EECSE 469 Computer Architecture I
Christopher Andrade, Brayden Lam
HW 6

1. Convert C to assembly. Assume all variables are integers, and initialized before these code blocks. Assume that there is code before and after these blocks, and that each part is independent of the others.

a. **Assume a is stored in R0, b is stored in R1, c is stored in R2, and d is stored in R6. Use ONLY registers to hold intermediate values in the assembly for someFunc().**

```
int main() {
      int a = 4;
      int b = 2;
      int c = 5;
      int d = someFunc(b, a, c);
}

int someFunc(int a, int b, int c) {
      return ((a + c) * b) - c;
}

main:
      // code before function call
      MOV R0, #4        // a = 4
      MOV R1, #2        // b = 2
      MOV R2, #5        // c = 5

      MOV R3, R1        // move b to R3 (to use R3 for first parameter)
      MOV R4, R0        // move a to R4 (to use R4 for second parameter)
      MOV R5, R2        // move c to R5 (to use R5 for third parameter)
      BL someFunc       // Call someFunc

      MOV R6, R0        // store result of someFunc in d (R6)
      // code after function call
someFunc:
      ADD R3, R3, R5    // a + c
      MUL R3, R3, R4    // (a + c) * b
      SUB R0, R3, R5    // ((a + c) * b) - c

      MOV PC, LR        // returns to caller
```

b. Repeat Part a, but this time use the stack to save the previous value of R4 in someFunc() so that it is retained when someFunc() returns to main().

```
main:
      // code before function call

      MOV R0, #4         // a = 4
      MOV R1, #2         // b = 2
      MOV R2, #5         // c = 5

      MOV R3, R1         // move b to R3 (to use R3 for first parameter)
      MOV R4, R0         // move a to R4 (to use R4 for second parameter)
      MOV R5, R2         // move c to R5 (to use R5 for third parameter)
      BL someFunc        // Call someFunc

      MOV R6, R0         // store result of someFunc in d (R6)

      // code after function call
someFunc:
      PUSH {R4}          // Store R4 on stack

      ADD R3, R3, R5     // a + c
      MUL R3, R3, R4     // (a + c) * b
      SUB R0, R3, R5     // ((a + c) * b) - c

      POP {R4}     // Restore R4 from stack

      MOV PC, LR         // returns to caller
```

c. Use registers and/or the stack to implement the following recursive code.

```
int main() {
      summation(5);
}

int summation(int x) {
      if(x <= 1) {
            return 1;
      }
      return x + summation(x-1);
}

main:
      // code before function call

      MOV R0, #5        // to call summation with #5
      BL summation// call summation function
                        // R0 now contains result of summation

      // code after function call
summation:
      MOV R1, #0        // Ensure R1 starts at 0
      PUSH {LR}         // Push LR back to main onto stack

recursiveCase:
      CMP R0, #1        // CMP for BLE
      BLE baseCase// Branch to baseCase of x <= 1

      ADD R1, R1, R0    // Add current x to previous sum of xs
      SUB R0, R0, #1    // decrement x for next call

      B recursiveCase   // branch to recursive case

baseCase:
      ADD R0, R1, R0    // add R0 (1) to R1 and store in R0

      POP {LR}          // pop LR from stack
      MOV PC, LR        // return to main
```

2. Represent the following fractions using 8 integer bits and 8 fraction bits:

a. **4.25**

`00000100.01000000`

b. **218.375**

`11011010.01100000`

c. **150.1875**

`10010110.00110000`

3. Represent the following base-10 floating-point values in binary, please show your work:

a. **26.25**

positive, sign bit is 0

convert to binary, 11010.01

convert to binary scientific notation, $1.101001 * 2^{4}$

convert exponent, 4 + 127 = 131, 131 in binary = 10000011

place into 32-bit number, `0 | 10000011 | 10100100000000000000000`

`0 | 10000011 | 10100100000000000000000`

b. **1250.3125**

positive, sign bit is 0

convert to binary, 10011100010.0101

convert to binary scientific notation, $1.00111000100101 * 2^{10}$

convert exponent, 10 + 127 = 137, 137 in binary = 10001001

place into 32-bit number, `0 | 10001001 | 00111000100101000000000`

`0 | 10001001 | 00111000100101000000000`

c. **-469.0**

negative, sign bit is 1

convert to binary, 111010101.

convert to binary scientific notation, $1.11010101 * 2^{8}$

convert exponent, 8 + 127 = 135, 135 in binary = 10000111

pace into 32-bit number, `1 | 10000111 | 11010101000000000000000`

`1 | 10000111 | 11010101000000000000000`

4. Represent the following binary floating-point values in base-10, please show your work (the fields have been separated):

a. **`0 | 10000100 | 11001110000000000000000`**
   sign bit is 0, positive
   exponent of 10000100 = 132, 132 -127 = 5, 2^5 is the exponent.
   mantissa is 1 + ½ + ¼ + 1/32 + 1/64 + 1/128 = 1.8046875
   1.8046875 * 2^5 = 57.75
   57.75
b. **`1 | 10000111 | 01101000001000000000000`**
   sign bit is 1, negative
   exponent of 1000111 = 135, 135 - 127 = 8, 2^8 is the exponent.
   mantissa is 1 + ¼ + ⅛ + 1/32 + 1/2048 = 1.406738281
   1.406738281 * 2^8 = 360.125
   -360.125
c. **`1 | 01111110 | 00100000000000000000000`**
   sign bit is 1, negative
   exponent of 126, 126 - 127 = -1, 2^{-1} is the exponent.
   mantissa is 1 + ⅛ = 1.125
   1.125 * 2^{-1} = .5625
   -.5625