**Overview**

- **Shift Operations**
  - Field Insertion

- **Multiplication Operations**
  - Multiplication
  - Long Multiplication
  - Multiplication and accumulation
  - Signed and unsigned multiplications

---

**Review: ARM Instructions So far**

- add
- sub
- mov
- and
- bic
- orr
- eor
- Data Processing Instructions with shift and rotate
  - lsl, lsr, asr, ror

---

**Review: Masking via Logical AND**

- AND: 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:
    
    $\begin{align*}
    1011 & 0110 & 1010 & 0100 & 0011 & 1101 & 1001 & 1010 \\
    0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 1111 & 1111 \\
    \end{align*}$

  - Mask: $\begin{align*}
    0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 1111 & 1111 \\
    \end{align*}$

  - The result of anding these two is:
    
    $\begin{align*}
    0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 0000 & 1001 & 1010 \\
    \end{align*}$
**Review: Masking via Logical BIC**

- **BIC (AND NOT):** Note that BIC-ing a bit with 1 produces a 0 at the output while BIC-ing a bit with 0 produces the original bit.

- This can be used to create a mask.
  - Example:
    
    | Mask: 0000 0000 0000 0000 0000 0000 1111 1111 |
    |---|---|---|---|---|---|---|---|
    | 1011 0110 1010 0100 0011 1101 1001 1010 |

  - The result of BIC-ing these two is:
    | Mask: 0000 0000 0000 0000 0000 0000 0000 0000 |
    |---|---|---|---|---|---|---|---|
    | 1011 0110 1010 0100 0011 1101 1001 1010 |

**Extracting a field of bits (#1/2)**

- Extract bit field from bit 9 (left bit no.) to bit 2 (size=8 bits) of register v1, place in rightmost part of register a1

<table>
<thead>
<tr>
<th>31</th>
<th>9876543210</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>000000000000000000000000</td>
</tr>
<tr>
<td>al</td>
<td>000000000000000000000000</td>
</tr>
</tbody>
</table>

- Shift field as far left as possible (9 → 31) and then as far right as possible (31 → 7)

<table>
<thead>
<tr>
<th>31</th>
<th>9876543210</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>000000000000000000000000</td>
</tr>
<tr>
<td>al</td>
<td>000000000000000000000000</td>
</tr>
</tbody>
</table>

**Extracting a field of bits (#2/2)**

- Move field left 2 bits, mask out field, OR in field

| mov a1, v1, lsl #2 ; 8 bits to left end (31-9) |
|---|---|
| | 000000000000000000000000 |
| | v1 |
| | 000000000000000000000000 |
| | al |

| mov a1, al, lsr #24 ; 8 bits to right end (7-0) |
|---|---|
| | 000000000000000000000000 |
| | al |
| | 000000000000000000000000 |
| | a1 |

**Inserting a field of bits**

- Insert bit field into bit 9 (left bit no.) to bit 2 (size=8 bits) of register a1 from rightmost part of register v1 (rest is 0)

| mov a2, v1, lsl #2 ; field left 2 |
|---|---|
| bic a1, a1, #0x3FC ; mask out 9-2 |
| orr a1, a1, a2 ; OR in field |
| bic stands for ‘bit clear’, where ‘1’ in the second operand clears the corresponding bit in the first |
Bit manipulation in C (#1/2)

° Convert C code to ARM ASM

° Bit Fields in C (Word as 32 bits vs int/unsigned!)

```
struct {
  unsigned int ready: 1; /* bit 0 */
  unsigned int enable: 1; /* bit 1 */
} rec;
rec.enable = 1;
rec.ready = 0;
printf("%d %d", rec.enable, rec.ready);
```

