

Learning Outcomes

Upon completing this assignment, students should be able to

- Use a string of characters to represent a binary encoding of an integer.
- Learn to use the short (short **int**) type.
- Perform logical and arithmetic operations on bit strings.
- Check if the encoded integer is zero or negative.

Introduction

The Nand2Tetris computer uses **sixteen-bit words** for all memory and register contents. This assignment will build a data type, `BitString` that supports converting to/from Java `int` values, printing, reading, and logically operating on these bit strings.

Method

Getting Started

Start a new Gradle project with a package called `bits` for the `BitString` class and a simulator package where `BitStringClient` will live.

The `BitString` class will contain an **array** of 16 `char`, each of which will be set to `'0'` or `'1'` (a sixteen bit string or memory word). Each bit's index (in the array) is the power of 2 that place represents. This means that printing out the bits, left-to-right, would count **down** rather than up. It mean *addition* will go from low to high indices.

1. `BitString` needs a *primary* constructor that takes a `short` as its parameter. `short` stands for short `int` (the term used in C/C++) and in Java this is a 16-bit integer.

You can use the same algorithm we have used in class to generate the bit values for each of the sixteen characters in the array:

```
BitString b = new BitString(1091)
```

produces a `BitString` with an array (index 0 is on the left):

'1'	'1'	'0'	'0'	'0'	'0'	'1'	'0'	'0'	'0'	'1'	'0'	'0'	'0'	'0'	'0'
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

And `toString` should produce the string `"0000 0100 0100 0011"` (note that bit 0 is to the right, now). You should have the constructor and a `toString` method at this point.

Write `BitStringClient` to use `BitString` and its primary constructor. Test out your work (use on-line conversion programs to get the right answers). Note that *negative* values will set bit 15; there will be more on negative numbers below.

2. `BitString` needs two more constructors: the default constructor should create a `BitString` with the value 0 (has all zero characters); the copy constructor should take a `BitString` as its parameter and copy the contents of the original into a new array of `char`.
3. You will want to convert back to a `short` from a `BitString`; call the method `BitString.toShort()`. Use Horner's Method to evaluate the bit string as a polynomial with the value of x being two (remember that we used this as part of hashing).

You need to use an `int` for the calculation and then cast it to a `short` because otherwise Java will detect an overflow error:

```
int value = 0;
for (int i = 16; i > 0; i--) {
    int j = i - 1;
    value = value * 2 + (whichever value is in bit[j])
}
return value;
```

4. Implement bitwise and, or, and invert methods that **return a new value**. That is, `a.and(b)` does not change `a` (or `b`), but rather creates a brand new `BitString` that has the result of anding the two bit strings together.
5. Implement two boolean methods, `zero` that returns true if the binary string contains all zero bits and `negative` that returns true if bit 15 is set. Note that our `toShort` method converts negative numbers correctly so you can see what bit patterns go with what numbers.
6. Finally, `add` adds the two bit strings together, **ignoring** overflow. Add each bit, from 0 to 15, remembering if there was a carry from the previous place. If the sum of bit from `a` and bit from `b` and carry is **odd**, then the bit in the sum is 1. If the sum of those three values is greater than 1, then the carry (to the next column) is 1.

Documentation

README

Must document how you tested. How do you know that it is right?

Must include instructions on how to **compile** and how to **run** the program as submitted.

Deliverables

Submission medium: git to Gitea at `cs-devel.potsdam.edu`.