ARM Logic Instructions

- **AND** \( \text{dst} = \text{src1} \text{ AND} \text{src2} \)
  \( \text{src1} \& \text{src2} \)
- **EOR** \( \text{dst} = \text{src1} \text{ EOR} \text{src2} \)
  \( \text{src1} \^ \text{src2} \)
- **ORR** \( \text{dst} = \text{src1} \text{ OR} \text{src2} \)
  \( \text{src1} | \text{src2} \)
- **MVN** \( \text{dst} = \text{NOT} \text{src2} \)
  \( \sim\text{src2} \)
- **ORN** \( \text{dst} = \text{src1} \text{ NOR} \text{src2} \)
  \( \sim(\text{src1} | \text{src2}) \)
- **BIC** \( \text{dst} = \text{src1} \text{ AND} \text{NOT} \text{src2} \)
  \( (\text{src1} & \sim\text{src2}) \)

**Examples**

- **ORR** \( R0, R1, R2 \) ; \( R0 = R1 \text{ OR} R2 \)
- **AND** \( R0, R0, \#0x0F \) ; \( R0 = R0 \text{ AND} 0x0F \)
- **EORS** \( R1, R3, R0 \) ; \( R1 = R3 \text{ EOR} R0 \text{ + set condition code flags} \)
AND

- $\text{dst} = \text{src1} \& \text{src2}$

- each bit in $\text{dst}$ is the AND of the corresponding bits in $\text{src1}$ and $\text{src2}$ (see truth table)

- can be used to clear selected bits

  MOV  R0, #0xAA
  AND  R0, R0, #0x0F

<table>
<thead>
<tr>
<th>src1</th>
<th>src2</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

  clears most significant bits

- if $\text{src2}$ used as a mask, clears bit if corresponding bit in mask is 0
OR

- \( \text{dst} = \text{src1} \mid \text{src2} \)

- each bit in \( \text{dst} \) is the OR of the corresponding bits in \( \text{src1} \) and \( \text{src2} \) (see truth table)

- can be used to set selected bits

### Truth Table

<table>
<thead>
<tr>
<th>src1</th>
<th>src2</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- if \( \text{src2} \) used as a mask, sets bit if corresponding bit in mask is 1

```
MOV R0, #0x0A
ORR R0, R0, #0xF0
```

<table>
<thead>
<tr>
<th>0x0A</th>
<th>0000 1010</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xF0</td>
<td>1111 0000</td>
</tr>
<tr>
<td>0xFA</td>
<td>1111 1010</td>
</tr>
</tbody>
</table>

sets most significant bits
EOR / XOR

- $\text{dst} = \text{src1} \oplus \text{src2}$

- each bit in $\text{dst}$ is the EOR of the corresponding bits in src1 and src2 (see truth table)

- can be used to invert selected bits

MOV R0, #0x0A
EOR R0, R0, #0x0F

<table>
<thead>
<tr>
<th>src1</th>
<th>src2</th>
<th>EOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Truth table

- if src2 used as a mask, inverts bit if corresponding bit in mask is 1
MVN (Move NOT)

- \( \text{dst} = \sim \text{src2} \)

- each bit in \( \text{dst} \) is the inverse (\( \sim \) or NOT) of the corresponding bit in \( \text{src2} \) (see truth table)

- can be used to invert ALL bits

<table>
<thead>
<tr>
<th>src2</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOV</th>
<th>R0, #0x0A</th>
<th>~</th>
<th>0x0000000A</th>
<th>0101</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVN</td>
<td>R0, R0</td>
<td></td>
<td>0xFFFFFFF5</td>
<td>0101</td>
</tr>
</tbody>
</table>

\[ \text{inverts ALL bits} \]
ORN (NOR)

- \( \text{dst} = \sim (\text{src1} \mid \text{src2}) \)

- each bit in \( \text{dst} \) is the NOR of the corresponding bits in \( \text{src1} \) and \( \text{src2} \) (see truth table)

MOV R0, #0x0A
ORN R0, R0, #0x0F
BIC (bit clear)

- $dst = src1 \& \sim src2$
- each bit in $dst$ is set to $src1 \& \sim src2$ using the corresponding bits in $src1$ and $src2$ (see truth table)
- can be used to clear selected bits

