Carry and Overflow

Young W. Lim

2023-11-11 Sat

Outline

Based on

1. "Self-service Linux: Mastering the Art of Problem Determination",

Mark Wilding

1. "Computer Architecture: A Programmer's Perspective", Bryant & O'Hallaron

Compling 32-bit program on 64-bit gcc

- `gcc -v`
- `gcc -m32 t.c`
- `sudo apt-get install gcc-multilib`
- `sudo apt-get install g++-multilib`
- `gcc-multilib`
- `g++-multilib`
- `gcc -m32`
- `objdump -m i386`

TOC: Overview

Carry flag and overflow flag

- considering carry and overflow flags in x86
- do not confuse the carry flag
  with the overflow flag
  in integer arithmetic.
- the *ALU* always sets these flags appropriately
  when doing any integer math.
- these flags can occur on its *own*, or *both* together.

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Signed and unsigned computations

- the CPU's ALU doesn't care or know whether signed or unsigned computations are performed;
- the *ALU* just performs integer arithmetic and sets the flags appropriately.
- It's up to the *programmer* to know which flag to check after the arithmetic is done.

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Flags for an unsigned number

- if a word is treated as an unsigned number,
  - the carry flag must be used to check
    if the result is fit into n-bit or $(n+1)$-bit number
  - the overflow flag is *irrelevant*
    to an unsigned number arithmetic

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Flags for a signed number

- if a word is treated as an signed number,
  - the carry flag is *irrelevant*
    to an signed number arithmetic
  - the overflow flag must be used to check
    if the result is wrong or not

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Detecting errors in usigned and signed arithmetic (1)

	unsigned integer arithmetic	signed integer arithmetic
CF Carry Flag	detects *overflows* extends an *n-bit* result into an (*n+1*)-bit result	
OF Overflow Flag		detects *overflows* errors the result cannot be used

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Detecting errors in usigned and signed arithmetic (2)

- unsigned integer arithmetic *overflow*
  is indicated by the carry flag
  - $P + P$ `CF=1` $\rightarrow$ carry out – the result is too large for an *n-bit* integer
  - $P - P$ `CF=1` $\rightarrow$ borrow in – the result is too small for an *n-bit* integer
- signed integer arithmetic *overflow*
  is indicated by the overflow flag
  - $P + P \rightarrow N$ `OF=1` $\rightarrow$ overflow – the result is not correct
  - $N + N \rightarrow P$ `OF=1` $\rightarrow$ overflow – the result is not correct
- P (positive), N (negative)

https://stackoverflow.com/questions/47333458/assembly-x86-64-setting-carry-flag-fo

Detecting errors in usigned and signed arithmetic (3)

- unsigned integer arithmetic *overflow*
  is indicated by the carry flag
  - the *overflowed* n-bit result can be extended
    into ($n+1$)-bit result by using the carry flag
- signed integer arithmetic *overflow*
  is indicated by the overflow flag
  - the *overflowed* n-bit result cannot be used

https://stackoverflow.com/questions/47333458/assembly-x86-64-setting-carry-flag-fo

The verb to overflow v.s. the overflow flag (1)

- Do not confuse the English verb *to overflow* with the overflow flag in the ALU.
- The verb *to overflow* is used casually to indicate that some math result doesn't fit in the number of bits available;
- it could be integer math, or floating-point math, or whatever.
- The overflow flag is set specifically by the ALU it isn't the same as the casual English verb "to overflow"

`http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt`

The verb to overflow v.s. the overflow flag (2)

- In English, we may say
  "the binary/integer math overflowed
  the number of bits available for the result,
  causing the carry flag to come on".
- Note how this English usage of the verb "to overflow"
  is not the same as saying the overflow flag is on".
- A math result can overflow (the verb)
  the number of bits available
  without turning on the ALU overflow flag

