Project 3 Report

Christopher Badolato

badolato.christopher@knights.ucf.edu

EEL 3801-0011: Computer Organization

4/8/2019

1.0 Project Description

The purpose of this design project is to create a QuadBitCipher of two user input values. This cipher will, compare these two values, 4-bit by 4-bit (Hex by Hex) to create a new output z. We will "break" the value into sections of 4-bits to see which set of 4-bits (4-bits from input x, 4-bits from input y) is smaller. The smaller of the two will be stored onto our output z, creating a new 32-bit integer. Once we have this new 32-bit value we will break it up into 4-bit groups and count the frequency of each. We will then print out the frequency array which will represent our cipher.

2.0 Program Design

To begin, QuadBitCipher: our main function will first ask the user for input integers representing 32 binary bits each. Inputs x and y will be stored into registers $t0 and $t1. We then move these values into our registers $a1 and $a2. We do this to pass these values as parameters to the QuadMinMixer function which will be called immediately after with jal QuadMinMixer.

We are now in our QuadMinMixer function. We can first begin by creating space on the stack by adding a (-12) to the ($sp) register. We can then load our three values to save ($ra, $s0, $s1) to the stack.

Next, we need to create a mask value initially storing 0x0000000f. We can bitwise AND this mask with our input to create a 4-bit copy of each 4-bit group of the inputs. ANDing anything and 0 will result in 0 but ANDing anything with 1 will create a copy of the value (works like multiplication). Shifting this mask left each iteration by 4 will result in us creating a mask of each 4-bit sets of the 32-bit inputs. We also need to create a result variable ($t1), that initially will store 0, or 32-bits of 0. Since the OR operation will essentially add two binary bits together, we can continuously OR our result with the 4-bit values found in each iteration. We iterate until our 32-bit result is full or 8 times. Filling in the result 4-bit at a time from left to right (LSB to MSB).

So, for each iteration we first create temporary masks by ANDing each input ($a1, $a2) with our mask ($t0), which is shift left immediately after each iteration by 4. We store these temporary masks into (result of x&&mask ($s0) and result of y&&mask ($s1)). Essentially, ($s0 = $t0 && $a1). We can now compare these values because they represent the same group of 4-bit powers of 2. Using bge we compare ($s0 and s$1). If ($s0) is greater, we can take our branch xIsGreater(1-8), which will OR ($s1) to the result ($t1) since y is the minimum of the two values then continue to the next iteration. Otherwise, we can assume x is the minimum of the two values and OR ($s0) with result ($t1) then jump to yIsGreater(1-8):. Which skips over xIsGreater(1-8):. These branches will continue until the result ($t1) is full with 8 hex bits, or 8 groups of 4-bit binary.

After the function has finished, we can pop the values ($ra, $s0, $s1) off the stack and return the space by adding 12 to ($sp). We can as well return the result ($t0) to QuadBitCipher by storing it into $v0. We can follow **Figure 1** to understand how this function runs.

Once we have returned from the QuadMinMixer function our output result z is stored within ($s0) from ($v0). We can now overwrite $v0 for printing or other syscall functions. We then will load the base address of the count array into register ($s1) as well as reset our mask ($t0) to 0x0000000f to mask the first 4 bits of z.

We then proceed to copy these first 4 Least significant bits by ANDing our new input z ($s0) with the mask in ($t0) and storing the result in ($t1). Since our value is now representing the 4 least significant bits, this value can represent the index of that value on the count array. The value stored in ($t1) will represent the index of the array but, we must first multiply this value by 4 to get the proper address with offset. So, we will shift the value ($t1 << 2) left two then store it where we stored the base address of the count array ($s1). Now that we have the proper index with offset, we can load the frequency count of that value($s1) into ($t2), add one to ($t2), then store ($t2) back onto the count array stored in ($s1). Once we have updated the frequency on the count array, we need to subtract the offset and get ($s1) back to its base address. Subtracting ($t1) from ($s1) will take us right back to our base address!

We will continue this process for all 8 sets of 4-bits. As we proceed down the value, we must change our shift value each iteration. We will shift the mask as usually by 4 but, our indexing variable ($t1) will be shifted all the way right so that it can represent the value we are taking the frequency of THEN, it is shift back left to multiply ($t1) by four as before. As opposed to shifting right, then shifting left, we can save instructions by just shift right by the total spaces away from the 4 least significant bits minus 2. So, after our first iteration we will be shifting the value right to find our index offset. This can be visualized in **figure 2**.

