ENS1161
Computer Fundamentals

Lecture 08
Representation of Numbers in a Computer

Dr Włodzimierz Górnisiewicz – School of Engineering

Outline

01. Representation of Numbers in a Computer
02. Signed and unsigned integers
03. Representation of unsigned integers
04. Representation of signed integers
05. Definition of 8-bit 2's complement representation
06. Finding the 1's complement of a number
07. Converting an integer –m to 2's complement form
08. Converting a 2's complement form to an integer
09. A hexadecimal version of the 1's complement method
10. Addition of two bytes and the C, N and V flags
11. ALU structure
12. Examples of additions and flags
13. "Overflow" (when the correct answer is too big)
14. Interpreting the output as an unsigned integer
15. Interpreting the output as a signed integer
16. The "questions that flags N and V are asked"
17. How to extend to 16 bits
18. Summarising the N & V flags
19. Looking at both interpretations

Lecture’s Major Objectives

After completing this section, students should be able to:

- find the 2's complement representation of a decimal number
- find the decimal represented by a 2's complement number
- add two 8-bit binary numbers and find the C, N and V flags
- interpret the result of an addition as an unsigned integer
- interpret the result of an addition as a signed integer
- expand the result of an addition to 16 bits if 8 bits are not sufficient

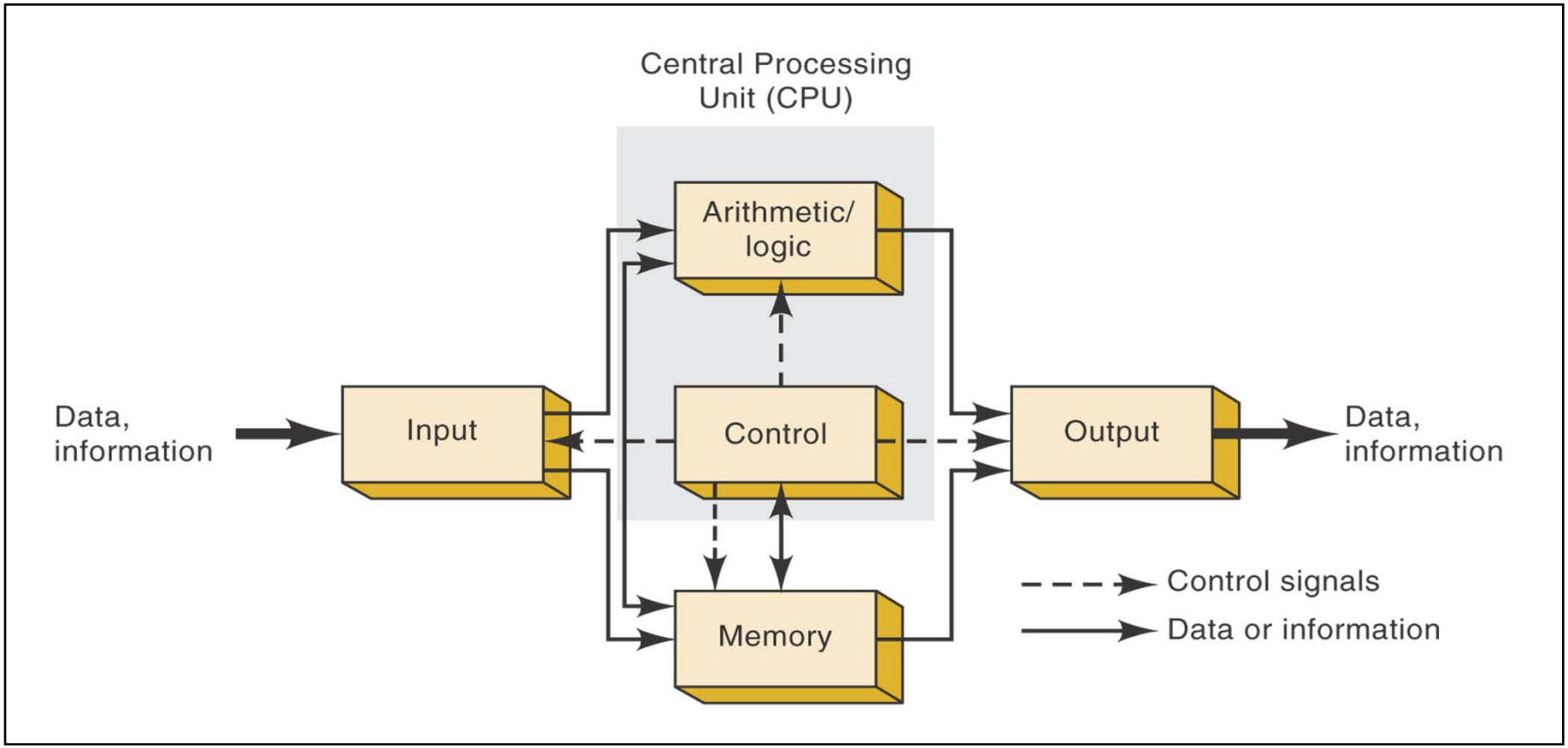
Representation of Numbers in a Computer

In this section we are concerned with integers, how they are represented in a computer and how they are added.

In a computer, numbers are usually represented as sequences of binary digits ("bits") of fixed length.
For example an integer may be stored in 1 byte (8 bits) or sometimes it might require two bytes (16 bits).

When talking about 8-bit binary numbers, for example, the rightmost bit is called "bit 0" and the leftmost "bit 7".
Bit 0 is also called the "least significant bit" (LSB) because it represents 1, and bit 7 is called the "most significant bit“ (MSB) because it represents 128.

Representation of Numbers in a Computer



Functional diagram of a digital computer.

Signed and unsigned integers

The set of integers consists of zero and the positive and negative "whole numbers", such as 37, 1496, –28, –355, etc, that is numbers that have no fractional part.

