Digital Electronics and Microprocessor Systems (ELEC211)

Dave McIntosh and **Valerio Selis**

dmc@liverpool.ac.uk
V.Selis@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL

Outline

- Use of the carry flag.
  - ADCS instruction
- Use of the negative flag.
- Use of the overflow flag.
- Other number formats.
  - Floating point.
  - IEEE 754.
  - Single precision.

Use of the carry flag

The carry flag can be used in two ways:

1) To determine if a conditional branch is executed or not.
   - E.g. "**B**C**C** label" will only **B**ranch if the **C**arry flag is **C**lear.
2) The other use of the carry flag is in the addition and subtraction instructions
   - Addition with carry instruction, ADCS
   - Subtraction with carry instruction, SBCS

Add with carry

The instruction ADCS ‘add with carry’ adds together the two values and adds the value of the carry flag (0 if clear, 1 if set).

E.g. ADCS r0, r1 ; would do r0 = r0 + r1 + Carry

- If initially r0=3, r1=5, and the carry flag is clear (C=0).
  The result would be r0 = 3 + 5 + 0 = 8
- If initially r0=3, r1=5, and the carry flag is set (C=1).
  The result would be r0 = 3 + 5 + 1 = 9

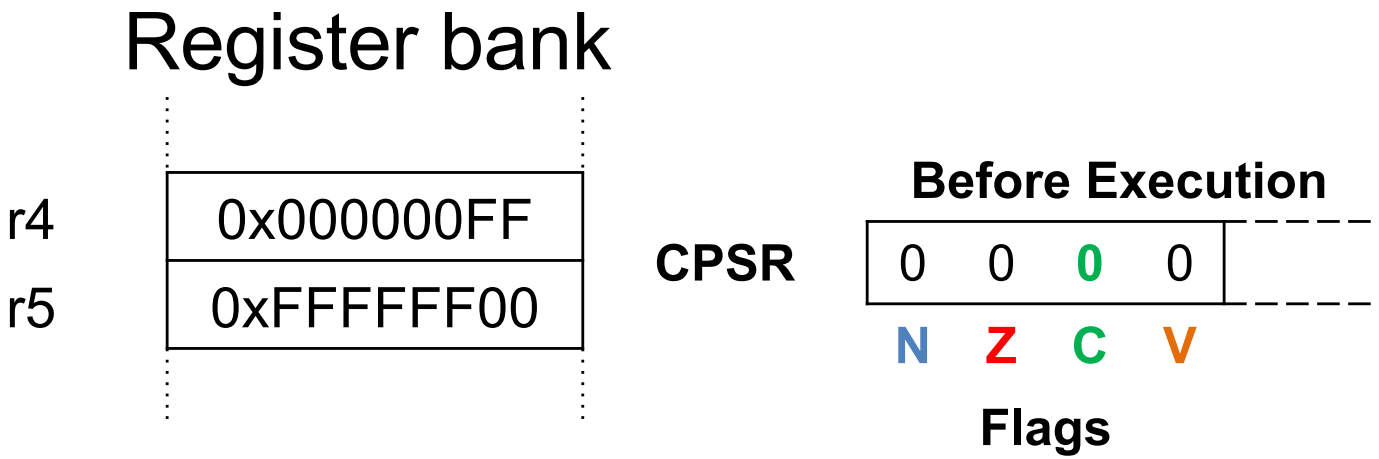
Question

What value is held in register r4 after the ‘add with carry’ instruction, ADCS r4, r5, is executed assuming the initial value in register r4 is 0x000000FF (= $+255_{10}$) and the value in r5 is 0xFFFFFF00 (= -256_{10}) when

(i) the carry flag is clear and

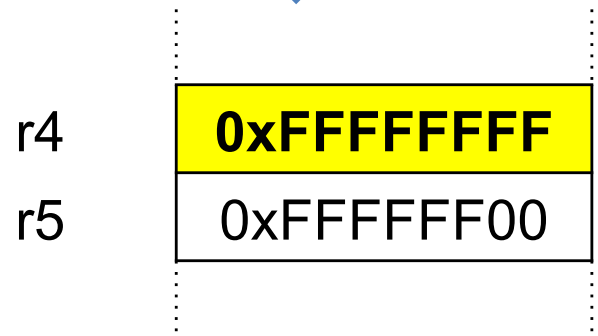
(ii) the carry flag is set?

