# A Note on Carry and Overflow

Paul Ossientis

April 16, 2017

## 1 Introduction

In this note we assume given a natural number $q = 2^n$ where $n \in \{8, 16, 32, 64\}$. We denote $\mathbf{Z}_q$ the ring of integers modulo $q$. The purpose of this note is to provide a formal presentation of the mathematics underlying some operations of a computer's CPU and explain the notions of *carry* and *overflow* within that formal framework. It all starts with the realization that computer hardware is designed to perfectly perform the operation of addition $+$ on the ring $\mathbf{Z}_q$. There is no approximation, there is no error, no overflow, the result is always perfect and exact: computer hardware naturally operates on $\mathbf{Z}_q$. It is also very good at inverting bits: so given $x \in \mathbf{Z}_q$ it can easily compute $\neg x$. Furthermore, since $x + \neg x = q - 1$ we have the equality $-x = \neg x + 1$. It follows that a CPU can easily (and exactly) compute the opposit $-x$ of any $x \in \mathbf{Z}_q$, simply by incrementing $\neg x$. Hence we see that not only is the addition $+$ a natural primitive of computer hardware, but so is the subtraction $-$ defined by $x - y = x + (-y)$ for all $x, y \in \mathbf{Z}_q$.

**Definition 1** *For all $x \in \mathbf{Z}_q$, we define $x^*$ the unique integer with the property:*

$$x^* = x \ mod \ q \quad and \quad x^* \in [0, q[$$

## 2 Carry for Addition

Users of computer hardware are not interested in $\mathbf{Z}_q$. One of their first interests is to perform the operation of addition $+$ on the set of natural numbers $\mathbf{N}$, an operation commonly referred to as *unsigned addition* by computer scientists. While it is possible (and indeed is a built-in feature of many modern computer languages such as Python or Haskell) to handle every possible values of $\mathbf{N}$ (as permitted by a computer's memory), for historical reasons and the purpose of this note on *carry* and *overflow*, it is important to restrict our attention to natural numbers which can be represented as *unsigned integers* within an $n$-bits register. These *representable* natural numbers are exactly those belonging the interval $[0, q[$ and the representation is exactly the associated integer modulo $q$. In other words, the *representable* natural numbers are the range of the mapping $x \to x^*$ from $\mathbf{Z}_q$ to $\mathbf{N}$, and for all $x \in \mathbf{Z}_q$, $x$ is the representation (in computer hardware) of the natural number $x^*$. Now, given two *representable* natural numbers $x^*$ and $y^*$, the users of computer hardware are interested in computing the sum $x^* + y^*$, while their machine only knows about $x + y$. Luckily, the following proposition shows that computing $x + y$ allows us to infer the value of $x^* + y^*$, provided the latter is also *representable*:

**Proposition 1** *For all $x, y \in \mathbf{Z}_q$ the following are equivalent:*

$$\begin{array}{ll} (i) & x^* + y^* \in [0, q[ \\ (ii) & (x + y)^* = x^* + y^* \end{array}$$

**Proof**
For all $x \in \mathbf{Z}_q$, from definition (1) $x^*$ is an element of $[0, q[$. It follows that $(i)$ is an immediate consequence of $(ii)$. Conversely, if we assume that $(i)$ is true, then since it is clear that $x^* + y^* = x + y$ modulo $q$, we conclude from definition (1) that $(x + y)^* = x^* + y^*$, which completes our proof. **.**

Hence we see that as long as $x^* + y^*$ is a *representable* natural number, it does not matter that our CPU should only know about addition in $\mathbf{Z}_q$: adding the two representations $x$ and $y$ gives us a representation of $x^* + y^*$ which is as good as we can hope. However, there are cases when $x^* + y^*$ is not a *representable* natural number, in which case equality $(ii)$ of proposition (1) does not hold. One way to think about this situation is saying that *the result of $x + y$ is wrong*. As it turns out, there is nothing wrong with $x + y$ which is a perfectly correct answer to the question of adding two numbers in $\mathbf{Z}_q$. However, $x + y$ is not a representation of $x^* + y^*$, and in that sense, it is clearly wrong. This is where the notion of *carry for addition* naturally comes in:

**Definition 2** *We call* carry for addition *the map $c : \mathbf{Z}_q \times \mathbf{Z}_q \to 2$ defined by:*

$$c(x, y) = 1 \quad \Leftrightarrow \quad x^* + y^* \notin [0, q[$$

By convention we denote $2 = \{0, 1\}$ which is our choice of boolean type, and the *carry for addition* is therefore a boolean function defined on the cartesian product $\mathbf{Z}_q \times \mathbf{Z}_q$, the purpose of which is to flag any situation where the result of $x + y$ is *wrong* or (to phrase it more accurately) any situation where $x + y$ is not a representation of the natural number $x^* + y^*$.

