Bit Manipulation

Dae Hyun Kim

EECS
Washington State University

Overview

- You use bitwise operations to generate special signals, perform special math functions, and so on.

- We will assume that an n-bit register is composed of n 1-bit flip-flops.
  - $A = a_{n-1}a_{n-2} \dots a_0$
  - $B = b_{n-1}b_{n-2} \dots b_0$

- Available bitwise operations (A or B could be a constant)
  - INV C, A: $C = c_{n-1}c_{n-2} \dots c_0$ where $c_k = \overline{a_k}$
  - AND C, A, B: $C = c_{n-1}c_{n-2} \dots c_0$ where $c_k = a_k \cdot b_k$
  - OR C, A, B: $C = c_{n-1}c_{n-2} \dots c_0$ where $c_k = a_k + b_k$
  - EOR C, A, B: $C = c_{n-1}c_{n-2} \dots c_0$ where $c_k = a_k \oplus b_k$ (exclusive OR)

- Logical shift operations
  - LSL C, A, #m: Shift each bit in A to the left by m bits
  - LSR C, A, #m: Shift each bit in A to the right by m bits

Bit Masking

- Assume all the registers are 8-bit.

- Masking to 1
  - OR C, A, 0x01: $C = a_7 a_6 \ldots a_1 1$

- Masking to 0
  - AND C, A, 0xFE: $C = a_7 a_6 \ldots a_1 0$

- Inversion (if INV is not supported)
  - EOR C, A, 0x01: $C = a_7 a_6 \ldots a_1 \overline{a_0}$

Example

- Suppose $X = x_7 x_6 \dots x_0$. The meanings of the bits are as follows:
  - x_7: 0 - No overflow, 1 - Overflow
  - x_6: 0 - Positive, 1 - Negative
  - x_5: 0 - Under normal operations, 1 - Under testing
  - x_4: 0 - No interrupt, 1 - Interrupt
  - ...

- If you want to check whether there is an interrupt,
  - AND C, X, #0x10
  - Then, compare C with #16

Example

- Packing two 8-bit values in a 16-bit register
  - $A = XXXX\ XXXX\ a_7a_6a_5a_4\ a_3a_2a_1a_0$
  - $B = XXXX\ XXXX\ b_7b_6b_5b_4\ b_3b_2b_1b_0$
  - Generate $C = b_7b_6b_5b_4\ b_3b_2b_1b_0\ a_7a_6a_5a_4\ a_3a_2a_1a_0$

Example

- Packing two 8-bit values in a 16-bit register
  - $A = XXXX\ XXXX\ a_7a_6a_5a_4\ a_3a_2a_1a_0$
  - $B = XXXX\ XXXX\ b_7b_6b_5b_4\ b_3b_2b_1b_0$
  - Generate $C = b_7b_6b_5b_4\ b_3b_2b_1b_0\ a_7a_6a_5a_4\ a_3a_2a_1a_0$

  - LSL C, B, #8
  - AND D, A, #0x00FF
  - OR C, C, D

Numerical Operations by Logical Operations

- MUL C, A, #2 ($C = A * 2$)
  - LSL C, A, #1 (shift left by 1 bit)

- DIV C, A, #2 ($C = \frac{A}{2}$)
  - LSR C, A, #1 (shift right by 1 bit)

- Rationale
  - Numerical operations such as multiplication and division take many clock cycles.
  - Logical operations generally take one clock cycle.

- How to interpret given operands and the result of an operation on the operands depends on the programmer.

Example

- What is the numerical meaning of "AND C, A, 0xF0" if C and A have unsigned binary numbers?