Computer science uses the concepts of signed integers and unsigned integers.
Unsigned integers are the non-negative integers, in other words 0 and the positive integers. Some applications involve positive whole numbers only.

Signed integers include all the integers, that is positive, negative and zero, and so a sign is necessary to distinguish between positive and negative numbers.

Some applications require the use of both positive and negative whole numbers.

Representation of unsigned integers

Representation of unsigned integers is no different from that of ordinary binary numbers, except that there may be extra zeros added in front to "pad out" to 8 bits or 16 bits.

The binary representation of 55 is 110111, its 8-bit representation is 0011 0111.

If we use 8 bits, the range of values for unsigned integers is:

0000 0000 to 1111 1111
(0 to 255 in decimal)

With 16 bits the range for unsigned integers is:

0000 0000 0000 0000 to 1111 1111 1111 1111
(0 to 65535 in decimal)

Representation of signed integers

In our work with signed integers we will use the method called 2's complement representation.

Mostly we will be concerned with 8-bit 2’s complement representation, but sometimes we will use 16-bit 2’s complement representation.

To represent signed integers we separate the possible 8-bit numbers into two groups:

0000 0000 to 0111 1111 represent positive integers (or zero)
1000 0000 to 1111 1111 represent negative integers.

Representation of signed integers

With this arrangement we can represent signed integers from –128 to 127:

0000 0000 to 0111 1111 represent the numbers 0 to 127
1000 0000 to 1111 1111 represent the numbers from –128 to –1.

Notice that bit 7 indicates the sign:

- if bit 7 is 0, the number is positive (or zero)
- if bit 7 is 1, the number is negative

Remember:

When dealing with signed integers

- if a number starts with 0, it is positive (or zero)
- if a number starts with 1, it is negative

Definition of 8-bit 2's complement representation

(i) For an integer from 0 to 127, the 2's complement representation is the same as its usual binary representation.

(ii) For an integer –m from –128 to –1 the 2's complement representation is the binary representation of 256 – m.

In practice, we use (i), but in place of (ii) we use an algorithm ("recipe") based on the so-called 1's complement of the number.

Finding the 1's complement of a number

The procedure to find the 1's complement of a number is very easy.

Simply replace each 1 by a 0 and vice versa.

For example:

the 1's complement of 1111 1110 is 0000 0001
the 1's complement of 0001 1001 is 1110 0110
the 1's complement of 1001 1001 is 0110 0110

Converting an integer –m to 2's complement form

To find the 2's complement representation of a negative integer –m, the "recipe" is:

- drop the minus sign
- convert the decimal number m to 8-bit binary
- find the 1's complement and add 1

Example :

Find the 2's complement representation	–95
The binary representation of 95 is	0101 1111
The 1's complement is:	1010 0000
Add 1	1
	1010 0001

– 95 is 1010 0001

Converting an integer –m to 2's complement form

Example:

Find the 2's complement representation of	–46
The binary representation of 46 is	0010 1110
The 1's complement is:	1101 0001
Add 1	1
	1101 0010

–46 is 1101 0010

Converting an integer –m to 2's complement form

Example:

Find the 2's complement representation of	–68
The binary representation of 68 is	0100 0100
The 1's complement is:	1011 1011
Add 1	1
	1011 1100

– 68 is 1011 1100

Converting a 2's complement form to an integer

To convert a positive number to decimal we may use any methods of conversion from an ordinary binary number.

To convert a negative number in 2's complement form to decimal, we use the fact that the process of finding the 2's complement is its own inverse.

Converting a 2's complement form to an integer

Suppose we start with	1010 0011
Its 1's complement is	0101 1100
Add 1	1
The result is	0101 1101
Now starting with	0101 1101
Its 1's complement is	1010 0010
Add 1	1
The result is	1010 0011 - original number.

- find the 1's complement and add 1
- convert the binary number to decimal
- attach a minus sign

Converting a 2's complement form to an integer

Example:

Convert the 2's complement number 1011 1001 to decimal

The number is negative.

The 2's complement number is	1011 1001
Its 1's complement is	0100 0110
Add 1	1
The result is	0100 0111

$$0100\ 0111_2 = 71_{10}$$

$$1011\ 1001_2 = -71_{10}$$

Converting a 2's complement form to an integer

Example:

Convert the 2's complement number 1101 0111 to decimal

The number is negative	
The 2's complement number is	1101 0111
Its 1's complement is	0010 1000
Add 1	1
The result is	0010 1001

$$0010\ 1001_2 = 41_{10}$$

$$1101\ 0111_2 = -41_{10}$$

Converting a 2's complement form to an integer

Example:

Convert the 2's complement number 0010 0101 to decimal

The number is positive

So its 2's complement representation is just its ordinary binary representation.

$$0010\ 0101_2 = 25_{16} = 37_{10}$$

$$0010\ 0101_2 = 37_{10}$$

A hexadecimal version of the 1's complement method

To convert negative integer to 8-bit 2's complement :

Using binary

- drop the minus sign
- convert the decimal number to binary
- find the 1's complement and add 1

Using hexadecimal

- drop the minus sign
- convert the decimal number to hex
- subtract from FF and add 1
- convert to binary

A hexadecimal version of the 1's complement method

To convert the 8-bit 2's complement of a negative integer to decimal:

Using binary

- find the 1's complement and add 1
- convert the binary number to decimal
- attach a minus sign

Using hexadecimal

- convert from binary to hex
- subtract from FF and add 1
- convert the hex number to decimal
- attach a minus sign

Addition of two bytes and the C, N and V flags

When the microprocessor "adds two 8-bit binary numbers", it simply processes two bytes of binary data.

The operation is performed by the Arithmetic and Logic Unit (ALU) in the heart of the microprocessor. The ALU is given two 8-bit binary numbers with the instruction to "add".

When the ALU carries out the addition, it also assigns values to various "flags", in particular the N, V and C flags.