So we now understand what the *carry* is, a notion which we have defined in the context of unsigned addition. As we shall see, there will be a notion of carry for (unsigned) subtraction, and similar notions for signed addition and subtraction (called *overflow* rather than *carry* for signed operations). What all these notions have in common is their purpose: to warn a user of computer hardware that the result of a primitive operation perfomed by the CPU on elements of $\mathbf{Z}_q$ (which are representations of elements of $\mathbf{N}$ or $\mathbf{Z}$ as the case may be), does not yield a representation of the result to the corresponding operation on natural numbers or signed integers. In short, *carry* and *overflow* are designed to tell us when the result of an operation is *wrong*.

For those writing computer software, the ability to compute the carry flag $c(x, y)$ is crucial, as we need to know when $x + y$ ceases to be an accurate representation of the result we care about. For this reason, designers of computer hardware have made the computation of the carry flag for addition, one of the fundamental primitives of a CPU. So whenever an addition $x+y$ is perfomed, the carry flag will be set or cleared, depending on whether $c(x, y) = 1$ or $c(x, y) = 0$. The ability to test this carry flag and introduce branching instructions in the code which depend on the outcome is also one of the primitives of computer hardware. While we have not yet discussed the *carry* following a subtraction, this carry is essentially the same bit being set or cleared after the operation is performed, and as will shall see the ability to test this flag allows us to compare:

**Definition 3** *Given $x, y \in \mathbf{Z}_q$ we say that $x \leq y$ if and only if $x^* \leq y^*$.*

Mathematically speaking, the order $\leq$ thus defined on $\mathbf{Z}_q$ may not be very interesting as it is not compatible with addition (the inequality $x \leq y$ does not imply $x + z \leq y + z$). However, it is interesting to us as it is a relation which can be tested by a CPU (as mentioned, it can be tested using the *carry for subtraction*). It is also interesting because it is defined to perfectly mimic the inequality $x^* \leq y^*$ satisfied or not by the natural numbers represented by $x$ and $y$. Note that having defined the relation $\leq$ on $\mathbf{Z}_q$, we have implicitely defined $>$, $\geq$ and $<$, and since we can test these conditions (the *zero* flag allows us to test equality), we are able to compute the corresponding binary min and max functions. So while definition (2) relies on the quantity $x^* + y^*$ which we cannot compute, the following proposition defines the *carry for addition* in terms of conditions which can easily be tested in assembly language:

**Proposition 2** *For all $x, y \in \mathbf{Z}_q$, the following are equivalent:*

$$\begin{array}{rl}
(i) & c(x, y) = 1 \\
(ii) & x + y < \min(x, y) \\
(iii) & x + y < \max(x, y) \\
(iv) & x + y < x \\
(v) & x + y < y
\end{array}$$

*where $c : \mathbf{Z}_q \times \mathbf{Z}_q \to 2$ is the carry for addition of definition (2), and the order $<$ is derived from the relation $\leq$ of definition (3) defined on $\mathbf{Z}_q$.*

**Proof**
Since for all $x, y \in \mathbf{Z}_q$, $\min(x, y) \leq x, y \leq \max(x, y)$ we immediately have the implications $(ii) \Rightarrow (iv)$, $(ii) \Rightarrow (v)$, $(iv) \Rightarrow (iii)$ and $(v) \Rightarrow (iii)$. In order to complete the proof, it remains to show that $(i) \Rightarrow (ii)$ and $(iii) \Rightarrow (i)$. We first show that $(i) \Rightarrow (ii)$. So we assume that $c(x, y) = 1$, and need to show that $x + y < \min(x, y)$. In other words we need to show that both inequalities $x + y < x$ and $x + y < y$ hold. However from definition (2), our assumption is equivalent to $x^* + y^* \notin [0, q[$, and we know from definition (1) that both $x^*$ and

$y^*$ are elements of $[0, q[$. It follows that $x^* + y^*$ must be an element of $[0, 2q[$ without being an element of $[0, q[$. Hence it must be an element of $[q, 2q[$, from which we see that $x^* + y^* - q$ is an element of $[0, q[$, while being equal to $x + y$ modulo $q$. From definition (1) it follows that $(x + y)^* = x^* + y^* - q$, and since both $y^* - q < 0$ and $x^* - q < 0$ we obtain $(x + y)^* < x^*$ and $(x + y)^* < y^*$ which by virtue of definition (3) is equivalent to $(x + y) < x$ and $(x + y) < y$ as requested. We now prove that $(iii) \Rightarrow (i)$. So we assume that $x + y < \max(x, y)$ and we need to show that $c(x, y) = 1$, or equivalently that $x^* + y^* \notin [0, q[$. However, our assumption implies that $x + y < x$ or $x + y < y$. So we shall distinguish two cases: first we assume that $x + y < x$. From definiton (3), this means that $(x + y)^* < x^*$. Hence, it is impossible that $x^* + y^* \in [0, q[$ as this would imply that $(x + y)^* = x^* + y^*$ (being equal to $x + y$ modulo $q$) and consequently $x^* + y^* < x^*$, yielding the contradiction $y^* < 0$. Likewise, if we assume that $x + y < y$ then $x^* + y^* \in [0, q[$ implies the contradiction $x^* < 0$. .

