Machine Representation of Numbers

Digital computers store numbers in special digital electronic devices called

Registers

- Digital computers store numbers in special digital electronic devices called **Registers**
- **Registers** consist of a fixed number of storage elements.
- Each storage element is capable of storing one bit of data (either 0 or 1).
- Thus, every register has a *finite* number of bits

Registers consist of a fixed number of storage elements.

Each storage element is capable of storing one bit of data (either 0 or 1).

Thus, (every) register has a *finite* number of bits

- The register ***size*** is the number of storage bits in this register (*n*).
- Register size is typically a power of 2, e.g. 8, 16, 32, 64, etc.
- A register of *size n* can represent (store) a Number of *Distinct Values* (= 2^n).
- Registers are, thus, capable of holding binary numbers.
- Numbers stored in registers may be either *unsigned* or *signed* numbers. For example, **13**

The register *size* is the number of storage bits in this register (*n*).

Register size is typically a power of 2, e.g. 8, 16, 32, 64, etc.

A register of *size n* can represent (store) a Number of *Distinct Values* (= 2^n).equal to two to the power of n

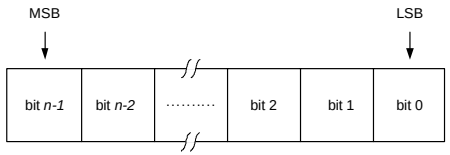
Registers are, thus, capable of holding binary numbers

Numbers stored in registers may be either (*unsigned*) or (*signed*) numbers. For example, **13** is an unsigned number while **+13** and **–13** are (signed) numbers.

is an unsigned number while **+13** and **–13** are signed numbers.

Unsigned_Reg.vsd

Unsigned Number Representation



N-Bit Register holding an n-Bit Unsigned Number

A register of n-bits, can store any unsigned number that has n-bits or less.

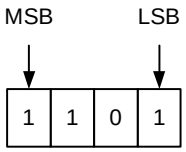
Typically the rightmost bit of the register is designated as the least significant bit (LSB), while the leftmost bit is the most-significant bit (MSB)

(When representing an integer), this register can hold values from 0 up to two to the n, minus 1.

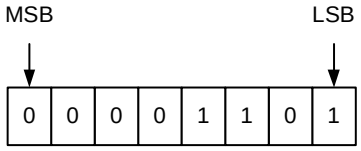
Example

Show how the value $(13)_{ten}$ (or **D** in Hexadecimal) is stored in a 4-bit register and in an 8-bit register

Unsigned_13.vsd



4-Bit Register Storing 13



8-Bit Register Storing 13

Zeros will be used to pad the binary representation of 13 in the 8-bit register

Signed Number Representation

- In addition to magnitude information, a signed number representation must also indicate whether the number is positive or negative.
- Two major techniques are used to represent signed numbers:
  1. Signed Magnitude Representation
  2. Complement method
  - Radix (R`s) Complement (2`s Complement)
  - Diminished Radix (R-1`s) Complement (1`s Complement)

In signed number representation, both the magnitude and sign information of the number must be represented.

Two major techniques are used to represent signed numbers.

One. is the (signed-magnitude) method, while the other is the complement representation method.

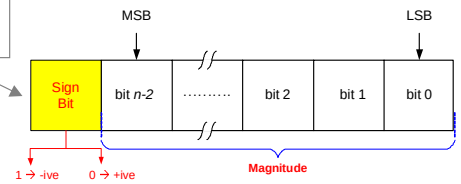
Two complement methods have been commonly used.

The first is the radix complement, (or the R's complement) while,

The second is the diminished radix complement or the R minus one's complement..

Signed Magnitude Number Representation

Signed_Mag.vsd



Signed-Magnitude Number Representation in N-Bit Register

The first signed number representation method is the (Signed-Magnitude) nethod

- •Independent Representation of The Sign and The Magnitude
- The leftmost bit is used as a *Sign Bit*.
- The remaining n-1 bits are used to represent the magnitude of the number

In this method, both the sign of the number and its magnitude are represented independent from one another

The leftmost bit is dedicated for use as a *Sign Bit*

For a register of size n-bits, one bit is used exclusively as a sign bit while the remaining (n-1) bits are used to represent the (magnitude).

- The *Sign Bit* :
  - = 0 → +ive number
  - = 1 → -ive number.