It is the programmer task to interpret the instruction result depends on the type integer being process - signed or unsigned.

Addition of two bytes and the C, N and V flags

The unsigned integers requires to use the C flag to determine whether the 8 bit output needs to be extended to 16 bits in order to give a correct answer.

The signed integers require use the V and N flags to determine the sign of the answer, and also to determine whether the 8-bit output needs to be extended to 16 bits in order to give a correct answer.

Addition of two bytes and the C, N and V flags

Each flag is given the value 1 or 0 during the addition operation as follows.

C: is equal to the carry out of bit 7.
C is called "the carry flag".

N: is equal to the MSB (bit 7) of the 8-bit output.
N is sometimes called the "negative flag" or the "sign bit".

V: is calculated from the carry bit into bit 7 and the carry bit out of bit 7.
V is sometimes called the "sign bit error flag".

Addition of two bytes and the C, N and V flags

When the result of the addition is to be interpreted as a signed integer, V is used :

- together with N, to determine the sign of the answer, and
- to determine whether the 8-bit output represents the correct answer or whether needs to be extended to 16 bits.

Addition of two bytes and the C, N and V flags

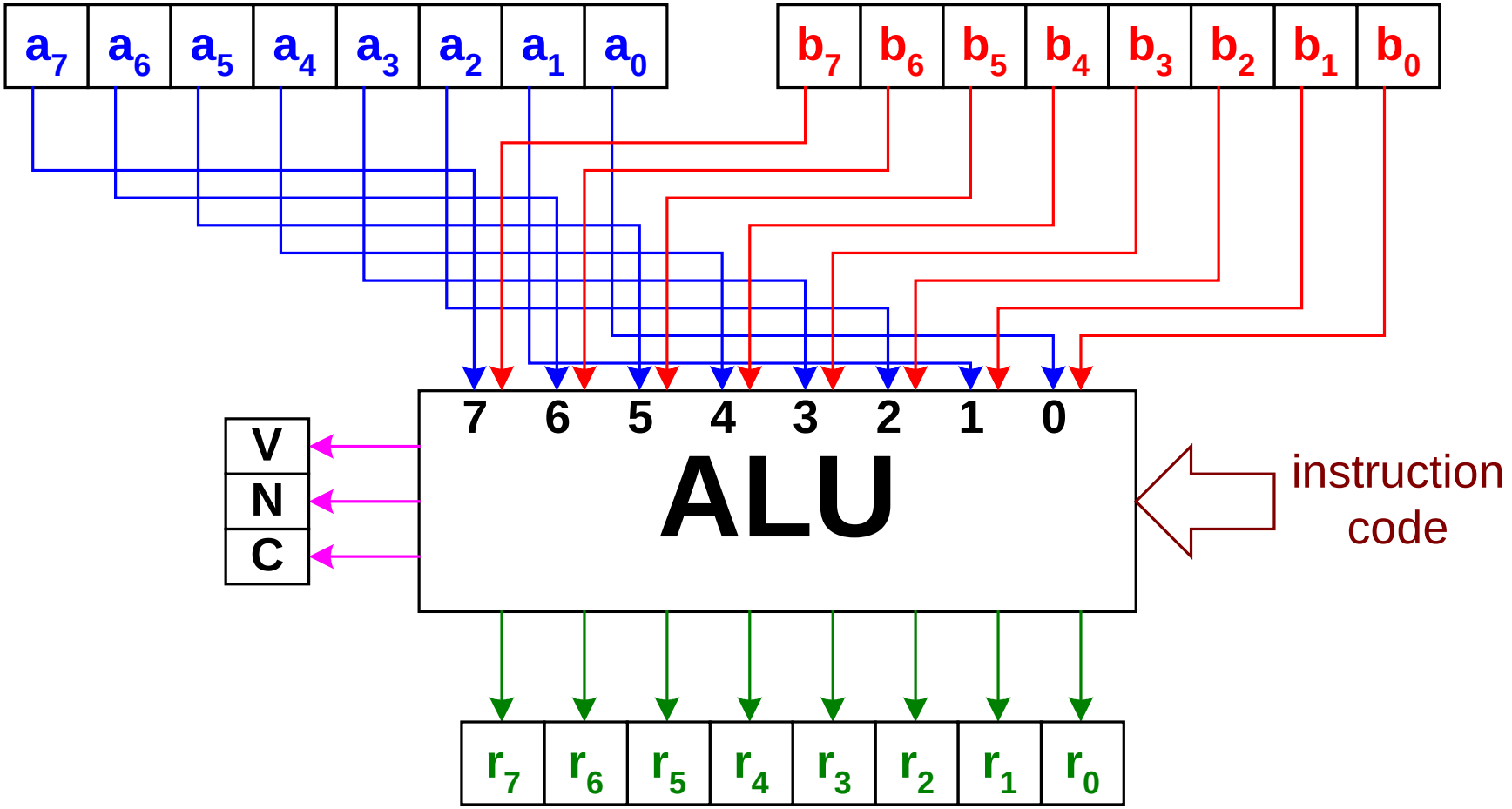
We calculate V using:

$$V = (\text{carry into bit 7}) \oplus (\text{carry out of bit 7})$$

The truth table for exclusive or is as shown

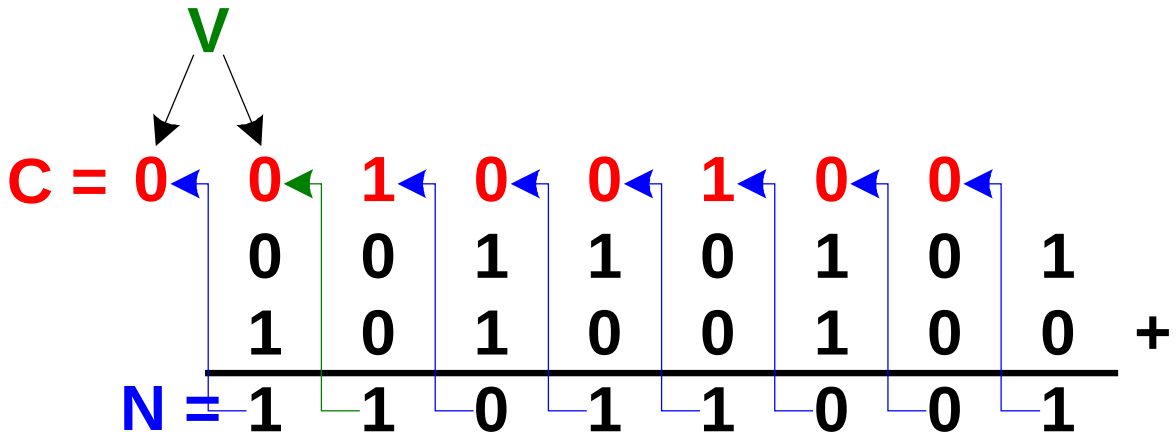
p	q	p⊕q
0	0	0
0	1	1
1	0	1
1	1	0

ALU structure



Examples of additions and flags

Example 1
When the two 8-bit binary numbers 00110101 and 10100100 are added the result is an 8-bit output and the N, V and C flags are given values of 0 or 1:



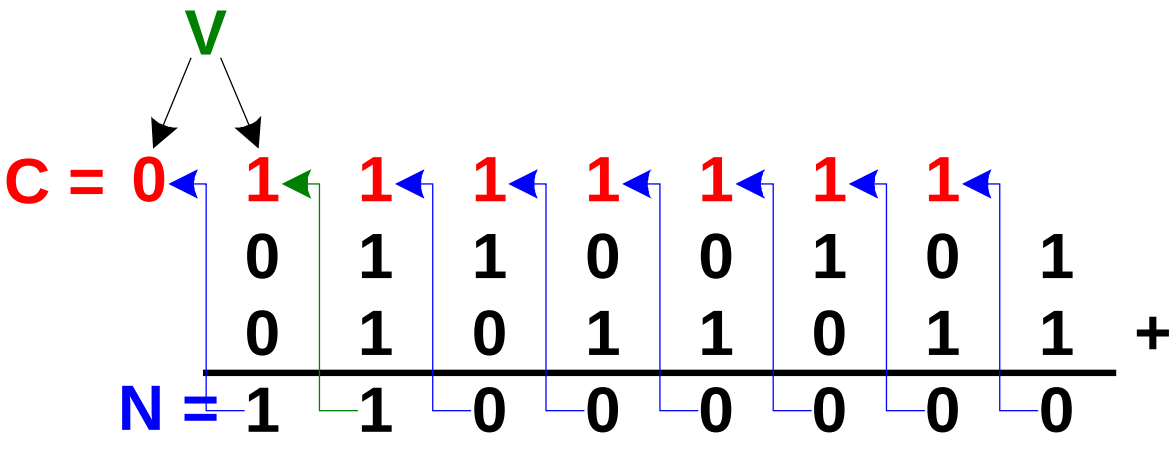
C	N	V
0	1	0

We can see that:

N = 1, because N = bit 7 of the "answer"
V = 0, because (carry into bit 7) = (carry out of bit 7)
C = 0, because C = the carry out of bit 7