## 3 Carry for Subtraction

Given two *representable* natural numbers $x^*$ and $y^*$, users of computer hardware commonly want to compute their difference $x^* - y^*$. Unfortunately, the only result available to them is the difference $x - y \in \mathbf{Z}_q$ performed by the CPU on their representations $x$, $y$. In light of our analysis for addition, the first question we should ask is whether $x - y$ is a representation of $x^* - y^*$. As before, as long as the sought out result is *representable*, the answer is 'yes':

**Proposition 3** *For all $x, y \in \mathbf{Z}_q$, the following are equivalent:*

$$\begin{array}{ll} (i) & x^* - y^* \in [0, q[ \\ (ii) & x^* \geq y^* \\ (iii) & (x - y)^* = x^* - y^* \end{array}$$

**Proof**
Since both $x^*$ and $y^*$ are always elements of $[0, q[$, the difference $x^* - y^*$ is always an element of $] - q, q[$. Hence the equivalence between $(i)$ and $(ii)$ is clear. It remains to show the equivalence between $(ii)$ and $(iii)$. First we show $(ii) \Rightarrow (iii)$. So suppose $x^* \geq y^*$. Since $x^* - y^*$ is an element of $] - q, q[$, it is in fact an element of $[0, q[$ while being equal to $x - y$ modulo $q$. It follows from definition (1) that $x^* - y^* = (x - y)^*$ as requested. Conversely, if we assume that $x^* - y^* = (x - y)^*$ then in particular $x^* - y^*$is an element of $[0, q[$, so $x^* \geq y^*$. .

This in turn immediately motivates the following definition, whose purpose it is to highlight any situation where the result of $x - y$ is *wrong*, that is a situation where $x - y$ ceases to be a representation of the difference $x^* - y^*$.

**Definition 4** *We call* carry for subtraction *the map $c : \mathbf{Z}_q \times \mathbf{Z}_q \to 2$ defined by:*

$$c(x, y) = 1 \quad \Leftrightarrow \quad x^* < y^*$$

In similar fashion to proposition (2) we state:

**Proposition 4** *For all $x, y \in \mathbf{Z}_q$, the following are equivalent:*

$$\begin{array}{ll} (i) & c(x,y) = 1 \\ (ii) & x < y \end{array}$$

*where $c : \mathbf{Z}_q \times \mathbf{Z}_q \to 2$ is the carry for subtraction of definition (4), and the order $<$ is derived from the relation $\leq$ of definition (3) defined on $\mathbf{Z}_q$.*

**Proof**
Immediate from definition (4) and the fact that $x < y$ is equivalent $x^* < y^*$. ∎

The point of proposition (4) is to be able to validate in software the consistency between testing for the carry flag after a subtraction $x - y$, and testing for the condition $x < y$. Whenever a 'sub x y' instructtion is performed, the carry flag should be set if and only if the condition $x < y$ holds.

## 4 Overflow for Addition

Users of computer hardware are not interested in $\mathbf{Z}_q$, and they may not be interested in the set of natural numbers $\mathbf{N}$ either. Given two integers $u, v \in \mathbf{Z}$ they often want to compute their sums $u + v$, an operation commonly referred to as *signed addition* by computer scientists. Unfortunately, as we can guess by now, a CPU has no interstanding of integers in $\mathbf{Z}$ (it only knows about $\mathbf{Z}_q$) let alone of how to compute their sum. We humans have to come up with a trick to obtain the answers we want from a machine which is limited to operating on $\mathbf{Z}_q$. When faced with operations on $\mathbf{N}$, we designed a successful strategy: first we restricted our domain to a subset of $\mathbf{N}$ namely those natural numbers belonging to $[0, q[$. Then we decided to give every natural number $x^*$ of this domain a representation $x$ in $\mathbf{Z}_q$. Then we let the CPU operate on the representations $x$ and $y$ of our chosen natural numbers, hoping to obtain a representation of the sum $x^* + y^*$ or the difference $x^* - y^*$ as the case may be. This strategy works well, but is not immune to failure, which led to the notions of *carry* for addition and subtraction giving software designers the ability to know if and when the CPU representations $x+y$ and $x-y$ in $\mathbf{Z}_q$ are indeed what they want. We shall now spell out a similar strategy for $\mathbf{Z}$, that of *two's complement representation*:

**Definition 5** *We say that $n \in \mathbf{Z}$ is* representable *if and only if $n \in [-q/2, q/2[$.*

Definition (5) describes our domain. We have now defined the term *representable* in the context of signed addition (addition in $\mathbf{Z}$) which should not be confused with *representable* in the context of unsigned addition (addition in $\mathbf{N}$). A *representable* natural number (aka unsigned integer) belongs to $[0, q[$, while a *representable* integer (aka signed integer) belongs to $[-q/2, q/2[$. Now given a *representable* integer in $[-q/2, q/2[$, we need to define its representation in $\mathbf{Z}_q$:

**Definition 6** *The bijection $r : \mathbf{Z}_q \to \mathbf{Z} \cap [-q/2, q/2[$ defined by:*

$$\forall x \in \mathbf{Z}_q \ , \ r(x) = \left\{ \begin{array}{lll} x^* & \text{if} & x \in [0, q/2[ \\ x^* - q & \text{if} & x \in [q/2, q[ \end{array} \right\}$$

