

# ALU instructions



R. Ferrero, P. Bernardi

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

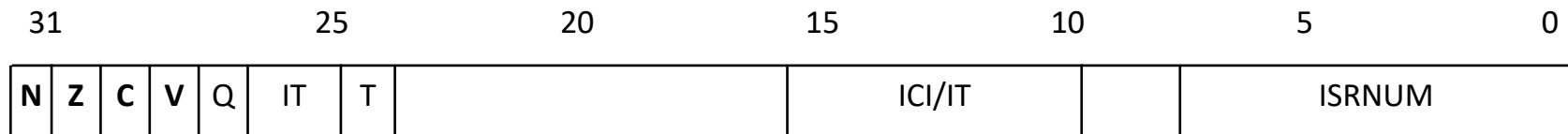
Torino - Italy

This work is licensed under the Creative Commons (CC BY-SA) License. To view a copy  
of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/>



# Flags

- The uppermost nibble of the program status register contains the flags.



- Flags are set and cleared by means of:
  - comparison instructions, e.g., CMP and TST
  - ALU instructions if they have the suffix ‘S’
  - a direct write to the program status register.

# Carry flag C

- After a sum, C = 1 if the result size is 33 bit.

$$\begin{array}{r} 1010\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001 + \\ 1011\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101 \\ \hline \end{array}$$

C = 1 0101 0110 0110 0110 0110 0110 0110 0110

- After a subtraction, C is inverted. Here C = 1:

$$\begin{array}{r} 1101\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 - \\ 0101\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \\ \hline \end{array}$$

1000 0000 0000 0000 0000 0000 0000 0000

- After a move or logical instruction, C is the result of an inline barrel shifter operation.

## Carry Flag C

In unsigned arithmetic, watch the carry flag to detect errors.

In signed arithmetic, the carry is used for comparisons together with other flags .

1. The carry flag is set if the addition of two numbers causes a carry out of the most significant (leftmost) bits added.
  - $1111 + 0001 = 0000$  (carry flag is turned on)
2. The carry (borrow) flag is also set if the subtraction of two numbers requires a borrow into the most significant (leftmost) bits subtracted.  
the borrow =  $1 - C$

- $0000 - 0001 = 1111$  (carry flag is turned on)

Otherwise, the carry flag is turned off (zero).

- $0111 + 0001 = 1000$  (carry flag is turned off [zero])
- $1000 - 0001 = 0111$  (carry flag is turned on, so borrow = 0)

# Negative flag N

- It corresponds to the first bit of the result.
- If N = 1, a 2's complement number is negative.
- Example:  $-4 + (-3) = -7 \rightarrow N = 1$

$$\begin{array}{r} 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1100 + \\ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1101 = \\ \hline \textcolor{red}{N} = 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1001 \end{array}$$

- Why N = 1 in the following sum?

$$\begin{array}{r} 0111 \ 0001 \ 0001 \ 0001 \ 0001 \ 0001 \ 0001 \ 0001 + \\ 0011 \ 0101 \ 0101 \ 0101 \ 0101 \ 0101 \ 0101 \ 0101 = \\ \hline \textcolor{red}{N} = 1010 \ 0110 \ 0110 \ 0110 \ 0110 \ 0110 \ 0110 \ 0110 \ 0110 \end{array}$$

# Overflow flag V

- It is set if, in a sum of values with the same sign, there is a change in the MSB

This is a carry

$$\begin{array}{r} 1010\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001 + \\ 1011\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101 = \\ \hline (\text{V} = 1)\ 1\ 0101\ 0110\ 0110\ 0110\ 0110\ 0110\ 0110\ 0110 \end{array}$$

- V = 0 if both conditions occur: the result is right

$$\begin{array}{r} 1110\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001 + \\ 1011\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101 = \\ \hline (\text{V} = 0)\ 1\ 1001\ 0110\ 0110\ 0110\ 0110\ 0110\ 0110\ 0110 \end{array}$$

## Zero flag Z

- It is set if the result is zero.

$$\begin{array}{r} 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101 + \\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011 = \\ \hline Z = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \end{array}$$

# Comparison instructions

- They compare the value or test some bits:

compare/test <Rd>, <operand2>

- They set the flags without updating Rd.

