Computer System Organization Recitation [Fall 2017]

CSCI-UA 201-006

R2: C variable/pointer/array/bit operators

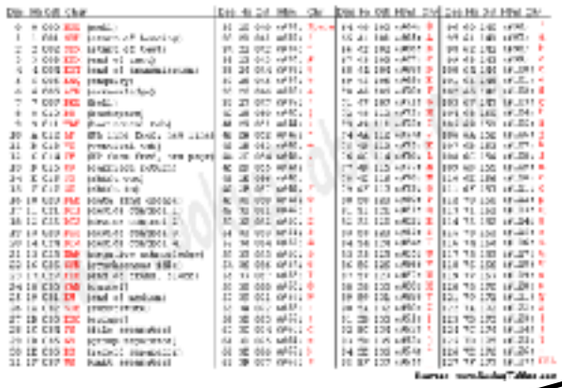
Variable in C

- A variable defines three information.
  - The memory address(ADDR) of that variable.
  - The number of bytes used by the variable.
  - How to interpret the content stored in ADDR.

Variable in C

char c = ‘A’;

***(0x1000F000)**
=‘A’



unsigned int a = 255;

***(0x1000F008)**
= 00000000 00000000 00000000 11111111
= 255_{10}

float b = 0.5;

***(0x1000F010)**
= 0 01111110 00000000000000000000000
= + 1.(0…0) * 2^(126 - 127)
= 1.0 * 2 ^ -1
= 0.1_2
= 0.5_{10}

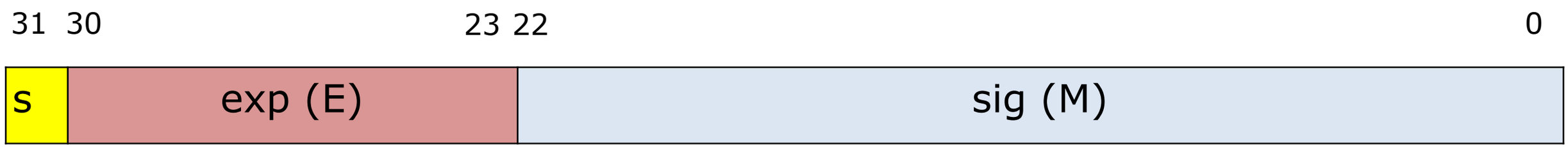
0x42	0x1000F000
	0x1000F001
	0x1000F004
	0x1000F008
0x000000FF	
	0x1000F00C
	0x1000F010
0x3F000000	