*is called* the two's complement representation mapping. *Given $x \in \mathbf{Z}_q$, we say that $x$ is the* two's complement representation *of the integer $r(x) \in [-q/2, q/2[$.*

Now consider the simplest case when $q = 2^8$. We have 256 elements of $\mathbf{Z}_q$, from 0 to 255. According to definition (6), we have $r(0) = 0$ and $r(1) = 1$ etc. up to $r(127) = 127$. Then we see that $r(128) = -128$, $r(129) = -127$, etc. up to $r(255) = -1$. The two's complement representation of $0 \in \mathbf{Z}$ is $0 \in \mathbf{Z}_q$. The two's complement representation of $-128 \in \mathbf{Z}$ is $128 \in \mathbf{Z}_q$ and the two's complement representation of $-1 \in \mathbf{Z}$ is $255 \in \mathbf{Z}_q$.

**Proposition 5** *The map $r : \mathbf{Z}_q \to \mathbf{Z} \cap [-q/2, q/2[$ is indeed a bijection.*

**Proof**
First we show that $r$ is injective. So suppose $r(x) = r(y)$ for some $x, y \in \mathbf{Z}_q$. We need to show that $x = y$ or equivalently that $x^* = y^*$. Consider four cases:

1. $x \in [0, q/2[$ and $y \in [0, q/2[$: then $r(x) = x^*$ and $r(y) = y^*$ and the equality $r(x) = r(y)$ yield $x^* = y^*$ as requested.

2. $x \in [0, q/2[$ and $y \in [q/2, q[$: then $r(x) = x^*$ and $r(y) = y^* - q$ and the equality $r(x) = r(y)$ yields $x^* = y^* - q < 0$ which is a contradiction. Hence this case cannot occur whenever $r(x) = r(y)$.

3. $x \in [q/2, q[$ and $y \in [0, q/2[$: a similar argument shows that this case cannot occur whenever $r(x) = r(y)$.

4. $x \in [q/2, q[$ and $y \in [q/2, q[$: then $r(x) = x^* - q$ and $r(y) = y^* - q$ and the equality $r(x) = r(y)$ yields $x^* = y^*$ as requested.

We now show that $r$ is surjective. So let $n$ be an integer in $[-q/2, q/2[$. We need to show the existence of $x \in \mathbf{Z}_q$ such that $n = r(x)$. Define $x$ simply to be the class of $n$ modulo $q$. We need to check that $r(x) = n$. Consider two cases:

1. $n \in [-q/2, 0[$: then $n + q \in [q/2, q[$. Hence we see that $n + q \in [0, q[$ while being equal to $x$ modulo $q$. It follows that $x^* = n + q \in [q/2, q[$ from which we obtain $r(x) = x^* - q = n$ as requested.

2. $n \in [0, q/2[$: then $n \in [0, q[$ while being equal to $x$ modulo $q$. It follows that $x^* = n \in [0, q/2[$ from which we obtain $r(x) = x^* = n$ as requested.

.

Hence we see that the *representable* integers belonging to $[-q/2, q/2[$ are in fact the range of the two's complement representation mapping $r : \mathbf{Z}_q \to \mathbf{Z}$. If $r(x)$ and $r(y)$ are two *representable* integers, the fundamental question for us is whether $x + y$ is a representation of the sum $r(x) + r(y)$. As it turns out, the answer is once again 'yes' provided of course that $r(x) + r(y)$ is *representable*. This result may be regarded as somewhat of a miracle: despite its relative complexity, the two's complement representation allows us to compute the sum $r(x) + r(y) \in \mathbf{Z}$ simply by performing the standard operation $x + y \in \mathbf{Z}_q$.

**Proposition 6** *For all $x, y \in \mathbf{Z}_q$, the following are equivalent:*

$$\begin{array}{ll} (i) & r(x) + r(y) \in [-q/2, q/2[ \\ (ii) & r(x+y) = r(x) + r(y) \end{array}$$

**Proof**
The implication $(ii) \Rightarrow (i)$ is clear. It remains to show $(i) \Rightarrow (ii)$, for which we shall consider four different cases:

1. $x \in [0, q/2[$ and $y \in [0, q/2[$: in this case we have $r(x) = x^*$ and $r(y) = y^*$ and condition $(i)$ reduces to $x^* + y^* \in [-q/2, q/2[$ which in fact is equivalent to $x^* + y^* \in [0, q/2[$ (since $x^* + y^*$ is never negative). So if $(i)$ is true, we see that $x^* + y^* \in [0, q[$ while being equal to $x + y$ modulo $q$, which in turn implies that $(x+y)^* = x^* + y^* \in [0, q/2[$ from which we obtain $r(x+y) = (x+y)^* = x^* + y^* = r(x) + r(y)$ which is condition $(ii)$.