Once we have finished taking the frequency of hex bits in our output(z) we print out the frequency array by creating a loop through 16 (There are 16 hex digits (0-15)) using ($t3) as an iterator, and load the address of the LAST value in the array in our case 60($s1) into register ($t4) to be printed. We can subtract 4 from the value ($s1) to iterate DOWN the count array. This will print the array backwards.

Finally, the minimum required Cache size in Bytes to achieve a Cache Hit Rate of at least 90% is 256 bytes, using 8 blocks and 8 cache block size (in words).

Figure 1 (QuadMinMixer function)

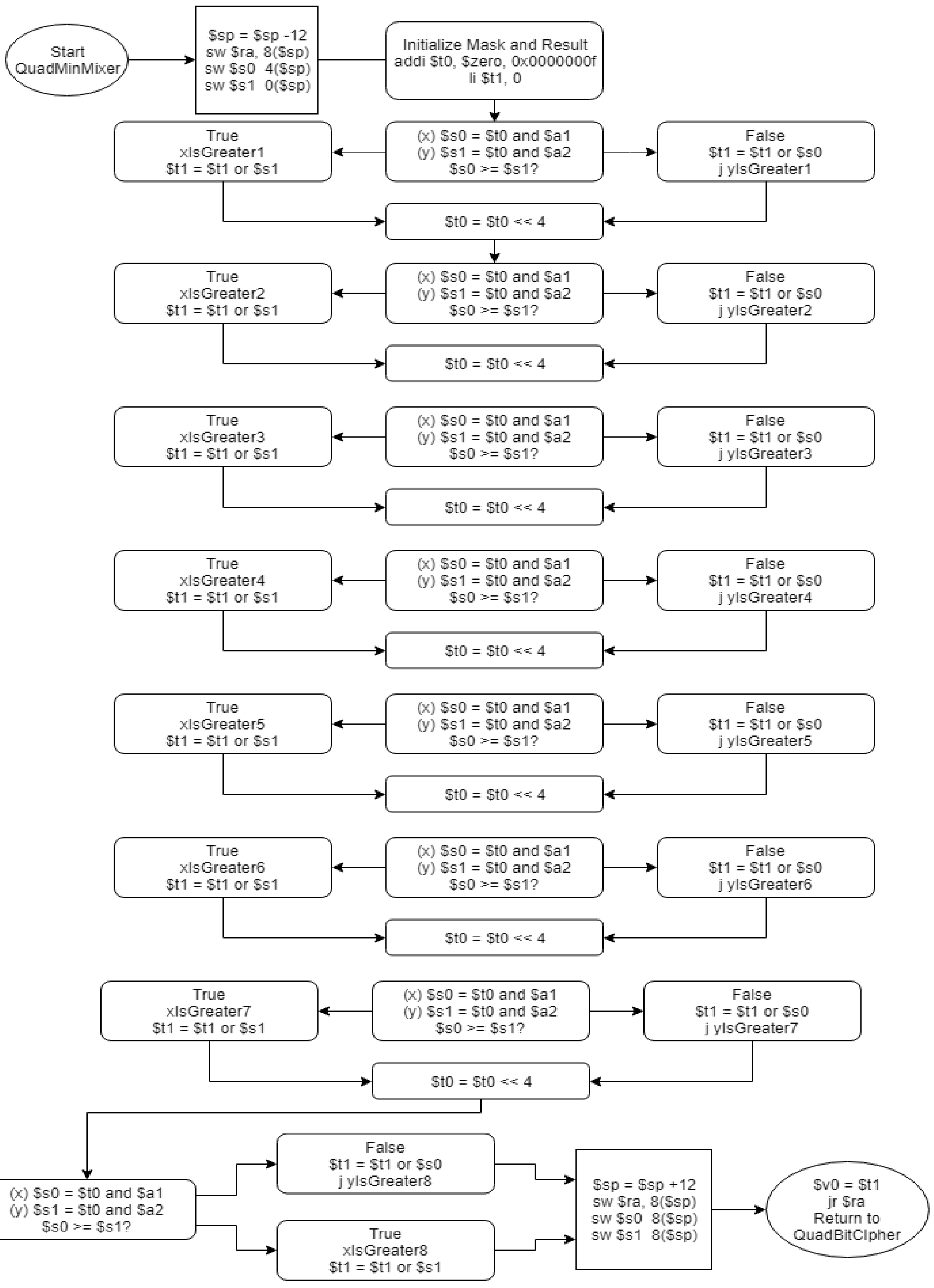
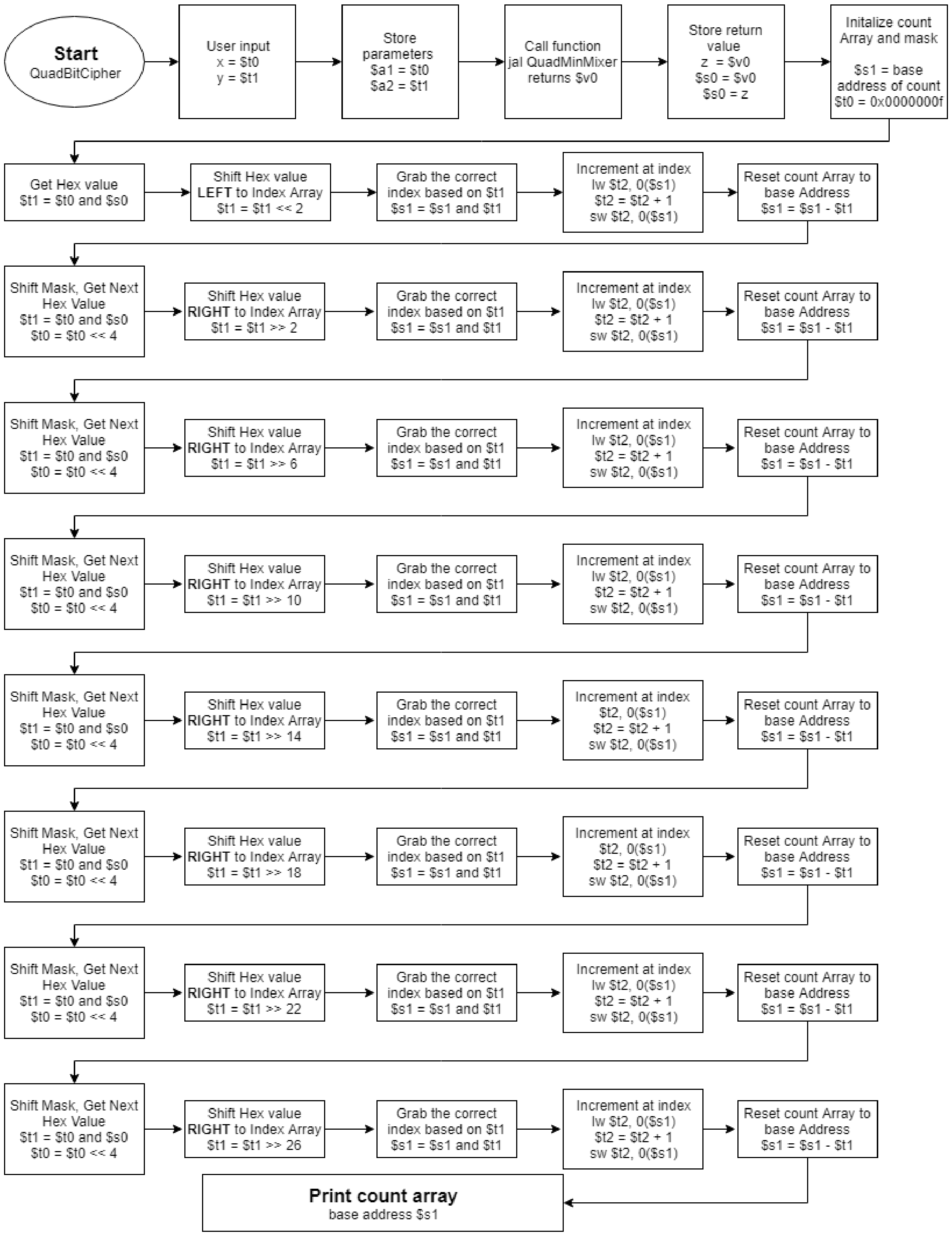


Figure 2 (Main QuadBitCipher)



3.0 Symbol and Label Tables

Symbol Tables

Registers used in main function (QuadBitCipher)

Register	Assignment
count .word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	Stores and initializes the frequency array
$v0	Stores our return value, from QuadMinMixer. Also used throughout the program to store our syscall value.
Sa1	
$a2	
$s0	When QuadMinMixer is finished we store the return value in our register $s0. We will count the frequency of hex values from this. It is z.
$s1	Stores base address of our frequency array “count”.
$t0	Stores our input x. Is overwritten to store: Mask position value. Originally (0x0000000f)
$t1	Stores our input y. Is overwritten to store: Result of ANDing mask with input z($s0) for each set of 4-bits.
$t2	Value used to load the CURRENT frequency in the count array. We will load the value from the count array. Increment it, then store it back onto the array lw $t2, 0($s1) add $t2, $t2, 1 sw $t2, 0($s1)
$t3	Counter for print loop. Incremented to 16 to print frequency array.
$t4	Used to load value at current index of frequency array in our printLoop.