Normalized representation in computer

$r_{10} = \pm M * 2^E$, where $1 <= M < 2$

$M = (1.b_1b_2b_3 \ldots b_n)_2$

M: significant, E: exponent

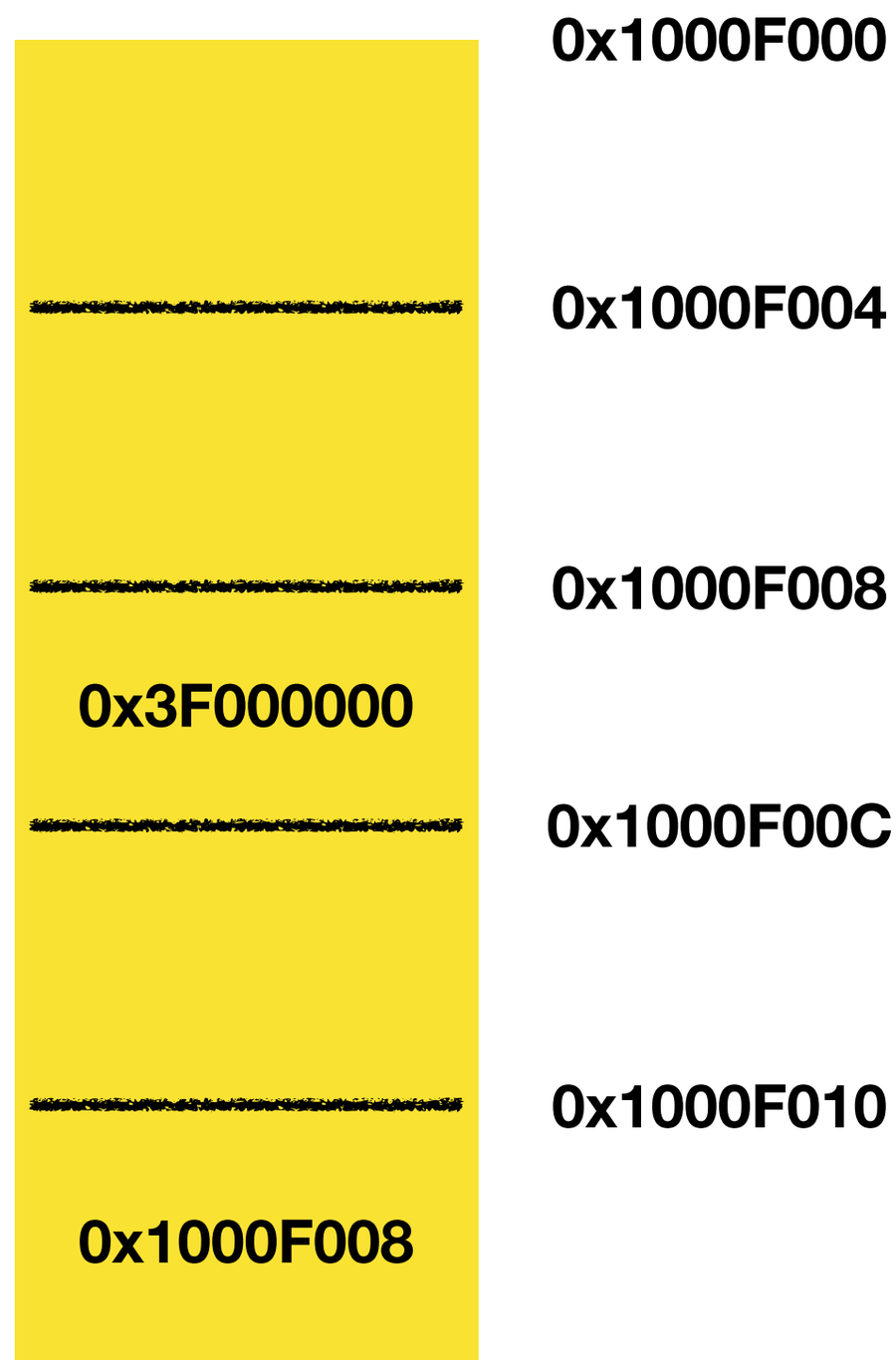


$(1.b_1b_2b_3 \ldots b_n)_2$

Pointer in C

- A pointer is just a variable with a unique interpretation.
  - The memory address(ADDR) of that variable.
  - The number of bytes used by the variable.
  - How to interpret the content stored in ADDR.
    - It represents a memory address.

Pointer in C



unsigned int a = 0x3F000000;

*(0x1000F008) = 0x3F000000

unsigned int *ptr = &a;

printf("%p", ptr); *(0x1000F010) = 0x1000F008

printf("%u", *ptr); *(*(0x1000F010)) = *(0x1000F008) = 0x3F000000

Array in C

- An array is just a variable with different byte lengths.
  - The memory address(ADDR) of that variable.
  - The number of bytes used by the variable.
    - It contains multiple values.
  - How to interpret the content stored in ADDR.
    - Compiler doesn't deference it unless you use [] symbol.

Pointers in C

unsigned int grades[] = {0, 1, 2, 4, 8};

unsigned int *ptr = grades;