2. $x \in [0, q/2[$ and $y \in [q/2, q[$: in this case we have $r(x) = x^*$ and $r(y) = y^* - q$ and condition $(i)$ reduces to $x^* + y^* - q \in [-q/2, q/2[$ or equivalently $x^* + y^* \in [q/2, 3q/2[$ which is always true in this case (since $x^* \in [0, q/2[$ and $y^* \in [q/2, q[$). So we simply need to prove that $r(x+y) = r(x) + r(y)$, for which we shall consider two further cases: First we assume that $x^* + y^* - q \in [-q/2, 0[$. Then we see that $x^* + y^* \in [q/2, q[$ while being equal to $x + y$ modulo $q$. It follows that $(x+y)^* = x^* + y^* \in [q/2, q[$ and consequently $r(x+y) = (x+y)^* - q = x^* + y^* - q = r(x) + r(y)$ as requested. We now assume that $x^* + y^* - q \in [0, q/2[$. Since it is also equal to $x + y$ modulo $q$ we obtain $(x+y)^* = x^* + y^* - q \in [0, q/2[$. It follows that $r(x+y) = (x+y)^* = x^* + y^* - q = r(x) + r(y)$ as requested.

3. $x \in [q/2, q[$ and $y \in [0, q/2[$: addition being commutative in $\mathbf{Z}$ and $\mathbf{Z}_q$, we can conclude that $r(x+y) = r(x) + r(y)$ from the previous case.

4. $x \in [q/2, q[$ and $y \in [q/2, q[$: in this case we have $r(x) = x^* - q$ and $r(y) = y^* - q$ and condition $(i)$ reduces to $x^* + y^* - 2q \in [-q/2, q/2[$ which is $x^* + y^* - q \in [q/2, 3q/2[$ which is in fact equivalent to $x^* + y^* - q \in [q/2, q[$ (since $x^* \in [q/2, q[$ and $y^* \in [q/2, q[$). Hence we see that $(i)$ implies $x^* + y^* - q \in [q/2, q[$ while being equal to $x + y$ modulo $q$. It follows that $(x+y)^* = x^* + y^* - q \in [q/2, q[$ and consequently $r(x+y) = (x+y)^* - q = x^* + y^* - 2q = r(x) + r(y)$ as requested.

.

As we can see from propostion (6) there may be cases where $x + y \in \mathbf{Z}_q$ fails to be a representation of the sum $r(x) + r(y) \in \mathbf{Z}$. The condition under which this anomaly arises is called an *overflow* rather than *carry*:

**Definition 7** *We call* overflow for addition *the map $o : \mathbf{Z}_q \times \mathbf{Z}_q \to 2$ defined by:*

$$o(x, y) = 1 \quad \Leftrightarrow \quad r(x) + r(y) \notin [-q/2, q/2[$$

As before, this definition is not terribly useful, unless we have some way of determining when an overflow situation arises:

**Proposition 7** *For all $x, y \in \mathbf{Z}_q$, we have $o(x, y) = 1$ if and only if:*

$$x \in [0, q/2[ \text{ and } y \in [0, q/2[ \text{ and } x + y \in [q/2, q[$$

*or:*

$$x \in [q/2, q[ \text{ and } y \in [q/2, q[ \text{ and } x + y \in [0, q[$$

*where $o : \mathbf{Z}_q \times \mathbf{Z}_q \to 2$ is the overflow for addition of definition (7).*

**Proof**
First we assume that $x \in [0, q/2[$, $y \in [0, q/2[$ and $x + y \in [q/2, q[$, and we need to show that $o(x, y) = 1$ or equivalently $r(x) + r(y) \notin [-q/2, q/2[$. However in this case, we have $r(x) = x^*$ and $r(y) = y^*$ and $x^* + y^* \in [0, q[$ (since $x^* \in [0, q/2[$ and $y^* \in [0, q/2[$). Since $x^* + y^*$ is equal to $x + y$ modulo $q$ we obtain $(x + y)^* = x^* + y^*$ and the assumption $x + y \in [q/2, q[$ therefore yields $x^* + y^* \in [q/2, q[$ which is $r(x) + r(y) \in [q/2, q[$. So we see that $r(x) + r(y)$ does not belong to $[-q/2, q/2[$ as requested.

Next we assume that $x \in [q/2, q[$, $y \in [q/2, q[$ and $x + y \in [0, q/2[$, and we need to show that $o(x, y) = 1$ or equivalently $r(x) + r(y) \notin [-q/2, q/2[$. However in this case, we have $r(x) = x^* - q$ and $r(y) = y^* - q$ and $x^* + y^* - q \in [0, q[$ (since $x^* \in [q/2, q[$ and $y^* \in [q/2, q[$). Since $x^* + y^* - q$ is equal to $x + y$ modulo $q$ we obtain $(x + y)^* = x^* + y^* - q$ and the assumption $x + y \in [0, q/2[$ therefore yields $x^* + y^* - q \in [0, q/2[$ which is $r(x) + r(y) \in [-q, -q/2[$. So we see that $r(x) + r(y)$ does not belong to $[-q/2, q/2[$ as requested.