Answer – carry flag clear



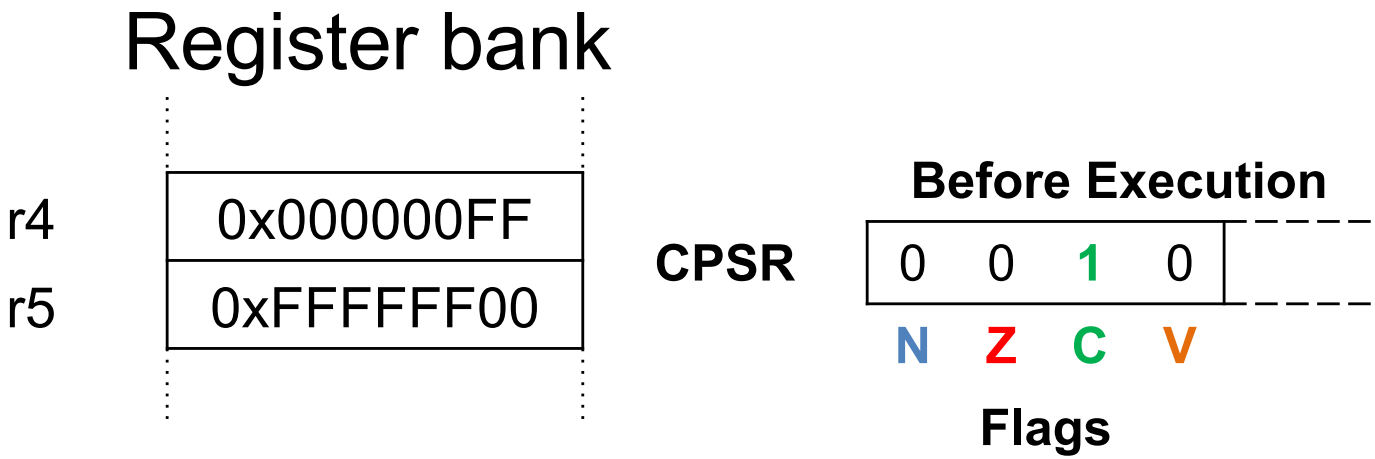
ADCS r4, r5

```
  0x 00 00 00 FF
+ 0x FF FF FF 00
+                0
  0x FF FF FF FF
```



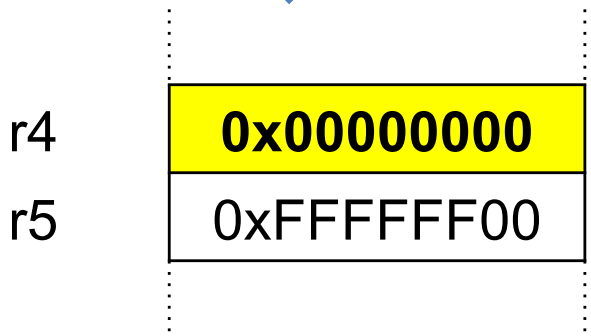
UNIVERSITY OF LIVERPOOL

Answer – carry flag set



ADCS r4, r5

```
  0x 00 00 00 FF
+ 0x FF FF FF 00
+                1
  0x 00 00 00 00
```



UNIVERSITY OF LIVERPOOL

Using add with carry

ADCS can be used to add together numbers greater than (2^{32}-1)

E.g. $120{,}000{,}000{,}000_{10}$ added to $130{,}000{,}000{,}000_{10}$ which in hex is 0x**1B**F08EB000 added to 0x**1E**449A9400

Both numbers are 37 bits in length and each one can be stored in two registers.

E.g. $120{,}000{,}000{,}000_{10}$ could be stored in registers, r0 and r1.
Register r0 could hold the value **0xF08EB000** and r1 could hold the value **0x0000001B**.