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

CF (carry flag) and OF (overflow flag) computation

ADD (addition)	SUB (subtraction)
CF = C_n	CF = $\overline{C_n}$
OF = $C_n \bigoplus C_{n-1}$	OF = $C_n \bigoplus C_{n-1}$
a 2's complement addition $A + B = A + B + 0$	a transformed addition $A - B = A + \overline{B} + 1$
$\{C_n, S_{n-1}\} = a_{n-1} + b_{n-1} + c_{n-1}$	$\{C_n, S_{n-1}\} = a_{n-1} + \overline{b_{n-1}} + c_{n-1}$
$\{C_{n-1}, S_{n-2}\} = a_{n-2} + b_{n-2} + c_{n-2}$	$\{C_{n-1}, S_{n-2}\} = a_{n-2} + \overline{b_{n-2}} + c_{n-2}$

https://www.csie.ntu.edu.tw/~cyy/courses/assembly/12fall/lectures/handouts/lec14_

TOC: Carry flag

TOC: Examples of signed and unsigned integer arithmetic

Examples of signed and unsigned integer arithmetic (1)

- 0xFFFFBDC3 as a signed (negative) number -0x0000423D (-16957_{10})

```
                  F    F    F    F    B    D    C    3
-0xFFFFBDC3    0x1111_1111_1111_1111_1011_1101_1100_0011
 0x0000423D   -0x0000_0000_0000_0000_0100_0010_0011_1100 (1's complement)
 0x0000423D   -0x0000_0000_0000_0000_0100_0010_0011_1101 (1's complement)
                0    0    0    0    4    2    3    D

                0    0    0    0    4    2    3    D
-0x0000423D   -0x0000_0000_0000_0000_0100_0010_0011_1101
 0x0000BDC2    0x1111_1111_1111_1111_1011_1101_1100_0010 (1's complement)
 0xFFFFBDC3    0x1111_1111_1111_1111_1011_1101_1100_0011 (2's complement)
                F    F    F    F    B    D    C    3
```

- 0xFFFFBDC3 as an unsigned (positive) number ($+4294950339_{10}$)

https://stackoverflow.com/questions/47333458/assembly-x86-64-setting-carry-flag-fo

Examples of signed and unsigned integer arithmetic (2)

- 0x0000195D - 0x0000618D : unsigned subtraction
  subtraction by hand

```
                          0    0    0    0    1    9    5    D
  0x0000195D     0x0000_0000_0000_0000_0001_1001_0101_1101
- 0x0000618D     0x0000_0000_0000_0000_0110_0001_1000_1101
                          0    0    0    0    6    1    8    D
 ----------------------------------------------------------
  0xFFFFB7D0   1 0x1111_1111_1111_1111_1011_0111_1101_0000 (hand subtraction)
               1      F    F    F    F    B    7    D    0
               .
               V borrow (CF=1) : unsigned integer overflow
```

  - A borrow is indicated by the carry flag (CF=1)
    - whenever an unsigned integer overflow happened
    - $A - B$, when $A < B$, for non-negative integers A, B

https://stackoverflow.com/questions/47333458/assembly-x86-64-setting-carry-flag-fo

Examples of signed and unsigned integer arithmetic (3)

- 0x0000195D + (-0x0000618D) : signed subtraction
  the transformed addition using the 2's complement of subtrahend

```
                         0    0    0    0    6    1    8    D
 -0x0000618D    -0x0000_0000_0000_0000_0110_0001_1000_1101
  0xFFFF9E73     0x1111_1111_1111_1111_1001_1110_0111_0011 (2's complement)
                         F    F    F    F    8    E    7    3

  0x0000195D     0x0000_0000_0000_0000_0001_1001_0101_1101 (+0x0000195D)
+ 0xFFFF9E73     0x1111_1111_1111_1111_1001_1110_0111_0011 (-0x0000618D)
 -----------------------------------------------------------
  0xFFFFB7D0  0  0x1111_1111_1111_1111_1011_0111_1101_0000 (hand addition)
              0          F    F    F    F    B    7    D    0
 -0x00004830  .  0x0000_0000_0000_0000_0100_1000_0011_0000 (2's complement)
              .          0    0    0    0    4    8    3    0
              V  no carry in the transformed addition (Cn=0) --> (CF=1)
```

  - signed integer overflow is indicated
    not by the carry flag (CF), but by the overflow flag (OF)
    - the carry flag is set by the inverted carry of a transformed addition

https://stackoverflow.com/questions/47333458/assembly-x86-64-setting-carry-flag-fo