Brian Kernighan & Dennis Ritchie:  
The C Programming Language, 2nd Ed., PP 150

Bit manipulation in C (#2/2)

```
struct {
  unsigned int ready: 1; /* bit 0 */
  unsigned int enable: 1; /* bit 1 */
} rec; /* v1 */
rec.enable = 1;
rec.ready = 0;
printf("%d %d", rec.enable, rec.ready);
orr v1,v1, #0x2           ;1 in bit 1
bic v1,v1  #1             ;0 in bit 0,
ldr a1, =LCO              ;printf format
mov a2, v1, lsr #1       ;just bit 1
and a2, a2,0x0001       ;mask down to 1
and a3, v1, 0x0001     ;just bit 0
bl  printf             ;call printf
```

Multiply by Power of 2 via Shift Left (#1/3)

° In decimal:
  • Multiplying by 10 is same as shifting left by 1:
    - 71410 x 1010 = 714010
    - 5610 x 1010 = 56010
  • Multiplying by 100 is same as shifting left by 2:
    - 71410 x 10010 = 7140010
    - 5610 x 10010 = 560010
  • Multiplying by 10^n is same as shifting left by n

Multiply by Power of 2 via Shift Left (#2/3)

° In binary:
  • Multiplying by 2 is same as shifting left by 1:
    - 112 x 102 = 1102
    - 10102 x 102 = 101002
  • Multiplying by 4 is same as shifting left by 2:
    - 112 x 1002 = 11002
    - 10102 x 1002 = 1010002
  • Multiplying by 2^n is same as shifting left by n
**Multiply by Power of 2 via Shift Left (#3/3)**

- Since shifting is so much faster than multiplication (you can imagine how complicated multiplication is), a good compiler usually notices when C code multiplies by a power of 2 and compiles it to a shift instruction:

  ```c
  a *= 8; // in C
  ```

  would compile to:

  ```arm
  mov a0,a0,lsl #3 // in ARM
  ```

**Shift, Add and Subtract for Multiplication**

<table>
<thead>
<tr>
<th>Add and Subtract Examples:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f = 5g ) /* ( f = (4+1) \times g ) */ (in C)</td>
</tr>
</tbody>
</table>
| \( f = 105 \times g \) /* \( f = (15 \times 7) \times g \) */ (in C) | \( f = (16-1) \times (8-1) \times g \)
| \( f = 16 \times 7 \times g \) | rsb \( v1,v2,v2 \ lsl \ #4 \); \( v1 = -v2 + v2 \times 16 \) (in ARM)
| \( f = (16-1) \times g \) | \( f = (8-1) \times f \) |

**Shift, Add and Subtract for Division**

- ARM does not have division.
- Division \( A/B \) produces a quotient and a remainder.
- It should be done via sequence of subtraction and shifting (See Experiment 3)
- For \( B \) in \( A/B \) a constant value (eg 10) simpler technique via Shift, Add and Subtract is available (Will be discussed later)

**Shift Right Arithmetic; Divide by 2??**

- Shifting left by \( n \) is same as Multiplying by \( 2^n \)
- Shifting right by \( n \) bits would seem to be the same as dividing by \( 2^n \)
- Problem is signed integers
  - Zero fill is wrong for negative numbers
- Shift Right Arithmetic (\textit{asr}); sign extends (replicaes sign bit):
  - \( \text{1111 1111 1111 1000} = -8 \)
  - \( \text{1111 1111 1111 1100} = -4 \)
  - \( \text{1111 1111 1111 1110} = -2 \)
  - \( \text{1111 1111 1111 1111} = -1 \)
Is asr really divide by 2?

