

# Bitstring Operations

## CS 350, Lecture 8

### A. Why?

- Bitstring shifts correspond to multiplication and division by two.
- Bitwise operations with masks enable us to manipulate particular bits of a bitstring.
- On/off switches are natural to use with the voltages that represent binary data.
- Transistor circuits act as switches.

### B. Outcomes

At the end of today, you should:

- Be able to perform bitstring operations.

### C. Bitstring Operations

- Bitstrings
  - Can use an  $n$ -bit bitstring to record  $n$  true/false values.
    - Text numbers the bits 0, 1, ... starting from the right.
  - Can specify individual bits using index in square brackets
    - Example: If  $Z = 011$ , then  $Z[0] = Z[1] = 1$ ,  $Z[2] = 0$ .
  - Can select ranges using index : index in square brackets
    - Example: If  $Z = 011$ , then  $Z[0:1] = 11$ ,  $Z[1:2] = 01$ ,  $Z[0:2] = Z = 011$ .
- **Shift left/right** [multiply and divide by 2]. Examples:
  - Left shift  $k$  bits with zero fill corresponds to multiplying by  $2^k$ .
    - Left shift 111101 3 bits unsigned adding zeros from right yields 111101000.
    - Left shift 001101 1 bit left in 2's complement, get 011010. ( $13 * 2 = 26$ )
    - If you left shift 001101 two bits left you get 110100 = - 001100 = -12 (overflow)
    - Left shift 111101 3 bits (filling **in with** zeros from right) yields 101000 = - 011000. ( $-3 * 8 = -24$ )
    - Left shift 111101 four bits yields 010000, which overflowed.
    - Successive 1-bit left shifts of 111101 (=  $-3_{10}$ ) yield 111010 (=  $-6_{10}$ ) then 110100 (=  $-12_{10}$ ) then 101000 (=  $-24_{10}$ ) then 010000 (=  $16_{10}$ ; if sign bit's carry-in  $\neq$  its carry out, then arithmetically, multiplying by 2 caused overflow).
  - Right shift can correspond to division, but ....
    - Right shift 110101 3 bits (**filling in with zeros** from left) yields 000110.
      - Works correctly as division if unsigned.
    - Right shift 110101 3 bits (**filling in with sign bit**) yields 111110.
      - In 2's complement,  $110101 = -001011 = -11_{10}$ ;  $111110 = -000010 = -2$ .
      - $-11 / 8 = -2 + 5$

- Right shift  $k$  bits with sign bit fill corresponds to dividing by  $2^k$ .
  - E.g., successive 1-bit right shifts of 010101 ( $= 21_{10}$ ) yield 001010 ( $= 10_{10}$ ) then 000101 ( $= 5_{10}$ ) then 000010 ( $= 2_{10}$ ) then 000001 ( $= 1_{10}$ ) then 000000 ( $= 0$ )
  - E.g., successive 1-bit right shifts of 110101 ( $= -001011 = -11_{10}$ ) yield 111010 ( $= -000110 = -6$ ) then 111101 ( $= -000011 = -3$ ) then 111110 ( $= -000010 = -2$ ) then 111111 ( $= -1$ ) then 11111 ( $= -1$ ) etc.
- **Circular shift** — copy bits that fall off to the other side.
  - Circular shift left 110101 one bit repeatedly yields 101011, 010111, 101110, etc.
  - Circular shift right 110101 one bit repeatedly yields 111010, 011101, 101110, etc.

### • Bitwise operations

- Unary operation (e.g., NOT): Perform operation on each bit of string.
  - E.g., bitwise NOT of 101101 = 010010.
  - Same as 1's complement of bitstring considered as a number.
- Binary operations (e.g., OR): Perform operation on corresponding bits from to strings. Examples:
  - Bitwise OR of 101100 and 100001 is 101101.
  - Bitwise AND of 101100 and 100001 is 100000.

### • Bit Masks

- Masks are bitstrings used to help us work with only some specified bits of a bitstring.
- **Test for bit**: To see if  $X[k]$  (bit  $k$  of bitstring  $X$ ) is 1:
  1. Create a mask bitstring  $M$  that's all 0's except at bit  $k$ :  $M[i] = 0$  (for  $i \neq k$ ) and  $M[k] = 1$ .
  2. Take bitwise  $X$  AND  $M$  if result is all 0's, then bit  $X[k] = 0$ .
  3. If result was not  $\neq$  all 0's, then  $X[k] = 1$ .
- **Set bit (to 1)**: To set bit  $X[k]$  to 1
  1. Create a mask  $M$  of all 0's except  $M[k] = 1$ :  $M[i] = 0$  (for  $i \neq k$ ) and  $M[k] = 1$ .
  2. Set  $X$  to bitwise  $X$  OR  $M$ .
    - For  $i \neq k$ , the new  $X[i] = \text{original } X[i] \text{ OR } 0$ , which equals the original  $X[i]$ .
    - new  $X[k] = \text{original } X[k] \text{ OR } 1$ , which equals 1.
- **Unset bit (to 0)**: To unset bit  $X[k]$  to 0:
  1. Create a mask  $M$  of all 1's except at  $k$ :  $M[i] = 1$  if  $i \neq k$ , and  $M[k] = 0$ .
  2. Set  $X$  to bitwise  $X$  AND  $M$ .
    - For  $i \neq k$ , the new  $X[i] = \text{original } X[i] \text{ AND } 1$ , which equals the original  $X[i]$ .
    - new  $X[k] = \text{original } X[k] \text{ AND } 0$ , which equals 0.
- **Flip bit** (from 1 to 0 or vice versa):

1. Create a mask  $M$  of all 0's except  $M[k] = 1$ :  $M[i] = 0$  (for  $i \neq k$ ) and  $M[k] = 1$ .
2. Set  $X$  to bitwise  $X \text{ XOR } M$ .
  - For  $i \neq k$ , the new  $X[i] = \text{original } X[i] \text{ XOR } 0$ , which equals the original  $X[i]$ .
  - new  $X[k] = \text{original } X[k] \text{ XOR } 1$ , which equals  $\text{NOT } X[k]$ .