# CS 301 Assignment 3

Student ID: 200482797

Owen Monus

March 16th, 2024

1. (a) Represent the following decimal numbers in twos complement using 16 bits: +512, -29.
   512 → 0000001000000000
   -29 → 1111111111100011

   (b) Represent the following twos complement values in decimal:
   1101011 → -21
   0101101 → 45

2. (a) The $r$'s complement of an $n$-digit number $N$ in base $r$ is defined as $r^n - N$ for $N \neq 0$ and 0 for $N = 0$. Find the tens complement of the decimal number $13,250$.

   13,250's 10's complement is 86,750.

   (b) Calculate $72,530 - 13,250$ using tens complement arithmetic. Assume that the rules are similar to those for twos complement arithmetic.

   Take 10s comp of 13250, add that to 72530.
   $72530 + 86750 = 159,280$

3. Use the Booth's algorithm to multiply 23 (multiplicand) by 29 (multiplier), where each number is represented using 6 bits.

```
// Booth's Algorithm with M = 23, N = 29
A = 0
B = 010111 // 23
C = 011101 // 29
C_-1 = 0
n = 6

while(n > 0) {
    if (LSB(C) == 0 && C_-1 == 1)
        A = A + B
    if (LSB(C) == 1 && C_-1 == 0)
        A = A - B

    n = n - 1
    A||C||C_-1 = (A||C||C_-1) >>> 1
}
```

Table 1: Booth's Algorithm: $23 \times 29$

| Operation | n | A | C | $C_{-1}$ | B |
|---|---|---|---|---|---|
| initial value | $6_{10}$ | 000000 | 011101 | 0 | 010111 |
| $A = A - B$ | $6_{10}$ | 101001 | 011101 | 0 | 010111 |
| right shift | $6_{10}$ | 110100 | 101110 | 1 | 010111 |
| $A = A + B$ | $5_{10}$ | 001011 | 101110 | 1 | 010111 |
| right shift | $5_{10}$ | 000101 | 110111 | 0 | 010111 |
| $A = A - B$ | $4_{10}$ | 101110 | 110111 | 0 | 010111 |
| right shift | $4_{10}$ | 110111 | 011011 | 1 | 010111 |
| right shift | $3_{10}$ | 111011 | 101101 | 1 | 010111 |
| right shift | $2_{10}$ | 111101 | 110110 | 1 | 010111 |
| $A = A + B$ | $1_{10}$ | 010100 | 110110 | 1 | 010111 |
| right shift | $1_{10}$ | 001010 | 011011 | 0 | 010111 |

Result = 0b001010011011 = $667_{10}$

4. 
```
// One instruction machine
LOAD E
MUL F
STORE T
LOAD D
SUB T
STORE X
LOAD B
MUL C
ADD A
DIV X
STORE X

// Two instruction machine
MOVE X,F
MUL X,E
MOVE T,D
SUB T,X

MOVE X,C
MUL X,B
ADD X,A
DIV X,T

// Three instruction machine
MUL T,E,F
SUB T,D,T
MUL X,B,C
ADD X,X,A
DIV X,X,T
```

5. Both an arithmetic left shift and a logical left shift correspond to a multiplication by 2 when there is no overflow. If overflow occurs, arithmetic and logical left shift operations produce different results, but the arithmetic left shift retains the sign of the number.

Table 2: Shifts for Decimal Numbers

| $d$ | 2's | LLS | ALS | Condition |
|---|---|---|---|---|
| -16 | 10000 | 00000 | 10000 | overflow |
| -15 | 10001 | 00001 | 10010 | overflow |
| -14 | 10010 | 00100 | 10100 | overflow |
| -13 | 10011 | 00011 | 10110 | overflow |
| -12 | 10100 | 01000 | 11000 | overflow |
| -11 | 10101 | 01010 | 11010 | overflow |
| -10 | 10110 | 01100 | 11100 | overflow |
| -9 | 10111 | 01110 | 11110 | overflow |
| -8 | 11000 | 10000 | 10000 | no overflow |
| -7 | 11001 | 10010 | 10010 | no overflow |
| -6 | 11010 | 10100 | 10100 | no overflow |
| -5 | 11011 | 10110 | 10110 | no overflow |
| -4 | 11100 | 11000 | 11000 | no overflow |
| -3 | 11101 | 11010 | 11010 | no overflow |
| -2 | 11110 | 11100 | 11100 | no overflow |
| -1 | 11111 | 11110 | 11110 | no overflow |
| 0 | 00000 | 00000 | 00000 | no overflow |
| 1 | 00001 | 00010 | 00010 | no overflow |
| 2 | 00010 | 00100 | 00100 | no overflow |
| 3 | 00011 | 00110 | 00110 | no overflow |
| 4 | 00100 | 01000 | 01000 | no overflow |
| 5 | 00101 | 01010 | 01010 | no overflow |
| 6 | 00110 | 01100 | 01100 | no overflow |
| 7 | 00111 | 01110 | 01110 | no overflow |
| 8 | 01000 | 10000 | 00000 | overflow |
| 9 | 01001 | 10010 | 00010 | overflow |
| 10 | 01010 | 10100 | 00100 | overflow |
| 11 | 01011 | 10110 | 00110 | overflow |
| 12 | 01100 | 11000 | 01000 | overflow |
| 13 | 01101 | 11010 | 01010 | overflow |
| 14 | 01110 | 11100 | 01100 | overflow |
| 15 | 01111 | 11110 | 01110 | overflow |

As shown in the table, when an operation results in overflow, the Arithmetic Left Shift and Logical Left Shift result in different results. The Arithmetic Left Shift retains the sign of the number, whereas the Logical Left Shift does not.