- The second operand can be:

- a register with an optional shift
- a constant obtained by shifting left an 8-bit value
- a constant of the form 0x00XY00XY
- a constant of the form 0xXY00XY00
- a constant of the form 0xXYXYXXY.

# CMP and CMN

- CMP (compare) subtracts operand2 from Rd and updates the flags.
- CMN (compare negative) adds operand2 to Rd and updates the flags.
  - The execution of the arithmetic operations at the base of the CMP and CMN is not modifying the content of the operands.
- Examples with r0 = #12
  - CMP r0, #10 -> N = 0, Z = 0, C = 1, V = 0
  - CMP r0, #12 -> N = 0, Z = 1, C = 1, V = 0
  - CMP r0, #14 -> N = 1, Z = 0, C = 0, V = 0
- The assembler can change CMP and CMN:
  - CMP r0, #-8 becomes CMN r0, #8

# TST and TEQ

- TST (test) computes the logical AND between operand2 and Rd; then updates all the flags except V.
- TEQ (test equivalence) computes the logical EOR between operand2 and Rd; then updates all the flags except V.
- Examples with r0 = #1
  - TST r1, r0, LSL#4 -> is the 4th bit of r1 set?
  - TEQ r2, r3 -> are r2 and r3 equal?

# Accessing Program Status Register

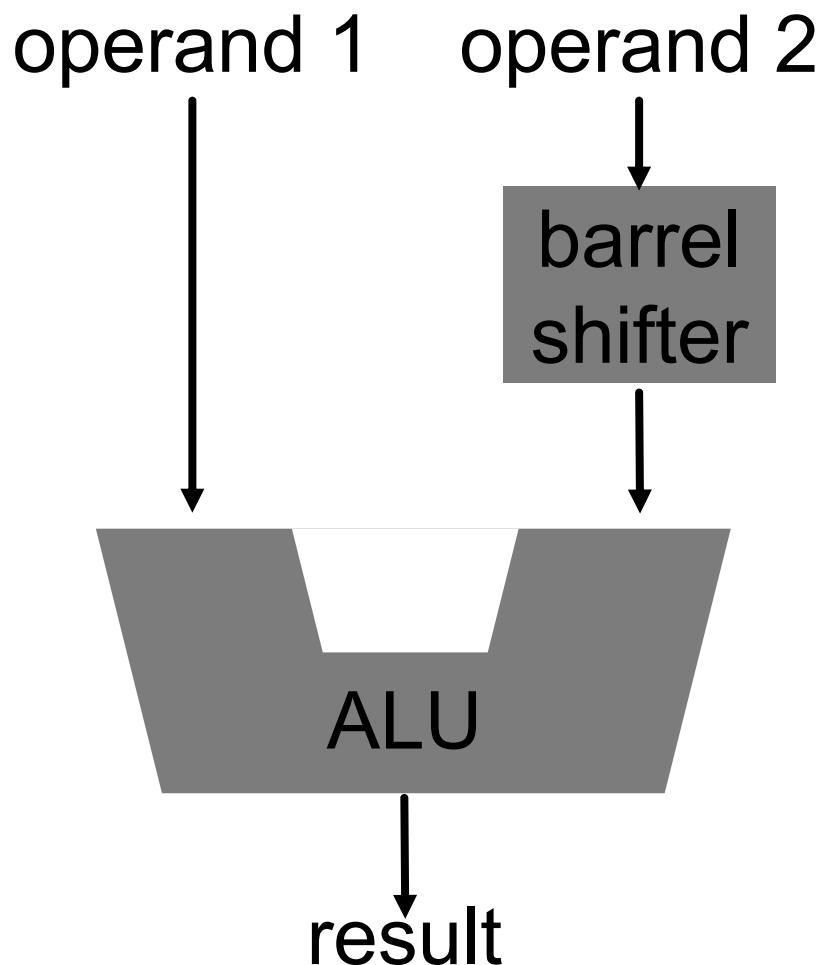
- MRS <Rn>, <Sreg> copies a special register into a register.
- MSR <Sreg>, <Rn> copies a general purpose register into a special register.
- Sreg can be APSR, EPSR, IPSR, and PSR.
- MRS r0, APSR reads the flags and copies them to the uppermost nibble of r0.