Registers used in quadMinMixer function.

register	Assignment
$sp	“stack pointer”, used to keep track of the caller, as well as the passed parameters to QuadMinMixer.
$ra	Stores return address of the call of QuadMinMixer in QuadBitCipher
$a1	Stores input x as a parameter to the function.
$a2	Stores input y as a parameter to the function.
$s0	Masked value. Stores each iteration’s group of four bits masked from the input x. Is || (or’d) with result stored in $t1 each iteration to form output z. Represents **tempx** in C prototype.
$s1	Masked value. Stores each iteration’s group of four bits masked from the input y. Is || (or’d) with result stored in $t1 each iteration to form output z.

	Represents **tempy** in C prototype.
$t0	Stores the mask value that we will AND && with each input value to create a temporary value that we can compare. Then we can decide the minimum. Shifted left by 4 each iteration. Represents **mask** in C prototype.
$t1	Stores output result that is created 4-bits at a time starting from the right least significant 4-bits by ORing it with either $s0 or $s1 Represents **results** in C prototype
$v0	After all 8 iterations of 4-bits has finished we will store $t1 result into $v0 to be returned to our main function.

Label Tables

QuadBitCipher Labels

Label	Use
QuadMinMixer	Calls function QuadMinMixer
printLoop:	Loops 16 times to print the frequency array after occupation.

QuadMinMixer Labels

Label	Use
QuadMinMixer:	Represents the start of the function.
xIsGreater1-xIsGreater8	Act as if else statements We will compare the values stored in $s0 and $s1 (our masked 4 bits from inputs x and y) If x is greater, we will branch to xIsGreater1 which will add $s1 (y's 4-bits) to the result. Otherwise,
yIsGreater1-yIsGreater8	Otherwise we can assume y is greater. Which will add $s0 (x's 4-bits) then JUMP us to yIsGreater1, which will continue us on to the next iteration. Each iteration contains one of each of the two labels above to create this "if else" statement.

4.0 Learning Coverage

- Using the stack to store previous iterations of values for potential to use program recursively.
- Storing values on an integer array using store word and load word. As well as how we can store a base address, then change the address to access what we need from the data.
- Indexing of an array with a previously unknown value by shifting loaded base address by memory offset.

- Creation of function in MIPS using parameters, up to 4 or as many as we'd like. As well as returning a value from a function.
- Printing the values of an array in order with Loops or by unrolling the loops.

5.0 Prototype in C

Code can be easily copy and pasted into IDE or preferred text editor.

```
/*
Christopher Badolato
4/8/2019
EEL 3801 0011
Project 3
This program is a C prototype of the QuadMinMixer as well as the QuadBitCipher
Takes two 32 bit input values, grabs the minimum of each group of four bits between the two and
creates
a new output Z. The Hexadecimal frequency is then taken of the output value Z.
*/

#include <stdio.h>

   //Function prototypes
unsigned QuadMinMixer(unsigned x, unsigned y);

int main (){
      //Initialize frequency array, loop counter as well as temps and mask.
   int count[16], i;
   unsigned x, y, z, temp;
   unsigned mask = 0x0000000F;
      //get user inputs of x and y.
   scanf("%u", &x);
   scanf("%u", &y);
      //Call QuadMinMixer with x and y and store the output in z.
      //Print the output
   z = QuadMinMixer(x,y);
   printf("QuadMinMixer = ");
   printf("0x%x\n", z);
     //Initialize count array.
   for(i = 0; i < 16; i++){
      count[i] = 0;
   }
      //To start our cipher we must first mask our first set of 4 bits from Z.
      //and also increment the frequency count at those 4-bits
   temp = mask & z;
   count[temp]++;
      //After the first iteration we can loop through the rest.
      //We shift the mask by 4 bits each loop creating a mask of z each iteration
```

```
        //as well as taking a frequency count at the index of that temporary stored mask.
    for(i = 0; i < 7; i++){
         z = z >> 4;
         temp = z & mask;
         count[temp]++;
    }
        //Finally we can the array (backwards for our expected output.)
    printf("QuadBitCipher = ");
    for(i = 15; i >= 0; i--){
        printf("%d", count[i]);
    }
    return 0;
}

unsigned QuadMinMixer(unsigned x, unsigned y){
    unsigned tempx = 0, tempy = 0, result = 0;
    int i;
    unsigned mask = 0x0000000F;
        //create the temps for our FIRST set of 4-bits to be compared.
    tempx = x & mask;
    tempy = y & mask;
        //We went to "or" or add the minimum of the two temp values to the result.
        //if x is larger take y.
        //otherwise take x.
    if(tempx >= tempy){
        result = result | tempy;
    }
    else{
        result = result | tempx;
    }
        //After our first iteration we can continue shifting the mask left until we've masked the other 7
        //remaining groups of 4-bits or hex value.
    for(i=0; i < 7; i++){
        mask = mask << 4;
        tempx = x & mask;
        tempy = y & mask;
            //if x is greater take y
            //other wise take x.
        if(tempx >= tempy){
            result = result | tempy;
        }
        else{
            result = result | tempx;
        }
    }
        //Finally we return the results.
    return result;
}
```