<table>
<thead>
<tr>
<th>src1</th>
<th>src2</th>
<th>BIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

truth table

MOV R0, #0xAA
BIC R0, R0, #0xF0

<table>
<thead>
<tr>
<th>0xAA</th>
<th>1010 1010</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIC</td>
<td>0xF0</td>
</tr>
<tr>
<td></td>
<td>1111 0000</td>
</tr>
<tr>
<td></td>
<td>0x0A</td>
</tr>
<tr>
<td></td>
<td>0000 1010</td>
</tr>
</tbody>
</table>

clear selected bits

- if $src2$ used as a mask, clears bit if corresponding bit in mask is 1
How to Clear Bits

• write ARM instructions to clear bits 3 and 4 of R1 (LS bit is bit 0)

\[
\begin{align*}
\text{LDR } & \text{R1, } =0\times12345678 \quad ; \text{load test value} \\
\text{LDR } & \text{R2, } =0\timesFFFFFFE7 \quad ; \text{AND mask to clear bits} \\
\text{AND } & \text{R1, R1, R2} \quad ; \text{R1} = 0\times12345660
\end{align*}
\]

<table>
<thead>
<tr>
<th>(0x\ldots78)</th>
<th>0111 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp; (0x\ldotsE7)</td>
<td>1110 0111</td>
</tr>
<tr>
<td>(0x\ldots60)</td>
<td>0110 0000</td>
</tr>
</tbody>
</table>

• alternatively, the BIC (Bit Clear) instruction can be used with a mask of 1’s in the bit positions that need to be cleared

\[
\begin{align*}
\text{LDR } & \text{R1, } =0\times12345678 \quad ; \text{load test value} \\
\text{LDR } & \text{R2, } =0\times00000018 \quad ; \text{AND mask to clear bits 3 and 4} \\
\text{BIC } & \text{R1, R1, R2} \quad ; \text{R1} = 0\times12345660
\end{align*}
\]

• in this case, can use an immediate mask saving one instruction

\[
\begin{align*}
\text{LDR } & \text{R1, } =0\times12345678 \quad ; \text{load test value} \\
\text{BIC } & \text{R1, R1, } #0\times18 \quad ; \text{R1} = 0\times12345660
\end{align*}
\]
Logic and Shift Instructions

How to Invert Bits

- write ARM instructions to invert bits 2 .. 5 of R1 (LS bit is bit 0)
- EOR mask = 0x3C (invert bit if corresponding bit in mask is 1)

LDR R1, =0x12345678  ; load test value
LDR R2, =0x0000003C  ; EOR mask to invert bits 2 .. 5
EOR R1, R1, R2       ; R1 = 0x12345644

```
<table>
<thead>
<tr>
<th>0x...78</th>
<th>0111 1000</th>
</tr>
</thead>
</table>
^       | 0x...3C  | 0011 1100 |
        | 0x...44  | 0100 0100 |

inverts bits 2, 3, 4 and 5
```

- in this case, can use an immediate mask saving one instruction

LDR R1, =0x12345678  ; load test value
EOR R1, R1, #0x3C    ; R1 = 0x12345644
ARM Shift and Rotate

- Logical Shift Left (LSL)  
  \[ a << n \]  // logical shift left n places

- Logical Shift Right (LSR)  
  \[ a >> n \]  // logical shift right n places

- Arithmetic Shift Right (ASR)

- Rotate Right (ROR)

- Rotate Right with eXtend (RRX)

- NB: these are NOT instructions in the same sense as ADD, SUB or ORR
Logical Shift Left (LSL)

- LSL one place (LSL #1)
  - 0 shifted into LSB, MSB discarded

  32 bits
  \[
  \begin{array}{c}
  00000001 \  \ 11111111 \  \ 00000000 \  \ 01111111 \\
  \end{array}
  \]

  0x00FF00FF => 0x01FE01FE

- LSL 3 places (LSL #3)

  32 bits
  \[
  \begin{array}{c}
  00000001 \  \ 11111111 \  \ 00000000 \  \ 01111111 \\
  \end{array}
  \]

  0x00FF00FF => 0x07F807F8

- can LSL 0 to 31 places
Logical Shift Right (LSR)

- LSR one place (LSR #1)
  - 0 shifted into MSB, LSB discarded

\[ \text{0x00FF00FF} \Rightarrow \text{0x007F807F} \]

- LSR 3 places (LSR #3)

\[ \text{0x00FF00FF} \Rightarrow \text{0x001FE01F} \]

- can LSR 0 to 31 places
ARM shift instructions

- ARM has NO dedicated shift/rotate instructions
- instead, ALL instructions can optionally shift/rotate the src2 operand before it is used as input for the ALU operation (ADD, SUB, ...)

