Microprocessor Systems

ELE 271

Laboratory 2:

Bitwise Operators

Due 31JAN2023

By: Nick Bean , Toussaint Falangani , [Taylor Brookes](mailto:taylor.brookes@maine.edu)

**Introduction**

The goal of this short Laboratory exercise is to learn and practice with bitwise operations using the C language. Bitwise operations are particularly useful in manipulating single bits or combinations of bits in a byte or word. With these bitwise operations we can use the function of masking to manipulate only the object segment of the particular stream of data. We used the concept of masking with bit wise operations to perform a series of data manipulations against code that was already written.

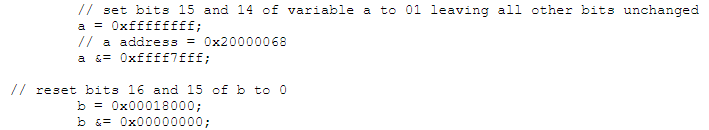
**Part 1**

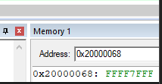
To set a bit, you need to OR the target word with another word with 1 in that specific bit position and 0 in all other words with the target. All 0's in the other positions ensure that the existing 1's in the target are as it is during OR, and the 1 in the specific positions ensures that the target gets the 1 in that position.

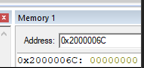
The provided code segment had a series of variables already established for manipulation (unsigned integers a,b,c and float x), and by examining the memory locations both before and following the manipulation we could look at the effects of our masking to confirm if the desired operation was successful.

Problem one - set bits 14 and 15 of variable a to ‘01’

For this application we used the bit wise &= (and gets) function, so the original value of 0xffffffff could be altered to the desired value of 0xffff7fff, while leaving all other values unchanged. The same principal was used to manipulate the specified bits of value b if the next problem.



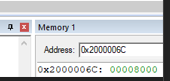




(Demonstration of the first two solutions)

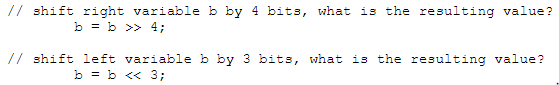
We then replaced the 1th byte of variable b with 0x80, we accomplished this with the |= (or gets) function because at this time of manipulation the value of b was 0x00000000, so logically or seemed to fit the situation.

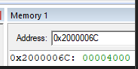


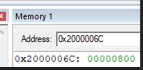


(Demonstration of or gets function )

Next we manipulated variable b with shift right and shift left functions, because the variable was an unsigned integer the values that were shifted into the existing number were 0’s

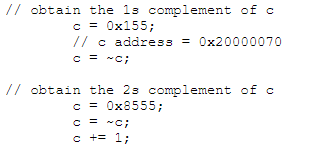


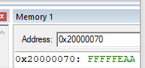


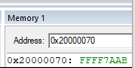


(demonstration of shift left and right functions )

Next we manipulated the c variable to obtain various complementary values, the ones complement could be attained with the ~ (not) function and the 2’s complement is similar except it must be increment up one after the not function is applied.

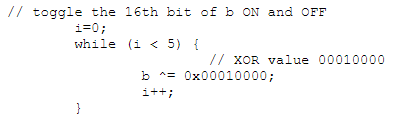


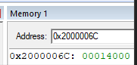


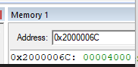


(Finding the various complementary values of c)

Following the complements we created a loop to alternate a singular bit value of b between 0 and 1, arbitrarily we decided to do the loop 5 times to avoid an infinite loop but provide the proof of concept of the ^= (XOR) function.





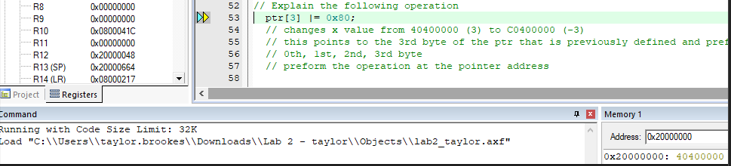


(A while loop, which cycles the 16th bit of b 5 times and exits)

The final and most interesting part of the experiment had to deal with manipulating the x value by addressing the pointer previously assigned.

// Determine the address pointed to by ptr[3]

// 0x2000000C, address of x, R1



In order to determine the address pointed to pointed to by **ptr[3]**, we use the pointer arithmetic: **char \*ptr3 = ptr + 3;** which leads us to the address of x, R1.

// Explain the following operation

ptr[3] |= 0x80;

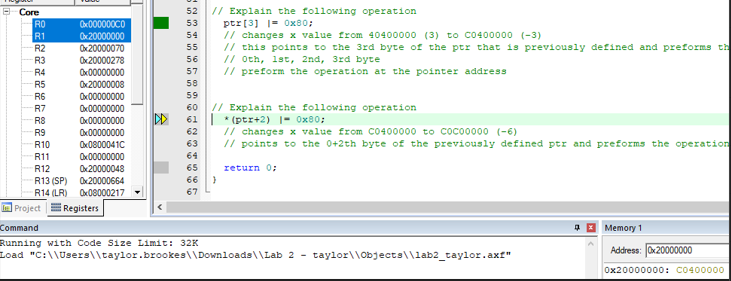
// changes x value from 40400000 (3) to C0400000 (-3)

// this points to the 3rd byte of the ptr that is previously defined and preforms the operation

// 0th, 1st, 2nd, 3rd byte

// preform the operation at the pointer address

The operation **ptr[3] |= 0x80;** sets the 7th bit of the 4th byte of the memory representation of **x** to 1 using a bitwise OR operation. The **|=** operator performs a bitwise OR operation between the current value of **ptr[3]** and the constant **0x80**, and stores the result back in **ptr[3]**. This operation sets the 7th bit of **ptr[3]** to 1, leaving the other bits unchanged.



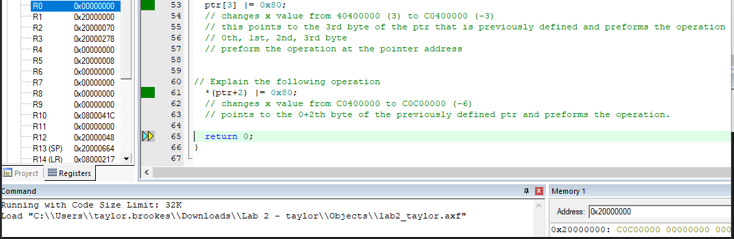
// Explain the following operation

\*(ptr+2) |= 0x80;

// changes x value from C0400000 to C0C00000 (-6)

// points to the 0+2th byte of the previously defined ptr and preforms the operation.

The operation **\* (ptr+2) |= 0x80;** sets the 7th bit of the 3rd byte of the memory representation of **x** to 1 using a bitwise OR operation. The **ptr+2** expression calculates the address of the 3rd byte in the memory representation of **x**. The **\*** operator dereferences the pointer, so the expression **\*(ptr+2)** accesses the value stored at the 3rd byte of **x**. The **|=** operator performs a bitwise OR operation between the current value of **\*(ptr+2)** and the constant **0x80**, and stores the result back in **\*(ptr+2)**. This operation sets the 7th bit of **\*(ptr+2)** to 1, leaving the other bits unchanged.



**Conclusion**

Bit masking is the skill that was widely explored in the experiment, it's a great skill to manipulate specific datasets while maintaining the integrity of the other information contained. Specifically the pointer problems are an excellent method of changing only a specific set of bytes from an information set. With that method and a known quantity of input you can select which specific byte to format and allow the other information to remain unchanged.