ICSI 404 – Assignment 4, the multiplier and the ALU

This assignment builds on the previous assignment.

Part 1

With the ability to add and subtract, we can now multiply. You must create a new class, the multiplier. It should have the following as a method:

public static longword multiply (longword a, longword b)

You **must** use the multiplication method shown in class (shifting and using the rippleAdder). You may use a loop. You may not use built-in math. Multiplying 2 32-bit numbers should yield a 64-bit number. We are ignoring the UPPER 32 bits and keeping only the lower 32-bits.

You must provide a test file (multiplier_test.java) that implements void runTests() and call it from your main, along with your existing tests. As with the other tests, these tests must be independent of each other and there must be reasonable coverage. You cannot reasonably test all of the billions of possible combinations, but you can test a few representative samples. Ensure that you test with positive and negative numbers.

Part 2

Now that we have created several mathematical operations, we need to abstract this into a calculator, called an ALU. The ALU will accept 2 longwords and 4 bits. These 4 bits will indicate which operation is to be executed on the 2 longwords. One option for this interface:

public static longword doOp(bit[] operation, longword a, longword b)

Another option is to accept the bits as four different parameters. This is up to you. The mapping of bits to operations must look like this:

1000 – and

1001 – or

1010 – xor

1011 – not

1100 – left shift

1101 – right shift

1110 – add

1111 – subtract

0111 - multiply

You **must** compare bits, not use getSigned() or getUnsigned() to determine the operation.

You must provide a test file (ALU_test.java) that implements void runTests() and call it from your main, along with your existing tests. As with the other tests, these tests must be independent of each other and there must be reasonable coverage. You cannot reasonably test all of the billions of possible combinations, but you can test a few representative samples.

You must submit buildable .java files for credit.

Rubric	Poor	OK	Good	Great
Comments	None/Excessive (0)	"What" not "Why", few (5)	Some "what" comments or missing some (7)	Anything not obvious has reasoning (10)
Variable/Function naming	Single letters everywhere (0)	Lots of abbreviations (5)	Full words most of the time (8)	Full words, descriptive (10)
Unit Tests	None (0)	Partial Coverage (7)	All methods covered, needs more cases (13)	All methods/cases covered (20)
Multiply	None (0)	Attempted (10)	Some cases work (20)	Completely working (30)
ALU	None (0)	Attempted (10)	Some cases work (20)	Completely working (30)