For positive numbers, a zero is stored in the sign bit.
For negative numbers, a one is stored in the sign bit.

Example

Show how the signed-magnitude representations of +6, -6, +13 and –13 using a 4-Bit register and an 8-Bit register

As an example,
Let us see how to represent plus six, minus six, plus thirteen and minus thirteen in a signed magnitude representation using a 4-bit register and an 8-bit register.

Solution

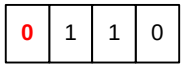
- **For a 4-bit register**, the leftmost bit is used as a sign bit, which leaves 3 bits only to represent the magnitude.

For a 4-bit register, the leftmost bit is used as a sign bit, which leaves 3 bits only to represent the magnitude

- The largest magnitude representable in 3-bits is 7. Accordingly, we cannot use a 4-bit register to represent +13 or –13.

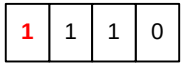
The largest magnitude representable in 3-bits is 7.

Accordingly, we cannot use a 4-bit register to represent plus thirteen or minus thirteen.



Signed-Magnitude Representation of +6

Here is the representation of Plus Six with a zero in the sign bit representing the plus sign and binary 6 value representing the magnitude.



Signed-Magnitude Representation of -6

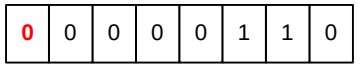
The representation of Minus Six is the same as that of Plus six except for the sign bit which holds a one instead of a zero to represent the minus sign rather than the plus..

- **For an 8-bit register**, the leftmost bit is a sign bit, which leaves 7 bits to represent the magnitude.
- The largest magnitude representable in 7-bits is 127 (= 2^7-1).

Now consider an eight-Bit register.

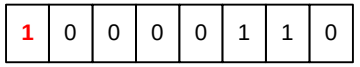
The leftmost bit is dedicated for the sign, while the remaining 7 bits will represent the magnitude

Thus the register can represent magnitudes up to 2 to the power of seven minus one or one hundred and twenty seven



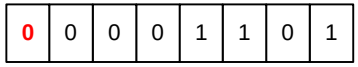
Signed-Magnitude Representation of +6

Here is the representation of Plus Six with a zero in the sign bit representing the plus sign and binary 6 value representing the magnitude.



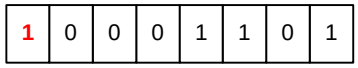
Signed-Magnitude Representation of -6

The representation of Minus Six is the same as that of Plus six except for the sign bit which holds a one instead of a zero to represent the minus rather than the plus sign.



Signed-Magnitude Representation of +13

Here is the representation of Plus thirteen with a zero in the sign bit representing the plus sign and binary 6 value representing the magnitude.



Signed-Magnitude Representation of -13

The representation of Minus thirteen is the same as that of Plus six except for the sign bit which stores a one instead of a zero to represent the minus sign.

PUT QUIZ SM1 here (See SM1_A04_Quiz.rtf)

Notes

- Signed magnitude method has Two representations for 0 → {+0 , -0} → nuisance for implementation
  - Harder to implement addition/subtraction
  - Multiplication & division less problematic
- Signed magnitude method has a symmetric range of representation $\{-(2^{n-1} -1) : +(2^{n-1} -1)\}$

Here are some concluding notes.

The signed magnitude method has Two representations for 0 a plus zero where the sign bit is zero and the magnitude is zero. (AND) a minus zero where the sign bit is 1 but the magnitude is zero.

Having two representations for zero is an implementation nuisance.

It causes addition and subtraction to be Harder to implement while it makes Multiplication & division less problematic

The signed magnitude method has a symmetric range of representation between plus and minus two to the power of minus one (minus) one

The Second signed number representation method is the (Complement) nethod

Complement Representation

- Positive Numbers (+N) Are Represented in Exactly the Same Way as in Signed Magnitude System

In this method, Positive Numbers Are Represented in (Exactly the Same Way) as in Signed Magnitude System

- Negative Numbers (-N) Are Represented by the *Complement* of N (**N'**)

Negative Numbers, however, Are Represented by the (*Complement*) of that number .

The complement of some number N will be referrd to as N-Prime

Define the Complement of a number N as: **N' =**

M -N *where, **M** = Some Constant*

- The negation of some number N, i.e **–N**, is represented by the *Complement* (**N'**) of that number.

