Digital Electronics and Microprocessor Systems (ELEC211)

Dave McIntosh and **Valerio Selis**

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

UNIVERSITY OF LIVERPOOL

Outline

- Negative numbers
  - Sign & Magnitude
  - 2’s complement
- Setting the N flag
- Setting the V flag
- Setting the C flag

Negative numbers

There are two main methods for representing negative numbers in microprocessors.

These are:

1) Sign magnitude
2) Two's complement

In both methods the most significant bit (m.s.b.) indicates the sign (sign bit)

- '1' indicates a negative number
- '0' indicates a positive number

The sign bit

The following applies to both ‘sign magnitude’ and ‘two’s complement’ methods:

- If the m.s.b. of a binary number is **1**, then the number is negative
- if the m.s.b. of a binary number is **0**, then the number is positive.
- If the most significant digit of a hexadecimal number is **8, 9, A, B, C, D, E or F**, then it is a negative number.
- If the most significant digit of a hexadecimal number is **0, 1, 2, 3, 4, 5, 6 or 7**, then it is a positive number.

Sign magnitude

Using the sign magnitude method, a negative number is the same as a positive number but with the m.s.b. or ‘sign bit’ equal to 1.

For example, for 16 bit numbers:

Decimal	Binary	Hexadecimal
+160	0000000010100000	0x00A0
−160	1000000010100000	0x80A0
+20640	0101000010100000	0x50A0
−20640	1101000010100000	0xD0A0

The magnitude of a ‘sign magnitude’ number is easy to find: simply AND the number with 0x7FFFFFFF (mask).

Sign magnitude

However sign magnitude numbers <u>cannot</u> be used for arithmetic.

For example: 3 + (−3) should be **0**.

Decimal	Hexadecimal
3	0x00000003
+ (−3)	+ 0x80000003
?	0x80000006

The answer is **−6** rather than **0** !

2's complement

In the two's complement method, a negative number, −x, is given by the value (2^n − x) for an *n* bit binary number.

For example −3 in 32 bits is:

0x100000000	(2^n, where $n = 32$)
−3	(−x)
0xFFFFFFFD	2's complement

So 0xFFFFFFFD is the 2's complement representation of −3 in 32 bits.

The two's complement method automatically sets the m.s.b. or 'sign bit' to 1.

2's complement

The following method can be used to find a 2's complement representation of a negative number

For example, -20640_{10}

1st : find the positive value: 0x000050A0 or
$0000\ 0000\ 0000\ 0000\ 0101\ 0000\ 1010\ 0000_2$

2nd : invert all bits (0 ➔ 1, 1 ➔ 0): 0xFFFFAF5F or
$1111\ 1111\ 1111\ 1111\ 1010\ 1111\ 0101\ 1111_2$

3rd : add 1 to the result: 0xFFFFAF5F + 1 = 0xFFFFAF60

Result: 0xFFFFAF60 is the 2's complement representation of -20640_{10}

2's complement

The inversion of bits can be implemented in hexadecimal rather than binary as follows:

Original No:	0 1 2 3 4 5 6 7 8 9 A B C D E F
Inverted No:	F E D C B A 9 8 7 6 5 4 3 2 1 0

1st : if the positive value is: 0x000050A0

2nd : invert all bits 0x000050A0 ➔ 0xFFFFAF5F

3rd : add 1 to the result:
0xFFFFAF5F + 1 = 0xFFFFAF60

Result: 0xFFFFAF60 is the 2's complement representation of -20640_{10}

Question

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

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

What is the two's complement of the following numbers in 32 bits?

$$-1,500,000,000_{10} \quad (1,500,000,000_{10} = 0x59682F00)$$
$$-211_{10} \quad (211_{10} = 0x000000D3)$$
$$-2017_{10} \quad (2017_{10} = 0x000007E1)$$

- $-1,500,000,000_{10} = 0xA697D101, -211_{10} = 0xFFFFFF2E, -2017_{10} = 0xFFFFF820$
- $-1,500,000,000_{10} = 0xA697D0FF, -211_{10} = 0xFFFFFF2C, -2017_{10} = 0xFFFFF81E$
- $-1,500,000,000_{10} = 0xF697D0FF, -211_{10} = 0xFFFFFF2C, -2017_{10} = 0xFFFFF81E$
- $-1,500,000,000_{10} = 0xA697D100, -211_{10} = 0xFFFFFF2D, -2017_{10} = 0xFFFFF81F$

Total Results: 0

2's complement

This method also works in reverse.

Original No:	0 1 2 3 4 5 6 7 8 9 A B C D E F
Inverted No:	F E D C B A 9 8 7 6 5 4 3 2 1 0

1st : if the negative value in 2's complement is:
0xFFFFFF60 (-160_{10})

2nd : invert all bits 0xFFFFFF60 ➔ 0x0000009F

3rd : add 1 to the result:
0x0000009F + 1 = 0x000000A0

Result: 0x000000A0 is the positive value 160_{10}

2's complement

Unlike sign magnitude, arithmetic is simple in two's complement.

For example (16 bit number): $3 + (-3)$ should be **0**.

Decimal	Binary
3	0000 0000 0000 0011
+ (−3)	+ 1111 1111 1111 1101
0	1 0000 0000 0000 0000

Note that, if the **carry bit** (17^{th} bit) is ignored, the answer is **0** which is correct.

