1 Purpose

This assignment has 2 parts:

(1) If the value of N is not valid, the program will continuly print my student ID, which is `PB22051087`. As long as the program finds that the value of N is valid, it calculates $N!$ and prints the result.

(2) If there is a interrupt signal from keyboard, the program stop print my ID, and check if the input frome keyboard is valid. If the valid input will be stored as the value of N so that the program could calculate $N!$.

2 Principles

In order to realize the above targets, we need to solve the following problems.

2.1 Interrupt-Driven I/O

Several things must be **TRUE** for an I/O device to actually interrupt the program that is running:

(1) The I/O deveice must want service;

(2) The device must have the right to request the service;

(3) The deice request must be more urgent than what the processor is currently doing.

To make sure all of those are true, we did the following things.

2.1.1 The I/O deveice must want service

This part will be done by LC-3 system. As long as we type a character, the ready bit of `KBSR` will be set, which means the I/O device want service.

2.1.2 The device must have the right to request the service

Whether an I/O device has interrupt authority is determined by its interrupt enable bit. If it is set, the device has the right to interrupt current assignment.

In the given start code, we use following instructions to set the interrupt enable bit:

```
LDI     R0, KBSR        ;xFE00
LD      R1, MASK        ;MASK is 0100 0000 0000 0000(x4000)
NOT     R1, R1          ;1011 1111 1111 1111
AND     R0, R0, R1      ;Clear the 14 bit of KBSR
NOT     R1, R1          ;0100 0000 0000 0000
ADD     R0, R0, R1      ;Set the 14 bit of KBSR
STI     R0, KBSR
```

After executing above instructions, the I/O device has the right to interrupt.

2.1.3 The deice request must be more urgent than what the processor is currently doing

This LC-3 system gives the interrupt request 4 level priority. To know this, I checked the `PSR` of interrupt request. When interrupt occured the `PSR` is `x040_`, which means the priority level of interrupt request is 4.

If we want the interrupt to occur when we type a character, we need to make sure that the priority level of our user program is lower than 4. So in the given start code, we will execute following instructions:

```
LD      R0, PSR         ;1000 0000 0000 0010(x8002)
ADD     R6, R6, #-1
STR     R0, R6, #0      ;Push PSR into SSP
LD      R0, PC          ;x3000 which is the start address of user program
ADD     R6, R6, #-1
STR     R0, R6, #0      ;Push PC into SSP
RTI                     ;Pop PC, Pop PSR
```

After executing above instructions, the LC-3 will start to run the user program and the priority level of user program is 0, which is lower than 4. So the interrupt driven by keyboard could occur.

2.1.4 Interrupt vector table

To make sure that the interrupt request can be finished. We also need the **interrupt vector table**. It will guide the `PC` to jump to the location which stores the interrupt request. The given start code uses following instructions to do this job:

```
LD      R0, VEC         ;x0180 the address of interrupt vector table
LD      R1, ISR         ;x1000 which is the start address of interrupt request
STR     R1, R0, #0      ;x1000 -> x0180
```

After executing above instructions, we stored the interrupt vector table in memory. When interrupt occurs, the start address of interrupt request will be loaded into `PC` (`Mem[x0180] -> PC`).

2.2 Print my ID

This part is easy to realize.

(1) Print my ID using `LEA` and `TRAP x22`;

(2) Delay.

The delay function is as follow:

```
DELAY       ST      R1, Save_R1         ;Save the fomer value of R1
            LD      R1, COUNT           ;#2500
REP         ADD     R1, R1, #-1
            BRp     REP                 ;Loop 2500 times
            LD      R1, Save_R1         ;Load the fomer value of R1
            RET
```

2.3 Interrupt Request

This part is easy to realize.

(1) Load the input stored in `KBDR` into `R0`;

(2) Check if `x0030` $\leqslant$ `R0` $\leqslant$ `x0039` is true;

(3) If it is true, prompt corresponding message using `LEA` and `TRAP` and store the input as the value of N;

(4) Otherwise prompt corresponding message using `LEA` and `TRAP`;

(5) `RTI`.

