Tutorial 4: Arithmetic and Logic Operations

Problem 1: Data Representation

Consider the number $A = 0xEEEEDDDD$. What is the value of this number in decimal? What is its value when represented in 8-bit unsigned and signed numbers? What is its value when logically shifted to the right by two bits and represented in 8-bit unsigned and signed numbers? What is its value when arithmetically shifted to the right by two bits and represented in 8-bit unsigned and signed numbers?

We can represent number $A = 0xEEEEDDDD$ in two ways; signed and unsigned integer representations:

Unsigned: $(+1) \times 2^{31} + 1 \times 2^{30} + 1 \times 2^{29} + \ldots + 1 \times 2^{0} = 4008631773$

Signed: $(-1) \times 2^{31} + 1 \times 2^{30} + 1 \times 2^{29} + \ldots + 1 \times 2^{0} = -286335523$

Representation of number $A = 0xEEEEDDDD$ in 8 bits only retains its lower 8 bits $0xDD$. Its signed and unsigned integer representations are:

Unsigned: $(+1) \times 2^{7} + 1 \times 2^{6} + 1 \times 2^{4} + \ldots + 1 \times 2^{0} = 221$

Signed: $(-1) \times 2^{7} + 1 \times 2^{6} + 1 \times 2^{4} + \ldots + 1 \times 2^{0} = -35$

Logical shifting of number $A = 0xEEEEDDDD$ to the right can be illustrated in binary as:

$(A = 0xEEEEDDDD) \gg \text{logical 2} = (A = 0b1110, 1110, 1110, 1110, 1101, 1101, 1101, 1101) \gg \text{logical 2} = (0b0011, 1011, 1011, 1011, 1011, 0111, 0111, 0111) = 0x3BBBB777$. Note that the emptied 2 bits from the left are filled with 0 and two bits from the right have disappeared.

Representation of number $A = 0x3BBBB777$ in 8 bits only retains its lower 8 bits $0x77$. Its signed and unsigned integer representations are:

Unsigned: $(+0) \times 2^{7} + 1 \times 2^{6} + 1 \times 2^{4} + \ldots + 1 \times 2^{0} = 119$

Signed: $(-0) \times 2^{7} + 1 \times 2^{6} + 1 \times 2^{4} + \ldots + 1 \times 2^{0} = 119$

Arithmetic shifting of number $A = 0xEEEEDDDD$ to the right can be illustrated in binary as:

$(A = 0xEEEEDDDD) \gg \text{arithmetic 2} = (A = 0b1110, 1110, 1110, 1110, 1101, 1101, 1101, 1101) \gg \text{arithmetic 2} = (0b1111, 1011, 1011, 1011, 1011, 0111, 0111, 0111) = 0xFBBBB777$. Note that the emptied 2 bits from the left are filled with sign of the number (–) and two bits from the right have disappeared.

Representation of number $A = 0xFBBBB777$ in 8 bits only retains its lower 8 bits $0x77$. Its signed and unsigned integer representations are:

Unsigned: $(+0) \times 2^{7} + 1 \times 2^{6} + 1 \times 2^{4} + \ldots + 1 \times 2^{0} = 119$

Signed: $(-0) \times 2^{7} + 1 \times 2^{6} + 1 \times 2^{4} + \ldots + 1 \times 2^{0} = 119$

Problem 2: Shift Operations in C

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 = 0x7722)?
```c
#include <stdio.h>

int main (void)
{
    short a = 0xDEE5;
    char b;

    b = a;
    printf("short = \"%d\"\n\n", a >> 1);
    printf("char = \"%d\"\n\n", b >> 1);
    return 0;
}
```

Figure 1: Program on Shift Operations
Problem 3: Rotate Operation

Consider the ARM Assembly code in Figure 4 that does rotation of bits in a register. Write an equivalent C version of this program. If (V1 = 0xEEBAE213), what would register A1 contain after the execution this instruction?

\[
\text{mov a1, v1, ror #8 ;a1 &lt; v1 &gt;&gt; 8 bits , a1[31:24] &lt; v1[7:0]}
\]

Figure 4: Assembly Program on Rotation
Figure 7: The Outputs of printf Statements

Note that in the C program in Figure 6 variable (a) is declared as (unsigned int). This will ensure that the shift operation (a >> 8) is logical and empties the 8 bits on the left. If (a) is declared as (int), then the shift operation (a >> 8) is interpreted as arithmetic shift right and will leave 1 in the emptied 8 bits on the left. That will cause C statement (b = (a >> 8)|(a << 24);) to produce (b = 0xFFEEBAE2).

Also note that in the C program in Figure 6 if variable (a) is declared as (int), then its equivalent assembly language program will require more than one instruction corresponding to the C statement (b = (a >> 8)|(a << 24);). The equivalent assembly instructions are provided in Figure 8.

```
mov a1, v1, asr #8 ;a1 (v1 >> arithematically shifted right by 8 bits)
orr a1, v1, lsl #24 ;a1 | ((v1 << arithematically shifted left by 24 bits)
```

Figure 8: Assembly Program on Rotation for int

However, if the variable (a) is guaranteed to only take negative values then a single assembly instruction given in Figure 9 is sufficient. That is because (asr) instruction fills the emptied bits with (1) and subsequent ORing with (1) through the (orr) instruction becomes redundant, as ORing anything with (1) produces (1).

```
mov a1, v1, asr #8 ;a1 (v1 >> arithematically shifted right by 8 bits)
```

Figure 9: Assembly Program on Rotation for int (If only negative values are possible)