Examples of signed and unsigned integer arithmetic (4)

- 0x0000195D - 0x0000618D
  - 0x0000195D - 0x0000618D : hand subtraction
    unsigned integer subtraction
  - 0x0000195D + (-0x0000618D) : the transformed addition
    using the 2's complement of the subtrahend
    signed integer subtraction
  - the result is 0xFFFFB7D0 (the two methods have the same bit pattern)
    - interpreting as a unsigned integer 4294948816_{10}
      0xFFFFB7D0 with a borrow (CF=1)
    - interpreting as a signed integer -18480_{10}
      -0x00004830 (meaningless CF=1)

https://stackoverflow.com/questions/47333458/assembly-x86-64-setting-carry-flag-fo

Examples of signed and unsigned integer arithmetic (5)

0xFFFFB7D0 with CF=1	the result of unsigned subtraction with unsigned integer overflow	4294948816_{10}
-0x00004830	the result of signed subtraction	-18480_{10}

https://stackoverflow.com/questions/47333458/assembly-x86-64-setting-carry-flag-fo

Examples of signed and unsigned integer arithmetic (6)

- `0x0000195D - 0x0000618D` : unsigned subtraction
  - there is an unsigned integer overflow
    so the carry flag will be set (`CF=1`) to indicate a borrow
    ($A - B$, when $A < B$, for non-negative integers A, B)
    (unsigned integers can't be negative),
- `0x0000195D + (-0x0000618D)` : signed subtraction
  - there is no signed integer overflow
    the overflow flag won't be set (`OF=0`)
  - signed overflw occurrs , in the transformed addition,
    - two *positive* numbers are added and
      the result is a *negative*, (P + P → N), or
    - two *negative* numbers are added and
      the result is a *positive*, (N + N → P)

https://stackoverflow.com/questions/47333458/assembly-x86-64-setting-carry-flag-f

TOC Carry flag in unsigned and signed computations

- Using the Carry Flag as a borrow
- Examples of signed and unsigned integer arithmetic

2's complement numbers : 4-bit

0111	(+7)	1000	(-8)
0110	(+6)	1001	(-7)
0101	(+5)	1010	(-6)
0100	(+4)	1011	(-5)
0011	(+3)	1100	(-4)
0010	(+2)	1101	(-3)
0001	(+1)	1110	(-2)
0000	(0)	1111	(-1)

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Addend and augend in a n-bit addition

n	bits	addened	A	$\{a_{n-1}, a_{n-2}, \cdots, a_1, a_0\}$
n	bits	augend	B	$\{b_{n-1}, b_{n-2}, \cdots, b_1, b_0\}$
(n+1)	bits	carry bits	C	$\{C_n, C_{n-1}, C_{n-2}, \cdots, C_1, C_0\}$
n	bits	sum bits	S	$\{S_{n-1}, S_{n-2}, \cdots, S_1, S_0\}$

external carry bits : C_n carry out, C_0 carry in

$$
\begin{array}{cccccc}
 & a_{n-1} & a_{n-2} & \cdots\cdots\cdots & a_1 & a_0 \\
 & b_{n-1} & b_{n-2} & \cdots\cdots\cdots & b_1 & b_0 \\
 & & & & & C_0 \\
\hline
C_n & S_{n-1} & S_{n-2} & \cdots\cdots\cdots & S_1 & S_0
\end{array}
$$

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Full adder operation in each bit position

full adder operation in the i^{th} bit position

$$\{C_{i+1}, S_i\} = a_i + b_i + C_i$$

$$\begin{array}{rr} & a_i \\ & b_i \\ & C_i \\ \hline C_{i+1} & S_i \end{array}$$