Examples of additions and flags

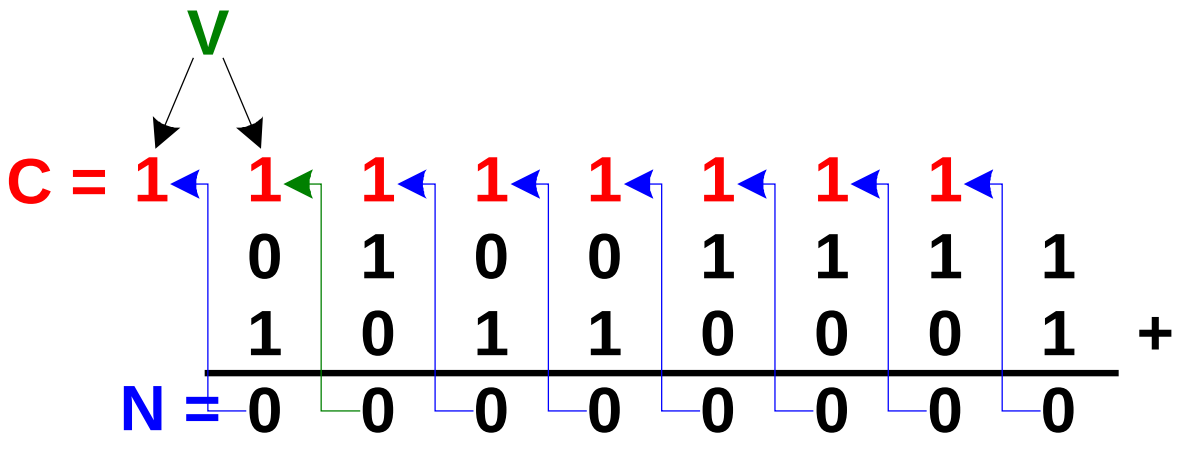
Example 2



C	N	V
0	1	1

Examples of additions and flags

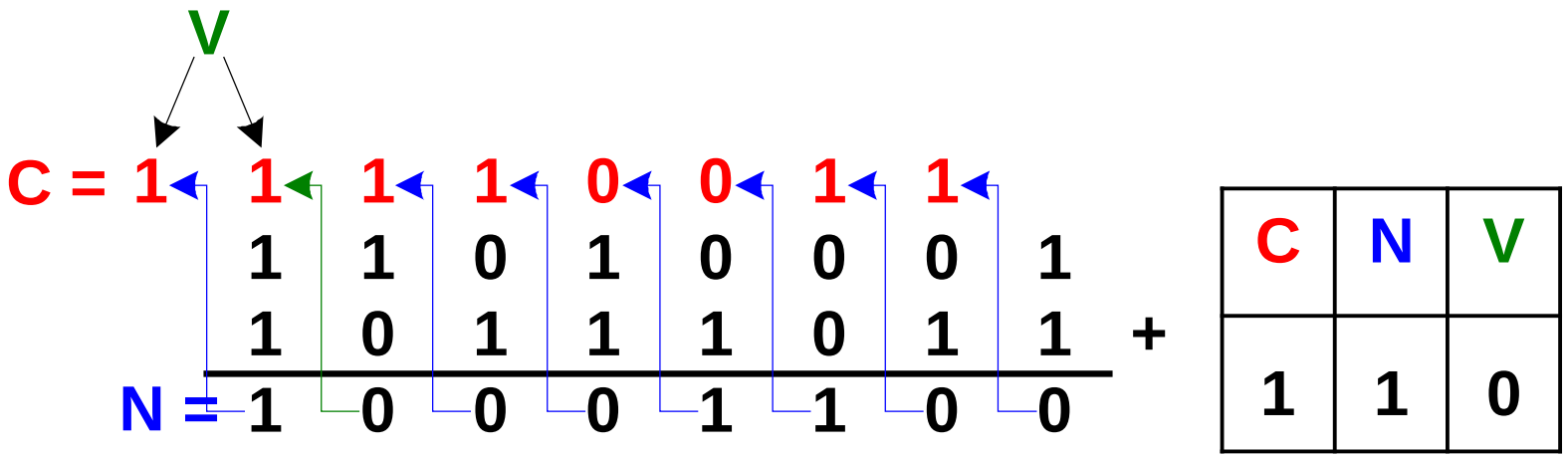
Example 3



C	N	V
1	0	0

Examples of additions and flags

Example 4



C	N	V
1	1	0

"Overflow" (when the correct answer is too big)

Sometimes the correct answer is bigger than 8 bits, and the program must detect this and extend the answer to 16-bits.

This is indicated by the flags:

- If the output is to be interpreted as an unsigned integer, the C flag is used.
- If the output is to be interpreted as a signed integer, the V and N flags are used.

Interpreting the output as an unsigned integer

If C = 0, the correct answer fits into 8 bits, but
if C = 1 then the correct answer does not fit into 8 bits and so the output must be extended to 16 bits.

Interpreting the output as an unsigned integer

Example 1
Consider the result of adding 168 and 137. The correct answer is 305 and it does not fit in 8 bits.
But how does the microprocessor tell us this?

If we add the binary representations of 168 and 137 by hand, we get:

168 → 1010 1000
137 → 1000 1001 +
(1) 0011 0001

However the output from the microprocessor would be just 8-bits and the flags:

8-bit output	C	N	V
00110001	1	0	1

and this is what we must interpret.

Interpreting the output as an unsigned integer

Since C = 1, the correct answer does not fit into 8 bits.

To extend the answer to 16 bits, we take the 8-bit output, put the value of C in front of it, and then put extra 0's in front to "pad out" the answer to 16 bits:

0000 0001 0011 0001 8-bit output + C + extra 0's

The correct answer is:

$$0000\ 0001\ 0011\ 0001_2 = 0131_{16} = 305_{10}$$

Interpreting the output as an unsigned integer

Example 2

Consider the result of adding 98 and 114. The correct answer is 212, which is inside the range for 8-bit unsigned integers. But how does the microprocessor tell us this?

If we add the binary representations of 98 and 114 by hand, we get:

98	→		0110 0010
114	→		0111 0010 +
		(0)	1101 0100

The output from the microprocessor would be just 8-bits and the flags:

8-bit output	C N V
11010100	0 1 1

and this is what we must interpret. Since C = 0, the correct answer fits into 8 bits, and so the correct answer is

$$1101\ 0100_2 = D4_{16} = 212_{10}$$

Interpreting the output as an unsigned integer

Example 3
How should we interpret the following output from the microprocessor as an unsigned integer?

8-bit output	C N V
01100000	1 0 1

Since C = 1, the correct answer does not fit into 8 bits. So the correct answer is:

$$0000\ 0001\ 0110\ 0000_2 = 160_{16} = 352_{10}$$

Interpreting the output as an unsigned integer

Example 4
How should we interpret the following output from the microprocessor as an unsigned integer?

8-bit output	C N V
00110110	1 0 1

Since C = 1, the correct answer does not fit into 8 bits. So the correct answer is:

$$0000\ 0001\ 0011\ 0110_2 = 136_{16} = 310_{10}$$

Interpreting the output as a signed integer

When interpreting the addition of two 8-bit binary integers as a signed integer, we must decide:

(i) the sign of the answer (positive or negative), and
(ii) whether the 8-bit output needs to be extended to 16 bits

For these tasks we need the N and V flags.

- The N flag is bit 7 (MSB) of the 8-bit output.
- The V flag is equal to (carry into bit 7)$\oplus$(carry out of bit7)

Interpreting the output as a signed integer

(i) Sign of the answer:
N is often called the "sign bit" because N = 1 indicates that the number is negative and N = 0 indicates the number is non-negative.
In the present context the sign depends on both N and V.

Put in very simple terms, sometimes N tells us the wrong answer, that is N may be "telling lies"; and V is the "lie-detector". V = 1 means "N is telling lies" and V = 0 means "N is not telling lies". (This is why V is sometimes called the "sign bit error flag".)

In summary, we have:

- If V = 0, then N = 1 means the answer is negative, and N = 0 means that the answer is positive.
- If V = 1, then N = 1 means the answer is positive, and N = 0 means that the answer is negative.

Interpreting the output as a signed integer

(ii) Possible extension of 8-bit output:

With 8 bits we can represent signed integers between –128 and +127.

Sometimes the correct answer lies beyond the ends of this range, and we need to use a 16-bit 2's complement number to represent the correct answer.

