = 50 + 2 = 52 R1 = 1
[3009] =50 + 1 = 51 R1 = 0
—-------------------------------------
[3010] = 50 Unaffected Z FLAG SET COMES OUT OF LOOP

Floating Points Errors



```cpp
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'

signed main() {
    printf ( " %.20f \n", 3.6);
    printf ( " %.20f \n", 0.1);
    printf ( " %.20f \n", 0.5);
    printf ( " %.20f \n", 0.75);
    printf ( " %.20f \n", 0.80);
    float a = 3.6; //Low Precision
    if(a==3.6) printf( "1 Done");
    cout<<"Size of Float"<<sizeof(a)<<endl;
    double b = 3.6;
    if(b==3.6) printf( "2 Done\n");
    cout<<"Size of Double"<<sizeof(b)<<endl;
```

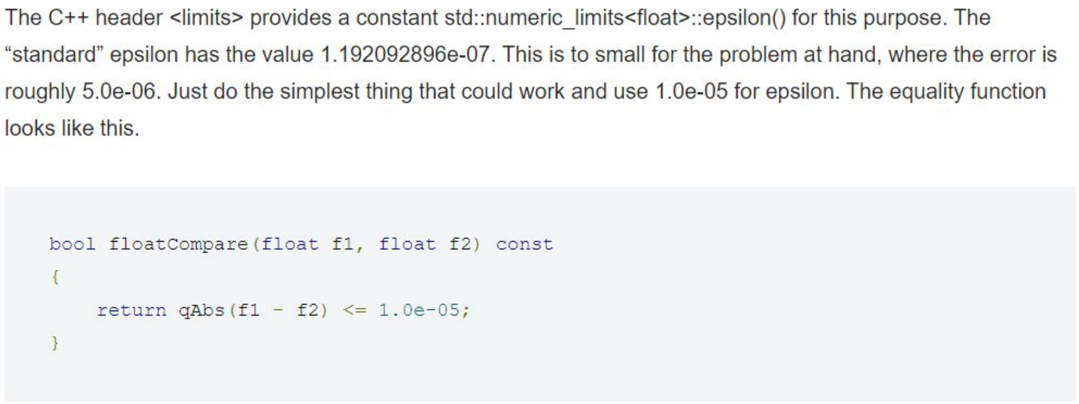
```
 3.60000000000000008882
 0.10000000000000000555
 0.50000000000000000000
 0.75000000000000000000
 0.80000000000000004441
Size of Float4
2 Done
Size of Double8

Process finished with exit code 0
```

How Computers Handle Comparison

- More Concretely the Computer Calculates the absolute difference between the two and if it is within the range of certain epsilon the comparison is assumed to be correct

- Higher the Bits for precision the greater the subtraction would be in the range of epsilon. The example in last slide denoted how the increase in bits lead to a more precision and a successful comparison



The C++ header <limits> provides a constant std::numeric_limits<float>::epsilon() for this purpose. The “standard” epsilon has the value 1.192092896e-07. This is to small for the problem at hand, where the error is roughly 5.0e-06. Just do the simplest thing that could work and use 1.0e-05 for epsilon. The equality function looks like this.

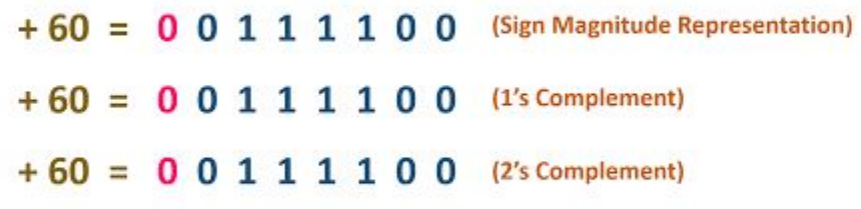
```
bool floatCompare(float f1, float f2) const
{
    return qAbs(f1 - f2) <= 1.0e-05;
}
```

Integer Representation - Positive NO

Positive NO:
For Positive No, Representation of the positive no is same in all different representation namely
Sign Magnitude, 1st Complement and 2nd Complement