6.0 Test Plan

We can test the mips program with the following inputs. These should stress test the program as long as we don't load a value larger than the max integer value.

Inputs	Outputs

X =1792801454
Y =2016082984
	QuadMinMixer = 0x682afa28
	QuadBitChipher = 1000020201000200

X = 1277564965
Y = 735994003
	QuadMinMixer = 0x2b261023
	QuadBitChipher = 0000100001001311

X =1111111111
Y =1111111111
	QuadMinMixer = 0x423a35c7
	QuadBitChipher = 0001010010112100

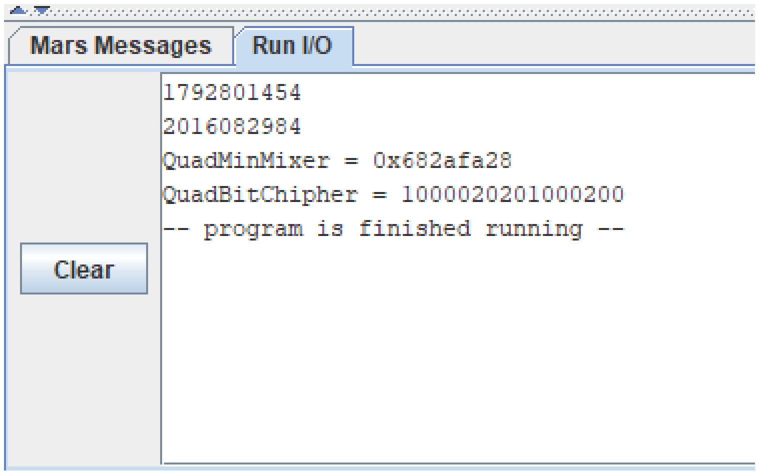
Test to make sure if else branches work.
X =0000000000
Y =1111111111
	QuadMinMixer = 0x00000000
	QuadBitChipher = 0000000000000008

Test to make sure if else branches work.
X =1111111111
Y =0000000000
	QuadMinMixer = 0x00000000
	QuadBitChipher = 0000000000000008

7.0 Test Results

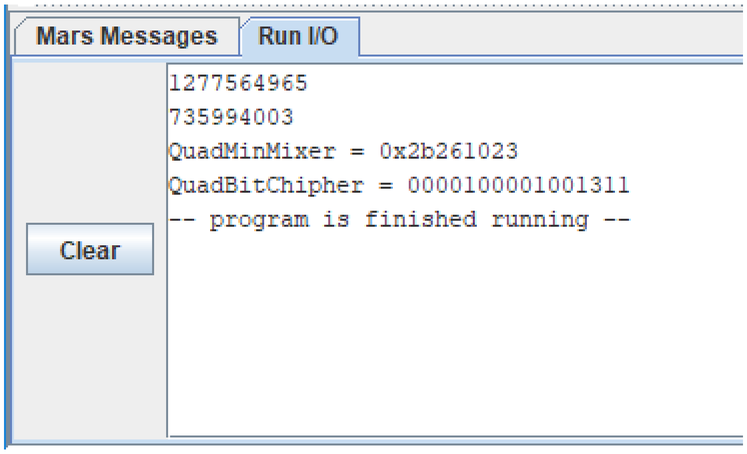
Outputs using the inputs from above.

