Bit-String Flicking

ACSL Contest #2

What are bit strings?

- Bit strings (strings of binary digits) are frequently manipulated bit-by-bit using the logical operators **NOT**, **AND**, **OR**, and **XOR**.
  - Example of a bit string: **1011011**
- Bits strings are manipulated as a unit using **SHIFT** and **CIRCULATE** operators.
- The bits on the left are called the *most significant bits* and those on the right are the *least significant bits*.

Why do they matter?

- Most high-level languages (e.g., Python, Java, C++), support bit-string operations.
- Programmers typically use bit strings to maintain a set of flags.
  - Suppose that a program supports 8 options, each of which can be either “on” or “off”. One could maintain this information using an array of size 8, or one could use a single variable (if it is internally stored using at least 8 bits or 1 byte, which is usually the case) and represent each option with a single bit.
  - In addition to saving space, the program is often cleaner if a single variable is involved rather than an array. Bits strings are often used to maintain a set where values are either in the set or not. Shifting of bits is also used to multiply or divide by powers of 2.
- Mastering this topic is essential for systems programming, programming in assembly language, optimizing code, and hardware design.

Bitwise Operators

- **NOT** is a unary operator that performs logical negation on each bit. **Bits that are 0 become 1, and those that are 1 become 0.**
  - For example: ~101110 has a value of 010001.
- **AND** is a binary operator that performs the logical AND of each bit in each of its operands. **The AND of two values is 1 only if both values are 1.**
  - For example, 1011011 and 011001 has a value of 001001. The AND function is often used to isolate the value of a bit in a bit-string or to clear the value of a bit in a bit-string.

Bitwise Operators

- **OR** is a binary operator that performs the logical OR of each bit in each of its operands. **The OR of two values is 1 only if one or both values are 1.**
  - For example, 1011011 or 0011001 has a value of 1011011. The OR function is often used to force the value of a bit in a bit-string to be 1, if it isn't already.
- **XOR** is a binary operator that performs the logical XOR of each bit in each of its operands. **The XOR of two values is 1 if the values are different and 0 if they are the same.**
  - For example, 1011011 xor 011001 = 110010. The XOR function is often used to change the value of a particular bit.

Important Info:

- All binary operators (AND, OR, or XOR) must operate on bit-strings that are of the same length.
- If the operands are not the same length, the shorter one is padded with 0's on the left as needed.
- For example, **11010 and 1110** would have value of **11010 and 01110 = 01010**.

Summary of Bitwise Operators

x	y	not x	x and y	x or y	x xor y
0	**0**	1	0	0	0
0	**1**	1	0	1	1
1	**0**	0	0	1	1
1	**1**	0	1	1	0

Shift Operators

- Shift operators manipulate single bit strings
- Two types:
  - Logical shifts
    - LSHIFT-x and RSHIFT-x: “ripple” the bit-string x positions in the indicated direction, either to the left or to the right. Bits shifted out are lost; zeros are shifted in at the other end.
  - Circulates
    - RCIRC-x and LCIRC-x: “ripple” the bit string x positions in the indicated direction. As each bit is shifted out one end, it is shifted in at the other end. The effect of this is that the bits remain in the same order on the other side of the string.

Examples of Shift Operators

x	(LSHIFT-2 x)	(RSHIFT-3 x)	(LCIRC-3 x)	(RCIRC-1 x)
01101	10100	00001	01011	10110
10	00	00	01	01
1110	1000	0001	0111	0111
1011011	1101100	0001011	1011101	1101101

Note: The size of a bit-string does not change with shifts, or circulates. If any bit strings are initially of different lengths, all shorter ones are padded with zeros in the left bits until all strings are of the same length.

Order of Precedence (Highest to Lowest)

---

1. NOT
2. SHIFT and CIRC
3. AND
4. XOR
5. OR

In other words, all unary operators are performed on a single operator first. Operators with equal precedence are evaluated left to right; all unary operators bind from right to left.

