Lab 6: Fixed point Conversion

Victor Yuan

Microcontrollers and Embedded Systems

March 23, 2017

Introduction:

The Goal of this lab was to write a program in assembly implementing conversion between a decimal fixed decimal point number to a binary-hexadecimal, binimal-heximal number, and converting a binary, binimal number to a decimal number.

Procedure:

Part1

Converting a non-integral number in base 10 to a binary fixed point number requires the number entered by the user to be first stored as a string to be manipulated. Once the number was stored as a string ldrb was used to access each number, character inside the whole string number. A compare was used to compare each number to the hex number 2e. 2e is the ascii representation for period(.) once 2e was seen in the string the program knew all the integral numbers had been accounted for. While each byte was loaded in a register if it was not 2e then it would either be 0x30 to 0x39, this program did not have error checking since it was assumed the user would input a correct number. 0x30 was then subtracted from the byte loaded inside the register to create the integral numbers 0 through 9. This was then multiplied by 10 and stored inside another register so as to create the numeric representation in decimal of the integer part. This integer was then called later in the program as a hex number by formatting the output as %x thus printing out the integer part.

The 4 decimal places entered after the radix point where converted into a integer representation in the same process as the number before however, changes to the format output was needed. The number was left shifted by 4, same thing as multiplying by 16 then divided by 10,000 for the to remove 4 decimal places. This would generate the first hexadecimal integer for the output. This was stored in variable hex1 to be output later. The integer would then be multiplied by 100 2 times to simulate being multiplied by 10,000. The original value would subtract this and stored back in the same register to continue this process. Until all 4 hex digits were stored.

Finally the digits where called and formatted in the form desired.

Part 2

For simplicity I wrote a separate program for part2 to clear up any registers. For binary to decimal conversion, I first used a loop to count how many 1 and 0s where in the string number before the radix point, 2e. Then a loop was constructed to multiply 1 or 0 with 2 to the power of the counter beforehand. After each loop the value was added to a separate register this allowed the value to be converted to a decimal integer and each loop the counter was decreased by 1.

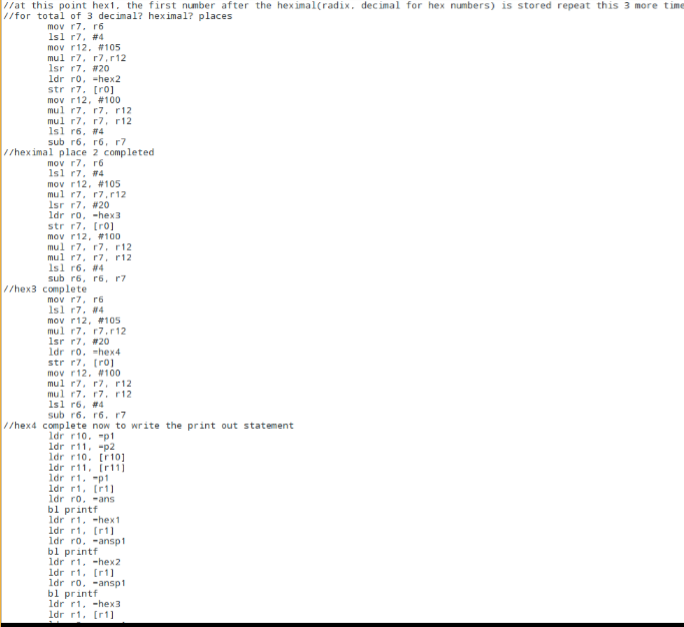
For the 8 decimal places conversion I had to manipulate more numbers. I multiplied constants until 50,000,000 was achieved. I would then load single bytes from the string number, it would either be 1 or a 0 then multiply it with the register containing 50,000,000. Whether if it’s a 1 or a 0 the result was added to a separate register and the register containing the 50,000,000 was right shifted by 1, divided by 2. This would then become 25,000,000 this process was looped 8 times to simulate 2^-1 to 2^-8. The total number would look like the expected decimal once the output was formatted.

Code:

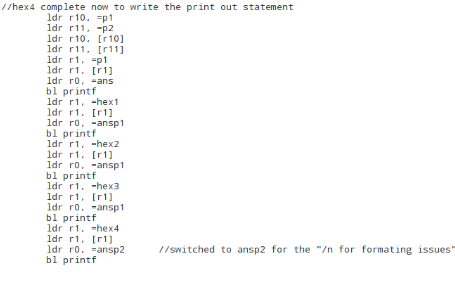
Part1:



First part of code where the numbers before and after the decimal were determined and placed into p1 and p2.