Using add with carry - example

E.g. $120{,}000{,}000{,}000_{10}$ added to $130{,}000{,}000{,}000_{10}$

or 0x1BF08EB000 added to 0x1E449A9400

r0 holds 0xF08EB000 and r1 holds 0x0000001B

r2 holds 0x449A9400 and r3 holds 0x0000001E

ADDS r4, r2, r0

r4 = r2 + r0 =

0xF08EB000 + 0x449A9400 = 0x135294400

The sum of r2 and r0 is greater than 0xFFFFFFFF so the carry flag is set (1) and r4 holds the lowest 32 bits: 0x35294400

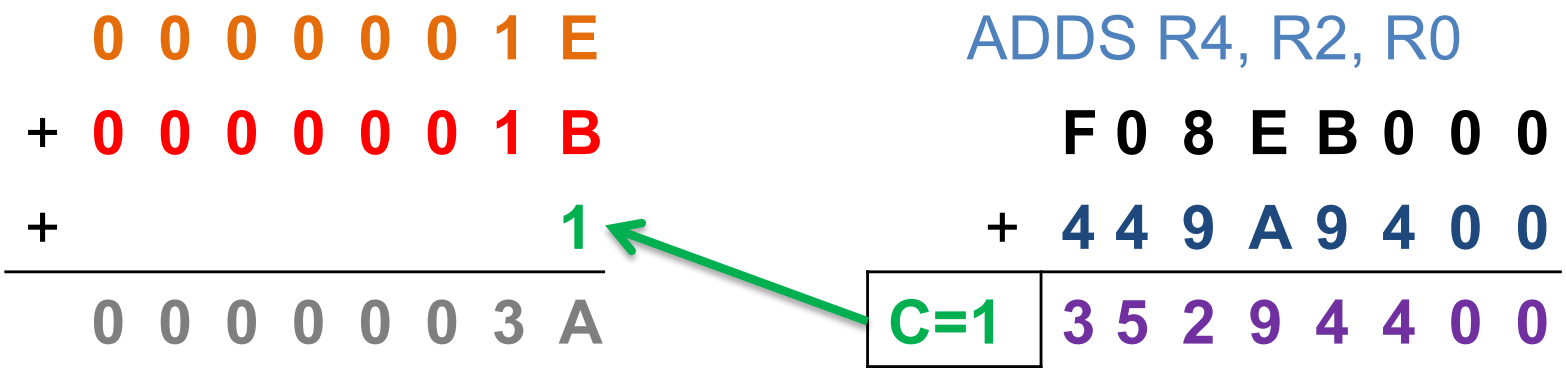
Using add with carry - example

Next instruction:

ADCS r3, r1

r3 = r3 + r1 + C =

0x0000001E + 0x0000001B + 1 = 0x0000003A



Taken together r3 and r4 hold the value

0x0000003A35294400 = $250{,}000{,}000{,}000_{10}$

The negative flag

If the m.s.b. or ‘sign bit’ is **1** the number is **negative**

if the m.s.b. or ‘sign bit’ is **0** the number is **positive**

To find out if a number is negative or positive we could use:

MOVS rx, rx

The value in register **rx** remains unchanged

The negative flag is updated:

- Set if the m.s.b. is **1**
- Cleared if the m.s.b. is **0**

The overflow flag

The overflow flag can be used to identify when a 2's complement result goes 'out of range'.

E.g. 0x59682F00 + 0x4190AB00 in binary.

```
  0101 1001 0110 1000 0010 1111 0000 0000
+ 0100 0001 1001 0000 1010 1011 0000 0000
  1001 10...   ← result (top 6 bits)
```

We are adding two positives numbers and the result is negative, there is an overflow into the sign bit and the overflow flag (**V**) is set.

Question

When poll is active, respond at **PollEv.com/elec211**

Text **ELEC211** to **22333** once to join

What happens to N,V,Z and C flags after each addition when r5 is 0x40000000 $(= 2^{30} = 1,073,741,824_{10})$

ADDS r4,r5,r6 ;r6:=$0x40000000 = 2^{30}$

ADDS r4,r5,r6 ;r6:=$0x80000000 = -2^{31}$

ADDS r4,r5,r6 ;r6:=$0xC0000000 = -2^{30}$

- 1st: N and V clear, Z and C set; 2nd: N set and Z, C, V clear; 3rd: Z and C clear, N and V set
- 1st: N set, V, Z and C clear; 2nd: N and V clear; Z, C set; 3rd: Z and C set, N and V clear
- 1st: N and V set, Z and C clear; 2nd: N set and Z, C, V clear; 3rd: Z and C set, N and V clear
- 1st: N and V set, Z and C clear; 2nd: N clear and Z, C, V set; 3rd: Z and C clear, N and V set

Total Results: 0

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

UNIVERSITY OF LIVERPOOL

Flags – summary

- The zero flag (**Z**) is set when the result (not including any carry out) is 0x00000000.
- The negative flag (**N**), is set when the most significant bit of the 32 bit result is 1.
- The carry flag (**C**) is set when the result (taken as an unsigned integer) is greater than $(2^{32}-1)$.
- The overflow flag (**V**) is set when the result (taken as a two's complement number) is greater than $(2^{31}-1)$ or less than -2^{31} (out of the allowed range).

