```
/*********************************************************************************
*       ESD Lab-1 Part-1 Assembly Language Programming
*       Device - 8051
/*********************************************************************************
*       Student Name  : Rushi James Macwan
*       Course        : Embedded System Design (Spring-19)
/*********************************************************************************
*
*This code has been written and produced by Rushi James Macwan with the help of
*Keil Software and Intel 8051 instruction set keeping in mind the CU Honor Code.
*Credits and courtesy to all the third-party software platforms.
*
*Credits and courtesy to the TA (Tristan Lennertz) for suggesting me some
*improvements to my code during the lab sign-off and actually helping me to
*make it even better. I sincerely thank him for his valuable support and
*guidance.
*
/*********************************************************************************
*Copyright (C) 2019 by Rushi James Macwan
*
*Redistribution, modification or use of this software in source or binary
*forms is permitted as long as the files maintain this copyright. Users are
*permitted to modify this and use it to learn about the field of embedded
*software. Rushi James Macwan is not liable for any misuse of this material
*any misuse of this material.
*
*********************************************************************************/

/*********************************************************************************
*Please Note:
*
*Some of these improvements that were made during the sign-off may not be
*actually reflected in the initial lab submission on the D2L platform as well
*on the sign-off sheet. I have tried to include as many improvements as possible
*to the code while finally submitting it online before the cut-off deadline and
*after the sign-off has taken place. I humbly look forward to receive appreciation
*for my work and efforts while I truly admit that I have been running a bit late.
*Thank you.
*
*********************************************************************************/


// CODE BEGINS HERE //


ORG 0000h                       // Code begins at 0x0000h memory space in the RAM

CLR C                           // Clear carry flag for error-free execution of code

MOV B, #0Ah                     /* Enter the value of Y in HEX */
```

```
MOV A, B                        // Moving the value of Y into the accumulator

JZ ERROR2                       // If the accumulator is zero, then a jump is performed
                                // specifying that the error for a zero divisor has
occurred


MOV A, #20h                     /* Enter the value of X in HEX */

/**********MULTIPLICATION PART BEGINS*********/

//MUL AB - Ideal instruction that could be used otherwise

RLC A                           // Rotating the accumulator contents left twice for
multiplication by 4

JC Error1                       // Assuming that (X*4) exceeds FFh or 255 value in dec,
the carry is set
                                // If the carry is set, the jump to ERROR1 label is
performed specifying
                                // that the overflow event has occurred

CLR C                           //After the first rotation left, the carry is cleared
otherwise a carry might not be detected after the second rotation
                                //and it might stop the system from generating
an error

RLC A                           // because we need Z = (X*4) / Y

JC Error1                       // Assuming that (X*4) exceeds FFh or 255 value in dec,
the carry is set
                                // If the carry is set, the jump to ERROR1 label is
performed specifying
                                // that the overflow event has occurred

BACK: MOV 20h, A                // Storing the LSB 8-bit value of (X*4) into the memory location
of 20h in RAM

/**********MULTIPLICATION PART ENDS*********/

/*********DIVISION PART BEGINS**********/

//DIV AB - Ideal instruction that could be used otherwise

MOV R0, #0h                     // R0 register is stored with a value of 0h and will work
as a counter

HERE:   SUBB A, B               // The value of divisor stored in B register is constantly
subtracted from the accumulator
```

```
                JZ STAGE1           // Check for the value of Accumulator to be zero (in case if the remainder is zero)
                                    // extra jump not required - let it just jump once and calculate the division accordingly

                JC STAGE2           // Check for the value of Accumulator carry to be set (in case if the subtraction has resulted into a neg integer)
                INC R0              // Continuously incrementing the counter for calculating the quotient
                JMP HERE

STAGE1:         MOV A, R0           // Storing the counter value in the accumulator
                MOV 21h, A          // Storing the value of quotient at 0x21h
                MOV 22h, #0h  // Storing the value of remainder (which is zero) at 0x22h
                JMP ENDLOOP

STAGE2:         ADD A, B            // Adding
                MOV 22h, A          // Storing the value of remainder at 0x22h
                MOV A, R0           // Storing the counter value in the accumulator
                MOV 21h, A          // Storing the value of quotient at 0x21h
                JMP ENDLOOP

/*********DIVISION PART ENDS***********/


ENDLOOP:        JMP WHILE           // Infinite loop where the MCU keeps jumping between the states
WHILE:          JMP ENDLOOP

/********Specifying the error conditions*******/

ERROR1: MOV 30h, #02h       // Error corresponding to the overflow of bytes
                CLR C               // Carry flag is cleared for safety reasons
                JMP ENDLOOP

ERROR2: MOV 30h, #01h       // Zero Divisor (Y = 0) error
                JMP ENDLOOP

end                                 // Program ends
```