2.4 Calculation of Factorial

This part is a little complex. We need to construct a recursive function:

$$n! = \begin{cases} 1 & n = 0 \\ n\,(n-1)! & n \geqslant 1 \end{cases}$$

To realize this function in LC-3, I did following things.

(1) Push `R7` into `USP`;

(2) Check if `R0` $= 0$ is true (Assume the value of N is stored in`R0`);

(3) If it is true, return 1;

(4) Otherwise Push `R0` into `USP`, `R0` $=$ `R0` -1, recursively calculate $(N-1)!$ (Using `JSR`);

(5) Pop `USP` into `R0`;

(6) Return $N\,(N-1)!$.

When I use the word "return" actually means several setps.

(1) Store the value that needs to be returned;

(2) Pop `USP` into `R7`;

(3) `RET`.

The following code implements $N\,(N-1)!$.

```
;N is stored in R0, the result of (N-1)! is stored in R1
            AND     R3, R3, #0          ;Temp value
AGAIN       ADD     R3, R1, R3
            ADD     R0, R0, #-1
            BRp     AGAIN               ;R3 + R1 for N times (N is at least 1)
            ADD     R1, R3, #0          ;Return the result
            BR      FINISH
```

2.5 Print the Result

The LC-3 system uses the `TRAP` instruction to output the value represented by `ASCII Code`. Therefore, the maximum decimal number that can be output is 9, but most of the program calculation results are greater than 9, so we need to split the program calculation results bit by bit and then output them.

In order to achieve this function, I did the following things.

(1) Push `R7` into `USP`;

(2) `R0` $=$ `R1` $/10$;

(3) Push `R1` $\%10$ into `USP`;

(4) `R1` $=$ `R0`;

(5) if `R1` $\neq 0$, go back to (2).

(6) Pop `USP` into `R0` and print `R0` using `TRAP` in a loop to print the result;

(7) Pop `USP` into `R7`;

(8) `RET`.

The following code implements (2) $\sim$ (6).

```
;Assume that the value of N! is stored in R1
            AND     R4, R4, #0
            ;Record the number of digits in the result of R1
            AND     R0, R0, #0              ;R0 for the result of R1/10
BACK        ADD     R3, R1, #-10            ;Temp value
            BRn     PUSH
            ADD     R1, R1, #-10            ;R1 -= 10
            ADD     R0, R0, #1              ;R0 += 1
            BR      BACK                    ;Loop untill R1 is smaller than 10

PUSH        ADD     R6, R6, #-1
            STR     R1, R6, #0              ;Push R1 (R1 % 10) into USP
            ADD     R4, R4, #1              ;R4 += 1
            ADD     R1, R0, #0              ;R1 = R1/10
            BRz     FINE                    ;If R1/10 == 0, go to print
            AND     R0, R0, #0              ;Clear R0
            BR      BACK                    ;Next digit

FINE        LD      R1, ASCIII_0            ;x0030 -> R1
AGAIN       LDR     R0, R6, #0              ;Pop one digit
            ADD     R0, R0, R1              ;Convert decimal digit to ASCII code
            TRAP    x21                     ;Output
            ADD     R6, R6, #1
            ADD     R4, R4, #-1
            BRp     AGAIN                   ;Loop untill R4 == 0
```

When all the above problems are solved, the goal of the first section can be easily achieved.

3 Procedure

To achieve the final goal, I did the following steps.

Step (1) **Initial**: Run the given code;

Step (2) **Check N**: Check the value of N, do Step (3) if the value is valid;

Step (3) **Print my ID**: print my ID and go back to Step (2);

Step (4) **Check**: If $N \geqslant 8$ is true, prompt the corresponding message and do Step (7);

Step (5) **Calculation**: Call for factorial function.

Step (6) **Output**: Prompt the corresponding message and call for print function.

