

Tutorial 3: Number Systems

Problem 1: Data Representation

Consider the number $A = 0xF2$. What is the value of this number when represented in; 8-bit unsigned number, 8-bit sign-magnitude, 8-bit one's complement, and 8-bit two's complement?

Unsigned: $1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1 = 128 + 64 + 32 + 16 + 2 = 242$

Signed-magnitude: $(-1)^1 \times (1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1) = -(64 + 32 + 16 + 2) = -114$

One's complement: $(1 \times 2^7 - 1) + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1 = 128 + 64 + 32 + 16 + 2 = -13$

Two's complement: $-1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1 = -128 + 64 + 32 + 16 + 2 = -14$

Problem 2: Data Types

Consider the C code in Figure 1. Answer the following questions.

What are the outputs of the `printf` statements?

What are the outputs of the `printf` statements if ($a = 0xFFFFFE5$) and ($b = 0xFFFFFE5$)?

```
#include <stdio.h>

int main (void)
{
    int a = 0xE5;
    char b = 0xE5;

    printf("integer = \"%d\"\n\n", a);
    printf("char = \"%d\"\n\n", b);

    return 0;
}
```

Figure 1: Program on Data Type Conversion

The number ($a = 0xE5$), being declared as an `int` (32-bit number), will be internally represented as ($a = 0x000000E5$) which is ($a = 14 \times 16^1 + 5 = 229$). The number ($b = 0xE5$) being declared as a `char` (8-bit number), will be internally represented as ($a = 0xE5 = 0b11100101 = -1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^2 + 1 = -27$). So, the output of the `printf` statements are.

```
integer = "229"

char = "-27"
```

Figure 2: The Outputs of `printf` Statements

The number ($a = 0xFFFFFE5$), being declared as an `int` (32-bit number), is simply a representation of 8-bit number ($0xE5 = -27$) in 32 bits. The number ($b = 0xFFFFFE5$) being declared as a `char` (8-bit number), has more bits than can accommodate. So all the bits after bit 7 will be discarded, and the number will be internally represented as ($a = 0xE5 = -27$). So, the output of the `printf` statements are.

```
integer = "-27"
char = "-27"
```

Figure 3: The Outputs of printf Statements

Problem 3: More Data Type Conversion

Consider the C code in Figure 4. Answer the following questions.

Which of the print statements will be printed out?

```
#include <stdio.h>

int main (void)
{
    unsigned int a =0xFFFFFFFF;
    int b =0xFFFFFFFF;
    char c = 0xFD;

    if (a < (unsigned) c)
        {printf("a < c \n\n");}
    if (b < (int) c)
        {printf("b < c \n\n");}

    return 0;
}
```

Figure 4: Program on Data Type Conversion

The statement (if (a < (unsigned) c)) treats both a and c as positive numbers (a = 0xFFFFFFFF = 4294967291) and (c = 0xFD = 253), and therefore the condition (a < c) is not true, and the first print statement does not execute.

The statement (if (b < (int) c)) treats both b and c as negative numbers (a = 0xFFFFFFFF = -5) and (c = 0xFD = -3), and therefore the condition (b < c) is true and the second print statement executes.

Problem 4: Binary Prefixes

How much is 2^{27} Bytes?

We need to access a memory organisation as large as 2.4 Mi Bytes. How many address lines are required for this purpose?

2^{27} Bytes = $2^7 \times 2^{20}$ Bytes = 128 mebi Bytes = 126 Mi Bytes.

$2.4 \text{ Mi} \rightarrow \lceil \log_2(2.4 \text{ Mi}) \rceil = \lceil \log_2(2.4) \rceil + \log_2(\text{Mi}) = 2 + \log_2(2^{20}) = 2 + 20 = 22 \text{ lines.}$