Sample Problem 1

Evaluate the following expression:

(101110 AND NOT 110110 OR (LSHIFT-3 101010))

Solution: The expression evaluates as follows:

(101110 AND **001001** OR (LSHIFT-3 101010))

(**001000** OR (LSHIFT-3 101010))

(001000 OR **010000**)

011000

Sample Problem 2

– – –

Evaluate the following expression:

(RSHIFT-1 (LCIRC-4 (RCIRC-2 01101)))

Solution: The expression evaluates as follows, starting at the innermost parentheses:

(RCIRC-2 01101) => 01011

(LCIRC-4 01011) => 10101

(RSHIFT-1 10101) = 01010

Problem 3

List all possible values of x (5 bits long) that solve the following equation.

(LSHIFT-1 (10110 XOR (RCIRC-3 x) AND 11011)) = 01100

Solution: Since x is a string 5 bits long, represent it by abcde.

(RCIRC-3 abcde) => cdeab

(cdeab AND 11011) => cd0ab

(10110 XOR cd0ab) => Cd1Ab (the capital letter is the NOT of its lower case)

(LSHIFT-1 Cd1Ab) => d1Ab0

So, d1Ab0 = 01100.

Meaning, we must have d=0, A=1 (hence a=0), b=0. Thus, the solution must be in the form 00*0*, where * is an “I-don’t-care”.

The four possible values of x are: 00000, 00001, 00100 and 00101.

Problem 4

Evaluate the following expression:

((RCIRC-14 (LCIRC-23 01101)) | (LSHIFT-1 10011) & (RSHIFT-2 10111))

Solution: The problem can be rewritten as

A | B & C

The AND has higher precedence than the OR.

The evaluation of expression A can be done in a straightforward way: (LCIRC-23 01101) is the same as (LCIRC-3 01101) which has a value of 01011, and (RCIRC-14 01011) is the same as (RCIRC-4 01011) which has a value of 10110. Another strategy is to offset the left and right circulates. So, ((RCIRC-14 (LCIRC-23 01101)) has the same value as (LCIRC-9 01101), which has the same value as (LCIRC-4 01101) which is also 11010.

Expressions B and C are pretty easy to evaluate:

B = (LSHIFT-1 10011) = 00110

C = (RSHIFT-2 10111) = 00101

The expression becomes

A | B & C = 10110 | 00110 & 00101 = 10110 | 00100 = 10110

Past Contests Senior

3. **Bit-String Flicking**
   Evaluate the following expression:

   (RCIRC-2 (LSHIFT-1 (LCIRC-2 (RSHIFT-1 10101))))

3. **Bit-String Flicking**
   (RCIRC-2 (LSHIFT-1 (LCIRC-2 (RSHIFT-1 10101))))
   = (RCIRC-2 (LSHIFT-1 (LCIRC-2 01010)))
   = (RCIRC-2 (LSHIFT-1 01001))
   = (RCIRC-2 10010)
   = 10100

Past Contests Senior

4. **Bit-String Flicking**

Solve for X (5-bits) in the following equation.

01011 OR (RCIRC-2 X) = 01111

4. Bit-String Flicking

Let X = abcde

LHS = 01011 OR (RCIRC-2 X)

= 01011 OR deabc

= d1a11

RHS = 01111

If d1a11 = 01111, then a = 1, b = *, c = *, d = 0, e = *

So X = 1**0*

Past Contests Intermediate

– – –

3. Bit-String Flicking

Evaluate the following expression:

NOT(10111 AND 01001) AND (10111 OR (NOT 01110))

3. Bit-String Flicking

NOT(10111 AND 01001) AND (10111 OR (NOT 01110))
= (NOT 00001) AND (10111 OR 10001)
= 11110 AND 10111
= 10110

Past Contests Intermediate

– – –

4. Bit-String Flicking

Evaluate the following expression:

(LCIRC-2 01110) OR (NOT 10101) AND (RSHIFT-1 01110)

4. Bit-String Flicking

(LCIRC-2 01110) OR (NOT 10101) AND (RSHIFT-1 01110)

= 11001 OR (01010 AND 00111)

= 11001 OR 00010

= 11011

Source Links

---

https://www.acsl.org/get-started/study-materials

https://www.categories.acsl.org/wiki/index.php?title=Main_Page