Testcase 1



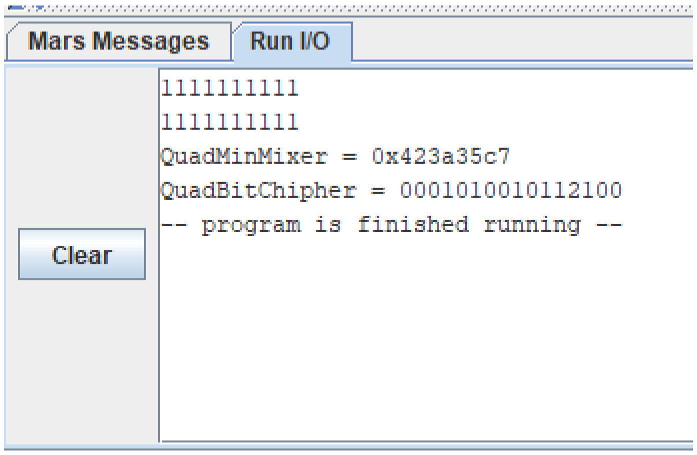
```
1792801454
2016082984
QuadMinMixer = 0x682afa28
QuadBitChipher = 1000020201000200
-- program is finished running --
```

Testcase 2



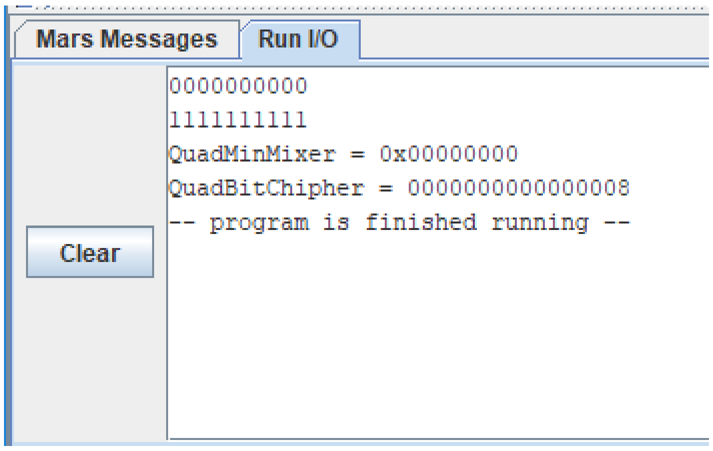
```
1277564965
735994003
QuadMinMixer = 0x2b261023
QuadBitChipher = 0000100001001311
-- program is finished running --
```

Testcase 3



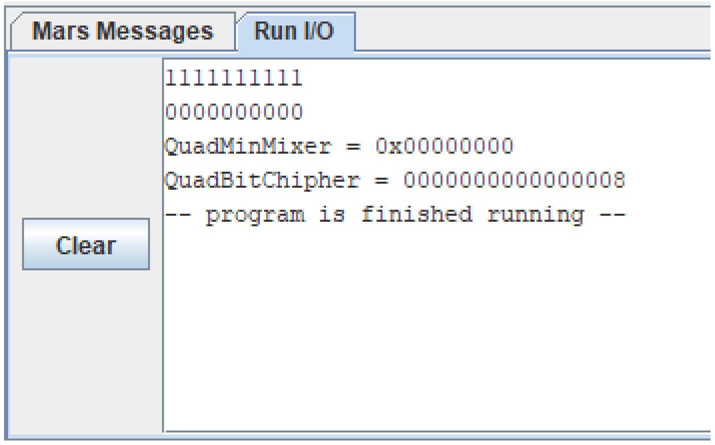
```
1111111111
1111111111
QuadMinMixer = 0x423a35c7
QuadBitChipher = 0001010010112100
-- program is finished running --
```

Testcase 4



```
0000000000
1111111111
QuadMinMixer = 0x00000000
QuadBitChipher = 0000000000000008
-- program is finished running --
```

Testcase 5



```
1111111111
0000000000
QuadMinMixer = 0x00000000
QuadBitChipher = 0000000000000008
-- program is finished running --
```

8.0 References

- EEL 3801 00219 DeMara Slides, Module 03-MIPS-ISA.pdf.
- Mars 4.5 Mips Simulator
- Recitation files
- Testing reference Sheet
- Based my MIPS code on this stack overflow code. Used to see how to use the stack and store to the stack frame https://stackoverflow.com/questions/15100476/mips-relevant-use-for-a-stack-pointer-sp-and-the-stack