- For N bit no.. (1 Bit for sign Representation and n-1 bit for no)
- Signed/1st complement -2^(n-1) to 2^(n-1)
- 2nd Complement -2^n to 2^(n-1)

Thus for Representing Positive no, one simply write the no in binary format ensuring a leading 0 at the MSB



+ 60 = 0 0 1 1 1 1 0 0 (Sign Magnitude Representation)

+ 60 = 0 0 1 1 1 1 0 0 (1's Complement)

+ 60 = 0 0 1 1 1 1 0 0 (2's Complement)

Integer Representation - Negative NO

To convert the no in Negative Representation

1. Write the no in Positive representation ensuring MSB is 0

- Sign Magnitude :
    - Change the Sign of MSB
- 1st Complement
    - Invert each of the bits from 0 to 1 and 1 to 0 (Vice Versa)
- 2nd Complement
    - Invert each of the Bits like above and add 1 to LSB

To find the value : Do the same process again, find the value and add Negative sign

Representation of 50 and -50 in Sign Magnitude 1st Complement and 2nd Complement

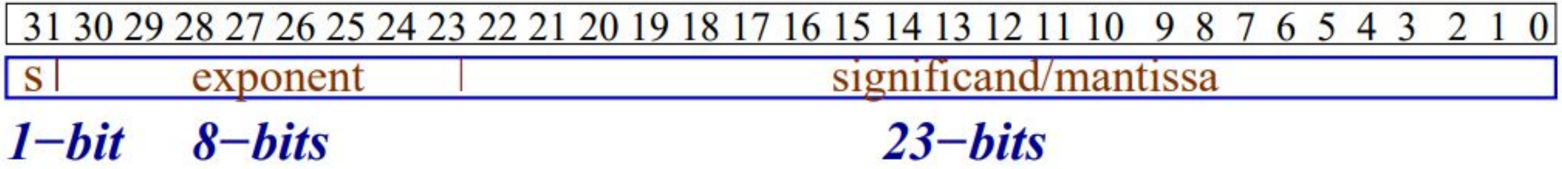
The 16-bit 2’s complement representation of an integer is 1111 1111 1111 0101; its decimal representation is

Let A = 1111 1010 and B = 0000 1010 be two 8-bit 2's complement numbers. Their product in 2's complement is

IEEE 754

- The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point arithmetic established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). The standard addressed many problems found in the diverse floating-point implementations that made them difficult to use reliably and portably. Many hardware floating-point units use the IEEE 754 standard.

- Thus Most of the binary floating-point representations follow the IEEE-754 standard. The data type **float uses IEEE 32-bit single precision format** and the **data type double uses IEEE 64-bit double precision format**. A floating-point constant is treated as a double precision number by GCC. This explains the output we had in screenshot of c program

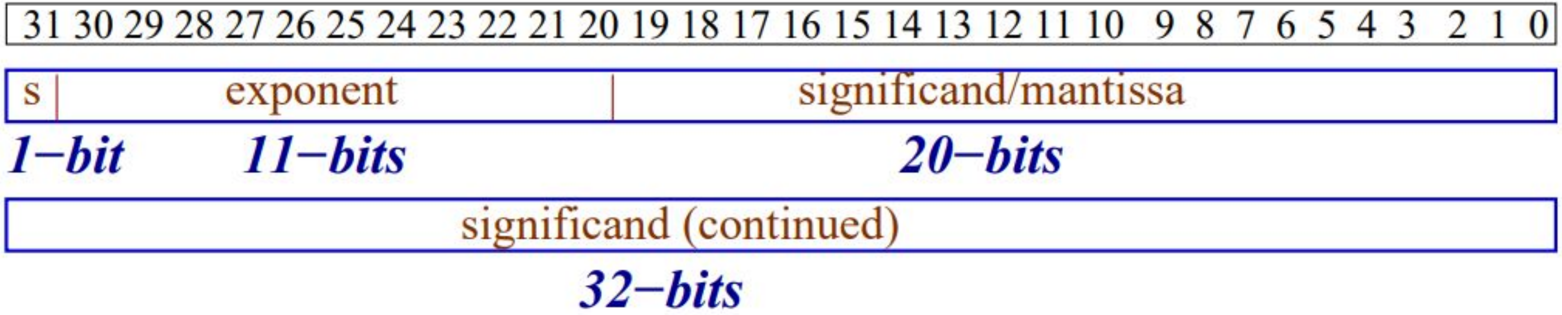


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

s exponent significand/mantissa

1−bit ***8−bits*** ***23−bits***

Single Precession (32−bit)



