Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$
\begin{array}{lcrcl}
x & = & 107_{(10)} & \mapsto & 1\ 0\ 7 \\
y & = & 14_{(10)} & \mapsto & \underline{\ 0\ 1\ 4}+ \\
c & = & & & \\
r & = & & & \underline{}
\end{array}
$$

but it *also* applies for $b = 2$

$$
\begin{array}{lcrcl}
x & = & 107_{(10)} & \mapsto & 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1 \\
y & = & 14_{(10)} & \mapsto & \underline{\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0}+ \\
c & = & & & \\
r & = & & & \underline{}
\end{array}
$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$
\begin{array}{rclcr}
x & = & 107_{(10)} & \mapsto & 1\ 0\ 7 \\
y & = & 14_{(10)} & \mapsto & \underline{0\ 1\ 4} + \\
c & = & & & 0 \\
r & = & & & \underline{}
\end{array}
$$

but it *also* applies for $b = 2$

$$
\begin{array}{rclcr}
x & = & 107_{(10)} & \mapsto & 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1 \\
y & = & 14_{(10)} & \mapsto & \underline{0\ 0\ 0\ 0\ 1\ 1\ 1\ 0} + \\
c & = & & & \\
r & = & & & \underline{}
\end{array}
$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$
\begin{array}{rcrcl}
x & = & 107_{(10)} & \mapsto & 1\ 0\ 7 \\
y & = & 14_{(10)} & \mapsto & \underline{0\ 1\ 4}\,+ \\
c & = & & & 1\ 0 \\
r & = & & & \underline{1}
\end{array}
$$

but it *also* applies for $b = 2$

$$
\begin{array}{rcrcl}
x & = & 107_{(10)} & \mapsto & 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1 \\
y & = & 14_{(10)} & \mapsto & \underline{0\ 0\ 0\ 0\ 1\ 1\ 1\ 0}\,+ \\
c & = & & & \\
r & = & & & \underline{}
\end{array}
$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$
\begin{array}{rcrcl}
x & = & 107_{(10)} & \mapsto & 1\ 0\ 7 \\
y & = & 14_{(10)} & \mapsto & \underline{0\ 1\ 4} + \\
c & = & & & 0\ 1\ 0 \\
r & = & & & \underline{2\ 1}
\end{array}
$$

but it *also* applies for $b = 2$

$$
\begin{array}{rcrcl}
x & = & 107_{(10)} & \mapsto & 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1 \\
y & = & 14_{(10)} & \mapsto & \underline{0\ 0\ 0\ 0\ 1\ 1\ 1\ 0} + \\
c & = & & & \\
r & = & & & \underline{}
\end{array}
$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$
\begin{array}{rcrcl}
x & = & 107_{(10)} & \mapsto & \;1\;0\;7 \\
y & = & 14_{(10)} & \mapsto & \underline{\;0\;1\;4}\,+ \\
c & = & & & 0\;0\;1\;0 \\
r & = & 121_{(10)} & \mapsto & \underline{\;1\;2\;1}
\end{array}
$$

but it *also* applies for $b = 2$

$$
\begin{array}{rcrcl}
x & = & 107_{(10)} & \mapsto & 0\;1\;1\;0\;1\;0\;1\;1 \\
y & = & 14_{(10)} & \mapsto & \underline{0\;0\;0\;0\;1\;1\;1\;0}\,+ \\
c & = & & & \\
r & = & & & \underline{}
\end{array}
$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$\begin{array}{rcrcrrrrl} x & = & 107_{(10)} & \mapsto & & 1 & 0 & 7 & \\ y & = & 14_{(10)} & \mapsto & & 0 & 1 & 4 & + \\ \hline c & = & & & 0 & 0 & 1 & 0 & \\ r & = & 121_{(10)} & \mapsto & & 1 & 2 & 1 & \\ \hline \end{array}$$

but it *also* applies for $b = 2$

$$\begin{array}{rcrcrrrrrrrrl} x & = & 107_{(10)} & \mapsto & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & \\ y & = & 14_{(10)} & \mapsto & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & + \\ \hline c & = & & & & & & & & & & 0 & \\ r & = & & & & & & & & & & & \\ \hline \end{array}$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$
\begin{array}{lclcr}
x & = & 107_{(10)} & \mapsto & 1\ 0\ 7 \\
y & = & 14_{(10)} & \mapsto & 0\ 1\ 4\ + \\
c & = & & & 0\ 0\ 1\ 0 \\
r & = & 121_{(10)} & \mapsto & 1\ 2\ 1
\end{array}
$$