Negative numbers in 4 bits

Hex	Binary	Sign magnitude	Two's complement
8	1000	0	−8
9	1001	−1	−7
A	1010	−2	−6
B	1011	−3	−5
C	1100	−4	−4
D	1101	−5	−3
E	1110	−6	−2
F	1111	−7	−1

Allowed ranges

	4 bits	16 bits	32 bits
Unsigned integer	0 to 15_{10} 0000_2 to 1111_2	0 to 65535_{10} 0000_{16} to $FFFF_{16}$	0 to $(2^{32} – 1)$ 00000000_{16} to $FFFFFFFF_{16}$
Sign magnitude	$–7_{10}$ to 7_{10} 1111_2 to 0111_2	$–32767_{10}$ to 32767_{10} $FFFF_{16}$ to $7FFF_{16}$	$–(2^{31}–1)$ to $(2^{31}–1)$ $FFFFFFFF_{16}$ to $7FFFFFFF_{16}$
Two's complement	$–8_{10}$ to 7_{10} 1000_2 to 0111_2	$–32768_{10}$ to 32767_{10} 8000_{16} to $7FFF_{16}$	$–2^{31}$ to $(2^{31}–1)$ 80000000_{16} to $7FFFFFFF_{16}$

UNIVERSITY OF LIVERPOOL

2’s complement: sign extension

When converting a two’s complement number from a 4 or 16 bit format to a format with more bits, the process of ‘sign extension’ is used.

For example

Decimal	4 bits	16 bits
7	0111	0000 0000 0000 0111
−6	1010	1111 1111 1111 1010

The extra 12 bits, extending the bits from 4 to 16 take the value of the sign bit shown in red. Likewise in hexadecimal:

Decimal	16 bits	32 bits
20165	0x4EC5	0x00004EC5
−32501	0x810B	0xFFFF810B

UNIVERSITY OF LIVERPOOL

2's complement: arithmetic

So if we add two very big numbers so that the sum is greater than $(2^{31}-1)$ then that number will be negative if we are working in 2's complement with 32 bits.

E.g. if we add $1{,}500{,}000{,}000_{10}$ to $1{,}100{,}000{,}000_{10}$ that is 0x59682F00 added to 0x4190AB00:

$$
\begin{array}{rcccc}
 & \text{0x} & 59 & 68 & \text{2F} & 00 \\
+ & \text{0x} & 41 & 90 & \text{AB} & 00 \\
\hline
 & \text{0x} & \text{9A} & \text{F8} & \text{DA} & 00
\end{array}
$$

In two's complement, the sum is a negative number $(= -1{,}694{,}967{,}296_{10})$ and it is clearly incorrect.

UNIVERSITY OF LIVERPOOL

Setting the N flag

```
    0x 59 68 2F 00
+   0x 41 90 AB 00
    --------------
    0x 9A F8 DA 00
```

If the instruction **ADDS** is used the negative flag (**N**) would be set but the carry flag (**C**) would be cleared because the sum is still a 32 bit result.

Any instruction that sets or clears the flags, e.g. MOVS, would set the negative flag if the value in the destination register has a sign bit equal to **1**.

Overflow

The allowed range for 32 bit in 2's complement is -2^{31} to $(2^{31}-1)$ or 0x80000000 to 0x7FFFFFFF.

When a 2's complement result goes 'out of range', we call it an 'overflow'.

E.g. for the last example in binary.

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

There is an overflow into the sign bit and the result 0x9AF8DA00 is out of range.

Setting the V flag

The ‘overflow’ flag (**V**) is set when the addition of 2 positive numbers gives in a negative result.

The overflow flag relates to 2’s complement numbers whereas the carry flag (**C**) relates to ‘unsigned’ integers.

	2’s complement	unsigned integer
0x5968 2F 00	$1{,}500{,}000{,}000_{10}$	$1{,}500{,}000{,}000_{10}$
+ 0x4190 AB 00	$+1{,}100{,}000{,}000_{10}$	$+1{,}100{,}000{,}000_{10}$
0x9AF8 DA00	$-1{,}694{,}967{,}296_{10}$	$2{,}600{,}000{,}000_{10}$

UNIVERSITY OF LIVERPOOL

Adding negative numbers

Using two's complement we can do sums, x + (−y)

E.g. if we add $1,500,000,00_{10}$ to $-1,100,000,000_{10}$

1st : find the 2's complement of $-1,100,000,000_{10}$, the positive value is: $1,100,000,000_{10}$ or 0x4190AB00

2nd : invert all bits 0x4190AB00 ➔ 0xBE6F54FF

3rd : add 1 to the result:
0xBE6F54FF + 1 = 0xBE6F5500

4th : add 0xBE6F5500 to 0x59682F00 ($1,500,000,000_{10}$)

UNIVERSITY OF LIVERPOOL

Adding negative numbers

$$
\begin{array}{rl}
 & \text{0x} \quad 59\ 68\ 2F\ 00 \\
+ & \text{0x} \quad BE\ 6F\ 55\ 00 \\
\hline
 & \text{0x}1\ 17\ D7\ 84\ 00
\end{array}
$$

Because we are working in two's complement, we can ignore the carry **1** and the answer in the lowest 32 bits is 0x17D78400 or $400{,}000{,}000_{10}$ which is correct.

There is a 'carry' indicating that the result is too big for a 32 bit unsigned integer

There is no overflow in two's complement

The signed number is positive because the m.s.b. is **0** (most sig. hex digit is 0x1 = 0001_2).

Adding negative numbers

```
  0101 1001 0110 1000 0010 1111 0000 0000
+ 1011 1110 0110 1111 0101 0101 0000 0000
  0001 01…       ← result (top 6 bits)
1 1111 00…       ← carry from previous column
```

The sign bit of the result is **0** indicating a positive result.

A carry out (**1**) occurs so the result is incorrect if considered as an unsigned integer, but the 32 bit result is correct if consider as a two's complement number.

Recommended reading

Fundamentals of Logic Design. Roth and Kinney.

- Chapter 1: Introduction Number Systems and Conversion.

Summary

- Negative numbers
- Setting the Negative flag
- Setting the Overflow flag
- Setting the Carry flag

Next class?

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