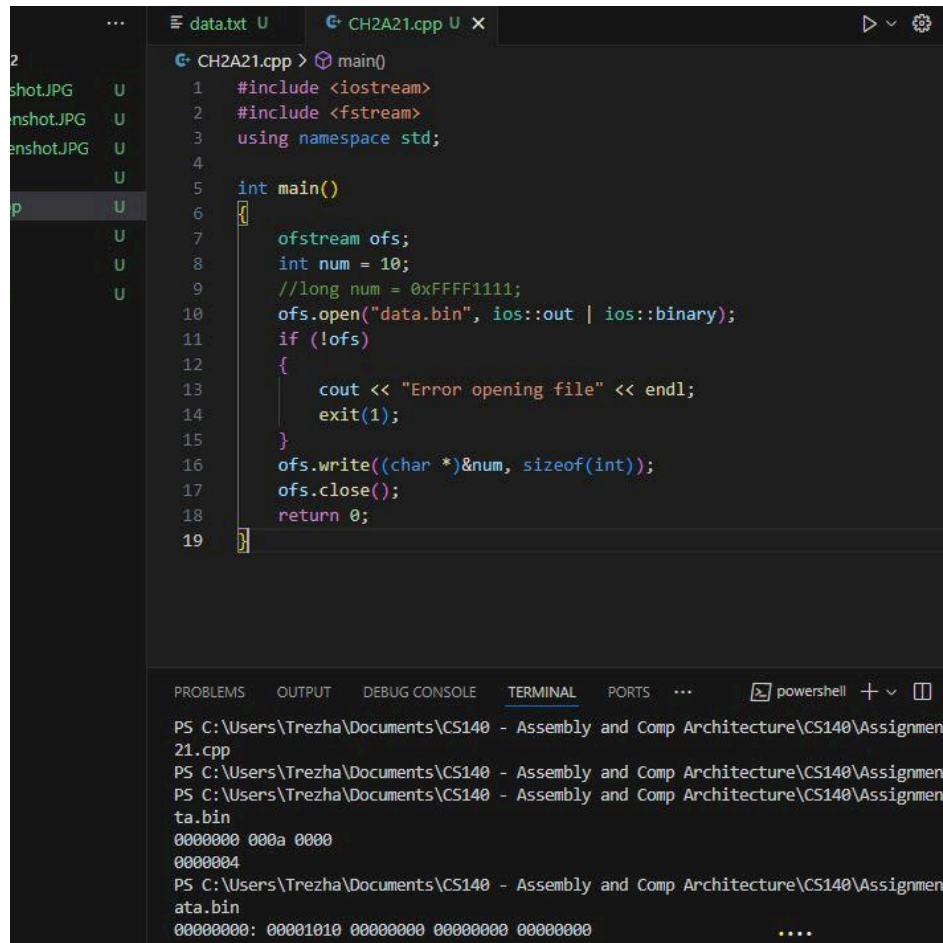


## Assignment 2-2-1 Byte Order of Integer Values

### 1. Integer value 10 to the binary file.



```
CH2A21.cpp > main()
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int main()
6  {
7      ofstream ofs;
8      int num = 10;
9      //long num = 0xFFFF1111;
10     ofs.open("data.bin", ios::out | ios::binary);
11     if (!ofs)
12     {
13         cout << "Error opening file" << endl;
14         exit(1);
15     }
16     ofs.write((char *)&num, sizeof(int));
17     ofs.close();
18     return 0;
19 }
```

```
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment 2-2-1> od -t x -b data.bin
00000000 000a 0000
00000004
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment 2-2-1> od -t x -b data.bin
00000000: 00001010 00000000 00000000 00000000 ....
```

(Notes: `xxd` command is used to dump the contents of a file in various formats, like hexadecimal and binary. The `-b` option specifies that the output should be in binary format.

Keep in mind: `od -x` typically uses little-endian)

**Explanation.** The output of using this command is in **Big Endian**. Because the Most significant byte (MSB) is (0a) and it's stored in the first byte (as the first pair read from right to left), ([2] 00=0000) ([1]: 0a=1010) = (1st byte = 0000 1010), followed by the remaining bytes which contain mainly 0s for padding because it's a 4 bytes, or 32 bits.