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Internal and external carry bits

external carrys	C_n output, C_0 input	
internal carrys	$\{C_{n-1}, C_{n-2}, \cdots\cdots\cdots, C_2, C_1\}$	output / input
sum bits	$\{S_{n-1}, S_{n-2}, \cdots\cdots\cdots, S_1, S_0\}$	output

$$
\begin{array}{cccccc}
 & a_{n-1} & a_{n-2} & \cdots\cdots\cdots & a_1 & a_0 \\
 & b_{n-1} & b_{n-2} & \cdots\cdots\cdots & b_1 & b_0 \\
\hline
C_n & C_{n-1} & C_{n-2} & \cdots\cdots\cdots & C_1 & C_0 \\
\hline
 & S_{n-1} & S_{n-2} & \cdots\cdots\cdots & S_1 & S_0
\end{array}
$$

$$
\begin{array}{cccccc}
 & a_{n-1} & a_{n-2} & \cdots\cdots\cdots & a_1 & a_0 \\
 & b_{n-1} & b_{n-2} & \cdots\cdots\cdots & b_1 & b_0 \\
 & & & & & C_0 \\
\hline
C_n & S_{n-1} & S_{n-2} & \cdots\cdots\cdots & S_1 & S_0
\end{array}
$$

`http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt`

Addition and Subtraction

- addition

$$\{C_n, S\} = A + B = A + B + 0$$

	a_{n-1}	a_{n-2}	$\cdots\cdots\cdots$	a_1	a_0
	b_{n-1}	b_{n-2}	$\cdots\cdots\cdots$	b_1	b_0
	C_{n-1}	C_{n-2}	$\cdots\cdots\cdots$	C_1	0
C_n	S_{n-1}	S_{n-2}	$\cdots\cdots\cdots$	S_1	S_0

- subtraction - transformed addition

$$\{C_n, S\} = A - B = A + \overline{B} + 1$$

	a_{n-1}	a_{n-2}	$\cdots\cdots\cdots$	a_1	a_0
	$\overline{b_{n-1}}$	$\overline{b_{n-2}}$	$\cdots\cdots\cdots$	$\overline{b_1}$	$\overline{b_0}$
	C_{n-1}	C_{n-2}	$\cdots\cdots\cdots$	C_1	1
C_n	S_{n-1}	S_{n-2}	$\cdots\cdots\cdots$	S_1	S_0

`http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt`

Using the Carry Flag as a borrow (1)

- a borrow (CF=1) occurs
  in the subtraction $A - B$
  when b is larger than a ($A < B$)
  as unsigned numbers
- Computer hardware can detect
  a borrow (CF=1) in subtraction
  by looking at whether a carry out (Cn) occurred
  in the transformed addition

https://stackoverflow.com/questions/47333458/assembly-x86-64-setting-carry-flag-fo

Using the Carry flag as a borrow (2)

- a borrow (CF=1) occurs
  in the subtraction $A - B$ ($A < B$)
  as unsigned numbers

- a carry out (Cn) in the transformed addition
  - If there is no carry (Cn=0)
    then there is a borrow (CF=1)
  - If there is a carry (Cn=1)
    then there is no borrow (CF=0)
  - CF = !Cn

https://stackoverflow.com/questions/47333458/assembly-x86-64-setting-carry-flag-fo

Using the Carry Flag as a borrow (3)

- the same *addition* and *subtraction* instructions
  are used for both unsigned and signed integer arithmetic.
  - no special *addition* and *subtraction* instructions
    for unsigned and signed integer arithmetic
- the only difference is
  - which flags you *test* afterwards and
  - how you *interpret* the result

`https://stackoverflow.com/questions/47333458/assembly-x86-64-setting-carry-flag-fo`

TOC Rules for the carry flag

2's complement numbers : 4-bit

0111	(+7)	1000	(-8)
0110	(+6)	1001	(-7)
0101	(+5)	1010	(-6)
0100	(+4)	1011	(-5)
0011	(+3)	1100	(-4)
0010	(+2)	1101	(-3)
0001	(+1)	1110	(-2)
0000	(0)	1111	(-1)

