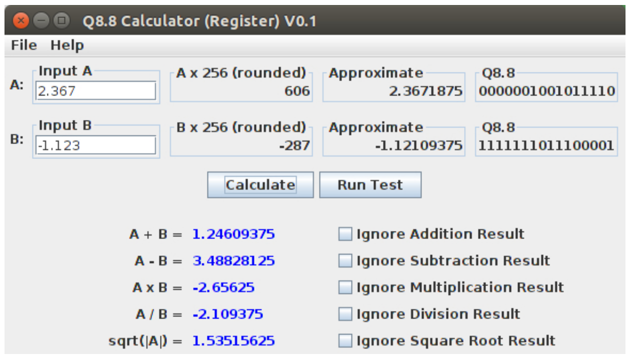
Project 1 - Floating-Point Operations without Floating-Point Instructions

CS 0447 — Computer Organization & Assembly Language

Check the Due Date on the CourseWeb

The purpose of this project is for you to practice writing assembly language to interact with input/output hardware. The hardware for this project is a very basic floating-point calculator as shown below:



A: Input A: 2.367 | A x 256 (rounded): 606 | Approximate: 2.3671875 | Q8.8: 0000001001011110

B: Input B: -1.123 | B x 256 (rounded): -287 | Approximate: -1.12109375 | Q8.8: 1111111011100001

A + B = 1.24609375 ☐ Ignore Addition Result
A - B = 3.48828125 ☐ Ignore Subtraction Result
A x B = -2.65625 ☐ Ignore Multiplication Result
A / B = -2.109375 ☐ Ignore Division Result
sqrt(|A|) = 1.53515625 ☐ Ignore Square Root Result

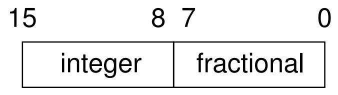
This tool can be found in `q88_calculator.zip` located in the CourseWeb under this project. Extract all files to your `[..]/mars4_5/mars/tools` directory. If you extract all files to the right directory, when you run the MARS program, you should see "Q8.8 Calculator (Register) V0.1" under the "Tools" menu.

Introduction to the Q (Number Format)

In some embedded systems, floating-point operations are not included in their implementation to keep the cost down. In other words, those embedded systems only have integer operations. To perform floating-point operations in such embedded systems, programmer must implement his/her own representation of floating-point numbers and perform calculation based on implementations.

One simple floating-point representation is called the Q number format. Q is a fixed point number format where the number of bits of integer and fractions are specified. For example, a Q8.8 number has 8 integer bit and 8 fractional bit. In this project, we will only focus on Q8.8 format.

A Q8.8 format is a 16-bit number. As mentioned earlier, it consists of 8 integer bit and 8 fractional bit. The 8 integer bit is at the top 8 bits (bit 8 to bit 15) and the fractional bit is at the bottom 8 bit (bit 0 to bit 7) as shown below:



A Q8.8 format uses Two's Complement format to represent negative value. To get a Q8.8 representation of a floating-point number, simply multiply the floating-point number by 256 and rounded to the nearest integer. For example, the Q8.8 representation of the number 3.25 is $3.25 \times 256 = 832$. Recall that 832 in 16-bit two's complement format is 0000001101000000 which is Q8.8 representation of 3.25. The Q8.8 format of -3.25 can be easily obtained as discussed in class by simply flip 3.25 bit-by-bit and add 1. Thus, the Q8.8 of -3.25 is 1111110011000000. To convert a Q8.8 format back to a floating-point number, simply divide the integer value of Q8.8 by 256. For example, suppose a Q8.8 number is 0011001100110000 which is 13104, the value is floating-point is $13104/256 = 51.1875$. Simply put, if x is a floating-point number, x in Q8.8 format is $x \times 256.0$ rounded to the closest integer.

