

Computer organization and architecture

Lesson 4

Condition flags

ARM instructions may set **condition flags** also called **status flags** based on whether the result is negative, zero, etc.

Flag	Name	Description
N	Negative	Instruction result is negative, i.e., bit 31 of the result is 1
Z	Zero	Instruction result is zero
C	Carry	Instruction causes a carry out
V	oVerflow	Instruction causes an overflow

Condition flags are set by the ALU and are held in the top 4 bits of the 32-bit **Current Program Status Register (CPSR)**

Subsequent instructions then execute conditionally, depending on the state of those condition flags.

The flags are set and cleared based on

- 1) Instructions that are specifically used for setting and clearing flags, such TST or CMP
- 2) Instructions that are told to set the flags by appending an “S” to the mnemonic.

Example:

```
SUBS R2, R3, R7 ; R2 = R3-R7,  
                ; and set the condition flags
```

Example: EORS would perform an exclusive OR operation and set the flags afterward

- 3) A direct write to the Program Status Register, where you explicitly set or clear flags

Instructions that affect condition flags:

Type	Instructions	Condition Flags
Add	ADDS, ADCS	N, Z, C, V
Subtract	SUBS, SBCS, RSBS, RSCS	N, Z, C, V
Compare	CMP, CMN	N, Z, C, V
Shifts	ASRS, LSLS, LSRS, RORS, RRXS	N, Z, C
Logical	ANDS, ORRS, EORS, BICS	N, Z, C
Test	TEQ, TST	N, Z, C
Move	MOVS, MVNS	N, Z, C
Multiply	MULS, MLAS, SMLALS, SMULLS, UMLALS, UMULLS	N, Z

When an S is appended to the instruction mnemonic, condition flags are changed

CMP – the compare instruction

It subtracts the 2nd source operand from the 1st and sets the condition flags based on the result

Example:

If the numbers are equal,

- the result is zero
- the Z flag is set

Example:

If the 1st number is an unsigned value that is higher than or the same as the 2nd, the subtraction will produce a carry out and the C flag is set.

Mnemonic	Name	CondEx
EQ	Equal	Z
NE	Not equal	\bar{Z}
CS/HS	Carry set / unsigned higher or same	C
CC/LO	Carry clear / unsigned lower	\bar{C}
MI	Minus / negative	N
PL	Plus / positive or zero	\bar{N}
VS	Overflow / overflow set	V
VC	No overflow / overflow clear	\bar{V}
HI	Unsigned higher	$\bar{Z}C$
LS	Unsigned lower or same	$Z \text{ OR } \bar{C}$
GE	Signed greater than or equal	$\bar{N} \oplus \bar{V}$
LT	Signed less than	$N \oplus V$
GT	Signed greater than	$\bar{Z}(\bar{N} \oplus \bar{V})$
LE	Signed less than or equal	$Z \text{ OR } (N \oplus V)$
AL (or none)	Always / unconditional	Ignored

The instruction mnemonic is followed by a **condition mnemonic** that indicates when to execute.

Example:

```
CMP R4, R5
ADDEQ R1, R2, R3
```

The CMP sets the Z flag if R4 and R5 are equal

The ADDEQ executes only if the Z flag is set.

CMN (compare negative) compares the first source to the negative of the second source by adding the two sources.

ARM instructions only encode positive immediates, so

CMN R2, #20 ; is used instead of CMP R2, #-20

TST (test) ANDs the source operands

– doesn't affect the V flag

 TST R2, #0xFF ; sets the Z flag
; if the low byte of R2 is 0

TEQ (test if equal) checks for equivalence by XOR-ing the sources.

– doesn't affect the V flag

The Z flag is set when they are equal and the N flag is set when the signs are different.

Condition mnemonics differ for signed and unsigned comparison.

Example: ARM provides two forms of \geq comparison:

HS (CS) is used for unsigned numbers

GE is used for signed numbers

For unsigned numbers, $A - B$ will produce a carry out (C) when $A \geq B$.



For signed numbers, $A - B$ will make **N** and V either both 0 or both 1 when $A \geq B$.

Unsigned Signed

$A = 1001_2$ $A = 9$ $A = -7$

$B = 0010_2$ $B = 2$ $B = 2$

$A - B:$ 1001 $NZCV = 0011_2$

$+ 1110$ **HS: TRUE**

10111 **GE: FALSE**

Unsigned Signed

$A = 0101_2$ $A = 5$ $A = 5$

$B = 1101_2$ $B = 13$ $B = -3$

$A - B:$ 0101 $NZCV = 1001_2$

$+ 0011$ **HS: FALSE**

1000 **GE: TRUE**

```
;R2 = #0x80000000, R3 = #0x00000001  
;R8 = -5, R9 = 7; R11 = 2;  
CMP R2, R3 ; R2-R3 = 0x7FFFFFFF, NZCV = 0011  
ADDEQ R4, R5, #78 ; does not execute because Z = 0  
ANDHS R7, R8, R9 ; executes because C=1  
ORRMI R10, R11, R12 ; does not execute as N = 0  
EORLT R12, R7, R10 ; executes as N = 0 and V = 1
```

$0x80000000 - 0x00000001 = 0x80000000 + 0xFFFFFFFF$
 $= 0x7FFFFFFF$ with a carry out (C=1)

The sources had opposite signs and the sign of the result differs from the sign of the first source, so the result overflows (V=1)

ANDHS executes because $R2 \geq R3$ (unsigned)

EORLT executes because $R2 < R3$ (signed)

ADDEQ and ORRMI do not execute because the result of $R2 - R3$ is not zero (i.e., $R2 \neq R3$) or negative.