Other number formats

Using 32 bits we can represent:

- Unsigned integers from 0 to $(2^{32} - 1)$
- Signed integers from -2^{31} to $(2^{31} - 1)$ using the two's complement format

What if we want to represent numbers bigger than 4.295×10^9 ($= 2^{32}$) or fractional number less than 1 but not equal to 0?

We can use two registers, that is 64 bits, to represent numbers up to 1.845×10^{19} but this uses twice as many bits and does not help with fractional numbers

Floating point numbers

IEEE 754 standard for floating point numbers have three basic components:

- The sign (**s**)
- The exponent (**n**)
- The fraction or mantissa (**a**)

Numbers are represented according to the following equation:

- $R = (-1)^s \cdot a \cdot 2^n$

Floating point numbers

IEEE 754 Single Precision Format:

In single precision:

- The sign (**s**) is 1 bit
- The mantissa (**a**) is 23 bits
- The exponent (**n**) is 8 bits

Floating point numbers – example

If we have number $+1995_{10}$ and we want to express it using the IEEE 754 Single Precision Format:

1st : we need to normalize the number, meaning that the m.s.b. of **a** is always **1** so this can be omitted

$+1995_{10} = +11111001011_2 \rightarrow +1.1111001011_2 \cdot 2^{10}$

2nd : $+1995_{10}$ is a positive number so **s** = 0

Binary point

3rd : to calculate **n** we need to consider a '**bias**' number which is 127 for single precision, so

$\mathbf{n} = 10_{10} + 127_{10} = 137_{10} = 10001001_2$

Result : in 32 bit IEEE 754 single precision format

0 10001001 11110010110000000000000

a = 23 bits

UNIVERSITY OF LIVERPOOL

Question

When poll is active, respond at **PollEv.com/elec211**

Text **ELEC211** to **22333** once to join

What is the IEEE 754 format of the following numbers?

-75_{10}

0.75_{10}

- $-75_{10} = 11000010110010110000000000000000_2, 0.75_{10} = 00111111011000000000000000000000_2$
- $-75_{10} = 11000010100101100000000000000000_2, 0.75_{10} = 00111111010000000000000000000000_2$
- $-75_{10} = 01000010100010110000000000000000_2, 0.75_{10} = 10111111011000000000000000000000_2$
- $-75_{10} = 11100010110010110000000000000000_2, 0.75_{10} = 00011111010000000000000000000000_2$

Total Results: 0

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

UNIVERSITY OF LIVERPOOL

IEEE 754 – special cases

IEEE 754 Single Precision Format special values:

- Zero is represented by a zero exponent and fraction although the sign bit can be either 0 or 1.
- Plus or minus infinity are represented by the maximum exponent value with a zero fraction and the appropriate sign bit, so 0x7F800000 is $+\infty$ and 0xFF800000 is $-\infty$.

IEEE 754 – special cases

- NaN (Not a Number) is indicated by the maximum exponent and a non-zero fraction, the sign bit can be either 0 or 1.

  NaN is used to indicate the result of an illegal floating point operation such as ‘divide by zero’ or ‘logarithm of a negative number’.

- Denormalized numbers are very small numbers that are just too small to normalize can be expressed as $\mathbf{a} \cdot 2^{-126}$, where **a** is of the form $\pm 0.xxxxxxxxxx_2$. These are represented by the sign bit, with a zero exponent followed by 23 bits of the fraction without the leading 0, i.e. $s00000000xxxxxxxxxxxxxxx_2$

Recommended reading and websites

ARM system-on-chip architecture

- Section 6.3: Floating-point data types

IEEE-754 Floating Point Converter

- https://www.h-schmidt.net/FloatConverter/IEEE754.html

Decimal to Hex, 2's complement and binary converter

- https://www.rapidtables.com/convert/number/decimal-to-hex.html

Summary

- Use of the carry, negative and overflow flags
- Floating point
- IEEE 754 standard
- Single precision

Next class?

Monday at 4 p.m. in the
Building 502,
Lecture Theatre 2
(502-LT2)