This can happen when we add two positive numbers or two negative numbers (but not when we add a positive and a negative number).

Interpreting the output as a signed integer

The V flag indicates whether the 8-bit output needs to be extended to 16 bits:

If V = 0, the interpretation of 8-bit output as a 2's complement number gives the correct answer, and so it is not necessary to extend the 8-bit output to 16 bits.

If V = 1, the interpretation of 8-bit output as a 2's complement number does not give the correct answer, and so the 8-bit output needs to be extended to 16 bits.

This is why V is sometimes called the "2's complement overflow flag".

The "questions that flags N and V are asked"

It may help to think of flags N and V as replying to certain questions, as follows.

N is asked: "Is the answer negative?"

N = 1 is the reply "Yes, the answer is negative"

N = 0 is the reply "No, the answer is not negative."

However N may be telling lies.

V is asked:

"Is N telling lies?" and

"Does the 8-bit output need to be extended to 16 bits?"

V = 1 is the reply

- "Yes, N is telling lies", and
- "Yes, the answer needs to be extended."

V = 0 is the reply

- "No, N is not telling lies" and
- "No, the answer does not need to be extended."

How to extend to 16 bits

There is one more rule that needs to be used.

When the 8-bit output needs to be extended to 16 bits, eight extra digits are placed at the front.

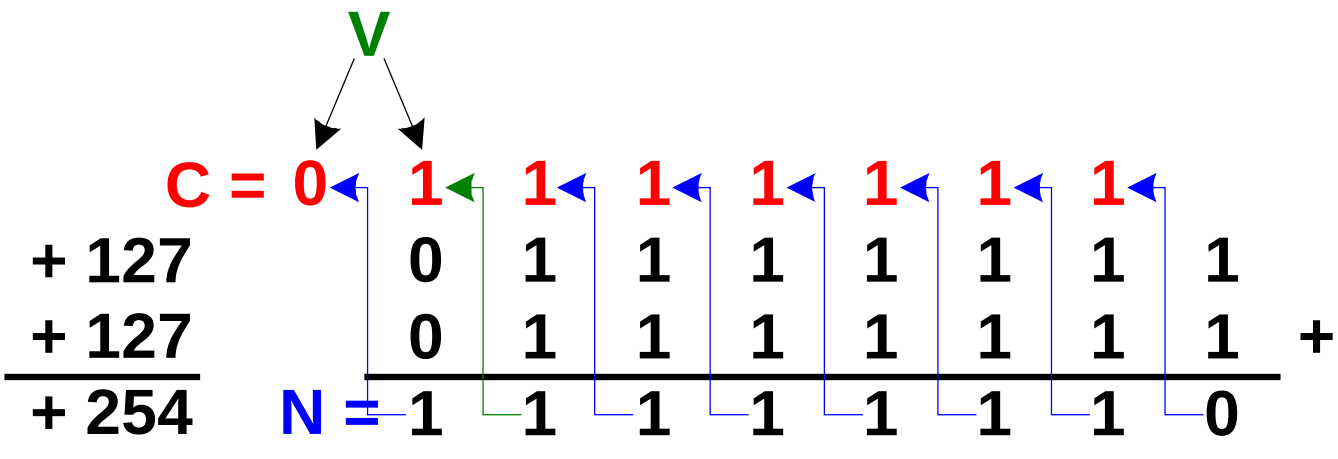
But it depends on whether the answer is positive or negative as to whether we use 0's or 1's to "pad out" the extra digits.

The rule is:

- If the answer is positive and the 8-bit output needs to be extended to 16 bits, then pad with 0's.
- If the answer is negative and the 8-bit output needs to be extended to 16 bits, then pad with 1's.

Summarising the N & V flags

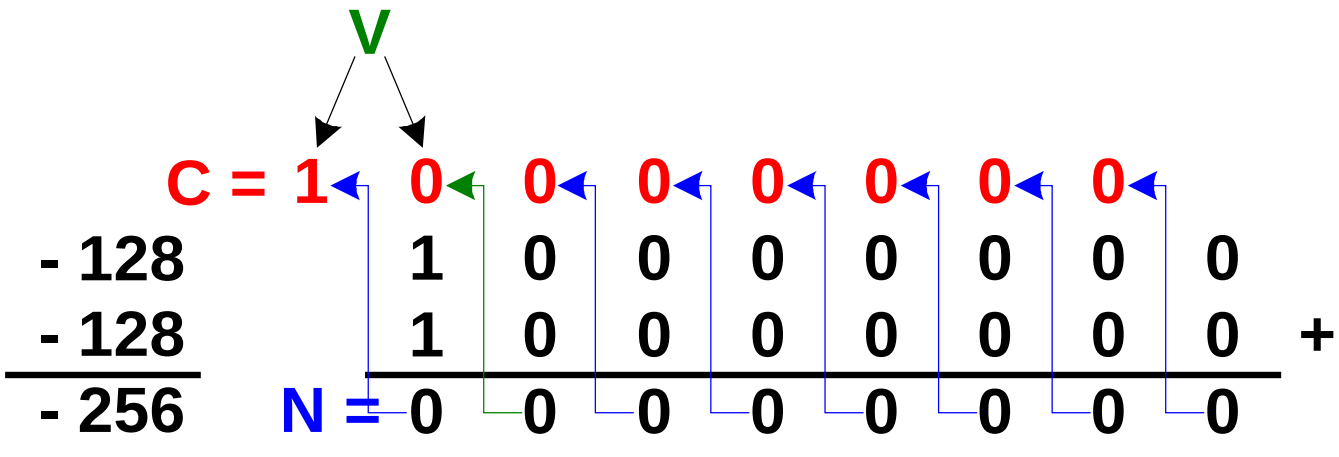
Two big positive numbers:



C	N	V
0	1	1

Summarising the N & V flags

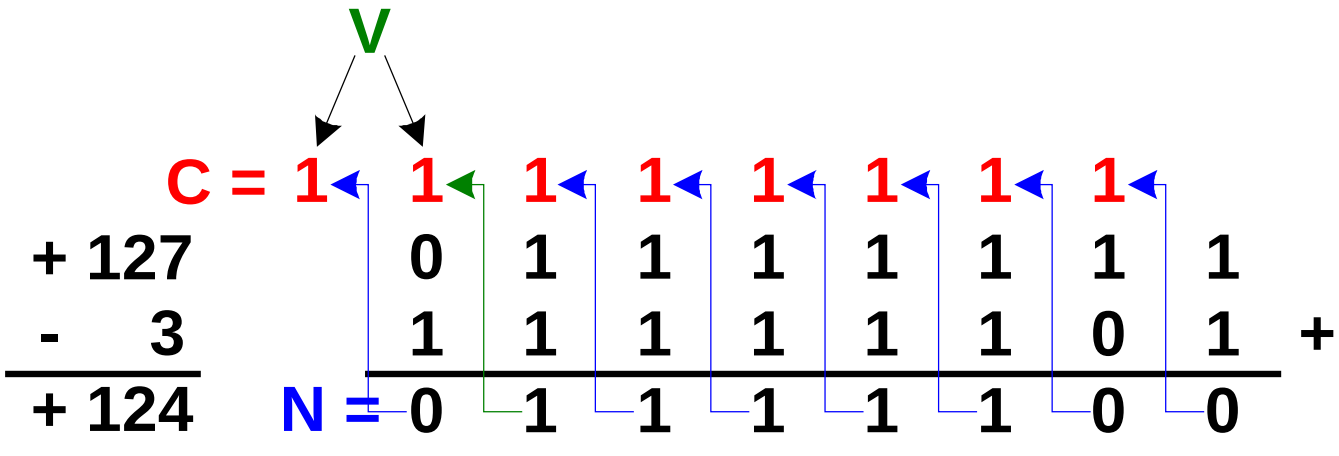
Two big negative numbers:



C	N	V
1	0	1

Summarising the N & V flags

Big positive number and small negative number:

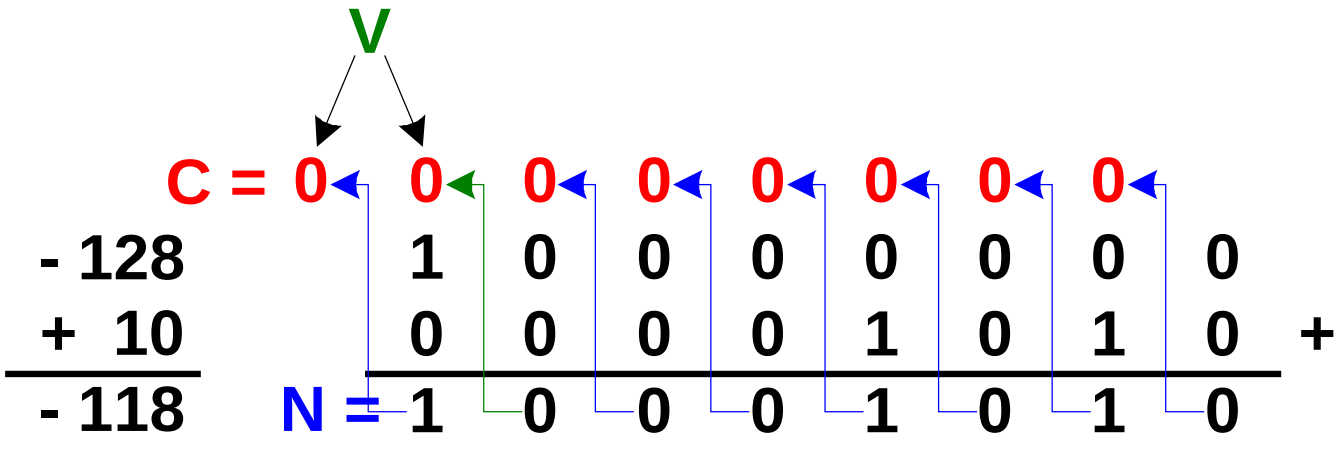


+

C	N	V
1	0	0

Summarising the N & V flags

Big negative number and small positive number:



C	N	V
0	1	0

Summarising the N & V flags

There are four different types of interpretations:

8-bit output	N & V flags	Result
1*** ****	N = 1, V = 0	negative, no ext'n needed
0*** ****	N = 0, V = 0	positive, no ext'n needed
1*** ****	N = 1, V = 1	positive, extend with 0's
0*** ****	N = 0, V = 1	negative, extend with 1's

Summarising the N & V flags

Example 1
When two signed integers are added, the 8-bit output and the N and V flags are as shown. Interpret the result as a signed integer.

1110 1101 N = 1 and V = 0

Solution:

N = 1: "the answer is negative"

V = 0: "N is not telling lies", so the answer IS negative

Also the 8-bit output does NOT need to be extended

The 2's complement number is	1110 1101
Its 1's complement is	0001 0010
Add 1	1
	0001 0011

$0001\ 0011_2 = 19_{10}$ so the answer is –19

Summarising the N & V flags

Example 2
When two signed integers are added, the 8-bit output and the N and V flags are as shown. Interpret the result as a signed integer.

0011 0101 N = 0 and V = 0

Solution:

N = 0: "the answer is positive"
V = 0: "N is not telling lies",

so the answer IS positive also the 8-bit output does NOT need to be extended.
Since the answer is positive, its 2's complement representation is just its ordinary binary representation.

$0011\ 0101_2 = 53_{10}$ so the answer is 53_{10}.

Summarising the N & V flags

Example 3

When two signed integers are added, the 8-bit output and the N and V flags are as shown. Interpret the result as a signed integer.