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

The 1st rule for setting the carry flag

1. The carry flag is set (CF = 1 : carry in addition)
   if the addition of two unsigned numbers causes a carry
   out of the most significant (leftmost) bits added.
   (*hand addition rule*)

```
signed addition         signed subtraction        unsigned addition

 1111   (-1)             1111   (-1)               1111   (15)
+0001  +(+1)            -1111  -(-1)              +0001  +( 1)
-----  -----            -----  -----              -----  -----
10000   ( 0)            10000   ( 0)              10000   (16)

 Cn=1 -> CF=1            Cn=1 -> CF=1              CF=1

 CF is not 16            CF is not 16              CF means 16
  S = 0000                S = 0000                  S = 0000
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

The 2nd rule for setting the carry flag

2. The carry flag is also set (CF = 1 : borrow in subtraction) if the subtraction of two unsigned numbers requires a borrow into the most significant (leftmost) bits subtracted. (*hand subtraction rule*)

```
signed addition          signed subtraction          unsigned subtraction

 0000   ( 0)              0000   ( 0)                  0000   ( 0)
+1111  +(-1)             -0001  -(+1)                 -0001  -( 1)
-----  -----             -----  -----                 -----  -----
01111   (-1)             01111   (-1)                 01111   (15) (-16)

Cn=0 -> CF=1             Cn=0 -> CF=1                 CF=1

CF is not -16            CF is not -16                CF means -16
 S = 1111                 S = 1111                     S = 1111
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Cases for clearing the carry flag (1/2)

- Otherwise, the carry flag is turned off (zero).

```
signed addition          signed subtraction          unsigned addition

 0111    (+7)             0111    (+7)                0111    ( 7)
+0001   +(+1)            -1111   -(-1)               +0001   +( 1)
-----   -----            -----   -----               -----   -----
01000    (-8)            01000    (-8)               01000    ( 8)

 Cn=0 -> CF=0             Cn=0 -> CF=0                CF=0
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Cases for clearing the carry flag (2/2)

- Otherwise, the carry flag is turned off (zero).

```
signed addition            signed subtraction          unsigned subtraction

 1000    (-8)               1000    (-8)                1000    ( 8)
+1111   +(-1)              -0001   -(+1)               -0001   -( 1)
-----   -----              -----   -----               -----   -----
10111    (+7)              10111    (+7)               00111    ( 7)

 Cn=1 -> CF=0              Cn=1 -> CF=0                CF=0
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

TOC Method for computing the carry flag

- Carry flag computation

Carry flag computation (1)

ADD (addition)	SUB (subtraction)
CF = C_n	CF = $\overline{C_n}$
normal carry of a 2's complement addition $A + B = A + B + 0$	inverted carry of a transformed addition $A - B = A + \overline{B} + 1$
$\{C_n, S_{n-1}\} = a_{n-1} + b_{n-1} + c_{n-1}$	$\{C_n, S_{n-1}\} = a_{n-1} + \overline{b_{n-1}} + c_{n-1}$

https://www.csie.ntu.edu.tw/~cyy/courses/assembly/12fall/lectures/handouts/lec14_

Carry flag computation (2)

- In unsigned arithmetic,
  - the carry flag is used to detect *overflow*
  - the carry flag is used to extend *n-bit* result into *(n+1)-bit* result
  - for addition, the carry flag is a carry out
  - for subtraction, the carry flag is a borrow in
- In signed arithmetic,
  - the carry flag is useless
  - the carry flag neither detects overflow nor extends n-bit result

`http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt`

Carry flag computation (3)

- In unsigned arithmetic,

Addition	`CF = 1` means carry out	when `Cn = 1`
Subtraction	`CF = 1` means borrow in	when `Cn = 0`

  - `CF` - Carry Flag in x86
  - `Cn` - the normal carry out
    - the carry out of a 2's complement addition for `ADD`
    - the carry out of a *transformed* addition for `SUB`

- In signed arithmetic,
  - the carry flag is useless

`http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt`

TOC: Overflow flag

TOC Overflow flag in unsigned and signed computations

- Overflow flag

Overflow flag (1)

- only need to look at the sign bits (leftmost) of the three numbers to decide if the overflow flag is turned on or off.
- overflow flag is based on signed arithmetic
- in signed arithmetic, watch the overflow flag to detect errors.
- in unsigned arithmetic, the overflow flag tells you nothing interesting

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Overflow flag (2)

- for signed (two's complement) arithmetic, overflow flag on means the answer is wrong
  - two positive numbers are added and the result is a negative, (P + P $\rightarrow$ N), or
  - two negative numbers are added and the result is a positive, (N + N $\rightarrow$ P)
  - opposite signed numbers are added, then no overflow
    - (P + N $\rightarrow$ P or N)
    - (N + P $\rightarrow$ P or N)
- for unsigned arithmetic, the overflow flag means nothing and should be ignored

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Overflow flag (3)

- the rules for two's complement detect errors by examining the sign of the result.
- a negative and positive added together cannot be wrong, because the sum is between the addends.
- mixed-sign addition never turns on the overflow flag.
- since both of the addends fit within the allowable range of numbers, and their sum is between them, it must fit as well.