Q8.8 Addition

Note that $(x \times 256.0) + (y \times 256.0) = (x + y) \times 256.0$. In other words, to perform $A + B$ where A and B are floating-point numbers in Q8.8 format, simply perform integer addition $A + B$. The result is a floating-point number in Q8.8 format. For example, consider two floating-point numbers is Q8.8 format, 1111110011000000 (-3.25) and 0011001100110000 (51.1857). Addition in binary is as follows:

$$1111110011000000 + 0011001100110000 = 0010111111110000$$

Note that 0010111111110000 is 12272 in decimal and $12272/256 = 47.9375$ which is the result of (-3.25) + 51.1875.

Q8.8 Subtraction

Similar to Q8.8 addition, $(x \times 256.0) - (y \times 256.0) = (x - y) \times 256.0$. Thus, to perform $A - B$ where A and B are floating-point number in Q8.8 format, simply perform integer subtraction $A - B$. The result is a floating-point number in Q8.8 format.

Q8.8 Multiplication

For multiplication, since $(x \times 256.0) \times (y \times 256.0) = (x \times y) \times (256.0 \times 256.0)$, a result of a multiplication is in Q16.16 format. Note that an 8-bit number multiplied by an 8-bit number results in a 16-bit number, and $(256.0 \times 256.0 = 2^{16}$. Thus, to perform $A \times B$ where A and B are floating-point numbers in Q8.8 format, simply perform integer multiplication $A \times B$. However, the result is in Q16.16 format as discussed earlier. So, we need to convert a Q16.16 back to Q8.8. For simplicity, simply shift the result to the right by 8 (same as divided by 256) to get Q16.8 format. For this project, we will only test with small numbers to ensure that the result of a multiplication is still in the range of Q8.8 format representation.

For this project, you are NOT ALLOWED to use any multiplication instructions. You must implement your multiplication operation using basic instructions such as addition, shifting, and bit-wise operations. Note that you only need to implement unsigned multiplication. Signed multiplication can be easily implemented by; (1) converting any negative operands to positive, (2) performing unsigned multiplication, and (3) converting the result based on signs of operands. **An unsigned multiplication method in binary will be discussed in a lecture**.

Q8.8 Division

For division, since $(x \times 256.0)/(y \times 256.0) = x/y$, a result of a division (quotient) is in Q8.0 format and the remainder ($x \texttt{ mod } y$) is also in Q8.0 format. Thus, to perform A/B where A and B are floating-point numbers in Q8.8 format, simply perform integer division A/B. Note that an integer division generally gives a quotient and a remainder. As discussed earlier, the quotient is in Q8.0 format (no fraction) and the remainder is in Q8.0 format. There are multiple ways to get the quotient in Q8.8 format. One simple way is to convert the dividend to Q8.16 by shifting left 8 times (multiply by 256). Then, perform integer division as usual to get a quotient in Q8.8 format. However, according to the division algorithm discussed in class, this work around requires us to shift the divisor left 24 times which results in a 40-bit number. Unfortunately, a register in MIPS is only 32-bit wide. One solution is to perform A/B to get a quotient1 and a remainder. Then perform (remainder $<<$ 8)$/B$ to get quotient2. The quotient in Q8.8 format is simply (quotient1 $<<$ 8)|quotient2.

For this project, you are NOT ALLOWED to use any division instructions. You must implement your multiplication operation using basic instruction such as addition, subtraction, comparisons, shifting, and bit-wise operation. Similar to multiplication, you only need to implement unsigned division and extend it to support signed division. **The unsigned division method in binary will be discussed in a lecture**.

Q8.8 Square Root

A square root of a Q8.8 number results in a Q4.4 number. We can use a simple trick by converting a Q8.8 number into Q8.16 to get a Q4.8 result. For this project, you only need to calculate the square root of the absolute value of the operand A only. The absolute value is required since a square root of a negative number is an imaginary number. **A square root method in binary will be discussed in a lecture**.