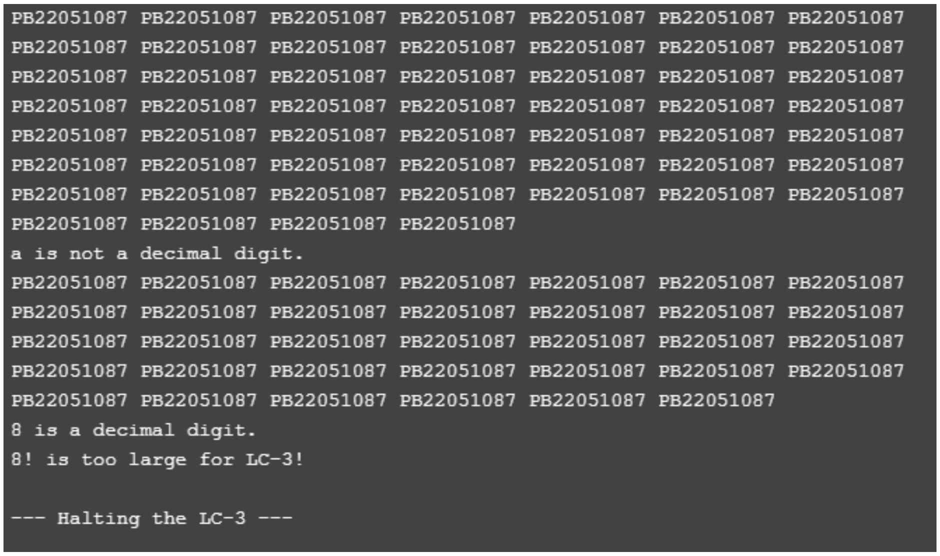
Step (7) **Done**: `HALT`.

The keyboard input interrupt will occur between Step (2) and Step (3). The implementation of interruption requirements has been discussed in detail in Section 2. After interrupt request has been served, the user program will start processing the valid N.

During my coding process, the problem I encountered was that the operation results could not be printed correctly. The reasons and solutions have been discussed in detail in Section 2.5.

4 Results

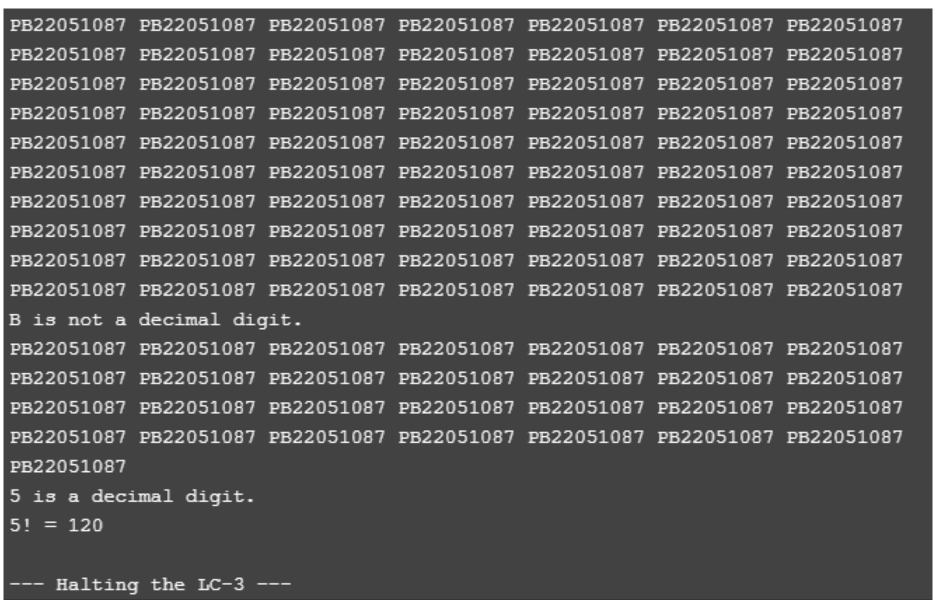
The following pictures are the results of running my program.



```
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087
a is not a decimal digit.
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
8 is a decimal digit.
8! is too large for LC-3!

--- Halting the LC-3 ---
```

Figure 1: Input a and 8



```
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
B is not a decimal digit.
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087 PB22051087
PB22051087
5 is a decimal digit.
5! = 120

--- Halting the LC-3 ---
```

Figure 2: Input B and 5

As you can see, the results are as expected.