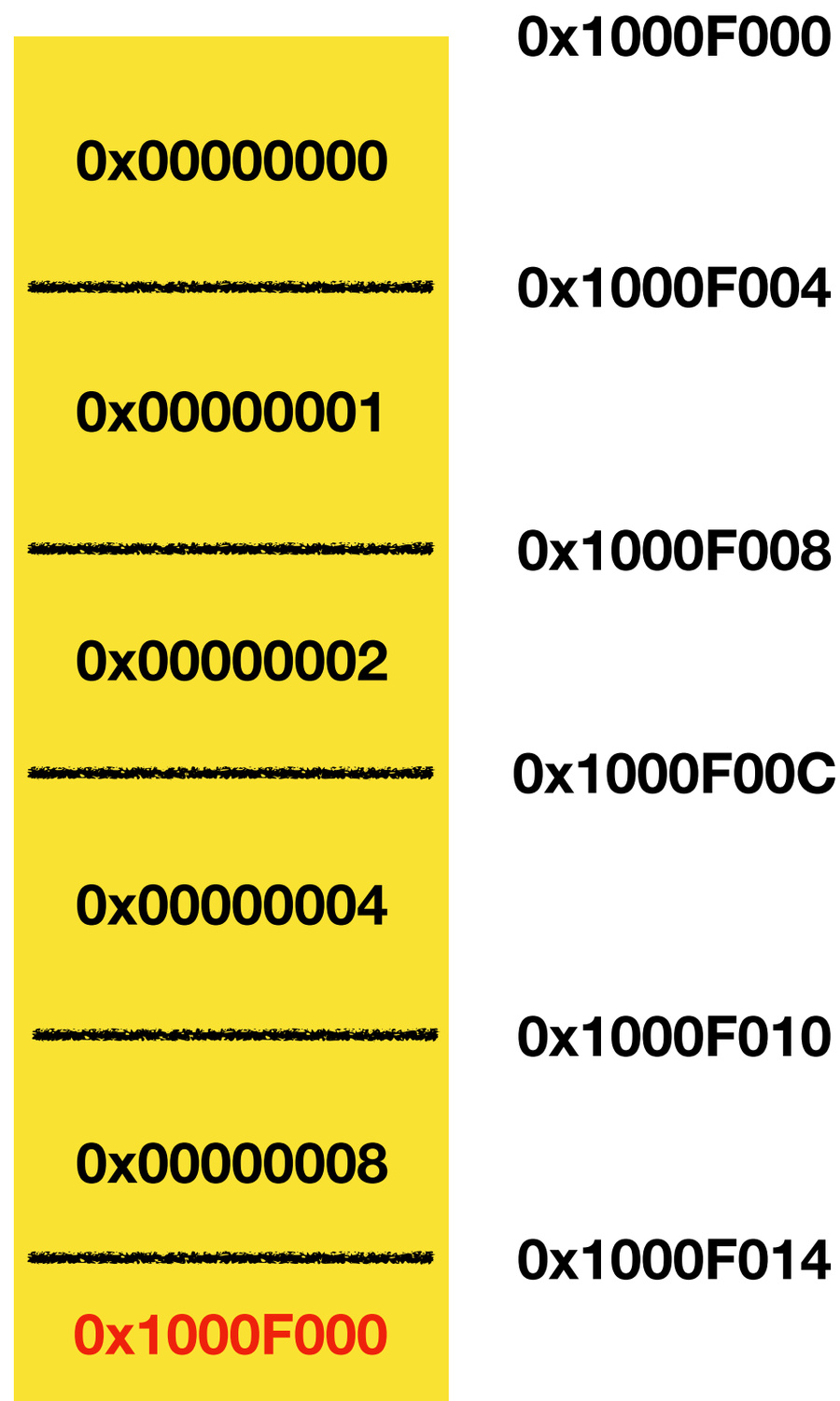
***(0x1000F004) = 0x1000F000**

grades = ptr;

grades[2]

***(ptr + 2)**

***(0x1000F000 + 8) = 0x2**



Exercise 1 (How to print 15)

```
void func1(unsigned num) {
    num += 1;
}
```

→ num(different num from main) += 1

```
void func2(unsigned* num) {
    *num += 1;
}
```

→ *(0x7F100000) += 1

```
int main() {
    unsigned num = ?;
    func1(num);
    funct2(&num);
    printf("%u", num);
}
```

~~func1(num);~~ → func1(?);

funct2(&num); → func2(0x7F100000);

num + 1 = 15

Lab1 : part1, par2.c, part4.c, part5.c

Exercise 2 (How to print 8)

```
int main() {
    unsigned short array[] = {0, 2, 4, 8, 16};
    int i = ?;
    int j = ?;
    unsigned short *ptr1 = &(array[i]);
    unsigned short *ptr2 = &(array[j]);
    unsigned int ptr1_int = (unsigned int) ptr1;
    unsigned int ptr2_int = (unsigned int) ptr2;
    printf("%u\n", ptr2_int - ptr1_int);
}
```