The negation of some number (N), that is minus N, is represented by the (*Complement*) of that number (that is N-Prime).

Important Property:

The following is a very important property of Complements

- The Complement of the Complement of some number *N* is the same number *N*.

(**N'**)**'** = M- (M-N) = N

- This is a required property to match the negation process since negating any number twice yields the original number , i.e. - (-**N**) = N}

This is a required property to match the negation (process) since since negating any number twice yields the original number

In other words, minus-minus-(N) equals N

Why Use the Complement Method ?

Through the proper choice of the constant **M**, the complement operation can be fairly simple and quite fast. A simple complement process allows:

- Simplified arithmetic operations since. subtraction can be totally replaced by addition and complementing.

Now, an important question is – (Why) (Use) (the) (Complement) (method)?

With the proper choice of the value of (M), complementing a number can be a very fast and efficient operation

A simple complement operation would lead to simplified and (LOW)-(Cost) arithmetic operations.

This is due to the fact that subtraction can be replaced by addition to the complement.
This means we can perform both addition and subtraction using only an (ADDER) hardware. In other words, no SUbtractor hardware is needed

- Lower cost, since no subtractor circuitry will be required and only an adder is needed.

Let us take a first look into computer arithmetic using the complement method

Complement Arithmetic

Negation is replaced by complementing. In other words, minus N is represented by the

Basic Rules

1. Negation is replaced by complementing (- N → N')
2. Subtraction is replaced by addition to the complement

   Thus, (X-Y) is replaced by (X+Y')

Likewise, Subtraction is replaced by Addition to the complement.

Thus X minus Y is replaced by X plus Y-Prime

Choice of M

The following discussion sheds some light on the issues involved in the choice of a proper value for M

- Consider the operation Z = X – Y, where

  both X and Y are positive numbers

- In complement arithmetic, Z is computed

  by adding X to the *complement* of Y

$$Z = X + Y'$$

Consider the following two possible cases:

First case Y > X → (Negative Result)

The result **Z** is **–ive** and is equal to **–(Y-X)**, i.e. it should be in the *complement form*:

$$Z = X + Y' = X + (M-Y)$$

To do that, we investigate the computation of the equation Z equals X

We now consider two cases. The first is when Y is greater than X, or in other words, the result of subtraction, Zee, is negative.

The second is when Y is (less) than X, or in other words, the result of subtraction, Zee, is positive.

Comsider The first is where Y is greater than X, yielding a negative result for, Zee which will equal minus the difference between Y and X.

Using the complement method, Zee is computed as X plus Y-Prime.

Replacing Y-prime with M minus Y, the resulting Zee Value is M minus the difference between Y and X.
This is the correct complement method representation of the answer. Since the negative result should be expressed in the complement form.

$$= M - (Y-X)$$

$$= \textit{Correct Answer in the Complement Form}$$

- Thus, in the case of a *negative result*, any value of M is possible and the only criterion for choosing M is the simplicity of the complement operation

Second case Y < X → (Positive Result)

The result **Z** is **+ive** equal to **+(X-Y)**. Using complement arithmetic we get:

$$Z = X + Y' = X + (M-Y)$$

$$= M + (X-Y)$$

In this case, any value of M will yield the correct result.

In the second case, Y is less than X, yielding a positive result for, Zee which equals the difference between X and Y.

Again, Using the complement method, Zee is computed as X plus Y-Prime.

Replacing Y-prime with M minus Y, the resulting Zee Value is M plus the difference between X and Y, while it should be the difference between X and Y. Thus, before reporting the final answer a correction step is required.

different from the **correct result +(X-Y)**

requires

- Thus, in the case of a *positive result*, a *correction step* is required for the final result. This places an additional constraint on the choice of M

Thus, in the case of a *positive result*, , irrespective of the value of M, a *correction step* is required for the final result.

This places an additional constraint on the choice of M

To summarize, there are two constraints on the choice of M

1. Simple and fast complement operation.
2. Elimination or simplification of the correction step

Thus, the choice of M should satisfy two conditions:

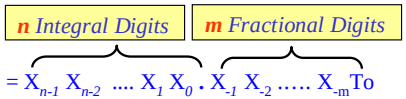
First., The Value of M should yield a fast and efficient complement operation.

