Problem Set 5 Solutions

1.a. Since the OUT trap handler was not checking the DSR before outputting, by the time the first data from the DDR was output, the DDR had already been written to several times. Therefore, it would not be possible to output all the data that was written to the DDR, i.e., some characters would not be displayed at all.

1.b. Since the letters A, B, C and D are being read multiple times, we infer that the GETC handler is not checking the KBSR before reading the KBDR.

2.a.

```
RO: x300B
R1: x300D
R2: x000A
R3: x1263
R4: x300B
```

2.b.

```
x300A Addr1:   x300B
x300B Addr2:   x000A
x300C Addr3:   x000A
x300D Addr4:   x300B
x300E Addr5:   x300D
```

3. Note: There are multiple examples where this situation can occur. As long as the condition (ST,STR,STI) instructions > (LD,LDR,LDI) instructions and the condition memory reads > memory writes both hold true.

Below, you can find a few examples that meet these conditions:

- 2 ST/STR + 2 STI and 2 LD/LDR + 1 LDI

  4 (store instructions) > 3 (load instructions) and
  Memory reads (6) > Memory writes (4)

- 2 STI and 1 LD/LDR + 0 LDI

  2 (store instructions) > 1 (load instructions) and
  Memory reads (3) > Memory writes (2)

- 2 STI and 1 LDI

  2 (store instructions) > 1 (load instructions) and
  Memory reads (4) > Memory writes (2)

- 3 ST/STR and 2 LDI + 0 LD/LDR

  3 (store instructions) > 2 (load instructions) and
  Memory reads (4) > Memory writes (3)

- 7 ST/STR and 3 LDI + 3 LD/LDR

  7 (store instructions) > 6 (load instructions) and
  Memory reads (9) > Memory writes (7)

- 7 ST/STR and 2 LDI + 4 LD/LDR

  7 (store instructions) > 6 (load instructions) and
  Memory reads (8) > Memory writes (7)

4.a. Note: There are multiple ways to execute this subroutine.

```
SET         ST R2, SaveR2
            LEA R2, MASK
            ADD R1, R1, R2
            LDR R1, R1, #0
            NOT R2, R1
            AND R0, R0, R2
            ADD R0, R0, R1
            LD R2, SaveR2
            RET
SaveR2      .BLKW 1
```

4.b. Note: There are multiple ways to execute this subroutine.

```
CLEAR       ST R2, SaveR2
            LEA R2, MASK
            LDR R1, R1, #0
            NOT R2, R1
            AND R0, R0, R2
            LD R2, SaveR2
            RET
SaveR2      .BLKW 1
```

5. If sequence starting at location in R0 is null, instruction at label A is never executed. However, if there is a string of characters at R0, the instruction at label A is executed an infinite number of times. (Why?) because the RET at END will always go back to LD R0,SAVER0.

TRAP handler needs to save the registers prior to using them within the handler. i.e.R1 in this case.

R7 must saved before using TRAP x21 and restored afterwards.

Code goes here:

```
        .ORIG x020F
        ST R1, SAVER1
        ST R7, SAVER7
START   LDR R1, R0, #0
        BRz DONE
        ST R0, SAVER0
        ADD R0, R1, #0
        TRAP x21
        LD R0, SAVER0
A       ADD R0, R0, #1
        BRnzp START
DONE     LD R1, SAVER1
        LD R7, SAVER7
        RET

SAVER0   .BLKW #1
SAVER1   .BLKW #1
SAVER7   .BLKW #1
        .END
```

6.a. 256 TRAP service routines can be implemented.
x0000- x00FF

6.b. RET stores the value of PC (before execution of the service routine) in R7 so that it can return control to the original program after execution of the service routine.
A BRnzp would not work because:

- the TRAP routine may not be reached by a 9 bit offset.
- if TRAP is called multiple times, the computer would not know which LABEL to go to (can change every time).

6.c. 2 memory accesses are made during TRAP instruction
1- access instruction in fetch
2- access trap vector table to get address of TRAP service routine

7.(a/b). There are multiple ways to execute this program. Notice that you did not have to write two separate programs. This programs accounts for the value in x6000 to be a multiple of 4 as well as not a multiple of 4.
Note: you did not have to write the sub routine mult_all

```
      .ORIG x3000
      LD R5, PTR
      LDI R6, CNT
      BRz DONEz   ;checks if more numbers to multiply(CNT=0)
MORE  LDR R1,R5,#0
      ADD R5,R5,#1
      ADD R6,R6,#-1
      BRz DONE1   ;continues if more numbers to multiply
      LDR R2,R5,#0      ;
      ADD R5,R5,#1
      ADD R6,R6,#-1
      BRz DONE2   ;continues if more numbers to multiply
      LDR R3,R5,#0
      ADD R5,R5,#1
      ADD R6,R6,#-1
      BRz DONE3   ;continues if more numbers to multiply
      LDR R4,R5,#0
      ADD R5,R5,#1
      ADD R6,R6,#-1
      BRnzp READY ;CNT is multiple of 4
DONEz AND R0,R0,#0
      BRnzp END
;
;(CNT = 4x+1) multiplies R1 by three 1's
DONE1 AND R2,R2,#0
      ADD R2,R2,#1      ;R2 = 1
      ADD R3,R2,#0      ;R3 = 1
      ADD R4,R2,#0      ;R4 = 1
      BRnzp READY
;
;(CNT = 4x+2) multiplies R1,R2 by two 1's
DONE2 AND R3,R3,#0
      ADD R3,R3,#1      ;R3 = 1
      ADD R4,R4,#0      ;R4 = 1
      BRnzp READY
;
;(CNT = 4x+3) multiplies R1,R2,R3 by 1
DONE3 AND R4,R4,#0
      ADD R4,R4,#1
READY JSR mult_all
      ADD R6,R6,#0
      BRz END           ;checks CNT
;
```

```
;if CNT is not zero takes R0 from subroutine and puts back into
memory to multiply more numbers
      ADD R5,R5,#-1
      STR R0,R5,#0
;add one back to CNT because R0 is back into memory
      ADD R6,R6,#1
      BRnzp MORE
;
;store result of multiplication in memory location RESULT
END   ST R0,RESULT
      HALT
RESULT      .BLKW 1
mult_all  … ;multiples R1,R2,R3,R4 and stores result in R0
          …
          …
          RET
PTR   .FILL x6001
CNT   .FILL x6000

.END
```