ptr2_int - ptr1_int == 8

(the address stored in ptr2) - (the address stored in ptr1) == 8

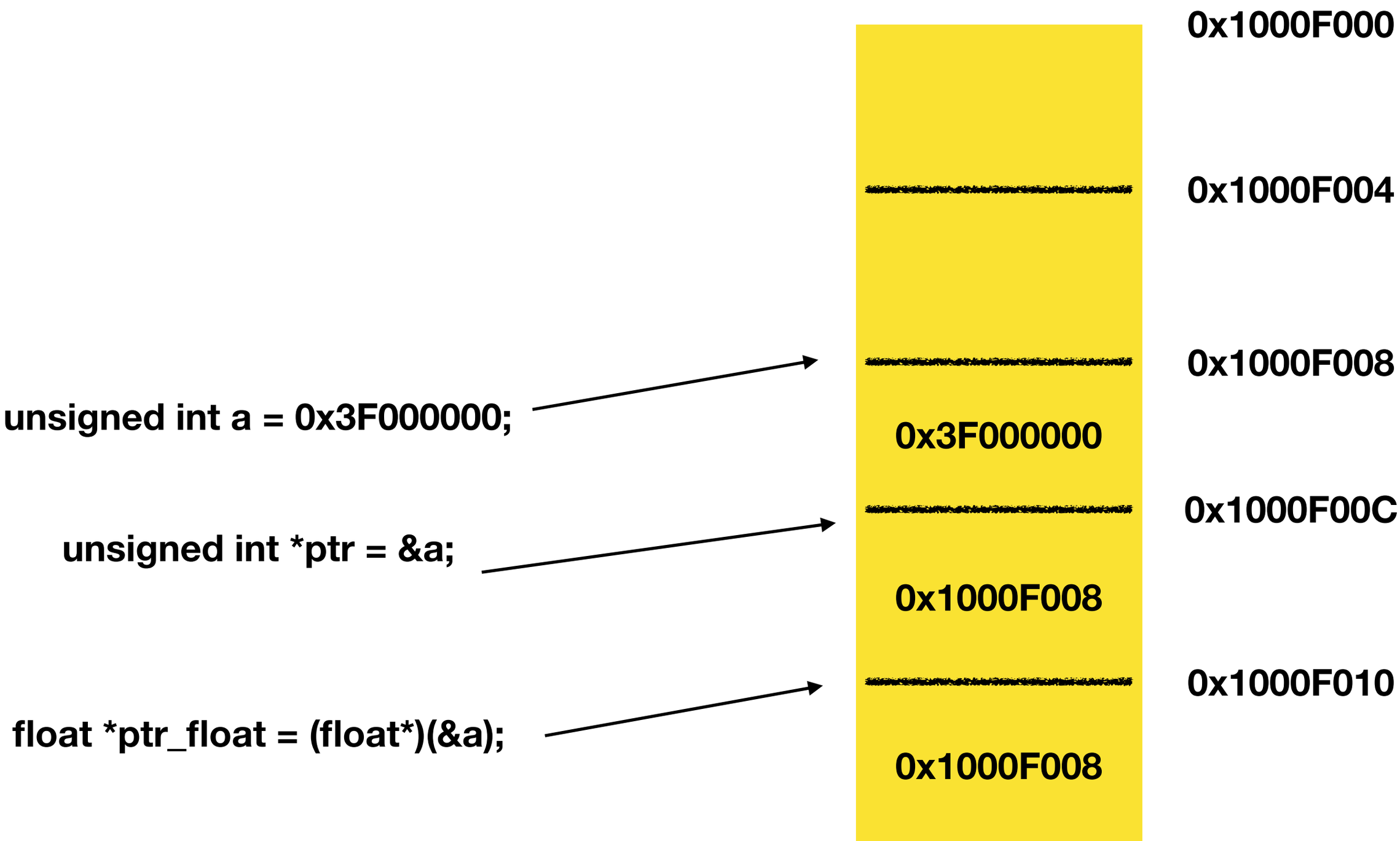
(the address of array[j]) - (the address of array[i]) == 8

unsigned short is two bytes

j - i == 4

Lab1 : part1, par2.c, part3.c, part4.c

Pointers in C



Exercise 3 (How to print 1056964608)

```
int main() {
    float answer = ?;

    unsigned* unsigned_ptr = (unsigned*)(&answer);
    printf("%u\n", *unsigned_ptr);
}
```

1056964608 = 0x3F000000

What is 0x3F000000 when interpreting it as a float?

0.5

Lab1 : part1, part2.c, part3.c, part5.c part6.c

Exercise 4 (How to print 262144)

```
int main() {
   unsigned answer = ?

    answer = (answer >> 3) << 5;
    printf("%u\n", answer);
}
```

262144 = 0x00040000 = b 0000,0000,0000,0100,0000,0000,0000,0000

do >> 5 : b ????,?000,0000,0000, 0010,0000,0000,0000

do << 3 : b ??00,0000,0000,0001, 0000,0000,0000, 0???

One more thing to know for lab 1: 0x00FF1112 & 0xFF == 0x00000012

Lab1 : part6.c

Lab1