but it *also* applies for $b = 2$

$$
\begin{array}{lclcr}
x & = & 107_{(10)} & \mapsto & 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1 \\
y & = & 14_{(10)} & \mapsto & 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ + \\
c & = & & & 0\ 0 \\
r & = & & & 1
\end{array}
$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$\begin{array}{lclcr}
x & = & 107_{(10)} & \mapsto & 1\ 0\ 7 \\
y & = & 14_{(10)} & \mapsto & 0\ 1\ 4\ + \\
c & = & & & 0\ 0\ 1\ 0 \\
r & = & 121_{(10)} & \mapsto & 1\ 2\ 1
\end{array}$$

but it *also* applies for $b = 2$

$$\begin{array}{lclcr}
x & = & 107_{(10)} & \mapsto & 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1 \\
y & = & 14_{(10)} & \mapsto & 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ + \\
c & = & & & 1\ 0\ 0 \\
r & = & & & 0\ 1
\end{array}$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), Add outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$
\begin{array}{rcrcr}
x & = & 107_{(10)} & \mapsto & 1\ 0\ 7 \\
y & = & 14_{(10)} & \mapsto & 0\ 1\ 4\ + \\
c & = & & & 0\ 0\ 1\ 0 \\
r & = & 121_{(10)} & \mapsto & 1\ 2\ 1
\end{array}
$$

but it *also* applies for $b = 2$

$$
\begin{array}{rcrcr}
x & = & 107_{(10)} & \mapsto & 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1 \\
y & = & 14_{(10)} & \mapsto & 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ + \\
c & = & & & 1\ 1\ 0\ 0 \\
r & = & & & 0\ 0\ 1
\end{array}
$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$
\begin{array}{lclcccccc}
x & = & 107_{(10)} & \mapsto & & 1 & 0 & 7 & \\
y & = & 14_{(10)} & \mapsto & & 0 & 1 & 4 & + \\
\cline{5-8}
c & = & & & 0 & 0 & 1 & 0 & \\
r & = & 121_{(10)} & \mapsto & & 1 & 2 & 1 & \\
\cline{5-8}
\end{array}
$$

but it *also* applies for $b = 2$

$$
\begin{array}{lclcccccccccc}
x & = & 107_{(10)} & \mapsto & & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
y & = & 14_{(10)} & \mapsto & & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \; + \\
\cline{5-13}
c & = & & & & & & & 1 & 1 & 1 & 0 & 0 \\
r & = & & & & & & & & 1 & 0 & 0 & 1 \\
\cline{5-13}
\end{array}
$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$\begin{array}{lclcr}
x & = & 107_{(10)} & \mapsto & 1\;0\;7 \\
y & = & 14_{(10)} & \mapsto & 0\;1\;4\;+ \\
c & = & & & 0\;0\;1\;0 \\
r & = & 121_{(10)} & \mapsto & 1\;2\;1
\end{array}$$

but it *also* applies for $b = 2$

$$\begin{array}{lclcr}
x & = & 107_{(10)} & \mapsto & 0\;1\;1\;0\;1\;0\;1\;1 \\
y & = & 14_{(10)} & \mapsto & 0\;0\;0\;0\;1\;1\;1\;0\;+ \\
c & = & & & 0\;1\;1\;1\;0\;0 \\
r & = & & & 1\;1\;0\;0\;1
\end{array}$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$
\begin{array}{rclcr}
x & = & 107_{(10)} & \mapsto & \begin{array}{cccc} & 1 & 0 & 7 \end{array} \\
y & = & 14_{(10)} & \mapsto & \begin{array}{cccc} & 0 & 1 & 4 \end{array} + \\
c & = & & & \begin{array}{cccc} 0 & 0 & 1 & 0 \end{array} \\
r & = & 121_{(10)} & \mapsto & \begin{array}{cccc} & 1 & 2 & 1 \end{array}
\end{array}
$$