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

TOC Rules for the overflow flag

- The 1st rule for setting the overflow flag
- The 2nd rule for setting the overflow flag
- Cases for clearing the overflow flag

The 1st rule for setting the overflow flag

1. If the sum of two signed numbers
   with the sign bits off (0, 0)
   yields a result number with the sign bit on (1)
   the overflow flag is turned on (OF =1 : $+, + \rightarrow -$)

```
signed addition          signed subtraction          unsigned addition

 0100    (+4)             0100    (+4)                0100    (4)
+0100   +(+4)            -1100   -(-4)               +0100   +(4)
-----   -----            -----   -----               -----   -----
01000    (-8)            01000    (-8)               01000    (8)
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

The 2nd rule for setting the overflow flag

2. If the sum of two numbers
   with the sign bits on `(1, 1)`
   yields a result number with the sign bit off `(0)`
   the overflow flag is turned on. (`OF` =1 : $-, - \rightarrow +$)

   `signed addition  (-7) + (-7) = (2) with a borrow (-16)`

   `1001 + 1001 = 1 0010` (2's complement addition) $(-, - \rightarrow +)$

   `signed subtraction (-7) - (7) = (2) with a borrow (-16)`

   `1001 - 0111 = 1 0010` (transformed subtraction)

   `unsigned addition (9) + (9) = (18)`

   `1001 + 1001 = 1 0010`

`http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt`

Cases for clearing the overflow flag (1)

- overflow flag is turned off. (OF = 0 : $+, + \to +$)

```
signed addition          signed subtraction          unsigned addition

 0011    (+3)             0011    (+3)                 0011    (3)
+0011   +(+3)            -1101   -(-3)                +0011   +(3)
-----   -----            -----   -----                -----   -----
00110    (+6)            00110    (+6)                00110    (6)
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Cases for clearing the overflow flag (2)

- overflow flag is turned off. ($\texttt{OF} = 0 : -, - \to -$)

```
signed addition       signed subtraction       unsigned addition

 1101    (-3)          1101    (-3)             1101    (13)
+1101   +(-3)         -0011   -(+3)            +1101   +(13)
-----   -----         -----   -----            -----   -----
11010    (-6)         11010    (-6)            11010    (26)
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Cases for clearing the overflow flag (3)

- overflow flag is turned off. (OF = 0 : $+, - \to +$)

```
signed addition          signed subtraction           unsigned addition

 0100    (+4)             0100    (+4)                 0100     (4)
+1101   +(-3)            -0011   -(+3)                +1101   +(13)
-----   -----            -----   -----                -----   -----
10001    (+1)            10001    (+1)                10001    (17)
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Cases for clearing the overflow flag (4)

- overflow flag is turned off. (OF = $0 : +, - \to -$)

```
signed addition        signed subtraction        unsigned addition

 0011    (+3)           0011    (+3)              0011     (3)
+1100   +(-4)          -0100   -(+4)             +1100   +(12)
-----   -----          -----   -----             -----   -----
01111    (-1)          01111    (-1)             01111    (15)
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Cases for clearing the overflow flag (5)

- overflow flag is turned off. (`OF` = $0 : -, + \to +$)

```
signed addition          signed subtraction          unsigned addition

 1101   +(-3)             0011   -(+3)               1101   +(13)
+0100    (+4)            -0100    (+4)              +0100     (4)
-----   -----            -----   -----              -----   -----
10001    (+1)            10001    (+1)              10001    (17)
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Cases for clearing the overflow flag (6)