Conversely we assume that $o(x, y) = 1$ or equivalently that $r(x) + r(y) \notin [-q/2, q/2[$ and we need to show that one of the two propeties of proposition (7) is satisfied. We shall distinguish four cases:

1. $x \in [0, q/2[$ and $y \in [0, q/2[$: in this case, we only need to prove that $x + y \in [q/2, q[$ in order to prove the first property of proposition (7). However, in this case we have $r(x) = x^*$ and $r(y) = y^*$ and the condition $o(x, y) = 1$ reduces to $x^* + y^* \notin [-q/2, q/2[$ which is equivalent to $x^* + y^* \notin [0, q/2[$ (since $x^* + y^*$ cannot be negative). However we know that $x^* + y^* \in [0, q[$ (since $x^* \in [0, q/2[$ and $y^* \in [0, q/2[$) and it follows that $x^* + y^* \in [q/2, q[$. Furthermore, since $x^* + y^*$ is equal to $x + y$ modulo $q$, we see that $(x + y)^* = x^* + y^*$ and consequently $(x + y)^* \in [q/2, q[$ which is $x + y \in [q/2, q[$ as requested.

2. $x \in [0, q/2[$ and $y \in [q/2, q[$: in this case we have $r(x) = x^*$ and $r(y) = y^* - q$ and the condition $o(x, y) = 1$ reduces to $x^* + y^* - q \notin [-q/2, q/2[$ which cannot happen (since $x^* \in [0, q/2[$ and $y^* \in [q/2, q[$). So this case can never happen when $o(x, y) = 1$ and there is nothing further to prove.

3. $x \in [q/2, q[$ and $y \in [0, q/2[$: likewise, having made the assumption $o(x, y) = 1$ this case can never happen and there is nothing to prove.

4. $x \in [q/2, q[$ and $y \in [q/2, q[$: in this case, we only need to prove that $x + y \in [0, q/2[$ in order to prove the second property of proposition (7). However, in this case we have $r(x) = x^* - q$ and $r(y) = y^* - q$ and the condition $o(x, y) = 1$ reduces to $x^* + y^* - 2q \notin [-q/2, q/2[$ which is $x^* + y^* - q \notin [q/2, 3q/2[$, which is equivalent to $x^* + y^* - q \notin [q/2, q[$ (since $x^* \in [q/2, q[$ and $y^* \in [q/2, q[$ and $x^* + y^* - q$ belongs to $[0, q[$ anyway). Hence, having assumed that $o(x, y) = 1$ we see that $x^* + y^* - q \in [0, q/2[$. Furthermore, since $x^* + y^* - q$ is equal to $x + y$ modulo $q$, we obtain $(x + y)^* = x^* + y^* - q$ and consequently $(x + y)^* \in [0, q/2[$ which is $x + y \in [0, q/2[$ as requested.

.

For all $x \in \mathbf{Z}_q$, we have $r(x) = x^* \geq 0$ when $x \in [0, q/2[$ and furthermore $r(x) = x^* - q < 0$ when $x \in [q/2, q[$. So $x$ represents a non-negative integer when $x \in [0, q/2[$ and it represents a negative integer when $x \in [q/2, q[$. If we think of the binary representation of $x \in \mathbf{Z}_q$, it is easy to see that the condition $x \in [0, q/2[$ corresponds to the highest order bit being cleared, while $x \in [q/2, q[$ corresponds to the highest order bit being set. This explains why the highest order bit is commonly called the *sign bit*. For this reason we define:

**Definition 8** *We call* sign bit mapping *the map $s : \mathbf{Z}_q \to 2$ defined by:*

$$s(x) = 1 \;\Leftrightarrow\; x \in [q/2, q[$$

In the light of definition (8) we can rephrase proposition (7) as follows:

**Proposition 8** *For all $x, y \in \mathbf{Z}_q$, we have $o(x, y) = 1$ if and only if:*

$$s(x) = 0 \text{ and } s(y) = 0 \text{ and } s(x + y) = 1$$

*or:*

$$s(x) = 1 \text{ and } s(y) = 1 \text{ and } s(x + y) = 0$$

*where $o : \mathbf{Z}_q \times \mathbf{Z}_q \to 2$ is the overflow for addition of definition (7), while $s : \mathbf{Z}_q \to 2$ is the sign bit mapping of definition (8).*

**Proof**
Follows immediately from proposition (7) and definition (8). .

## 5 Overflow for subtraction

Here we are now asking ourselves the same question again: when does the native CPU operation on $\mathbf{Z}_q$ yield a representation of the result from the corresponding operation on $\mathbf{Z}$? As before, as long as the result from the operation on $\mathbf{Z}$ is *representable*, the native CPU operation on $\mathbf{Z}_q$ provides a representation of it. Given two *representable* integers $r(x)$ and $r(y)$ such that $r(x) - r(y)$ is itself *representable*, the equation $r(x - y) = r(x) - r(y)$ fundamentally says: $x - y \in \mathbf{Z}_q$ is a representation (in computer hardware) of the difference $r(x) - r(y) \in \mathbf{Z}$.