Q8.8 Calculator Hardware

The Q8.8 calculator hardware allows user to enter two operands A and B in floating-point format. It will convert them into Q8.8 format and display some information as follows:

- $A \times 256$ **(rounded)**: This is the value of the operand A multiplied by 256 and rounded to the closest integer
- **Approximate**: This is the approximate value of the floating-point number in Q8.8 format. Note that not all floating-point value can be represented exactly by Q8.8 format. For example, $2.3 \times 256 = 588.8$ which must be rounded to 589 (0000001001001101_2). If we convert 589 back, we get $589/256.0 = 2.30078125$.
- **Q8.8**: This shows the given number in Q8.8 format (16-bit Two's complement)

When a user click the "Calculate" button, the Q8.8 calculator will put the operand A in the register `$a0` and put the operand B in register `$a1` (both in Q8.8 16-bit Two's complement format). Then the program will set the register `$t9` to 1 and wait until the value of `$t9` is back to 0 (by your program). Once it detects that `$t9` is back to 0, it will display and check the results (given by your program). If a result is in an acceptable range, it will display in blue. Otherwise, your result will be displayed in red.

What you need to do is to write a program that calculate $A + B$, $A - B$, $A \times B$, A/B, and $\sqrt{|A|}$. These results must by put in the following registers:

- `$v0` should contain the result of $A + B$ and $A - B$. Put the result of $A + B$ into the lower 16-bit and $A - B$ into the upper 16-bit.
- `$v1` should contain the result of $A \times B$ and A/B. Put the result of $A \times B$ into the lower 16-bit and A/B into the upper 16-bit.
- `$a2` should contain the result of $\sqrt{|A|}$.

Results must be put into three registers as shown below:

$v0	A – B	A + B
$v1	A / B	A x B
$a2		sqrt(|A|)

The Q8.8 calculator hardware also has a test feature. If a user press the "Run Test" button, the Q8.8 calculator will perform the following steps:

1. randomly generate numbers (in integer as well as floating-point) and put them as operand A and operand B,
2. set `$t9` to 1,
3. wait until `$t9` is changed to 0, check results,
4. Go back to step 1

It will keep running for roughly 6000 iterations (1000 iterations for integer operands and 5000 iterations for floating-point operands) and stop or until your program gives an incorrect result. You can tell the Q8.8 calculator to ignore a specific result of calculation by checking its associated check box. In doing so, if a result of that specific calculation is incorrect, it will keep running test. However, the incorrect result will be shown in red as usual.

To make your program compatible with this testing feature, the structure of your program should look like the following:

```
.text

wait:  beq $t9, $zero, wait          # Wait until $t9 is not zero (1)
       # Get operands A and B from $a0 and $a1
       # Calculate A + B and put the result in the lower 16-bit of $v0
       # Calculate A - B and put the result in the higher 16-bit of $v0
       # Calculate A * B and put the result in the lower 16-bit of $v1
       # Calculate A / B and put the result in the higher 16-bit of $v1
       # Calculate sqrt(|A|) and put the result in the lower 16-bit of $a2
       add $t9, $zero, $zero         # Set $t9 back to 0
       j   wait                      # Go back to wait
```

During your calculation, you **MUST NOT** use the register `$t9`.

Requirements

The following are requirements that you **must** follow:

- Your program should be named `q88.asm`
- Since the Q8.8 Calculator uses registers `$t9` to communicate between the program and the hardware, **do not use** `$t9` for any other purposes.
- No need to have functions in your program. In other words, do not use instructions `jal` and `jr` or stack pointer (`$sp`) unless you are really familiar with functions in assembly.
- No memory reference instructions are allowed (no load and store instructions).
- Your program should have only the text segment (`.text`) and no data segment (`.data`).
- You are not allowed to use any multiplication, division, and square root instructions. **You must implement your own signed multiplication, signed division, and square root algorithms by yourself**.
- You are **NOT ALLOWED** to use any floating-point instructions.

Grading Criteria

This project will be graded based on the functionality of your program. We will mainly test your program by simply click the "Run Test" button multiple times. If there is no error, you will get 100 points. If there are errors, points will be given according to the following:

Functionality	Points
Able to assemble	5
$A + B$	10
$A - B$	10
positive × positive	10
positive × negative	5
negative × positive	5
negative × negative	5
positive / positive	10
positive / negative	5
negative / positive	5
negative / negative	5
square root	25

Submission

The due date of this project is stated on the CourseWeb. Late submissions will not be accepted. You should submit the file `q88.asm` via CourseWeb.