- overflow flag is turned off. (`OF` = 0 : $-, + \rightarrow -$)

```
signed addition           signed subtraction          unsigned addition

 1100  +(-4)               0100  -(+4)                  1100  +(12)
+0011   (+3)              -0011   (+3)                 +0011    (3)
-----  -----              -----  -----                 -----  -----
01111   (-1)              01111   (-1)                 01111   (15)
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

TOC Method 1 for computing the overflow flag

Adding two numbers with the same sign

- overflow can only happen
  when adding two numbers of the same sign
  results in a different sign.
- signed binary arithmetic
- to detect overflow
  - only the sign bits are considered
  - the other bits are ignored

`http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt`

Overflow conditions for additions and subtractions

- with two operands and one result,
  three sign bits are considered
  $2^3 = 8$ possible combinations
- only two cases result in overflow for an addition
  - 0 0 1 $(p + p \rightarrow n)$
  - 1 1 0 $(n + n \rightarrow p)$
- only two cases are considered as overflow for an subtraction
  - 0 1 1 $(p - n \rightarrow n)$
  - 1 0 0 $(n - p \rightarrow p)$

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Overflow condition for an addition

- Overflow in an addition ($num1 + num2$)

```
   num1 num2 sum (num1 + num2)
   sign sign sign
  -----------------------------------------------------
        0 0 0
*OVER*  0 0 1 (adding two positives should be positive)
        0 1 0
        0 1 1
        1 0 0
        1 0 1
*OVER*  1 1 0 (adding two negatives should be negative)
        1 1 1
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Overflow conditions for a subtraction

- Overflow in a subtraction ($num1 - num2$)

```
   num1 num2 sub (num1 - num2)
   sign sign sign
  ---------------------------------------------------------------------
        0 0 0
        0 0 1
        0 1 0
*OVER*  0 1 1 (subtracting a negative is the same as adding a positive)
*OVER*  1 0 0 (subtracting a positive is the same as adding a negative)
        1 0 1
        1 1 0
        1 1 1
```

  - subtracting a *positive* number is the same as adding a *negative*
  - subtracting a *negative* number is the same as adding a *positive*

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Overflow in signed computations

- A computer might contain a small logic gate array
  that sets the overflow flag to "1"
  iff any one of the above four OV conditions is met.

- in signed computations,
  adding two numbers of the same sign
  must produce a result of the same sign,
  otherwise overflow happened.

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

TOC Method 2 for computing the overflow flag

- Carry into and carry out of the sign bit
- Overflow in 2's complement arithmetic
- Overflow flag $= C_n \bigoplus C_{n-1}$
- Examples
- C_n and C_{n-1} in a n-bit addition
- Overflow flag computation
- Examples of computing overflow flag
- Hexadecimal carry, octal carry, decimal carry
- No carry into the sign bit

Carry into and carry out of the sign bit

- When adding two binary values,consider
  - the carry *coming into* the leftmost place
    (carry into the sign bit)
  - the carry *going out of* that leftmost place.
    (carry out of the sign bit)
    this is the carry flag in the ALU

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Overflow in 2's complement arithmetic

- overflow in 2's complement happens when
  - there is a carry *into* the sign bit
    but no carry *out of* the sign bit.
  - there is no carry *into* the sign bit
    but a carry *out of* the sign bit.

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Overflow flag $= C_n \bigoplus C_{n-1}$

- the overflow flag is the XOR
  - of the carry *coming into* the sign bit
  - with the carry *going out of* the sign bit
- overflow happens when
  the carry in does not equal to the carry out