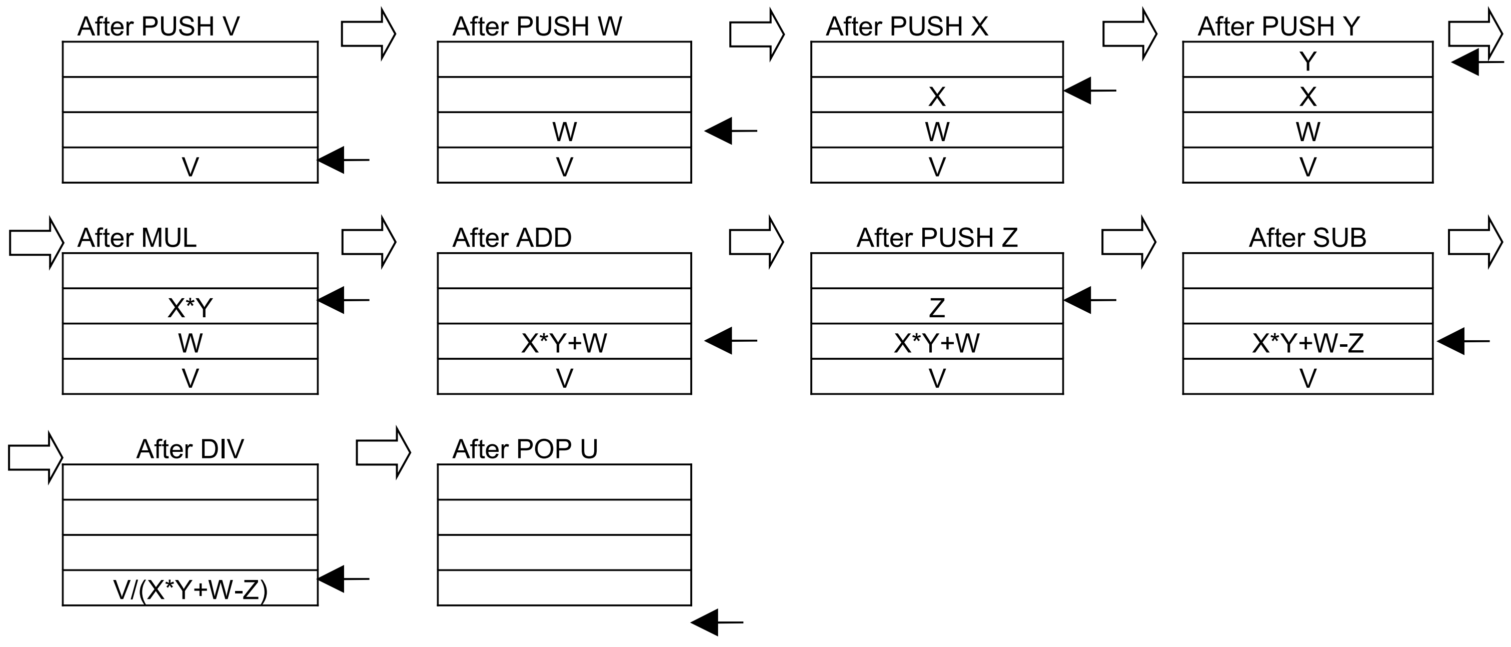
8. Need to save R7 so 1st service routine can return. Second RET overwrites the first RET value.

9. Stack is a storing mechanism. The concept of a stack is the specification of how it is to be accessed. That is, the defining ingredient of the stack is that the last thing you stored in it is the first things you remove from it. LAST IN FIRST OUT (LIFO)

Two Implementations and differences between them:
1. Stack in hardware: Stack pointer points to the top of the stack and data entries move during push or pop operations. (ex. Coin holder)
2. Stack in memory: Stack pointer points to the stack and moves during push or pop operations. Data entries do not move.

10.a. Picture of stack below: ◀— Indicates TOP of STACK, ⇨ Indicates next instruction,
Note: Though pointer to top of stack changes, the values PUSHED onto stack remain in memory.

Equation: U = V / (((X * Y) + W) – Z)



10.b. e = ((a*((b-c)+d))/(a+c))

Note: There are multiple solutions to this problem.

```
PUSH A
PUSH B
PUSH C
SUB
PUSH D
ADD
MUL
PUSH A
PUSH C
ADD
DIV
POP E
```