# ARM INSTRUCTIONS

Data Processing Instructions

perform arithmetic and logical operations on data values in registers

only instructions which modify data values

All other instructions just move data around and control the sequence of program execution

These instructions typically require two operands and produce a single result

- All operands are 32 bits wide and come from registers or are specified as literals in the instruction itself.
- The result, if there is one, is 32 bits wide and is placed in a register. (There is an exception here: long multiply instructions produce a 64-bit result)
- Each of the operand registers and the result register are independently specified in the instruction.
- The ARM uses a '3-address' format for these instructions.
- ADD r0, r1, r2 ; comment begins r0 = r1 + r2
- values in the source registers are 32 bits wide and may be considered to be either unsigned integers or signed 2's-complement integers.

# Arithmetic operations.

- These instructions perform binary arithmetic (addition, subtraction and reverse subtraction, which is subtraction with the operand order reversed) on two 32-bit operands.
- The operands may be unsigned or 2's-complement signed integers; the carry-in, when used, is the current value of the C bit in the CPSR.

# Arithmetic operations.

```
ADD     r0, r1, r2          ; r0 := r1 + r2
ADC     r0, r1, r2          ; r0 := r1 + r2 + C
SUB     r0, r1, r2          ; r0 := r1 - r2
SBC     r0, r1, r2          ; r0 := r1 - r2 + C - 1
RSB     r0, r1, r2          ; r0 := r2 - r1
RSC     r0, r1, r2          ; r0 := r2 - r1 + C - 1
```

RSB is reverse subtraction.

RSC is reverse subtract with carry.

# Bit-wise logical operations.

- perform the specified Boolean logic operation on each bit pair of the input operands, so in the first case r0[i]:= r1[i] AND r2[i] for each value of i from 0 to 31 inclusive, where r0[i] is the ith bit of r0.
- BIC stands for bit clear - every ' 1' in the second operand clears the corresponding bit in the first

```
AND     r0, r1, r2          ; r0 := r1 and r2
ORR     r0, r1, r2          ; r0 := r1 or r2
EOR     r0, r1, r2          ; r0 := r1 xor r2
BIC     r0, r1, r2          ; r0 := r1 and not r2
```

# Register movement operations

- These instructions ignore the first operand
- move the second operand (possibly bit-wise inverted) to the destination
- 'MVN' mnemonic stands for 'move negated'
- result register is set to the value obtained by inverting every bit in the source operand

```
MOV     r0, r2          ; r0 := r2
MVN     r0, r2          ; r0 := not r2
```

# Comparison operations.

- These instructions do not produce a result but just set the condition code bits (N, Z, C and V) in the CPSR according to the selected operation.
- The mnemonics stand for 'compare' (CMP), 'compare negated' (CMN), '(bit) test' (TST) and 'test equal' (TEQ).

```
CMP CMN  r1, r2          ; set cc on r1 - r2
TST TEQ  r1, r2          ; set cc on r1 + r2
         r1, r2          ; set cc on r1 and r2
         r1, r2          ; set cc on r1 xor r2
```

# Immediate operands

Add a constant to a register, replace the second source operand with an immediate value, a literal constant preceded by #1

ADD r3, r3, r1; r3 = r3 + 1

AND r8, r7, #&FF ; r8 = r7 + FF Hexadecimal values are specified by adding &

# Shifted register operands

- ADD r3, r2, r1, LSL #3
- allows the second register operand to be subject to a shift operation before it is combined with the first operand
- Single instruction executed in single clock cycle
- LSL - Logical shift left
- #3 - shift left 3 times, the number can be from 0 to 31. If 0, it indicates no shift
- LSL: logical shift left by 0 to 31 places; fill the vacated bits at the least significant end of the word with zeros.
- LSR: logical shift right by 0 to 31 places; fill the vacated bits at the most significant end of the word with zeros.
- ASL: Arithmetic shift left - synonym for LSL
- ASR: arithmetic shift right by 0 to 32 places; fill the vacated bits at the most significant end of the word with zeros if the source operand was positive or with ones if the source operand was negative
- ROR: rotate right by 0 to 32 places, the bits which fall off the least significant end of word are used, in order to fill the vacated bits at the most significant end of the word
- RRX: rotate right extended by 1 place, the vacated bit is filled with the old value of the C flag and the operand is shifted one place to the right.