1010 1101 N = 1 and V = 1

Solution:

N = 1: "the answer is negative"

V = 1: "N is telling lies",

so the answer is positive also the 8-bit output needs to be extended to 16 bits, which gives

0000 0000 1010 1101

Since the answer is positive, its 16-bit 2's complement representation is just its ordinary binary representation.

$0000\ 0000\ 1010\ 1101_2 = 173_{10}$ So the answer is 173_{10}

Summarising the N & V flags

Example 4

When two signed integers are added, the 8-bit output and the N and V flags are as shown. Interpret the result as a signed integer.

0110 0010 N = 0 and V = 1

Solution:

N = 0: "the answer is positive"

V = 1: "N is telling lies",

so the answer is negative also the 8-bit output needs to be extended to 16 bits, which gives

1111 1111 0110 0010

The 2's complement number is 1111 1111 0110 0010

Its 1's complement is 0000 0000 1001 1101

Add 1 1

0000 0000 1001 1110

$0000\ 0000\ 1001\ 1110_2 = 158_{10}$ so the answer is -158_{10}

Looking at both interpretations

In the following examples we look at the result of the addition of two 8 bit binary numbers.
We are not told whether these two numbers were unsigned or signed integers.

Our task is to interpret the result:

(i) in terms of unsigned integers, and
(ii) in terms of signed integers.

Looking at both interpretations

Example 1
Sum of 1001 1111 and 0111 1011
The output from the microprocessor would be just 8-bits and the flags:

		C N V
0001 1010	and	1 0 0

(i) Interpreting in terms of unsigned integers, we notice that C = 1 and so the correct answer does not fit into 8 bits. So, extending to 16 bits, the correct answer is:

$$0000\ 0001\ 0001\ 1010_2 = 011A_{16} = 282_{10}$$

[Check: Interpreting as unsigned integers, the numbers being added are 159 and 123, whose sum is 282, which lies outside the range of 8-bits.]

Looking at both interpretations

Example 1
Sum of 1001 1111 and 0111 1011
The output from the microprocessor would be just 8-bits and the flags:

		C	N	V
0001 1010	and	1	0	0

(ii) Interpreting in terms of signed integers, we notice that N = 0 and V = 0, so the correct answer is positive and fits into 8 bits. So the correct answer is:

$$0001\ 1010_2 = 1A_{16} = 26_{10}$$

[Check: Interpreting as signed integers, the numbers being added are –97 and 123, whose sum is 26, which lies within the range for 8-bits.]

Looking at both interpretations

Example 2
Sum of 1000 1111 and 1001 1011
The output from the microprocessor would be just 8-bits and the flags:

		C	N	V
0010 1010	and	1	0	1

(i) Interpreting in terms of unsigned integers, we notice that C = 1 and so the correct answer does not fit into 8 bits. So, extending to 16 bits, the correct answer is:

$$0000\ 0001\ 0010\ 1010_2 = 012A_{16} = 298_{10}$$

[Check: Interpreting as unsigned integers, the numbers being added are 143 and 155, whose sum is 298, which lies outside the range of 8-bits.]

Looking at both interpretations

Example 2
Sum of 1000 1111 and 1001 1011
The output from the microprocessor would be just 8-bits and the flags:

		C N V
0010 1010	and	1 0 1

(ii) Interpreting in terms of signed integers, we notice that N = 0 and V = 1, so the correct answer is negative and does not fit into 8 bits. Therefore we must extend to 16 bits by adding extra 1's in front. So the correct answer is:

$$1111\ 1111\ 0010\ 1010_2 = FF2A_{16} = -214_{10}$$

[Check: Interpreting as signed integers, the numbers being added are –113 and –101, whose sum is –214 , which lies outside the range for 8-bits.]

Looking at both interpretations

Example 3
Sum of 0101 0101 and 0011 1001
The output from the microprocessor would be just 8-bits and the flags:

		C	N	V
1000 1110	and	0	1	1

(i) Interpreting in terms of unsigned integers, we notice that C = 0 and so the correct answer fits into 8 bits. So the correct answer is:

$$1000\ 1110_2 = 8E_{16} = 142_{10}$$

[Check: Interpreting as unsigned integers, the numbers being added are 85 and 57, whose sum is 142, which lies inside the range of 8-bits.]

Looking at both interpretations

Example 3
Sum of 0101 0101 and 0011 1001
The output from the microprocessor would be just 8-bits and the flags:

		C N V
1000 1110	and	0 1 1

(ii) Interpreting in terms of signed integers, we notice that N = 1 and V = 1, so the correct answer is positive and does not fit into 8 bits. Therefore we must extend to 16 bits by adding extra 0's in front. So the correct answer is:

$$0000\ 0000\ 1000\ 1110_2 = 008E_{16} = 142_{10}$$

[Check: Interpreting as signed integers, the numbers being added are 85 and 57, whose sum is 142, which lies inside the range for 8-bits.]

Looking at both interpretations

Example 4
Sum of 1010 0100 and 0100 1001
The output from the microprocessor would be just 8-bits and the flags:

		C N V
1110 1101	and	0 1 0

(i) Interpreting in terms of unsigned integers, we notice that C = 0 and so the correct answer fits into 8 bits.
So the correct answer is:

$$1110\ 1101_2 = ED_{16} = 237_{10}$$

[Check: Interpreting as unsigned integers, the numbers being added are 164 and 73, whose sum is 237, which lies inside the range of 8-bits.]

Looking at both interpretations

Example 4
Sum of 1010 0100 and 0100 1001
The output from the microprocessor would be just 8-bits and the flags:

		C N V
1110 1101	and	0 1 0

(ii) Interpreting in terms of signed integers, we notice that N = 1 and V = 0, so the correct answer is negative and does fit into 8 bits.
So the correct answer is:

$$1110\ 1101_2 = ED_{16} = -19_{10}$$

[Check: Interpreting as signed integers, the numbers being added are –92 and 73, whose sum is –19, which lies inside the range for 8-bits.]

The End