EECSE 469 Computer Architecture I
Christopher Andrade, Brayden Lam
HW 5

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 from each other.

a. **Assume a is stored in R0, b is stored in R1, and c is stored in R2.**

```
a = 1;
b = 6;
c = a + b;
b = a - c;
c++;
a <<= 3;

MOV R0, #1
MOV R1, #6
ADD R2, R1, R0
SUB R1, R0, R2
ADD R2, R2, #1
LSL R0, R0, #3
```

b. **Assume a is stored in R0, b is stored in R1, and c is stored in R2.**

```
if (a < b) {
      a += c;
} else {
      a -= c;
}

CMP R0, R1
BLT TRUE
SUB R0, R0, R2
B END
TRUE:
      ADD R0, R0, R2
END:
```

c. **Assume a is stored in R0, b is stored in R1, and c is stored in R2, x is stored in R3.**

```
for(x = 10; x > 0; x--) {
      c = c + x;
      a = a + c;
      b = a + x;
}
MOV R3, #10
TOP:
      CMP R3, #0
      BLE END
      ADD R2, R2, R3
      ADD R0, R0, R2
      ADD R1, R0, R3
      SUB R3, R3, #1
      B TOP
END
```

d. **Assume a is stored in R0, b is stored in R1, and c is stored in R2.**

```
if ((a < 2 && a >= b) || (b != c)) {
      a *= 2;
} else {
      a = b;
}

CMP R0, #2                    // a < 2
BGE OR                        // AND fails, go to b != c
CMP R0, R1                    // a >= b
BLT ELSE                      // AND fails to to else
OR:
      CMP R1, R2              // b != c
      BEQ ELSE                // all logic fails go to else
LSL R0, R0, #1                // multiply a by 2
B END
ELSE:
      MOV R0, R1              // a = b
END
```

2. Provide comments for the following assembly at the lines indicated by a //. Comment as if you were giving a description about what the line does and why it does it, like in a lab. Also, provide a short description of the overall functionality. To get you started, here are some hints:

> **i. The code operates on an ASCII encoded string (so the string ends with the character '0'), which begins at memory index 32.**
> **ii. Each character in a string takes up one byte of storage in memory.**
>
> **Assuming "`LDR R0, [R0, #32]`" loads the address of the ASCII coded string to R0, confirmed in office hours.**

```
      MOV R0, #0
      LDR R0, [R0, #32] // Load the address of the string into register R0.
LOOP:
      LDRB R1, [R0, #0]
      CMP R1, #0        // Compare the current character in R1 to the null
                        // terminator to check if it's the end of the string.
      BEQ END
      CMP R1, #97 // Compare the current character in R1 to 'a' (ASCII
                        // value 0x61/97).
      BLT NEXT
      CMP R1, #122// Compare the current character in R1 to 'z' (ASCII
                        // value 0x7A/122), with previous compare instruction
                        // ensures that the current byte represents a
                        // lowercase letter.
      BGT NEXT
      SUB R1, R1, #32   // Convert the now confirmed lowercase letter to
                        // uppercase by subtracting 32 from its ASCII value

      STRB R1, [R0, #0] // Store the converted uppercase character back to
                        // the current address in memory.
NEXT:
      ADD R0, R0, #1    // Move to the next character in the string by
                        // incrementing the address in R0 by 1 byte.
      B LOOP
END:
```

Short description of the code:
This assembly code iterates through an ASCII encoded string which contains its address at memory index 32 and converts any lowercase characters to uppercase. The loop continues until it encounters the null terminator.