but it *also* applies for $b = 2$

$$
\begin{array}{rclcr}
x & = & 107_{(10)} & \mapsto & \begin{array}{cccccccc} 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \end{array} \\
y & = & 14_{(10)} & \mapsto & \begin{array}{cccccccc} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array} + \\
c & = & & & \begin{array}{cccccccc} & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{array} \\
r & = & & & \begin{array}{cccccccc} & & 1 & 1 & 1 & 0 & 0 & 1 \end{array}
\end{array}
$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$
\begin{array}{lclcrrrrl}
x & = & 107_{(10)} & \mapsto & & 1 & 0 & 7 & \\
y & = & 14_{(10)} & \mapsto & & 0 & 1 & 4 & + \\
\hline
c & = & & & 0 & 0 & 1 & 0 & \\
r & = & 121_{(10)} & \mapsto & & 1 & 2 & 1 & \\
\hline
\end{array}
$$

but it *also* applies for $b = 2$

$$
\begin{array}{lclcrrrrrrrrrl}
x & = & 107_{(10)} & \mapsto & & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & \\
y & = & 14_{(10)} & \mapsto & & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & + \\
\hline
c & = & & & & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \\
r & = & & & & & 1 & 1 & 1 & 1 & 0 & 0 & 1 & \\
\hline
\end{array}
$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), Add outputs the extra digit as a separate carry-out.

Example

Assuming $ci = 0$, the approach you know sets $b = 10$

$$
\begin{array}{lclcl}
x & = & 107_{(10)} & \mapsto & \;1\;0\;7 \\
y & = & 14_{(10)} & \mapsto & \underline{\;0\;1\;4} + \\
c & = & & & 0\;0\;1\;0 \\
r & = & 121_{(10)} & \mapsto & \underline{\;1\;2\;1}
\end{array}
$$

but it *also* applies for $b = 2$

$$
\begin{array}{lclcl}
x & = & 107_{(10)} & \mapsto & \;0\;1\;1\;0\;1\;0\;1\;1 \\
y & = & 14_{(10)} & \mapsto & \underline{\;0\;0\;0\;0\;1\;1\;1\;0} + \\
c & = & & & 0\;0\;0\;0\;1\;1\;1\;0\;0 \\
r & = & 121_{(10)} & \mapsto & \underline{\;0\;1\;1\;1\;1\;0\;0\;1}
\end{array}
$$

- Note that:
  - adding two n-digit integers produces an $(n + 1)$-digit result, and
  - instead of producing this directly (per the example), ADD outputs the extra digit as a separate carry-out.

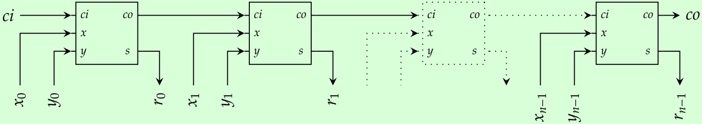
Algorithm (Add)

Input: Two unsigned, n-digit, base-b integers x and y, and a 1-digit carry-in ci

Output: An unsigned, n-digit, base-b integer $r = x + y$, and a 1-digit carry-out co

$r \leftarrow 0, c_0 \leftarrow ci$
for $i = 0$ **upto** $n - 1$ **step** $+1$ **do**
    $r_i \leftarrow (x_i + y_i + c_i) \bmod b$
    if $(x_i + y_i + c_i) < b$ **then** $c_{i+1} \leftarrow 0$ **else** $c_{i+1} \leftarrow 1$
end
$co \leftarrow c_n$
return r, co

Circuit


ci
ci
co
x
y
s
ci
co
x
y
s
ci
co
x
y
s
ci
co
x
y
s
co
x_0
y_0
r_0
x_1
y_1
r_1
x_{n-1}
y_{n-1}
r_{n-1}

Algorithm (Sub)

Input: Two unsigned, n-digit, base-b integers x and y, and a 1-digit borrow-in bi