\[
\text{src2 to ALU can be:}
\]

1) register with an optional shift/rotate
   - shift/rotate by constant number of places OR
   - by the number places specified in a register

2) 8 bit immediate value rotated right by an even number of places

- very ARM specific – unlike other CPUs
Shift using MOV

• ARM assembly language syntax to LSL src2 one place before MOV operation

\[
\text{MOV R1, R0, LSL \#1} \quad ; \quad R1 = R0 \ll 1
\]

• Logical shift left 5 places

\[
\text{MOV R1, R0, LSL \#5} \quad ; \quad R1 = R0 \ll 5
\]

• If no shift specified, the default is LSL \#0

\[
\text{MOV R1, R0, LSL \#0} \quad ; \quad R1 = R0 \text{ (NO shift)}
\]
Logical Shift Left

• LSL one place is the same as multiplying by 2 (if NO carry/overflow)

  LDR  R0, =0xFF           ; R0 = 0x00FF (255)
  MOV  R1, R0, LSL #1      ; R1 = 0x01FE (510)

• LSL n places is the same as multiplying by $2^n$

  LDR  R0, =0xFF           ; R0 = 0x00FF (255)
  MOV  R1, R0, LSL #4      ; R1 = 0x0FF0 (255 x $2^4 = 255 \times 16 = 4080$)

• works for signed and unsigned integers

  LDR  R0, =0xFFFFFFFF     ; R0 = 0xFFFFFFFF (-1)
  MOV  R1, R0, LSL #2      ; R1 = 0xFFFFFFFFC (-4) = R0 x 4
Logical Shift Right ...

• LSR one place is the same as integer division by 2 (if NO carry/overflow)

  LDR  R0, =0xFF ; R0 = 0xFF (255)
  MOV  R1, R0, LSR #1 ; R1 = 0x7F (127)

• LSR n places is the same as integer division by 2^n

  LDR  R0, =0xFF ; R0 = 0xFF (255)
  MOV  R1, R0, LSR #4 ; R1 = 0x0F (255 / 2^4 = 255 / 16 = 15)

• works for unsigned integers

• for signed integers use arithmetic shift right (ASR) which will be covered later
Shift ...

- can shift left or right by the number places specified in a register

```
MOV    R1, R0, LSL R2    ; R1 = R0 << R2
```

- R2 can be a variable rather than a constant

- if `MOVS` is used instead of `MOV`, the last bit shifted out (left or right) is stored in the CARRY flag

```
MOV    R0, #0x55    ; R0 = 0x55
MOVS   R1, R0, LSR #3    ; R1 = R0 >> 3
```

`LSL R2` places (LS 5 bits)

```
0x55 0101 0101
```

CARRY = 1

```
0x0A 0000 1010
```

CARRY = 1
Example Shift Operations

• shifts can be followed by any operation ALU operation

• what do the following instructions do?

  ADD   R0, R1, R1, LSL #3   ; R0 = R1 + R1 x 8 = R1 x 9
  RSB   R0, R5, R5, LSL #3   ; R0 = R5 x 8 - R5 = R5 x 7
  SUB   R0, R9, R8, LSR #4   ; R0 = R9 - R8/16

• write ARM instructions to set bit n of R0 where n is in R1 (n in range 0 .. 31)

  MOV   R1, #4               ; R1 = 4
  MOV   R2, #0x01            ; R2 = 1
  ORR   R0, R0, R2, LSL R1   ; R0 = R0 | R2 << n
Example Shift Operations...

- write ARM instructions to set \( R0 = n^{th} \) bit of \( R2 \) where \( n \) is in \( R1 \)

- example: if \( R2 = 0x55 \) and \( n \) is 4 then \( R0 = 1 \)

\[
\begin{align*}
\text{MOV} & \quad R0, \#1 \quad ; \quad R0 = 1 \\
\text{AND} & \quad R0, R0, R2, \text{LSR R1} \quad ; \quad R0 = R0 \& R2 >> R1
\end{align*}
\]

\( R0 = 0x01 \)
\( R2 = 0x55 \)
\( \text{barrel shifter} \)
\( \text{LSR R1} \ (n = 0x04) \)
\( R2 >> R1 = 0x05 \)

\( \text{ALU = AND} \)
\( R0 = R0 \& R2 >> R1 = 0x01 \& 0x05 = 1 \)
REMEMBER

• prepare for Mid-Term Test during Study Week

• ALL students Thurs 1\textsuperscript{st} Nov @ 9am in Goldsmith Hall (instead of Tutorial)