It is also possible to use a register value to specify the number of bits the second operand should be shifted by: ADD r5, r5, r3, LSL r2

This is a 4-address instruction.

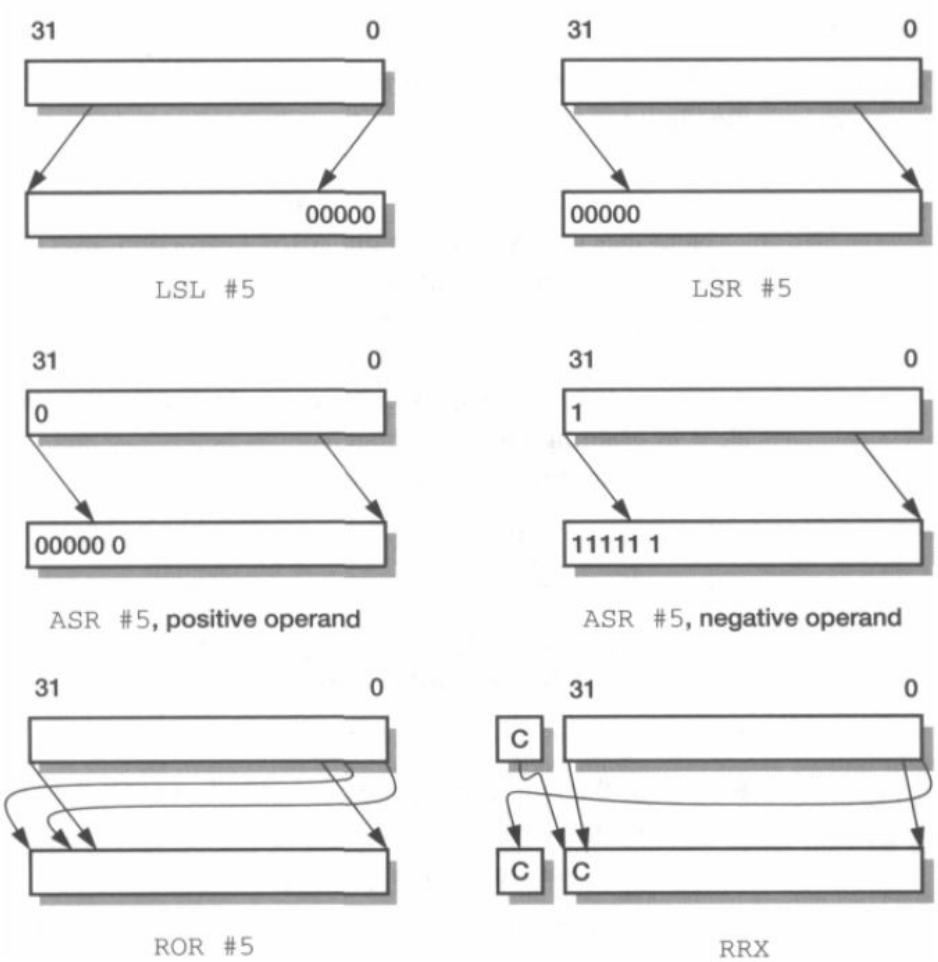


**Figure 3.1** ARM shift operations

# Setting the condition codes

Any data processing instruction can set the condition codes (N, Z, C and V) if the programmer wishes it to.

The comparison operations only set the condition codes, so there is no option with them, but for all other data processing instructions a specific request must be made.

```
ADDS    r2, r2, r0 ; 32-bit carry out -> C..
ADC     r3, r3, r1 ; .. and added into high word
```

An arithmetic operation (which here includes CMP and CMN) sets all the flags

according to the arithmetic result.

A logical or move operation does not produce a meaningful value for C or V, so these operations set N and Z according to the result but preserve V, and either preserve C when there is no shift operation, or set C to the value of the last bit to fall off the end of the shift.

the most important use of the condition codes, which is to control the program flow through the conditional branch instructions

Multiplies

MUL r4, r3, r2 ; r4 = r3 * r2

There are some important differences from the other arithmetic instructions:

- Immediate second operands are not supported.
- The result register must not be the same as the first source register.
- If the ' s' bit is set the V flag is preserved (as for a logical instruction) and the C flag is rendered meaningless.
- Multiplying two 32-bit integers gives a 64-bit result, the least significant 32 bits of which are placed in the result register and the rest are ignored.
- MLA r4, r3, r2, r1 ; r4 = (r3*r2 + r1)