- Divide +5 by 4 via asr 2; result should be 1
  - 0000 0000 0000 0000 0000 0000 0000 0101
  - = +1, so does work
- Divide -5 by 4 via asr 2; result should be -1
  - 1111 1111 1111 1111 1111 1111 1111 1011
  - = -2, not -1; Off by 1, so doesn’t always work
- Rounds to \(-\infty\)

MULTIPLY (unsigned): Terms, Example

- Paper and pencil example (unsigned):
  - Multiplicand 1000
  - Multiplier 1001
  - Product 01001000

- Solution 1:
  - 2's complement range for 4 bit number is \(-8 \leq a \leq 7 \Rightarrow -56 \leq a \times b \leq 64.\)
  - Can build a big lookup table (16 x 16 = 256) to map from binary bit patterns to multiplication results.
  - \(-8 \times -8 = 01000000 \Leftrightarrow 64\)
  - \(-8 \times -7 = 01001000 \Leftrightarrow 56\)
  - \(-2 \times -2 = 11000100 \Leftrightarrow 04\)
  - \(-1 \times -1 = 11100001 \Leftrightarrow 01\)
  - \(+7 \times +7 = 00110001 \Leftrightarrow 49\)

MULTIPLY (signed)? #(1/3)

- Paper and pencil example
  - Multiplicand 1000       -8
  - Multiplier 1001       -7
  - Product 01001000  72 \neq 56

- Solution 1:
  - 2’s complement range for 4 bit number is \(-8 \leq a \leq 7 \Rightarrow -56 \leq a \times b \leq 64.\)
  - Can build a big lookup table (16 x 16 = 256) to map from binary bit patterns to multiplication results.
  - \(-8 \times -8 = 01000000 \Leftrightarrow 64\)
  - \(-8 \times -7 = 01001000 \Leftrightarrow 56\)
  - \(-2 \times -2 = 11000100 \Leftrightarrow 04\)
  - \(-1 \times -1 = 11100001 \Leftrightarrow 01\)
  - \(+7 \times +7 = 00110001 \Leftrightarrow 49\)

MULTIPLY (signed)? #(2/3)

- Solution 2:
  - Multiplicand 1001       -7
  - Multiplier 1000       -8
  - Product 00111000  56 = 56

- Solution 2:
  - Multiplicand 1001       -7
  - Multiplier 1000       -8
  - Product 00111000  56 = 56
MULTIPLY (signed)? #(3/3)

Solution 2 (Another example):

<table>
<thead>
<tr>
<th>Multiplicand</th>
<th>1001</th>
<th>-7 x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier</td>
<td>1001</td>
<td>-7</td>
</tr>
<tr>
<td></td>
<td>1111001</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>0000000</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>000000</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>11001</td>
<td>-</td>
</tr>
<tr>
<td>Product</td>
<td>00110001</td>
<td>49</td>
</tr>
</tbody>
</table>

Multiplication Instructions

° The Basic ARM provides two multiplication instructions.

° Multiply
  • `mul Rd, Rm, Rs ; Rd = Rm * Rs`

° Multiply Accumulate - does addition for free
  • `mla Rd, Rm, Rs, Rn ; Rd = (Rm * Rs) + Rn`

° (Lower precision multiply instructions simply throws top 32 bits away)

° Restrictions on use:
  • Rd and Rm cannot be the same register
  - Can be avoided by swapping Rm and Rs around. This works because multiplication is commutative.
  • Cannot use PC.

These will be picked up by the assembler if overlooked.

° Operands can be considered signed or unsigned
  • Up to user to interpret correctly.

Multiplication Example

° Example:
  • in C: `a = b * c;`
  • in ARM:
    - let b be v1; let c be v2; and let a be v3 (It may be up to 64 bits)
    `mul v3, v2, v1 ; a = b * c`
    ; lower half of product into ; v3. Upper half is thrown up

° Note: Often, we only care about the lower half of the product.