`http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt`

Examples

- 4-bit signed 2's complement addition examples

```
 1100   (-4) (neg sign 1)
+0100   (+4) (pos sign 0)
========================
10000   ( 0) (pos sign 0)

carry in  1 (1+1+0)
carry out 1 (1+0+1)
1 XOR 1 = NO OVERFLOW
```

```
 1100 (-4) (neg sign 1)
+1000 (-8) (neg sign 1)
========================
10100 (+4) (pos sign 0)

carry in  0 (1+0+0)
carry out 1 (1+1+0)
0 XOR 1 = OVERFLOW!
```

```
 0100 (+4) (pos sign 0)
+0100 (+4) (pos sign 0)
========================
01000 (-8) (neg sign 1)

carry in  1 (1+1+0)
carry out 0 (0+0+1)
1 XOR 0 = OVERFLOW!
```

```
 1000 (-8) (neg sign 1)
+0100 (+4) (pos sign 0)
========================
01100 (-4) (neg sign 1)

carry in  0 (0+1+0)
carry out 0 (1+0+0)
0 XOR 0 = NO OVERFLOW
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

C_n and C_{n-1} in a n-bit addition

$(n-1)^{th}$ bit – MSB

- adding operations
  at the $(n-1)$ bit position
- $\{C_n, S_{n-1}\} =$
  $a_{n-1} + b_{n-1} + c_{n-1}$

$$
\begin{array}{cc}
 & \text{msb} \\
 & a_{n-1} \\
 & b_{n-1} \\
 & C_{n-1} \\
\hline
C_n & S_{n-1}
\end{array}
$$

- C_n :
  carry coming *out of* the msb

$(n-2)^{th}$ bit

- adding operations
  at the $(n-2)$ bit position
- $\{C_{n-1}, S_{n-2}\} =$
  $a_{n-2} + b_{n-2} + c_{n-2}$

$$
\begin{array}{cc}
\text{msb} & \\
 & a_{n-2} \\
 & b_{n-2} \\
 & C_{n-2} \\
\hline
C_{n-1} & S_{n-2}
\end{array}
$$

- C_{n-1} :
  carry coming *into* the msb

Overflow flag computation

ADD (addition)	SUB (subtraction)
OF = $C_n \bigoplus C_{n-1}$	OF = $C_n \bigoplus C_{n-1}$
a 2's complement addition $A + B = A + B + 0$	the transformed addition $A - B = A + \overline{B} + 1$
$\{C_n, S_{n-1}\}$ $= a_{n-1} + b_{n-1} + c_{n-1}$	$\{C_n, S_{n-1}\}$ $= a_{n-1} + \overline{b_{n-1}} + c_{n-1}$
$\{C_{n-1}, S_{n-2}\}$ $= a_{n-2} + b_{n-2} + c_{n-2}$	$\{C_{n-1}, S_{n-2}\}$ $= a_{n-2} + \overline{b_{n-2}} + c_{n-2}$

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Examples of computing overflow flag

- 4-bit signed addition examples

```
     +4        -7        +4        +6        -8        -4
     +4        -7        +1        -7        +1        -4
   -----    ------    ------     -----     -----     -----
     -8        +2        +5        -1        -7        -8

   0100      1001      0100      0110      1000      1100
   0100      1001      0001      1001      0001      1100
 ------    ------    ------    ------    ------    ------
  01000     10010     00000     00000     00000     11000
   1000      0010      0101      1111      1001      1000
 ------    ------    ------    ------    ------    ------
 C4 = 0    C4 = 1    C4 = 0    C4 = 0    C4 = 0    C4 = 1
 C3 = 1    C3 = 0    C3 = 0    C3 = 0    C3 = 0    C3 = 1
 ------    ------    ------    ------    ------    ------
 + +, -    - -, +    + +, +    + -, -    - +, -    - -, -
 ------    ------    ------    ------    ------    ------
 OF = 1    OF = 1    OF = 0    OF = 0    OF = 0    OF = 0
```

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Hexadecimal carry, octal carry, decimal carry

- Note that this XOR method
  only works with the binary carry that goes into the sign bit.
- not works with hexadecimal carry
  decimal carry, octal carry
  - the carry doesn't go into the sign bit
  - can't XOR that non-binary carry with the outgoing carry.

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

No carry into the sign bit

- Hexadecimal addition example
  (showing that XOR doesn't work for hex carry):
```
 8Ah
+8Ah
====
114h
```
- The hexadecimal carry of 1 resulting from A+A
  does not affect the sign bit.
- If you do the math in binary, you'll see
  that there is no carry into the sign bit;
  but, there is carry out of the sign bit.
  Therefore, the above example sets OVERFLOW on.
  (The example adds two negative numbers and gets a positive number.)

http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt