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>
</tr>
</thead>
</table>
## 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></td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assignment Instructions

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

Assignment in Assembly

- Example: `MVN r0, #0` (in ARM)
  
  Equivalent to: `a = -1` (in C)

  where ARM registers `r0` are associated with C variables `a`

  Since \( \sim0x00000000 == 0xFFFFFFFF \)
Bit wise operations

r0:  01101001
r1:  11000111

Which of the given results is incorrect?

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

- Note that ANDing a bit with 0 produces a 0 at the output while ANDing a bit with 1 produces the original bit.
- This can be used to create a mask.
  - Example:
    
    | 1011 0110 1010 0100 0011 1101 1001 1010 |
    |-----------------------------------------|
    | mask: 0000 0000 0000 0000 0000 1111 1111 1111 |
    | The result of ANDing these: |
    | 0000 0000 0000 0000 0000 1101 1001 1010 |
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 string to 1s.
  - For example, 0x12345678
    OR 0x0000FFFF
  
  Results in:
Uses for Logical Operators

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

- This can be used to force certain bits of a string to 0s.
  - For example, 0x12345678
  
  \[
  \text{BIC } 0x0000FFFF
  \]

  results in
Invert bits 0–2 of r0

A. AND r0, r0, #7
B. ORR r0, r0, #7
C. MVN r0, #7
D. EOR r0, r0, #7
Perform 1's complement on r0

A. EOR r0, r0, #0

B. MVN r0, r0

C. EOR r0, r0, 0xFFFFFFFF

D. BIC r0, r0, 0xFFFFFFFF
Shifts and Rotates

- **LSL** – logical shift by n bits – multiplication by \(2^n\)
  
  ![Diagram of LSL]

- **LSR** – logical shift by n bits – unsigned division by \(2^n\)
  
  ![Diagram of LSR]

- **ASR** – arithmetic shift by n bits – signed division by \(2^n\)
  
  ![Diagram of ASR]

- **ROR** – logical rotate by n bits – 32 bit rotate
  
  ![Diagram of ROR]
## Syntax in ARM and C

<table>
<thead>
<tr>
<th></th>
<th>In ARM</th>
<th>In C</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Addressing modes

- ADD r0, r0, r1
1. Register, optionally with shift operation
   - Shift value can either be:
     - 5 bit unsigned integer
     - Specified in bottom byte of another register.
   - Used for multiplication by constant

2. Immediate value
   - 8 bit number, with a range of 0-255.
     - Rotated right through even number of positions
   - Allows increased range of 32-bit constants to be loaded directly into registers
Conclusion

- Instructions so far:
  - Previously:
    ADD, SUB, MUL, MLA, [U|S]MULL, [U|S]MLAL
  - New instructions:
    RSB
    AND, ORR, EOR, BIC
    MOV, MVN
    LSL, LSR, ASR, ROR

- Shifting can only be done on the second source operand
- Constant multiplications possible using shifts and addition/subtractions