Second., The value of M should simplify the correction step

Now, Consider the number **X**, which has *n* integral digits and *m* fractional digits.

Thus, X equals X N minus one X N minus two all the way to X two X 1 X zero (point) X minus one X minus two all the way to X minus M.

Consider the number **X**, with *n* integral digits and *m* fractional digits, where



obtain the complement of X, two methods are commonly used. The two methods differ in the choice of the value of M.

1. Radix complement (R's Complement) method:
2. Diminished radix complement (R-1's Complement) method:

To obtain the complement of X, two methods are commonly used.

First. The radix complement method, commonly called the R's complement.

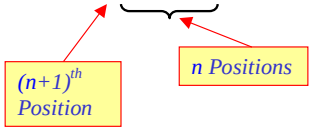
Second. The diminished radix complement method, commonly called the R minus one's complement

Radix Complement (R's Complement):

$$M = r^n$$

In the R's complement Method, M equals R to the power of n where n is the number of integral digits.

Note that = ***1***000….000



Please note that r to the power of n is an integer number that consists of n zeros followed by a one in the n plus one's position.

This is true for any radix.

Diminished Radix Complement (R-1's) Complement):

$$M = r^{n} - r^{-m}$$

where; r^{-m} = 000…00 . 00…00**1**

n Positions | m Positions

Thus; r^{-m} = 000…00 . 00…00**1**

= *Unit (one) in Least Position* (*ulp*)

OR $M = r^{n} - ulp$

Notes:

1. *For integer numbers, m=0*
2. *R's complement R-1's complement + ULP*

In the R minus one's complement Method, M equals r to the power of (n) minus r to the power of minus (m) where n is the number of integral digits while where m is the number of fractional digits

Actually, the value of r to the power of minus (m) corresponds to a number where all digits are zeros except for the emth fractional position where it has a one. That is a one in the least significant position only and zeros everywhere else.

This is true for any radix value.

r to the power of minus (m) is, thus, referred to as Unit in the Least Position, or U L P.

Note that for integer numbers, m equals zero.

Please note the following
One. For integer numbers, m equals zero

Two. The R's complement of a number equals its R minus one's complement (PLUS) U L P.

3. $(r^n - r^{-m})$ is the largest number that can be represented in *n*-integral digits and *m*-fractional digits

Three. Note that r to the n minus r to the minus m is the largest number that can be represented in *n*-integral digits and *m*-fractional digits. For example, in decimal systems it will be 9 filling all digits. For binary system all digits are all one's and so on.

The table below summarizes the radix complement computation of X for various number systems

Number System	R's Complement	Complement of X (X'_r)
Decimal	10's Complement	$X'_{10}=10^n-X$
Binary	2's Complement	$X'_2=2^n-X$
Octal	8's Complement	$X'_8=8^n-X$
Hexadecimal	16's Complement	$X'_{16}=16^n-X$

The shown table summarizes the R's complement computation of X for various number systems.

For the decimal system it is the ten's complement.

For the binary system it is the two's complement.

For the octal system it is the eight's

The shown table summarizes the (r-1)'s complement computation of X for various number systems

Number System	(R-1)'s Complement	Complement of X (X'_r)
Decimal	9's Complement	$X'_9=(10^n-10^{-m})-X$ = 99_9.999..9-X
Binary	1's Complement	$X'_1=(2^n-2^{-m})-X$ = 11_1.111..1-X
Octal	7's Complement	$X'_7=(8^n-8^{-m})-X$ = 77_7.777..7-X

The shown table summarizes the R minus one's complement computation of X for various number systems

For the decimal system it is the nine's complement.

For the binary system it is the one's complement.

Hexadecimal	F's Complement	$X'_F=(16^n-16^{-m})-X$ = FF□.FF..F-X

Here are few examples for computing the complement in various bases

Examples

This example deals with the decimal system and we are to compute the nine's and ten's complement of two numbers. One is an integer while the other is a real number.

Find the 9's and the 10's complement of the following decimal numbers:

a- 2357

b- 2895.786

Solution:

a- X = 2357→ *n=4,*

- $X'_9=(10^4 - ULP) - 2357$

  $= 9999 - 2357 = 7642$

The first number is twenty three fifty seven. The number of digits (n) is four.

Since the number has no fractional part (m) is zero and the nine's complement is equal to 9999 minus twenty three fifty seven.