### 2. Try with Hexadecimal Value

The image shows a screenshot of a code editor with a C++ file named CH2A21.cpp. The code defines a main function that opens a file named 'data.bin' in binary write mode. It then writes a long integer value of 0xFFFF1111 to the file. The program includes headers for iostream and fstream, and uses the std namespace. The output window at the bottom shows the execution of the program, displaying the file path, the file name, and the hexadecimal value 00000000 1111 ffff, which is the little-endian representation of the stored value.

```
CH2A21.cpp > main()
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int main()
6  {
7      ofstream ofs;
8      //int num = 10;
9      long num = 0xFFFF1111;
10     ofs.open("data.bin", ios::out | ios::binary);
11     if (!ofs)
12     {
13         cout << "Error opening file" << endl;
14         exit(1);
15     }
16     ofs.write((char *)&num, sizeof(int));
17     ofs.close();
18     return 0;
19 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ... powershell + -

```
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2>
21.cpp
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2>
bin
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2>
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2>
ta.bin
00000000 1111 ffff
00000004
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2>
ata.bin
00000000: 00010001 00010001 11111111 11111111
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2>
```

**Explanation:** The output for the Hexadecimal value is in **Little Endian**. The MSB is FF(11111111) and it's stored in the last byte. The least significant byte (LSB) is 11(00010001) and we can see that it's stored in the first byte. We can even see this in the output of `od -x`, which is 1111 ffff. The byte count is **4 bytes(32 bits in total)** which is used to store the number 0xFFFF1111 which makes sense because it's **declared as a long which allows num to be stored in 4 bytes (32 bits)**. The decimal conversion totals **4294906129**.

**3. Show the bytecode for the "num" = 429406129.**

The screenshot shows a Visual Studio Code editor with a C++ file named CH2A21.cpp. The code defines a main function that writes a long integer to a binary file named data.bin. The terminal output shows the compilation and execution of the program, followed by a hex dump of the file data.bin using the 'xxd -b' command. The hex dump shows the binary representation of the number 429406129, which is 11001100110000011011110110001 in binary.

```

1 #include <iostream>
2 #include <fstream>
3 using namespace std;
4
5 int main()
6 {
7     ofstream ofs;
8     //int num = 10;
9     //long num = 0xFFFF1111;
10    long num = 429406129;
11    ofs.open("data.bin", ios::out | ios::binary);
12    if (!ofs)
13    {
14        cout << "Error opening file" << endl;
15        exit(1);
16    }
17    ofs.write((char *)&num, sizeof(int));
18    ofs.close();
19    return 0;
20 }

```

```

PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2> g++ CH2A21.cpp
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2> ./a.exe
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2> od -x data.bin
00000000 37b1 1998
00000004
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2> xxd -b data.bin
00000000: 10110001 00110111 10011000 00011001          .7..
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2> bc
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
obase=2
ibase=10
429406129
11001100110000011011110110001
quit
PS C:\Users\Trezha\Documents\CS140 - Assembly and Comp Architecture\CS140\Assignment_2>

```

a.) Num uses 4 bytes (32 bits).

b.) No, they are in different Endians. There's also 3 bits missing but these bits aren't included in BC's output because they were at the front of the binary and leading bits that are 0 aren't included. But if we were to include them back in, we can see the pattern and what kind of endian the system uses.

[n] - endian order

BC	[6]0001 [5]1001	[8]1001 [7]1000	[2]0011 [1]0111	[4]1011 [3]0001
Xxd -b	[4]1011 [3]0001	[2]0011 [1]0111	[8]1001 [7]1000	[6]0001 [5]1001

Od -x data.bin = **37b1 1998**. What each "nibble"(4 bits) represents

Keep in mind: od - x typically uses little-endian so don't forget MSB and LSB are switched

	1st Byte (LSB)		2nd Byte		3rd Byte		4th Byte (MSB)	
Hex #	3	7	b	1	1	9	9	8
Bits	0011	0111	1011	0001	0001	1001	1001	1000

Therefore, we can see how real **binary numbers (BC)** uses **Big Endian** while **xxd -b** uses **Little Endian**.