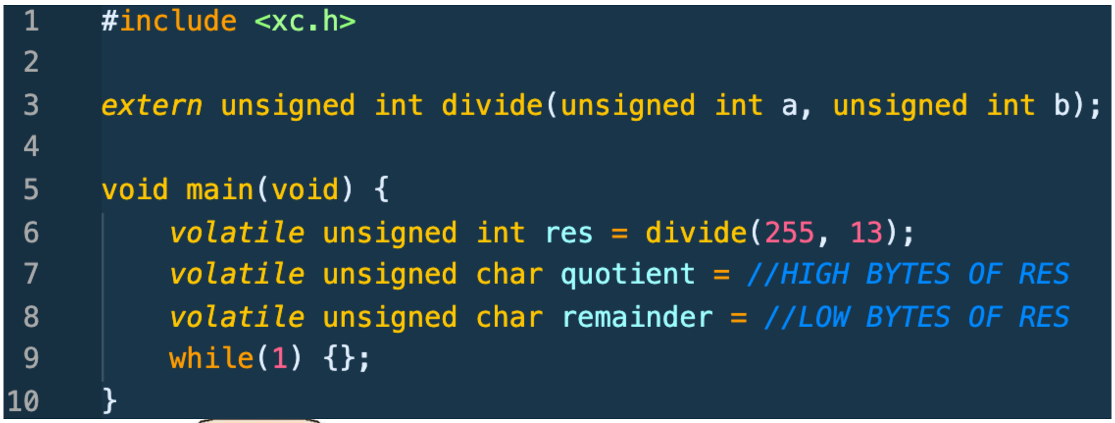
# Lab 5: Requirement Description

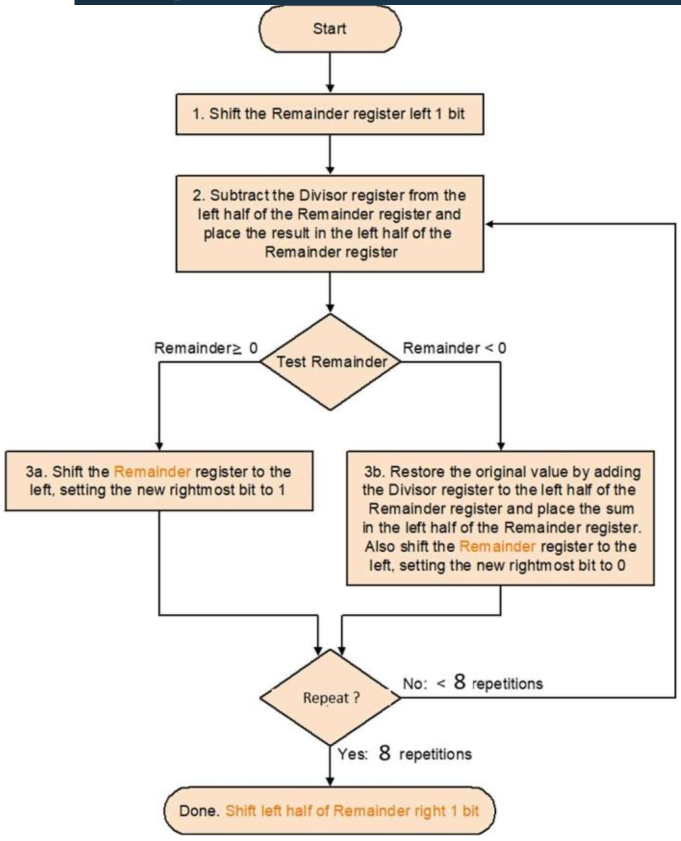
- 說明影片：
  https://youtu.be/tjyrBxM6HxI
- Basic (70%):
  - Description: Finish the following "main.c" code, which can call the divide function. Use PIC18 assembly only to implement the divider. This function should be able to input two integers, 8-bit unsigned integer divide by 4-bit unsigned integer. Then return one unsigned integer.
  - Standard of grading:
    1. Implement the following flow chart into a divisor.
    2. Enter the input as (255, 13), then show the output (decimal) quotient and remainder in WATCHes.
    3. No need to consider "dividing by 0" situation.

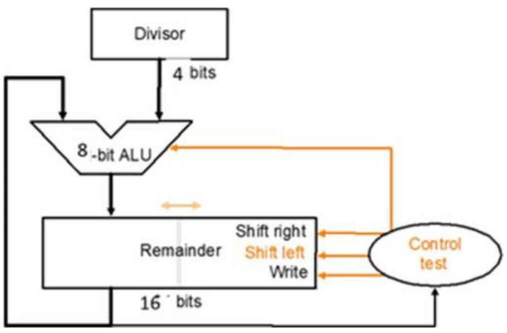


```c
#include <xc.h>

extern unsigned int divide(unsigned int a, unsigned int b);

void main(void) {
    volatile unsigned int res = divide(255, 13);
    volatile unsigned char quotient = //HIGH BYTES OF RES
    volatile unsigned char remainder = //LOW BYTES OF RES
    while(1) {};
}
```


