CS2033 Fall/2022 (pair programming)
Project 2 (100 pts)

Implement **Booth's algorithm** for 32-bit multiplicand and multiplier, and print the product as a 64-bit integer in binary. See the slides for Sec3.3 on Harvey if you need to review the algorithm. Please do NOT directly use the MIPS mult instruction to multiply the two inputs, as that will result in 0 points for this project. The mult instruction is only allowed when you convert keyboard string inputs to integer multiplicand or multiplier.

This project will be implemented with pair programming. You must work with your designated partner on this project. While each group will have only one program, every group member must individually submit it to Github classroom for autograding. Remember to add some comments at the beginning of your code, including group number and member names.

(1) Input requirements:
Your program must accept keyboard **string** inputs (use the syscall 8 exactly twice) and interpret them as **signed integers** for the multiplicand and multiplier. (Hint: in project 1, you have implemented the conversion from string input to integer output, so you may adapt that code for this project).
Both inputs can be negative or non-negative integers, and your program should be able to handle them.
You must check for non-numeric inputs and report any of them as "input error!". In addition to non-numeric inputs, note that a 32-bit signed integer exactly fits in a general purpose register in MIPS, and it has a range between -2^{31} (or -2,147,483,648) and $2^{31}-1$ (or 2,147,483,647). Your program must identify if the inputs are in this range. If not, print "input error!" for that input. (Hint: mult instruction is still used for string-to-integer conversion, you may want to use mfhi and mflo to help check if the converted integer is within the range)
Think further about whether the multiplicand can be equal to -2^{31}. The multiplicand and its opposite value are both needed by Booth's algorithm, will this multiplicand value cause any problem? If so, please take care of it in your code.

Note: do NOT use an infinite loop to receive keyboard input!

(2) Output requirements:
The output product is 64 bit. As long as the 32-bit inputs are properly checked, there will be no overflow to the product. As indicated in Booth's algorithm, your code will need one register for A, one for Q and one for Q_{-1}. The arithmetic shift right operation should be

performed on AQQ_{-1} together. As the multiplier is 32 bit, the loop of the algorithm should contain 32 iterations. At the end of each iteration, your program should print the current AQ (arithmetic-right-shifted) as an intermediate result. Since these are two registers, you will print their concatenated 64-bit binary. After the algorithm terminates, you must also print the final result as the last AQ in 64-bit binary.

(3) Sample output in QtSpim:

Please do NOT print any message to prompt for keyboard inputs. Instead, directly use syscall 8 twice: one to receive a multiplicand, and the next to receive a multiplier.

For example, when you hit the QtSpim run button, you key in 5 (and Enter) and 9 (and Enter) for the multiplicand and multiplier syscalls respectively, then the output of your code must use the following format:

```
 00000000000000000000000000000101
X00000000000000000000000000001001
----------
 1111111111111111111111111111110110000000000000000000000000000100
 0000000000000000000000000000000101000000000000000000000000000010
 0000000000000000000000000000000010100000000000000000000000000001
 1111111111111111111111111111110111010000000000000000000000000000
 0000000000000000000000000000000101101000000000000000000000000000
 0000000000000000000000000000000010110100000000000000000000000000
 0000000000000000000000000000000001011010000000000000000000000000
 0000000000000000000000000000000000101101000000000000000000000000
 0000000000000000000000000000000000010110100000000000000000000000
 0000000000000000000000000000000000001011010000000000000000000000
 0000000000000000000000000000000000000101101000000000000000000000
 0000000000000000000000000000000000000010110100000000000000000000
 0000000000000000000000000000000000000001011010000000000000000000
 0000000000000000000000000000000000000000101101000000000000000000
 0000000000000000000000000000000000000000010110100000000000000000
 0000000000000000000000000000000000000000001011010000000000000000
 0000000000000000000000000000000000000000000101101000000000000000
 0000000000000000000000000000000000000000000010110100000000000000
 0000000000000000000000000000000000000000000001011010000000000000
 0000000000000000000000000000000000000000000000101101000000000000
 0000000000000000000000000000000000000000000000010110100000000000
 0000000000000000000000000000000000000000000000001011010000000000
 0000000000000000000000000000000000000000000000000101101000000000
 0000000000000000000000000000000000000000000000000010110100000000
 0000000000000000000000000000000000000000000000000001011010000000
 0000000000000000000000000000000000000000000000000000101101000000
 0000000000000000000000000000000000000000000000000000010110100000
 0000000000000000000000000000000000000000000000000000001011010000
 0000000000000000000000000000000000000000000000000000000101101000
 0000000000000000000000000000000000000000000000000000000010110100
 0000000000000000000000000000000000000000000000000000000001011010
 0000000000000000000000000000000000000000000000000000000000101101
----------
 0000000000000000000000000000000000000000000000000000000000101101
```

- Row 1 and 2 are the 32-bit multiplicand (=5) and multiplier (=9) in binary. Note that the multiplicand is printed with a space character to start and a newline character to end. The multiplier is printed with an uppercase X to start and a newline to end
- The 32 rows between the two separating lines (each has ten hyphens) are the intermediate AQs in binary from each iteration. Each row is 64-bit with a space to start and a newline to end
- The last row is the final product, with a space to start and a newline to end (verify by yourself if the binary sequence in the picture is equal to 45)

If you enter any non-numeric input or out-of-range value to the multiplicand or the multiplier, your code should print “input error!” and return at once. For example, you key in **abc** for the multiplicand syscall in QtSpim, your code should print:

```
input error!
```

, which ends with a newline. If you key in 56 for the multiplicand syscall but 8000000000 (too large) for the multiplier syscall, it should also print the above message.

(4) Pair programming requirements

In pair programming, one will be a programmer and the other will be an observer. You and your partner must frequently switch between the two roles.

To summarize your pair programming experience, work with your partner on a one-page report, which should include:

- how you switched roles on different parts of the program
- how you handled difficulties and program bugs together
- the difference between pair programming on project2 and individual programming on project1

Name your report as reportX.pdf, where X is your group number.

(5) submission on Github classroom: https://classroom.github.com/a/T1w_qTnV

Your final source code must be named prj2.asm. Add both the source code and reportX.pdf to your local repo and push to Github.

To make the Github auto-grading work, you should strictly follow the required formats above to receive inputs and print results. In addition, please do NOT change run.sh and run1.sh in your project 2's git repo.

There will be 13 tests conducted on Github for each submission. Please test your code thoroughly on your local QtSpim before submitting it.

(5) flow-chart for your code