31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

s exponent significand/mantissa

1−bit ***11−bits*** ***20−bits***

significand (continued)

32−bits

Double Precession (64−bit)

Single Precession Normalized Number

- The normalized significand is $1.m$ (binary dot). The binary point is before bit-22 and the 1 (one) is not present explicitly.

- The sign bit $s = 1$ for a $-$ve number is zero (0) for a $+$ve number.

- The value of a normalized number is

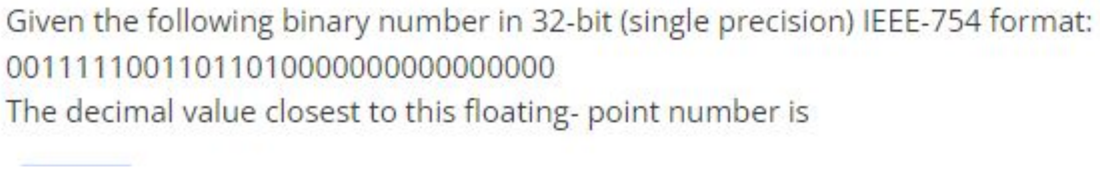
$$\boxed{(-1)^s \times 1.m \times 2^{e-127}}$$

Consider the following 32-bit pattern

1 1011 0110 011 0000 0000 0000 0000 0000

The value is

$$\begin{aligned} & (-1)^{1} \times 2^{10110110-01111111} \times 1.011 \\ = \; & -1.375 \times 2^{55} \\ = \; & -49539595901075456.0 \\ = \; & -4.9539595901075456 \times 10^{16} \end{aligned}$$



Given the following binary number in 32-bit (single precision) IEEE-754 format:
00111110011011010000000000000000
The decimal value closest to this floating- point number is

Exponent - 01111100 = 124

True exponent = 124 - 127 = -3

So in normalized form it will be

$= 1.1101101 * 2^{-3}$

$= 0.237$

$= 2.37 * 10^{-1} \approx 2.27 \ast 2^{-1}$

The decimal value 9.75 in IEEE single precision floating point

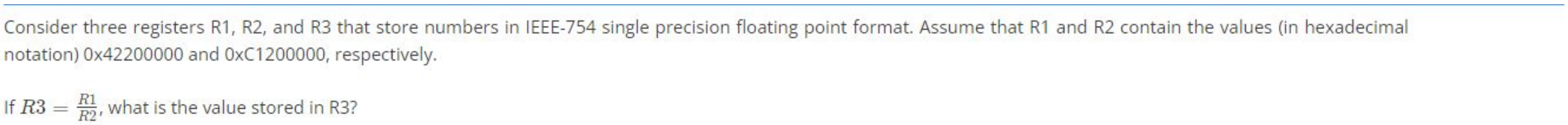
The decimal value 0.5 in IEEE single precision floating point

Floating Point Numbers

- **Special Numbers in IEEE 754 Standard**

Single Precision		Double Precision		Object Represented
E (8)	F (23)	E (11)	F (52)	
0	0	0	0	true zero (0)
0	nonzero	0	nonzero	± denormalized number
± 1-254	anything	± 1-2046	anything	± floating point number
± 255	0	± 2047	0	± infinity
255	nonzero	2047	nonzero	not a number (NaN)

Take Home



Consider three registers R1, R2, and R3 that store numbers in IEEE-754 single precision floating point format. Assume that R1 and R2 contain the values (in hexadecimal notation) 0x42200000 and 0xC1200000, respectively.

If $R3 = \frac{R1}{R2}$, what is the value stored in R3?