The N flag

This flag is useful when checking for a negative result.

A two's complement number is considered to be negative if the most significant bit is set.

Example: $-1 - 2 = -3$

```
MOV    r3, #-1
```

```
MOV    r4, #-2
```

```
ADDS   r5, r4, r3
```

$$\begin{array}{r} \text{FFFFFFFF} \\ + \text{FFFFFFFE} \\ \hline \text{FFFFFFFFD} \end{array}$$

The N bit is set in the CPSR as the most significant bit of register r5 is set as a result of the addition.

Example:

The addends are positive in two's complement notation, but the sum is negative,

```
MOV    r3, #0x7B000000
MOV    r4, #0x30000000
ADDS   r5, r4, r3
```

$$\begin{array}{r} 7B000000 \\ + 30000000 \\ \hline AB000000 \end{array}$$

Since the most significant bit is now set, this forces the N bit to be set

The result indicates that this positive sum cannot be represented in 32 bits, so the result effectively overflowed the precision we had available.

The overflow flag V is set.



The V flag

When performing an operation like addition or subtraction, if we calculate the V flag as an XOR of the carry bit going into the most significant bit of the result with the carry bit coming out of the most significant bit, then the V flag accurately indicates a signed overflow.

Overflow occurs if the result of an add, subtract, or compare is greater than or equal to 2^{31} , or less than -2^{31}

$$2^{31} = 0x80000000$$

Example:

Two signed values, assumed to be in two's complement representations, are added

$$\begin{array}{r} A1234567 \\ + B0000000 \\ \hline 151234567 \end{array}$$

which does not fit into 32 bits.

NZCV = 0011

We overflowed, since we added two large, negative numbers together, and the most significant bit of the 32-bit result is clear (notice the 5 in the most significant byte of the result).

The Z flag

The Z flag tells us is that the result of an operation produces zero.

All 32 bits must be zero.

Add #0xFFFFFFFF and #0x00000001

The result is 0 NZCV = 0110


Subtract #0xFFFFFFFF and #0xFFFFFFFF

The result is 0 NZCV = 0110

The C flag

The Carry flag is set if

– the result of an addition is $\geq 2^{32}$

```
LDR    r3, =0x7B000000   
LDR    r7, =0xF0000000  
ADDS   r4, r7, r3 ; value exceeds 32 bits
```

NZCV = 0010

– the result of a subtraction is positive

The Carry flag may also be set as the result of an inline barrel shifter operation in a move or logical instruction.

The carry flag is inverted after a subtraction operation, making the carry bit more like a borrow bit, primarily due to the way subtraction is implemented in hardware.

For example, these instructions will set the carry bit to a one, since the operation produces no carry out and the bit is inverted:

```
LDR    r0, =0xC0000000  
LDR    r2, =0x80000000  
SUBS   r4, r0, r2 ; r4 = r0 - r2
```

NZCV = 0010

Exercise 4.1

Run the code from the Exercise 1.1 (adding the numbers from 1 to 10) again but replace SUBS with SUB

```
        MOV    R0, #0
        MOV    R1, #10
again   ADD    R0, R0, R1
        SUB    R1, R1, #1
        BNE    again
```

What will be the result of this code?

Pay close attention to CPSR

Exercise 4.2

Run all examples in Keil from the pages in this lecture explaining N, Z, C, V flags.

Observe the N, Z, C, V bits in the CPSR

Exercise 4.3

Learn the difference between CMP and TST, TEQ

Compare the numbers #0xE and #0xF using these instructions

Compare the numbers #0xA0000000 and #0x60000000 using these instructions

Observe the CPSR

Exercise 4.4

Experiment with the instructions ANDS, ORRS, EORS, MVNS, and BICS.

Put such different numbers into R0 and R1, that the result for ANDS, ORRS, MVNS, and BICS would set different condition flags. Explain why those flags are set.

Exercise 4.5

Experiment with the instructions ASRS, LSLS, LSRS, RORS, and RRXS

Put such different numbers into R0 and R1, that the result for ASRS, LSLS, LSRS, RORS, and RRXS would set different condition flags. Explain why those flags are set.

Exercise 4.6

Translate each of the following conditions into a single ARM instruction:



a) Add registers r3 and r6 only if N is clear. Store the result in register r7.



b) Multiply registers r7 and r12, putting the results in register r3 only if C is set and Z is clear.

c) Compare registers r6 and r8 only if Z is clear. 