**Bitwise Operations Worksheet**

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Period: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

For this worksheet we’re going to revisit something we’ve covered a bit in CSCI 40, CSCI 41, and CSCI 26 – the notion of bitwise operations. Basically, you can do AND, OR, and NOT operations across one or more bytes at a time. Why is this useful? In short, they allow you to divide up a variable into multiple sub-variables, down to the level of individual bits. So rather than needing 32 different registers to hold 32 different booleans, you could use just one. Or maybe have 2 16 bit variables held in one register.

|  |  |
| --- | --- |
| **Examples of bitwise operations in C:** | **Equivalent commands in ASM:** |
| char foo = 3; //00000011 in binary | MOV R0, #3 |
| foo = ~foo; //Bitwise NOT – foo is now 11111100 | MVN R0, R0 |
| foo = foo | 1; //Bitwise OR – foo is now 11111101 | ORR R0, R0, #1 |
| foo = foo & 7; //Bitwise AND – foo is 00000101 | AND R0, R0, #7 |
| foo = foo << 2; //LSL by 2 – foo is now 00010100 | MOV R0, R0, LSL #2 |
| foo = foo ^ 2; //XOR 10100 and 10 = 10110 | EOR R0, R0, #2 |
| foo -= foo & 15; //Bitclear 1111 = foo is 10000 | BIC R0, R0, #15 |

For this worksheet you will start with some bit pattern held in R0, and write down the resulting bit pattern after each assembly command. The results carry over from one command to the next.

**AND**

To do a bitwise AND between two numbers, each matching bit is compared, and if both are true (i.e. 1), then it will be set to true in the result, otherwise 0. Example 00001111 & 10101010 gives us 00001010.

|  |  |
| --- | --- |
| **ASM Commands** | **The value of R0 after the command runs** |
| MOV R0, #15 | 00001111 |
| AND R0, R0, #14 | 00001110 |
| AND R0, R0, #31 |  |
| AND R0, R0, #3 |  |
| AND R0, R0, #1 |  |
| AND R0, R0, #255 | 00000000 |
| AND R0, R0, #42 |  |

**OR**

To do a bitwise OR between two numbers, each corresponding bit is compared, and a true is set in the corresponding place in the result if either was true. Example: 00001111 | 10101010 = 10101111.

|  |  |
| --- | --- |
| **ASM Commands** | **The value of R0 after the command runs** |
| MOV R0, #15 | 00001111 |
| ORR R0, R0, #17 | 00011111 |
| ORR R0, R0, #31 |  |
| AND R0, R0, #3 |  |
| ORR R0, R0, #4 |  |
| AND R0, R0, #255 | 00000111 |
| ORR R0, R0, #42 |  |

**NOT**

To do a bitwise NOT you just flip all the bits. You use the MVN command to do this.

|  |  |
| --- | --- |
| **ASM Commands** | **The value of R0 after the command runs** |
| MOV R0, #15 | 00001111 |
| MVN R0, #15 | 11110000 |
| MOV R0, #0 |  |
| MVN R0, R0 |  |
| AND R0, R0, #1 |  |
| MVN R0, R0 | 11111110 |
| MVN R0, #42 |  |

**XOR**

The XOR operation will return true if either of the bits is true, but not both. (And not if both are false either.) It means either-or, hence the EOR name in ARM. 00001111 ^ 10101010 = 10100101.

|  |  |
| --- | --- |
| **ASM Commands** | **The value of R0 after the command runs** |
| MOV R0, #15 | 00001111 |
| EOR R0, R0, #2 | 00001101 |
| AND R0, R0, #7 |  |
| ORR R0, R0, #3 |  |
| EOR R0, R0, #7 | 00000000 |
| EOR R0, R0, #31 |  |
| EOR R0, R0, #31 |  |

**BIC**

The Bitclear (BIC) command clears all of the bits that are true in the second parameter on the first parameter. It’s also called the NAND or not-and operation. A BIC with all 1s will clear all of the values to 0, but you can do so something like a bitclear with just the lower half ones to clear just half a register. For example, if R0 = 10101010, and we did a BIC R0, R0, #15, then the result would be 10100000.

|  |  |
| --- | --- |
| **ASM Commands** | **The value of R0 after the command runs** |
| MOV R0, #31 | 00011111 |
| BIC R0, R0, #7 | 00011000 |
| BIC R0, R0, #8 |  |
| ORR R0, R0, #3 |  |
| BIC R0, R0, #24 |  |
| EOR R0, R0, #3 |  |