**Proposition 9** *For all $x, y \in \mathbf{Z}_q$, the following are equivalent:*

$$\begin{array}{ll} (i) & r(x) - r(y) \in [-q/2, q/2[ \\ (ii) & r(x-y) = r(x) - r(y) \end{array}$$

**Proof**
The implication $(ii) \Rightarrow (i)$ is clear, so we only need to prove $(i) \Rightarrow (ii)$. So we assume that $r(x) - r(y) \in [-q/2, q/2[$ and we need to show the equality $r(x-y) = r(x) - r(y)$. We shall distinguish four cases:

1. $x \in [0, q/2[$ and $y \in [0, q/2[$: in this case we have $r(x) = x^*$ and $r(y) = y^*$ and our assumption reduces to $x^* - y^* \in [-q/2, q/2[$ which is necessarily true in this case (there is no need to assume it) since $x^* \in [0, q/2[$ and $y^* \in [0, q/2[$ implies $x^* - y^* \in ]-q/2, q/2[$. So we need to prove that $r(x-y) = x^* - y^*$. We shall distinguish two further cases: first we assume that $x^* - y^* \in ]-q/2, 0[$. Then $x^* - y^* + q$ belongs to $]q/2, q[$ while being equal to $x - y$ modulo $q$. It follows that $(x-y)^* = x^* - y^* + q \in ]q/2, q[$ and consequently $r(x-y) = (x-y)^* - q = x^* - y^* = r(x) - r(y)$ as requested. We now assume that $x^* - y^* \in [0, q/2[$. In this case we obtain $(x-y)^* = x^* - y^* \in [0, q/2[$ and consequently we have the equation $r(x-y) = (x-y)^* = x^* - y^* = r(x) - r(y)$ as requested.

2. $x \in [0, q/2[$ and $y \in [q/2, q[$: in this case we have $r(x) = x^*$ as well as $r(y) = y^* - q$ and our assumption reduces to $x^* - y^* + q \in [-q/2, q/2[$, while we need to prove that $r(x-y) = x^* - y^* + q$. However from $x^* \in [0, q/2[$ and $y^* \in [q/2, q[$ we obtain $x^* - y^* \in ]-q, 0[$ which is $x^* - y^* + q \in ]0, q[$. It follows that $x^* - y^* + q \in ]0, q/2[$ while being equal to $x - y$ modulo $q$. Hence we see that $(x-y)^* = x^* - y^* + q \in ]0, q/2[$ and consequently $r(x-y) = (x-y)^* = x^* - y^* + q$ as requested.

3. $x \in [q/2, q[$ and $y \in [0, q/2[$: in this case we have $r(x) = x^* - q$ and $r(y) = y^*$ and our assumption reduces to $x^* - y^* - q \in [-q/2, q/2[$ which is $x^* - y^* \in [q/2, 3q/2[$, while we need to prove that $r(x-y) = x^* - y^* - q$. However from $x^* \in [q/2, q[$ and $y^* \in [0, q/2[$ we obtain $x^* - y^* \in ]0, q[$. It follows that $x^* - y^* \in [q/2, q[$ while being equal to $x - y$ modulo $q$. Hence we see that $(x-y)^* = x^* - y^* \in [q/2, q[$ and consequently we have $r(x-y) = (x-y)^* - q = x^* - y^* - q$ as requested.

4. $x \in [q/2, q[$ and $y \in [q/2, q[$: in this case we have $r(x) = x^* - q$ as well as $r(y) = y^* - q$ and our assumption reduces to $x^* - y^* \in [-q/2, q/2[$, while we need to prove that $r(x-y) = x^* - y^*$. However, from $x^* \in [q/2, q[$ and $y^* \in [q/2, q[$ we obtain $x^* - y^* \in ]-q/2, q/2[$ so our assumption is always true in this case (there is no need to assume it). We shall distinguish two further cases: first we assume that $x^* - y^* \in ]-q/2, 0[$. Then $x^* - y^* + q \in ]q/2, q[$ while being equal to $x - y$ modulo $q$. It follows that we have $(x-y)^* = x^* - y^* + q \in ]q/2, q[$ and consequently we conclude $r(x-y) = (x-y)^* - q = x^* - y^*$ as requested. We now assume that

$x^* - y^* \in [0, q/2[$. Then $(x - y)^* = x^* - y^* \in [0, q/2[$ and consequently $r(x - y) = (x - y)^* = x^* - y^*$ as requested.

. In the light of proposition (9) we now define:

**Definition 9** *We call* overflow for subtraction *the map* $o\colon \mathbf{Z}_q \times \mathbf{Z}_q \to 2$ *defined by:*

$$o(x, y) = 1 \quad \Leftrightarrow \quad r(x) - r(y) \notin [-q/2, q/2[$$

Similarly to proposition (7) we now characterize the overflow for subtraction:

**Proposition 10** *For all $x, y \in \mathbf{Z}_q$, we have $o(x, y) = 1$ if and only if:*

$$x \in [0, q/2[ \text{ and } y \in [q/2, q[ \text{ and } x - y \in [q/2, q[$$

*or:*

$$x \in [q/2, q[ \text{ and } y \in [0, q/2[ \text{ and } x - y \in [0, q[$$

*where $o : \mathbf{Z}_q \times \mathbf{Z}_q \to 2$ is the overflow for subtraction of definition (7).*

**Proof**
First we assume that $x \in [0, q/2[$, $y \in [q/2, q[$ and $x - y \in [q/2, q[$, and we need to show that $o(x, y) = 1$ or equivalently $r(x) - r(y) \notin [-q/2, q/2[$. However in this case, we have $r(x) = x^*$ and $r(y) = y^* - q$ and $x^* - y^* + q \in ]0, q[$ (since $x^* \in [0, q/2[$ and $y^* \in [q/2, q[$). Since $x^* - y^* + q$ is equal to $x - y$ modulo $q$ we obtain $(x - y)^* = x^* - y^* + q$ and the assumption $x - y \in [q/2, q[$ therefore yields $x^* - y^* + q \in [q/2, q[$ which is $r(x) - r(y) \in [q/2, q[$. So we see that $r(x) - r(y)$ does not belong to $[-q/2, q/2[$ as requested.