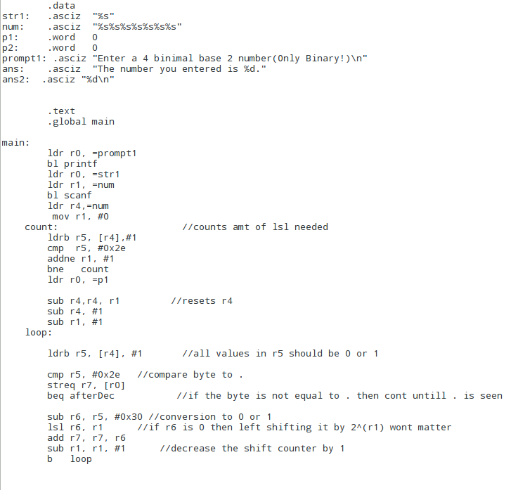


R7 and R6 held the same value r6 was the backup version for when I needed to access it again. R7 was multiplied by 16 divided by 10,000 stored in hex1, 2, 3, 4, then multiplied by 10,000 and subtracted from r6 so the process can repeat for all 4 hex numbers.

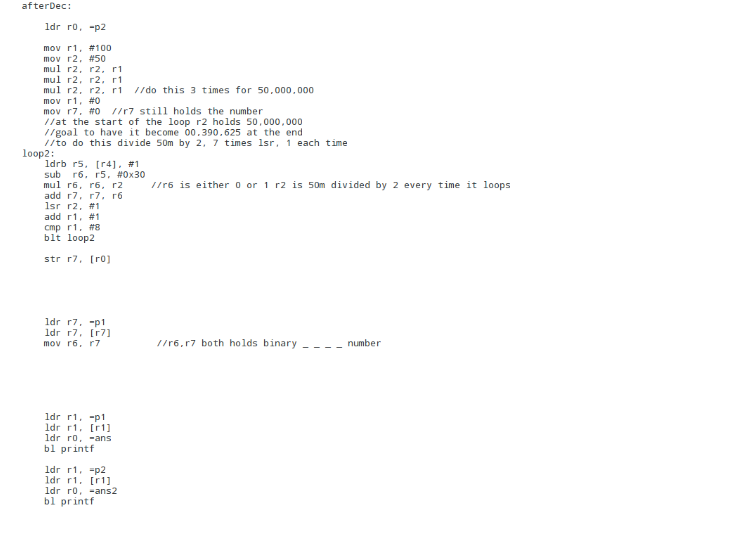


Print out commands.

Part2:



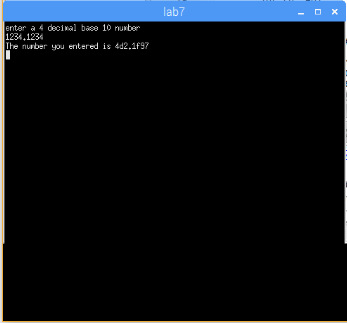
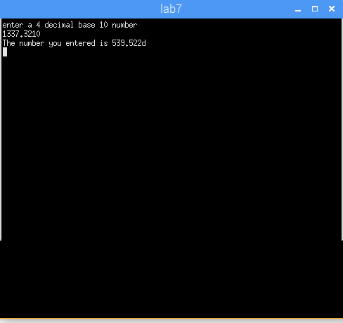
Count for determine the variable for 2^r1, used to calculate integral values before the decimal

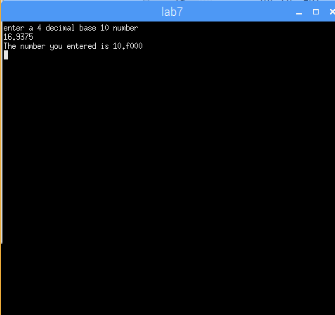


Loop 2 used for determining value used in numbers after decimal

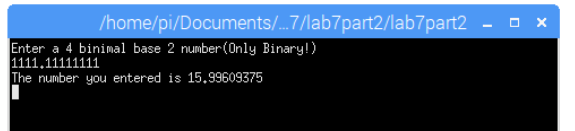
Results:

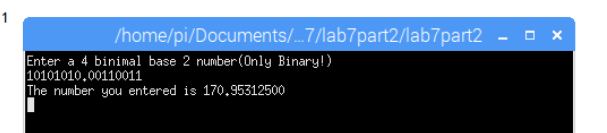
Part1:

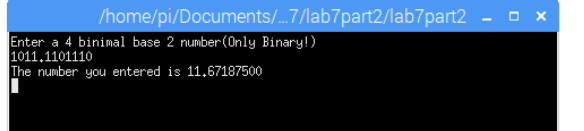
 



Part2:







The program was successful.

Conclusion:

There definitely a more efficient way of achieving the same result. A number stored in a memory location can be forced to display as a hexadecimal number by using %x. also an ascii value can be converted to numerical number by subtracted by 0x30.