Bitwise Instructions

CSE 30: Computer Organization and Systems Programming

Diba Mirza
Dept. of Computer Science and Engineering
University of California, San Diego
### Bit Wise Operators

- **Basic bitwise operator:**
  - **AND**
  - **OR**
  - **XOR**
  - **BIC (Bit Clear)**
  - **NOT**

#### Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A AND B</th>
<th>A OR B</th>
<th>NOT A</th>
<th>A XOR B</th>
<th>A BIC B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Boolean expression:**

\[
A \cdot \overline{B} + \overline{A} \cdot B
\]

\[
\text{inverting} \quad A
\]

\[
(A \text{ AND (NOT} B) \text{) OR ((NOT} A) \text{ AND B)}
\]
## Syntax in ARM and C

<table>
<thead>
<tr>
<th></th>
<th>In ARM</th>
<th>In C</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>AND r0, r1, r2</td>
<td>r0 = r1 &amp; r2;</td>
</tr>
<tr>
<td>OR</td>
<td>ORR r0, r1, r2</td>
<td>r0 = r1</td>
</tr>
<tr>
<td>XOR</td>
<td>EOR r0, r1, r2</td>
<td>r0 = r1 ^ r2;</td>
</tr>
<tr>
<td>BIC</td>
<td>BIC r0, r1, r2</td>
<td>r0 = r1 &amp; (~r2);</td>
</tr>
<tr>
<td>NOT</td>
<td>MVN r0, r1</td>
<td>r0 = ~r1;</td>
</tr>
</tbody>
</table>

Different ways to specify
What is the value in r0 after the above instruction is executed?

A. 1
B. 0
C. -1
D. None of the above
Assignment Instructions

- **MVN – Move Negative** – moves one complement of the operand into the register.

- **Assignment in Assembly**
  - Example: \( \text{MVN r0, } \#0 \) (in ARM)
  - Equivalent to: \( a = -1 \) (in C)

  where ARM registers \( r0 \) are associated with C variables \( a \)

  Since \( \sim 0x00000000 == 0xFFFFFFFF \)
Bit wise operations

A. ORR r3, r0, r1; 11101111
B. AND r3, r0, r1; 01000001
C. EOR r3, r0, r1; 10101110
D. BIC r3, r0, r1; 01101000

r0: 01101001
r1: 11000111

Which of the given results is incorrect?
Uses for Logical Operators

- Note that **AND**ing a bit with 0 produces a 0 at the output while **AND**ing a bit with 1 produces the original bit.

- This can be used to create a **mask**.

- **Example:**
  
  \[
  \begin{align*}
  \sigma_0 & : 1011 0110 1010 0100 0011 1101 1001 1010 \\
  \text{mask} & : 0000 0000 0000 0000 0000 1111 1111 1111 \\
  \sigma_3 & : 0000 0000 0000 0000 0000 1101 1001 1010 
  \end{align*}
  \]

- The result of **AND**ing these:

  \[
  \begin{align*}
  \text{AND } \sigma_3, \sigma_0, \text{ OxFFFF} \\
  \text{MUN } \sigma_1, \#0
  \end{align*}
  \]

You can generate it!
Uses for Logical Operators

- Similarly, note that ORing a bit with 1 produces a 1 at the output while ORing a bit with 0 produces the original bit.
- This can be used to force certain bits of a number to 1s.
  - For example, \(0x12345678\) OR \(0x0000FFFF\)
    
    Results in: \(0x1234FFFF\)
Uses for Logical Operators

- Finally, note that **BIC**ing a bit with 1 resets the bit (sets to 0) at the output while **BIC**ing a bit with 0 produces the original bit.

- This can be used to force certain bits of a string to 0s.
  - For example, `0x12345678` **BIC** `0x0000FFFF` results in `0x12340000`.

\[ \text{Set LS 2 bytes to zero} \]

\[ \text{Too Big} \]

\[ \text{Set least significant byte} \]

\[ \text{Just 8 bits} \]

\[ \text{Too big} \]
Invert bits 0–2 of r0

A. AND r0, r0, #7
   : Getting last 3 bits

B. ORR r0, r0, #7
   : Set last 3 bits to 2

C. MVN r0, #7
   : Generated a large constant 0xfffffffff8

D. EOR r0, r0, #7
   : Inverted the last 3 bits
   : Cleared the last 3 bits
   (Set the last 3 bits to 0)

E. BIC r0, r0, #7

\[
\begin{align*}
\text{r0} & : 101 \\
\text{210} & : \text{Unchanged} \\
\text{111} & : \text{Unchanged} \\
\text{010} & : \text{Unchanged}
\end{align*}
\]

\[
\begin{align*}
\text{AND} & : 101 \\
\text{OR} & : 111
\end{align*}
\]

\[
\begin{align*}
\text{#7 0's} & : 0111 \\
\text{1's} & : 1000
\end{align*}
\]

\[
\begin{align*}
\text{FFFFFFFFFFF8} & : \text{After flipping the bits of 7}
\end{align*}
\]