ARM Shifts and rotates
Addressing modes

CSE 30: Computer Organization and Systems Programming

Diba Mirza
Dept. of Computer Science and Engineering
University of California, San Diego
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

\[ \text{MOV r1, } \#-1 \]
\[ \text{EOR r0, r0, r1} \]
Shifts and Rotates

- **LSL** – logical shift by \( n \) bits – multiplication by \( 2^n \)

\[
\text{MOV } r3, \#2 \\
\text{LSL } r1, r0, r3 \\
\text{LSL } r1, r2, \#n \]

\[ r1 = 2^n \times r2 \]
Bit shifts and multiplication

LSL r0, r0,#4 is equivalent to which of the following expressions?

A. \( r0 = 4 \times r0 \)
B. \( r0 = 8 \times r0 \)
C. \( r0 = 16 \times r0 \)
D. \( r0 = r0/4 \)
Shifts and Rotates

- **LSR** – logical shift by \( n \) bits – unsigned division by \( 2^n \)

\[
\begin{array}{c}
0 \quad \rightarrow \quad 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\
\end{array}
\]

\[\text{LSR} \; x_1, \; x_0, \; \#2\]

\[\text{LSR} \; x_1, \; x_0, \; \#n\; ; \; \; \; r_1 = x_0 / 2^n \; \downarrow \; \text{unsigned} \]

4 \( \rightarrow \) 1 / 2^2

1 \( \downarrow \)
Shifts and Rotates

- ASR – arithmetic shift by n bits – signed division by $2^n$

```
-2
```

```
0 1 1 1 1 1 1 0
```

```
ASR r1, r0, #1
```

```
ASR r1, r0, #n
```

```
r1 = r0 / 2^n (signed)
```
Bit shifts and division

What is the result in r0 after the following code is executed

MOV r0, #%-8
ASR r0, r0, #2

A. 0x4
B. 0xFC
C. 0xFFFFFFFFFC
D. 0xFFFFFFFFFE
**Shifts and Rotates**

- **ROR** – logical rotate by n bits – 32 bit rotate

![Diagram showing ROR operation]
## 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><code>LSL r0, r1, #n</code> shift amount</td>
<td><code>r0 = r1 &lt;&lt; n;</code></td>
</tr>
<tr>
<td>LSR</td>
<td><code>LSR r0, r1, #n</code></td>
<td><code>r0 = (unsigned) r1 &gt;&gt; n;</code></td>
</tr>
<tr>
<td>ASR</td>
<td><code>ASR r0, r1, #n</code></td>
<td><code>r0 = (unsigned) r1 &gt;&gt; n;</code></td>
</tr>
<tr>
<td>ROR</td>
<td><code>ROR r0, r1, #n</code></td>
<td><code>r0 = (unsigned) r1 &gt;&gt; (32-n);</code></td>
</tr>
</tbody>
</table>

```c
int r1 = -8;
```
Rotates in C

Which of the following C statements is equivalent to the given ARM statement

ROR r0, r0, #24  @Assume ‘int x’ is mapped to r0

A. x>>24

B. x>>24 | ((unsigned) x<<8)

C. (unsigned)x>>24 | x<<8

D. (unsigned)x>>8 | x<<24
Addressing modes

- Register direct: all operands are registers

  ADD  r0, r1, r2

  Cannot be an immediate
  5 bits for the shift amount

  with shifts

  ADD  r0, r1, r2

  LSL  #2

  @ r0 = r1 + (r2 << 2)

  @ r0 = r1 + 4*r2

  LDR  r0, [r1]

  usual way

  Men[r1 + r2] = r0

  LDR  r0, [r1, r2, LSL #2]

  Men[r1 + 4*r2]
Addressing modes

- Register direct: all operands are registers

\[
\begin{align*}
\text{ADD} & \quad r_0, r_1, r_2 @ \quad r_0 = r_1 + r_2 \\
\text{ADD} & \quad r_0, r_1, \#2 @ \quad r_0 = r_1 + 4 \times r_2 \\
\text{ADD} & \quad r_0, r_1, \text{LSL} r_2 \quad @ \quad r_0 = r_1 + (r_2 >> 2)
\end{align*}
\]
Addressing modes

Formats for LDR/STR:

1. \texttt{LDR \ r0, [r1] @ yo2Mem [r1]}
2. \texttt{LDR \ r0, [r1, #y] @ yo2Mem [r1+y]}
3. \texttt{LDR \ r0, [r1, r2] @ yo2Mem [r1+r2]}
4. \texttt{LDR \ r0, [r1, r2, LSL #n] @ yo2Mem [r1+r2x2]}

In a int array, base address, index

\texttt{LDR \ r0, [r1, r2, LSL #2]}
Immediate Addressing

ADD r0, r1, 0xff  # 8 bit immediate

ADD r0, r1, 0xff [0000 0000] # even & 5 bits
The data processing instruction format has 12 bits available for operand2

```
<table>
<thead>
<tr>
<th>rotation (rot)</th>
<th>immediate (immed_8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
```

- 4 bit rotate value (0-15) is multiplied by two to give range 0-30 in steps of 2
- Rule to remember is “8-bits rotated right by an even number of bit positions”

```
0xFF000000
MOV r0, #0xFF, 8
Immed_8=0xFF, rot =4
```
Conclusion

- Instructions so far:
  - Previously:
    ADD, SUB, MUL,
  - New instructions:
    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