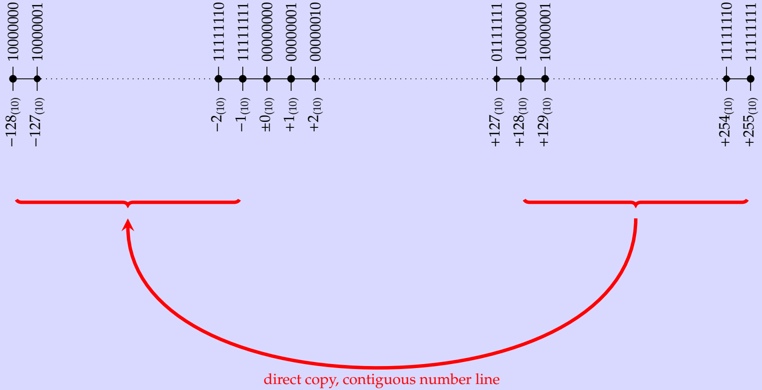
Output: An unsigned, n-digit, base-b integer $r = x - y$, and a 1-digit borrow-out bo

```
r ← 0, c_0 ← bi
for i = 0 upto n − 1 step +1 do
    r_i ← (x_i − y_i − c_i) mod b
    if (x_i − y_i − c_i) ≥ 0 then c_{i+1} ← 0 else c_{i+1} ← 1
end
bo ← c_n
return r, bo
```

Definition

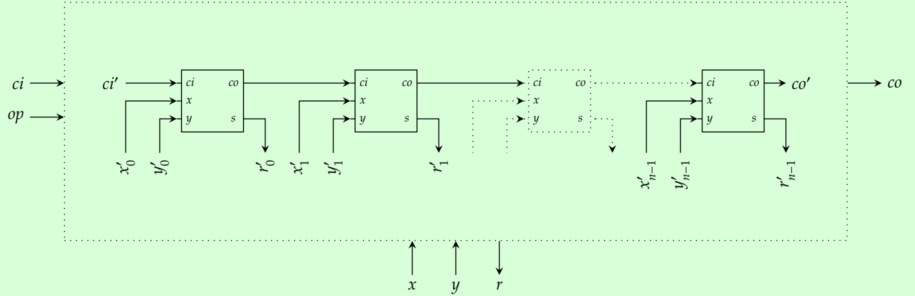
In two's-complement representation, for some y we have $-y \mapsto \neg y + 1$. Put more simply, to negate y we invert each bit y_i via a NOT gate, then add 1 to the result.

Example



- Question: we *know* $x - y \equiv x + (-y)$, and can *already* compute $x + y$, so

Circuit



given we want

$$r = \begin{cases} x + y + ci & \text{if } op = 0 \\ x - y - ci & \text{if } op = 1 \end{cases},$$

how do we control x', y' and ci' to get the correct result?

- So,

op	ci	r				
0	0	x	$+$	y	$+$	ci
0	1	x	$+$	y	$+$	ci
1	0	x	-	y	-	ci
1	1	x	-	y	-	ci

- So,

op	ci	r				
0	0	x	+	y	+	0
0	1	x	+	y	+	1
1	0	x	-	y	-	0
1	1	x	-	y	-	1

- So,

op	ci	r
0	0	$x + y + 0$
0	1	$x + y + 1$
1	0	$x + \neg y + 1 - 0$
1	1	$x + \neg y + 1 - 1$

- So,

op	ci	r
0	0	$x + y + 0$
0	1	$x + y + 1$
1	0	$x + \neg y + 1$
1	1	$x + \neg y + 0$

- So,

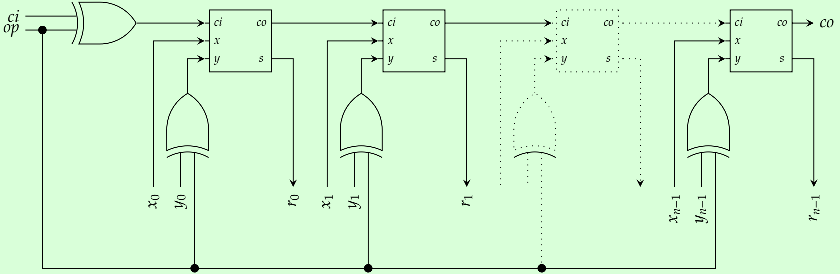
op	ci	r
0	0	$x + y + 0$
0	1	$x + y + 1$
1	0	$x + \neg y + 1$
1	1	$x + \neg y + 0$

- We can compute $x' + y' + ci'$, so we translate via

op	ci	x_i	y_i	ci'	x'_i	y'_i
0	0	0	0	0	0	0
0	1	1	1	1	1	1
1	0	0	0	1	0	1
1	1	1	1	0	1	0

i.e., $ci' = ci \oplus op$, $x'_i = x_i$ and $y'_i = y_i \oplus op$.