# ALU instructions and flags

- ALU can perform:
  - arithmetic operations
  - logic operations
  - shift and rotate operations.
- The flags are affected only if the suffix '**S**' is appended to the instruction.
- ```
LDR r0, =0xFFFFFFF9
ADDS r1, r0, #7
```

Flags are N = 0, Z = 1, C = 1, V = 0.

# Internal data path to ALU



Operand 1 is a register.

Operand 2 can be:

- shifted register
- shifted 8-bit constant
- 0x00XY00XY
- 0xXY00XY00
- 0xXYXYXYXY
- 12-bit constant (ADD and SUB only).

# Arithmetic instructions

- Addition
  - addition with carry
- Subtraction
  - subtraction with carry
  - reverse subtraction
- Multiplication
  - multiplication with accumulation
  - multiplication with subtraction
- Division

# Addition

- ADD <Rd>, <Rn>, <op2>       $Rd = Rn + op2$
- ADC <Rd>, <Rn>, <op2>  
     $Rd = Rn + op2 + C$
- ADDW is like ADD, but it takes only a 12-bit value and it can not update flags.
- With ADC it is possible to add 64-bit values:

ADDS r4, r0, r2

ADC r5, r1, r3

In the example  $r5, r4 = r1, r0 + r3, r2$

# Subtraction

- SUB <Rd>, <Rn>, <op2>       $Rd = Rn - op2$
- SBC <Rd>, <Rn>, <op2>  
     $Rd = Rn - op2 + C - 1$
- SUBW is like SUB, but it takes only a 12-bit value and it can not update flags.
- With SBC it is possible to subtract 64-bit values

SUBS r4, r0, r2

SBC r5, r1, r3

In the example  $r5, r4 = r1, r0 - r3, r2$

# Reverse subtraction

- RSB <Rd>, <Rn>, <op2>

$$Rd = op2 - Rn + C - 1$$

- Advantages:

- either one or the other operand can be shifted before the subtraction

```
SUB r0, r1, r2, LSL #2 ;r0 = r1 - r2*4
```

```
RSB r0, r2, r1, LSL #2 ;r0 = r1*4 - r2
```

- a register can be subtracted from a constant.

# Multiplication

- multiplication with 32-bit result

MUL <Rd>, <Rn>, <Rm>

- unsigned multiplication with 64-bit result

UMULL <Rd1>, <Rd2>, <Rn>, <Rm>

- signed multiplication with 64-bit result

SMULL <Rd1>, <Rd2>, <Rn>, <Rm>

- Note 1: there is no distinction between signed and unsigned multiplication with 32-bit result.
- Note 2: all operands must be registers.