Next we assume that $x \in [q/2, q[$, $y \in [0, q/2[$ and $x - y \in [0, q/2[$, and we need to show that $o(x, y) = 1$ or equivalently $r(x) - r(y) \notin [-q/2, q/2[$. However in this case, we have $r(x) = x^* - q$ and $r(y) = y^*$ and $x^* - y^* \in ]0, q[$ (since $x^* \in [q/2, q[$ and $y^* \in [0, q/2[$). Since $x^* - y^*$ is equal to $x - y$ modulo $q$ we obtain $(x - y)^* = x^* - y^*$ and the assumption $x - y \in [0, q/2[$ therefore yields $x^* - y^* \in [0, q/2[$ which is $r(x) - r(y) \in [-q, -q/2[$. So we see that $r(x) - r(y)$ does not belong to $[-q/2, q/2[$ as requested.

Conversely we assume $o(x, y) = 1$ or equivalently $r(x) - r(y) \notin [-q/2, q/2[$ and we need to show that one of the two propeties of proposition (10) is satisfied. We shall distinguish four cases:

1. $x \in [0, q/2[$ and $y \in [0, q/2[$: in this case we have $r(x) = x^*$ and $r(y) = y^*$ and the condition $o(x, y) = 1$ reduces to $x^* - y^* \notin [-q/2, q/2[$ which cannot happen (since $x^* \in [0, q/2[$ and $y^* \in [0, q/2[$). So this case can never happen when $o(x, y) = 1$ and there is nothing further to prove.

2. $x \in [0, q/2[$ and $y \in [q/2, q[$: in this case, we only need to prove that $x - y \in [q/2, q[$ in order to prove the first property of proposition (10). However, in this case we have $r(x) = x^*$ and $r(y) = y^* - q$ and the condition $o(x, y) = 1$ reduces to $x^* - y^* + q \notin [-q/2, q/2[$. However we know that $x^* - y^* + q \in ]0, q[$ (since $x^* \in [0, q/2[$ and $y^* \in [q/2, q[$) and it follows that $x^* - y^* + q \in [q/2, q[$. Furthermore, since $x^* - y^* + q$ is equal to $x - y$ modulo $q$, we see that $(x - y)^* = x^* - y^* + q$ and consequently $(x - y)^* \in [q/2, q[$ which is $x - y \in [q/2, q[$ as requested.

3. $x \in [q/2, q[$ and $y \in [0, q/2[$: in this case, we only need to prove that $x - y \in [0, q/2[$ in order to prove the second property of proposition (10). However, in this case we have $r(x) = x^* - q$ and $r(y) = y^*$ and the condition $o(x, y) = 1$ reduces to $x^* - y^* - q \notin [-q/2, q/2[$ or equivalently $x^* - y^* \notin [q/2, 3q/2[$. However, we know that $x^* - y^* \in ]0, q[$ (from the conditions $x^* \in [q/2, q[$ and $y^* \in [0, q/2[$). Hence, having assumed that $o(x, y) = 1$ we see that $x^* - y^* \in ]0, q/2[$. Furthermore, since $x^* - y^*$ is equal to $x - y$ modulo $q$, we obtain $(x - y)^* = x^* - y^*$ and consequently $(x - y)^* \in ]0, q/2[$ which is $x - y \in ]0, q/2[ \subseteq [0, q/2[$ as requested.

4. $x \in [q/2, q[$ and $y \in [q/2, q[$: in this case we have $r(x) = x^* - q$ and $r(y) = y^* - q$ and the condition $o(x, y) = 1$ reduces to $x^* - y^* \notin [-q/2, q/2[$ which cannot happen (since $x^* \in [q/2, q[$ and $y^* \in [q/2, q[$). So this case can never happen when $o(x, y) = 1$ and there is nothing further to prove.

.

Proposition (10) can equivalently be expressed in terms of sign bits.

**Proposition 11** *For all $x, y \in \mathbf{Z}_q$, we have $o(x, y) = 1$ if and only if:*

$$s(x) = 0 \textit{ and } s(y) = 1 \textit{ and } s(x - y) = 1$$

*or:*

$$s(x) = 1 \textit{ and } s(y) = 0 \textit{ and } s(x - y) = 0$$

*where $o : \mathbf{Z}_q \times \mathbf{Z}_q \to 2$ is the overflow for subtraction of definition (9), while $s : \mathbf{Z}_q \to 2$ is the sign bit mapping of definition (8).*

**Proof**
Follows immediately from proposition (10) and definition (8). .