Multiplication and Accumulate Example

° One example of use of `mla` is for string to number conversion: eg

Convert string="123" to value=123

1. `value = 0`
2. `loop = 0`
3. `len = length of string`
4. `Rd = value`
5. `while loop <> len`
   - `c = extract(string, len - loop, 1)`
   - `Rm = 10 ^ loop`
   - `Rs = ASC(c) - ASC (‘0’)`
   - `mla Rd, Rm, Rs, Rd`
   - `loop = loop + 1`
6. `endwhile`
Multiply-Long and Multiply-Accumulate Long

° Instructions are
  • MULL which gives RdHi,RdLo:=Rm*Rs
  • MLAL which gives RdHi,RdLo:=(Rm*Rs)+RdHi,RdLo

° However the full 64 bit of the result now matter (lower precision multiply instructions simply throws top 32 bits away)
  • Need to specify whether operands are signed or unsigned

° Therefore syntax of new instructions are:
  • umull RdLo,RdHi,Rm,Rs    ;RdHi,RdLo:=Rm*Rs
  • umlal RdLo,RdHi,Rm,Rs    ;RdHi,RdLo:=(Rm*Rs)+RdHi,RdLo
  • smull RdLo, RdHi, Rm, Rs ;RdHi,RdLo:=Rm*Rs (Signed)
  • smlal RdLo, RdHi, Rm, Rs ;RdHi,RdLo:=(Rm*Rs)+RdHi,RdLo (Signed)

° Not generated by some compilers. (Needs Hand coding)

Quiz

1. Specify instructions which will implement the following:
   a) \( a_1 = 16 \)
   b) \( a_2 = a_1 \times 4 \)
   c) \( a_1 = a_2 / 16 \) (r1 signed 2’s comp.)
   d) \( a_2 = a_3 \times 7 \)

2. What will the following instructions do?
   a) add \( a_1, a_2, a_2, lsl \#2 \)
   b) rsb \( a_3, a_2, \#0 \)

3. What does the following instruction sequence do?
   add \( a_1, a_2, a_2, lsl \#1 \)
   sub \( a_1, a_1, a_2, lsl \#4 \)
   add \( a_1, a_1, a_2, lsl \#7 \)

Quiz Solution (#1/2)

1. Specify instructions which will implement the following:
   a) \( a_1 = 16 \) \quad \text{mov} \ a_1, \ #16
   b) \( a_2 = a_1 \times 4 \) \quad \text{mov} \ a_2, \ a_1, \ lsl \#2
   c) \( a_1 = a_2 / 16 \) (r1 signed 2’s comp.) \quad \text{mov} \ a_1, \ a_2, \ asr \#4
   d) \( a_2 = a_3 \times 7 \) \quad \text{rsb} \ a_2, \ a_3, \ a_3, \ lsl \#3
       \quad \ a_2 = a_3^{*}(8-1)

       whereas \text{sub} \ a_2, \ a_3, \ a_3, \ lsl \#3 \ would \ give \ a_3^{*-7}

2. What will the following instructions do?
   a) add \( a_1, a_2, a_2, lsl \#2 \)
   \quad \text{a1= a2 + (a2 * 4) ie a1:=a2*5}
   b) rsb \ a_3, \ a_2, \ \#0
   \quad \text{r2=0-r1 ie r2:= -r1}

Division

° No Division Instruction in ARM

° Division has two be done in software through a sequence of shift/ subtract / add instruction.
  • General A/B implementation (See Experiment 3)
  • For B in A/B a constant value (eg 10) simpler technique via Shift, Add and Subtract is available (Will be discussed later)
Quiz Solution (#2/2)

3. What does the following instruction sequence do?
   add a1, a2, a2, lsl #1
   sub a1, a1, a2, lsl #4
   add a1, a1, a2, lsl #7

   $a1 = a2 + (a2 \times 2) = a2 \times 3$
   $a1 = a1 - (a2 \times 16) = (a2 \times 3) - (a2 \times 16) = a2 \times -13$
   $a1 = a1 + (a2 \times 128) = (a2 \times -13) + (a2 \times 128)$
     $= r1 \times 115$
   i.e. $a1 = a2 \times 115$

“And in Conclusion…”

- New Instructions:
  - mul
  - mla
  - umull
  - umlal
  - smull
  - smlal