Christopher Huynh
CMPE 12

Homework 3

1.
```
        .ORIG x3000
        LD R0, INPUT         ; load INPUT into R0
        AND R1, R1, #0       ; Clear R1

        AND R2, R2, #0       ; Clear R2
        ADD R2, R2, #1       ; Set R2 to 1 for bit checking

        AND R3, R3, #0       ; Clear R3 for 16bit counter

DO      ADD R3, R3, #1       ; add to counter
        AND R5, R5, #0
        AND R5, R0, R2       ; compare bit from R0 and R2
        BRp ONE              ; if bit is 1 go to ONE

ZERO     ADD R2, R2, R2      ; shift bits left
         BR CHECK

ONE      ADD R1, R1, #1      ; increase 1 bit counter
         BR ZERO

CHECK     AND R5, R5, #0
          ADD R5, R3, #-16   ; If counter has not checked
          BRn DO             ; all 16 bits keep looping

         HALT
INPUT    .FILL #4976

.END
```

2. R1 is not initialized with AND R1, R1, #0. Detected in run time. SUM is not defined. Detected in assembly time.

3. Advantages of doing I/O through a Trap routine is that the programmer doesn't need to know details of hardware data register, hardware status registers, and asynchronous nature of input.

4. a. 256 TRAP service routines can be implemented because there are 256 TRAP vector locations from x0000 to x00FF

b) 2 accesses to memory are made in a TRAP instruction. First is when we are fetching the instruction from memory. Second is when we are fetching source operands.

5. 1. 1111000000100001
2. x0430 ST R7, x043B
3. x0437 RET
4. HookemHorns

6. 1. 5 x 6

```
      0101   (5)
    x 0110   (6)
    ------
      0000
     01010
    010100
  0000000
  --------
  00011110   (30)
```

2. -6 x 4

```
  0110 |  1111 1010   (-6)
  1001 |  x    0100   (4)
 +   1 |  ----------
 ----- |  0006 0000
  1010 | 000000000
        111101000
      0000000000000
      -------------
           11101000 (-24)
```

3. 2 x 6

```
      0110   (6)
    x 0010   (2)
      0000
     01100
   0000000
+ 00000000
  00001100   (12)
```

4. -3 x -2

```
 0011 (3)        0010 (2)
 1100            1101
+   1           +   1
 1101 (-3)       1110 (-2)
```

```
                 11111101   (-3)
               x 11111110   (-2)
                 00000000
                111111010
               1111110100
              11111101000
             111111010000
            1111110100000
           11111101000000
        + 111111010000000
                 00000110   (6)
```

5. -8 x 7

```
  1000   (8)
  0111
+    1
  1000   (-8)
```

```
             11111000   (-8)
          x      0111   (7)
             11111000
            111110000
           1111100000
      + 00000000000
             11001000   (-56)
```