Ashkan Hosseini
CS315
9/21/2012

The Algorithms

Subtraction –

I got the idea behind this algorithm from CS380 which I am taking alongside this course. In that class the professor mentioned that in hardware if you don't have a subtraction as part of your ALU, you are still able to subtract by using the following property:

$A - B = A + \bar{B} + 1$

This property holds if you discard the last carry bit. So the algorithm is as follows (note that the addition algorithm from homework 1 is used here):

Input: two n-bit binary numbers A,B such that A>=B>0
Output: The difference of A and B
Take the inverse of B by flipping each bit
$A = A + \bar{B} + 1$
Discard last carry bit
Return A

The implementation of this algorithm is essentially the same as the algorithm itself; however there is one major addition that needs to be made. Since we don't discard the overflow carry but instead append to the digits in the addition implementation, we need to change to addition implementation to have the option to discard the last carry bit.

Division –

This algorithm is the iterative division algorithm covered in class:

```
Input: two n-bit binary numbers x,y where y>0
Output: the quotient and remainder of x divided by y
q=0
r=0
for i=n-1 down to 0:
        q=2q
        r=2r
        if x[i]=1: r=r+1
        if r>y:
                r=r-y
                q=q+1
return (q,r)
```

Similar to the multiplication algorithm in the previous homework, a shift function for multiplying by 2 is used and care has been taken to check correctness of c++ vectors in each stage of computation. The addition and subtraction parts of the algorithm make use of previous implementations of those algorithms for this class. Other than these things the implementation of the division algorithm is exactly the same as the algorithm itself.

Test Cases

For testing I decided not to test the boundaries of the implementations (what happens when you divide by zero) but instead created a test that made assumptions so I could test the validity of the results of expected values. So this test case is checking the correctness of the algorithm implementation when the values are within the expected domain per algorithm.

What this test does is to create random numbers and perform the operation we're testing. We create six random numbers starting with n = 4 bits in length and incrementing n by one per test up to n = 10.

Results:

Subtraction Test
x: 7
y: 2
x-y: 5

x: 8
y: 16
y-x: 8

x: 24
y: 25
y-x: 1

x: 88
y: 70
x-y: 18

x: 197
y: 123
x-y: 74

x: 220
y: 507
y-x: 287

Division Test
x: 1
y: 5
x/y quotient: 0
x/y remainder: 1

x: 8
y: 2
x/y quotient: 4
x/y remainder: 0

x: 26
y: 41
x/y quotient: 0
x/y remainder: 26

x: 72
y: 60
x/y quotient: 1
x/y remainder: 12

x: 67
y: 139
x/y quotient: 0
x/y remainder: 67

x: 24
y: 416
x/y quotient: 0
x/y remainder: 24