I used the RSA Encryption algorithm. I began with the ASCII character value as my number to be encrypted. I’ll use the first letter of the message ‘T’ as an example. Ord(‘T’)= 84. The encryption step plugs 84 into the equation m' = me mod n where m=84, m’ is the encrypted character and n and e are subject to the following constraints: given 2 primes p&q (7 & 17) n=p\*q= 119. e=5 is an arbitrary value less than n and relatively prime to φ. e & n act as the algorithms public key. I had additional constraints given the size of my registers and the immediate value for my addi function. Firstly n cannot be larger than 2^7 (the largest value possible for the addi function.) n had to be added whether it was through addi or addx. Every register was needed however so I had none to spare for a larger value. n had a lower bound restraint as well. Because values are modulated with respect to n, any value larger than n that must be encrypted will be lost. I had originally thought that I would need to make sure the result of the exponentiation step was below 2^7 which is why I chose such small p and q values (7& 17) but I was able to sidestep this restrain using the following rule of modular arithmetic.

((a \!\!\!\!\mod n)\,(b\!\!\!\!\mod n)) \!\!\!\!\mod n =  (ab) \!\!\!\!\mod n.

Using this rule I could modulate the register values after each multiplication was completed and therefore keep my registers at a reasonable size. However, due to my *legacy primes* all ASCII values above 119 are lost (or at least interpreted as ord(x-119).) This isn’t much of a problem considering all upper and lower case letters are below 119 except x,y,&z Decryption works according to the same equation as encryption m = m'd mod n except the encrypted value will be raised to the power of the private key which in this case is d or 77. Any intended recipient would have this value and n and could therefore easily decrypt the message.