# Multiplication with accumulation

- MLA <Rd>, <Rn>, <Rm>, <Ra>

$$Rd = Rn * Rm + Ra$$

- MLS <Rd>, <Rn>, <Rm>, <Ra>

$$Rd = Rn * Rm - Ra$$

- UMLAL <Rd1>, <Rd2>, <Rn>, <Rm>

$$Rd1, Rd2 = Rn * Rm + Rd1, Rd2$$

- SMLAL <Rd1>, <Rd2>, <Rn>, <Rm>

**same as UMLAL, but with signed values.**

# Division

- unsigned division

UDIV <Rd>, <Rn>, <Rm>

- signed division

SDIV <Rd>, <Rn>, <Rm>

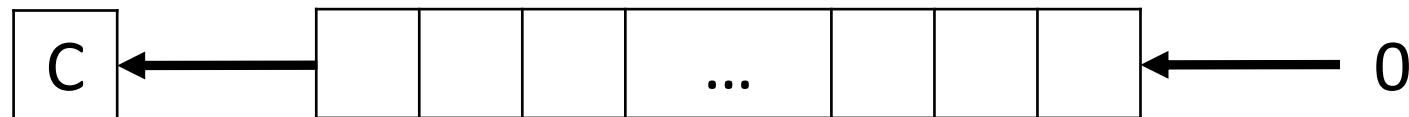
- If Rn is not exactly divisible by Rm, the result is rounded toward zero.
- UDIV and SDIV do not change the flags (the suffix 'S' can not be added).

# Logic instructions

- AND <Rd>, <Rn>, op2 ; Rn **AND** op2
- BIC <Rd>, <Rn>, op2 ; Rn **AND NOT** op2
- ORR <Rd>, <Rn>, op2 ; Rn **OR** op2
- EOR <Rd>, <Rn>, op2 ; Rn **XOR** op2
- ORN <Rd>, <Rn>, op2 ; Rn **OR NOT** op2
- MVN <Rd>, <Rn> ; **NOT** Rn

# Shift instructions

- LSL <Rd>, <Rn>, <op2>



- LSR <Rd>, <Rn>, <op2>

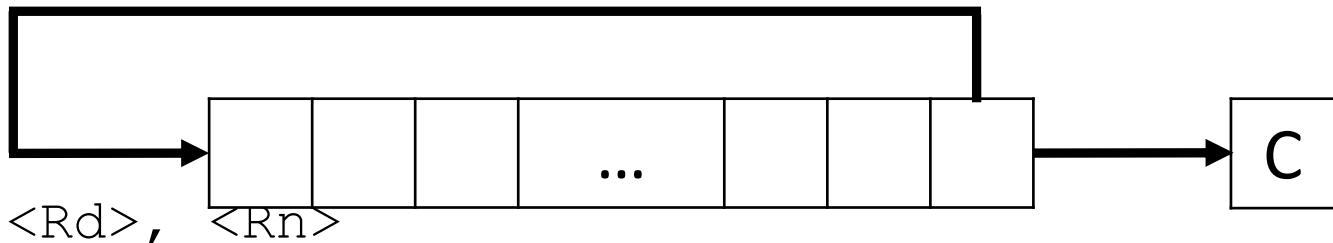


- ASR <Rd>, <Rn>, <op2>

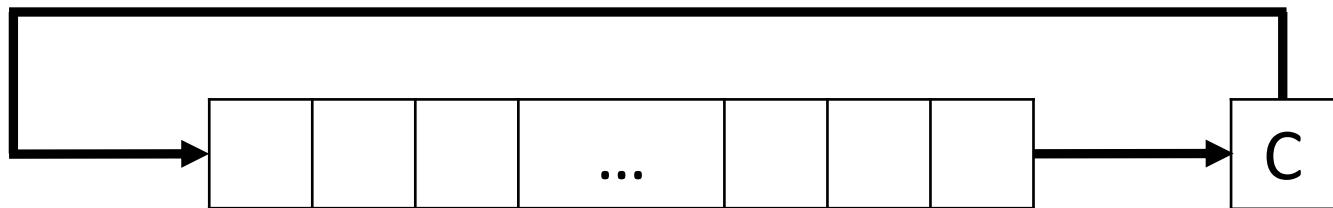


# Rotate instructions

- ROR <Rd>, <Rn>, <op2>



- RRX <Rd>, <Rn>



## Exercise: 64-bit routines for shift

- shift left  $r1, r0$  by one bit and save the result in  $r3, r2$ .
- logical shift right  $r1, r0$  by one bit and save the result in  $r5, r4$ .
- arithmetic shift right  $r1, r0$  by one bit and save the result in  $r7, r6$ .
- rotate right  $r1, r0$  by one bit and save the result in  $r9, r8$ .
- rotate right through carry flag  $r1, r0$  by one bit and save the result in  $r11, r10$ .