Example

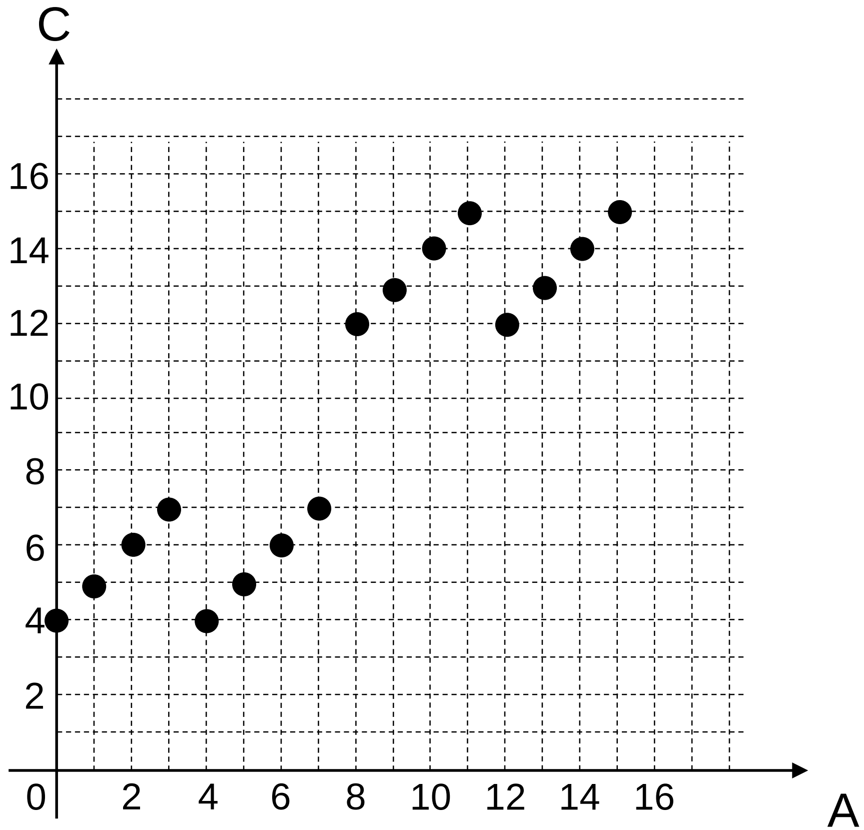
- What is the numerical meaning of “AND C, A, 0xF0” if C and A have unsigned binary numbers?
  - $C = a_7 a_6 a_5 a_4\,0000$
  - Let’s list some inputs and their outputs.
    - 0000 0000 (0) → 0000 0000 (0)
    - 0000 0001 (1) → 0000 0000 (0)
    - ...
    - 0000 1111 (15) → 0000 0000 (0)
    - 0001 0000 (16) → 0001 0000 (16)
    - 0001 0001 (17) → 0001 0000 (16)
    - ...
  - Thus, it implements the following function.
    - $C = \left[\frac{A}{16}\right] * 16$

Example

- What is the numerical meaning of “OR C, A, 0x04” if C and A have unsigned binary numbers?

Example

- What is the numerical meaning of "OR C, A, 0x04" if C and A have unsigned binary numbers?
  - $C = a_7 a_6 a_5 a_4\ a_3 1 a_1 a_0$
  - This means that if $a_2 = 0$, it is set to 1.
  - Let's list some inputs and their outputs.
    - 0000 0000 (0) → 0000 0100 (4)
    - 0000 0001 (1) → 0000 0101 (5)
    - 0000 0010 (2) → 0000 0110 (6)
    - 0000 0011 (3) → 0000 0111 (7)
    - 0000 0100 (4) → 0000 0100 (4)
    - 0000 0101 (5) → 0000 0101 (5)
    - 0000 0110 (6) → 0000 0110 (6)
    - 0000 0111 (7) → 0000 0111 (7)
    - 0000 1000 (8) → 0000 1100 (12) = 0000 1000 (8) + 0000 0101 (4)
    - 0000 1001 (9) → 0000 1101 (13) = 0000 1000 (8) + 0000 0101 (5)
    - ...
  - Thus, it implements the following function.


C
16
14
12
10
8
6
4
2
0
2
4
6
8
10
12
14
16
A