Circuit



Example

Consider a signed, 16-bit integer x, used within (or **cast** into) a signed, 32-bit integer; use as is produces the wrong result, whereas **sign extension** by padding with the sign bit is correct:

$$
\begin{array}{rclcr}
x & = & 1111111111111111_{(2)} & = & -1_{(10)} \\
 & & 00000000000000001111111111111111_{(2)} & = & 65535_{(10)} \\
 & & 11111111111111111111111111111111_{(2)} & = & -1_{(10)}
\end{array}
$$

- It's common to think of a left-shift (resp. right-shift) of x by y bits as multiplication (resp. division) by 2^y, e.g.,

$$2^y \cdot x = 2^y \cdot \sum_{i=0}^{n-1} x_i \cdot 2^i = \sum_{i=0}^{n-1} x_i \cdot 2^{i+y}.$$

- So one reason for having **arithmetic shift** is to preserve sign:

Example

Note that

$$\begin{array}{lcccl} -38_{(10)} & & & \mapsto & 11011010 \\ -38_{(10)}/2 & = & -19_{(10)} & \mapsto & 11101101 \end{array}$$

and right-shift by $y = 1$ bit should mean "divide by two". But if we use logical right-shift we get

$$\begin{array}{lcccl} r & = & x \gg_u y & = & 11011010 \gg_u 1 \\ & & & = & 01101101 \\ & & & \mapsto & 109_{(10)} \end{array}$$

whereas if we use arithmetic right-shift we get

$$\begin{array}{lcccl} r & = & x \gg_s y & = & 11011010 \gg_s 1 \\ & & & = & 11101101 \\ & & & \mapsto & -19_{(10)} \end{array}$$

as expected.

Example

Consider use of an *unsigned* representation:

$$
\begin{array}{rclccccccc}
x & = & 15_{(10)} & \mapsto & & 1 & 1 & 1 & 1 & \\
y & = & 1_{(10)} & \mapsto & & 0 & 0 & 0 & 1 & + \\
\cline{5-9}
c & = & & & 1 & 1 & 1 & 1 & 0 & \\
r & = & 0_{(10)} & \mapsto & & 0 & 0 & 0 & 0 & \\
\cline{5-9}
\end{array}
$$

Here, the carry-out indicates an error: the correct result $r = 16$ is too large for $n = 4$ bits.

Example

Consider use of a *signed* representation:

$$
\begin{array}{rclcccccccc}
x & = & -1_{(10)} & \mapsto & & 1 & 1 & 1 & 1 & \\
y & = & 1_{(10)} & \mapsto & & 0 & 0 & 0 & 1 & + \\
\hline
c & = & & & 1 & 1 & 1 & 1 & 0 & \\
r & = & 0_{(10)} & \mapsto & & 0 & 0 & 0 & 0 & \\
\hline
\end{array}
$$

Irrespective of the carry-out, the signs of inputs and output make sense: there is no overflow, so $r = 0$ is correct.

Example

Consider use of a *signed* representation:

$$
\begin{array}{rclcccccccc}
x & = & 7_{(10)} & \mapsto & & 0 & 1 & 1 & 1 & \\
y & = & 1_{(10)} & \mapsto & & 0 & 0 & 0 & 1 & + \\
\hline
c & = & & & 0 & 1 & 1 & 1 & 0 & \\
r & = & -8_{(10)} & \mapsto & & 1 & 0 & 0 & 0 & \\
\hline
\end{array}
$$

Irrespective of the carry-out, the signs of inputs and output make no sense: there is an overflow, so $r = -8$ is incorrect.

Listing (C)

```c
uint8_t add( uint8_t x, uint8_t y ) {
  asm goto( "add %0, %1       ;"
            "jo  %l[overflow] ;"
            "jc  %l[carry]    ;"
            "jmp %l[correct]  ;"

            : : "r" (x), "r" (y) : "cc" : overflow, carry, correct );

  overflow:
  printf( "overflow\n" );
  goto correct;

  carry:
  printf(    "carry\n" );
  goto correct;

  correct:
  return x + y;
}
```