Start
1. Shift the Remainder register left 1 bit
2. Subtract the Divisor register from the left half of the Remainder register and place the result in the left half of the Remainder register
Remainder≥ 0
Test Remainder
Remainder < 0
3a. Shift the Remainder register to the left, setting the new rightmost bit to 1
3b. Restore the original value by adding the Divisor register to the left half of the Remainder register and place the sum in the left half of the Remainder register. Also shift the Remainder register to the left, setting the new rightmost bit to 0
Repeat ?
No: < 8 repetitions
Yes: 8 repetitions
Done. Shift left half of Remainder right 1 bit



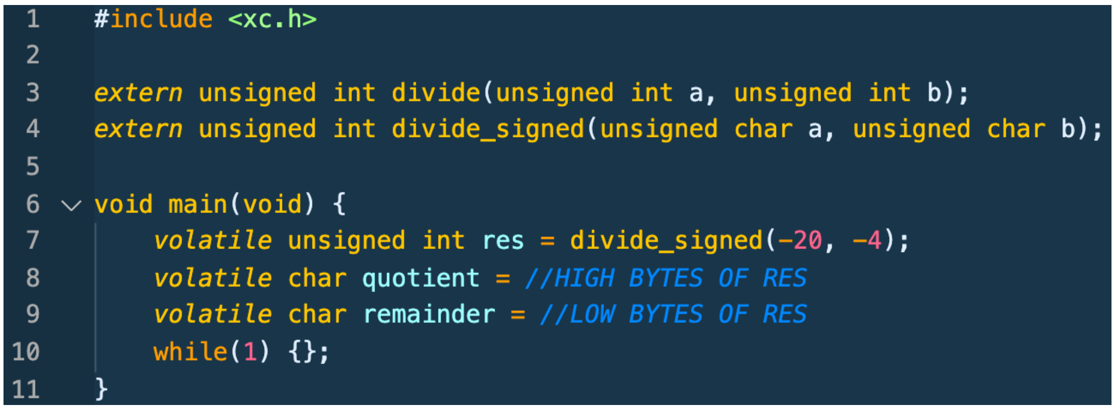
- Advanced (30%):
  - Description: Follow the basic question and modify its C code then finish the divide_signed in asm. The asm inputs will be 8-bit signed char divide by 4-bit signed char, then output an unsigned int. Finally store into two signed char then show in the WATCHes. Note that the signed data will be represent by two’ s complement. The quotient will be 8-bit and remainder will be 4-bit.
  - e.g. If quotient = -21, WATCHes will show 235(-21 + 256) in decimal.
  - Constrain: Dividend(-128~127), divisor(-8~7, without 0)
  - Standard of grading:
    1. You should NOT add more line of code in C but implement it in asm.
    2. Should pass the following testcase and each will worth 20% of grade: (-20, -4), (127, -6), (-12, -7), (-128, 5), (-3, -5).
    3. Since the quotient is 8-bit and remainder is 4-bit, then represented by 2’ s complement. Thus the anticipated output will be (5, 0), (235, 1), (1, 11), (231, 13), (0, 13).
    4. Follow 3., the relation between quotient and remainder is not fixed as long as you follow the principle that

       $$|divisor| > |remainder|$$

       E.g., devide_signed(127, -6) can be (235, 1) or (234, -5).

       $$127 \div (-6) = 21 \times (-6) + 1 = 22 \times (-6) + (-5)$$
    5. Don’ t need to consider “dividing by 0” situation.

Hint: try to predict the sign of outcome before the unsigned divisor.



```c
#include <xc.h>

extern unsigned int divide(unsigned int a, unsigned int b);
extern unsigned int divide_signed(unsigned char a, unsigned char b);

void main(void) {
    volatile unsigned int res = divide_signed(-20, -4);
    volatile char quotient = //HIGH BYTES OF RES
    volatile char remainder = //LOW BYTES OF RES
    while(1) {};
}
```

- Bonus (20%):

  Description:

  ***Given 16bit unsigned integer a, please implement：***

  ***Floor( sqrt(a) )***

  ***The function returns an 8bit unsigned integer.***

  Hint:

  1. Define your **FIND_SQUARE_ROOT_ALGO**
  2. Function definition above can be refered to C standard library
     https://www.cplusplus.com/reference/clibrary/

  Grading:

  1. **Mixing with C.** Implement the feature above in asm, and call by main function.
  2. **Using function signature as follow:**
     ```
     extern unsigned char mysqrt(unsigned int a);
     ```
  3. You should explain your code logic in detail.
  4. Show *floor(sqrt(0))*、*floor(sqrt(10))*、*floor(sqrt(15))*、*floor(sqrt(400))*、*floor(sqrt(800))*、*floor(sqrt(1300))*、*floor(sqrt(1600))* *in variable named result*. *(hint: 0x00、0x03、0x03、0x14、0x1C、0x24、0x28)*