HPPS 2024
Assignment one
From Logic to Arithmetic

Marcus Frehr Friis-Hansen lns611
Noah Wenneberg Junge qxk266

December 1, 2024

1 Introduction

The code is fully functional for its intended purpose. When run, the code passes all test. To run the code on windows, in WSL navigate to the directory where you saved the `c` file. Compile and name the code like so

```
gcc test_numbers.c -0 name
./name
```

2 Converting to and from C integers

`bits8_from_int` is fully functional and succusfully converges the unsigned integers into `struct bits8` by extracting each bit using bitwise operations.

`bits8_to_int` is also fully functional and reconstructs the integer by combining individual bits with left shifts or bit wise operator OR. with more time we would like to have made it work with signed integers

3 Implementing addition

This part is fully functional for unsigned addition. texttttbits8_add uses a helper function (`bit_add`) to perform addition for each bit, propagating the carry. The biggest challenge was definitely to ensure the correct carry propagation across all 8-bits. . With more time we would like to have made it more general, to also be able to add signed integers.

4 Implementing negation

`bit8_negate` is fully functional and performs two complement negation by inverting all bits and adding 1. When handling edge cases such as `0` (negation of `0` is `0`) and `-1` (negation of

-1 is 1). The function bit8_add was reused which was hinted at in the assignment.

The bits8_negate works for its general purpose, able to negate both signed and unsigned integers, therefore we wouldn't change this even with more time. Well we would have added a for loop, if it wasn't for the assignment constraints, so we wouldn't have to explicitly state to negate each bit.

5 Implementing multiplication

This function is fully fuctional for unsigned multiplication. bits8_mul performs binary multiplication by summing the appropriately shifted versions of the first operand for each bit in the second operand. Furthermore this function avoids loops and conditionals by explicitly handling all 8 bits.

We would like to have generalized it to also have it multiply signed integers, and possibly have simplified it using a for loop.

Additional questions

1. Does `bits8_add()` provide correct results if you pass in negative numbers in two´s complement representation? why or why not?

   Not i does not. The `bits8_add()` function is not able to handle singed addition correctly. For example if you were to add -5 and -3 it would result in incorrect outputs because it treats the inputs as unsigned.

2. Does `bits8_mul()` provide correct results if you pass in negative numbers in two´s complement representation?

   It does not handle signed multiplication correctly which can be seen for example if multiplying -5 and -5 it would result in incorrect output because the function assumes unsigned inputs.

3. How would you implement a function `bits8_sub()` for subtracting 8-bit numbers

   To implement bits8_sub(a, b), we would use the formula:

   $$a - b = a + (-b)$$

   Negate the second operand (b) using bits8_negate. Add the negated value to the first operand (a) using bits8_add.