The ten's complement is computed as ten to the power of four minus twenty three fifty seven, or, alternatively computed by adding a U L P to the nine's complement

- $X'_{10}=10^4 - 2357 = 7643$;
- *Alternatively*, $X'_{10}= X'_9 + 0001 =$ 7643

b- X = 2895.786→ *n=4, m=3*

- $X'_9=(10^4 - ULP) - 2895.786$

The second number has 4 integral digits and three fractional ones. That is, n equals 4 and m equals 3.

Thus, the nine's complement is ninety nine ninety nine point nine nine nine minus the number.

The ten's complement is computed as ten to the power of four minus the number, or, alternatively computed by adding a U L P to the nine's complement

= 9999.999 – 2895.786=

7104.213

- $X'_{10}=10^4 - 2895.786= 104.214$;
- *Alternatively*,

  $X'_{10} = X'_9 + 0000.001= 7104.214$

Example

Find the 1's and the 2's complement of the following binary numbers:

a- 110101010

b- 1010011011

c- 1010.001

Solution:

a- X = 110101010→ *n=9*,

This example deals with the binary system and we are to compute the one's and two's complement of three numbers. Two are integer numbers and the third is a real number.

The first number has nine bits, that is (n) equals 9.

The one's complement is computed by subtracting the number from nine one's..

The two's complement is computed as two to the power of nine minus the number, or, alternatively computed by adding a U L P to the one's complement

- $X'_1=(2^g-ULP)-110101010$

  $= 111111111 - 110101010$

  $= 001010101$

- $X'_2=2^g - 110101010$

  $= 1000000000 - 110101010$

  $= 001010110$

- *Alternatively*, $X'_2= X'_1 + ulp$

  $= 001010101 + 000000001$

  $= 001010110$

b- $X = 1010011011 \rightarrow n=10$,

- $X'_1=(2^{10}-ULP)-101001101$

  $= 1111111111 - 101001101$

  $= 010110010$

- $X'_2=2^{10} - 101001101$

  $= 10000000000 - 101001101 = 010110011$

- *Alternatively*, $X'_2 = X'_1 + ulp$

  $= 010110010 + 0000000001$

  $= 010110011$

c- $X = 1010.001 \rightarrow n=4, m=3$

- $X'_1 = (2^4 - ULP) - 1010.001$

  $= 1111.111 - 1010.001$

  $= 0101.110$

- $X'_2 = 2^4 - 1010.001$

  $= 10000 - 1010.001$

  $= 0101.111$

- *Alternatively*, $X'_2 = X'_1 + ulp$

  $= 0101.110 + 0000.001$

  $= 0101.111$

Important Notes:

1. The 1's complement of a number can be directly obtained by bitwise

The one's complement of a number can be directly obtained by bitwise complementing of each bit, that is-. each one is replaced by a zero and

complementing of each bit, i.e. each 1 is replaced by a 0 and each 0 is replaced by a 1.

- Example: X = 1 1 0 0 1 0 1 0 0 1
- X_1' = 0 0 1 1 0 1 0 1 1 0

Animation here is required where every bit of X_1' is shown as

2. The 2's complement of a number can be obtained visually as follows:
   - Scan the binary number from right to left.
   - 0's are replaced by 0's till the first 1 is encountered.
   - The first 1 is replaced by a 1 but from this point onwards each bit is

complemented replacing each 1 by

a 0 and each 0 by a 1

- Example: X = 1 1 0 0 1 0 1 0 0

- X_2' = 0 0 1 1 0 1 1 0 0

Animation here is required where every bit of X_1 is scanned right to left and the corresponding X_2' bit is

Example

Find the 7's and the 8's complement of the following octal numbers:

a- 6770

b- 541.736

Solution:

a- X = 6770→ *n=4,*

- $X'_7=(8^4-ULP)-6770$

  $= 7777 - 6770$

  $= 1007$

- $X'_8=8^4 - 6770$

  $= 1000000000 - 6770$

  $= 1010$

- *Alternatively*, $X'_8= X'_7 + ulp$

  $= 1007+ 0001$

  $= 1010$

b- X = 541.736→ *n=3*, → *m=4*

- $X'_7=(2^3-ULP)-541.736$

  $= 777.7777 - 541.736$

  $= 236.041$

- $X'_8=2^{10} - 541.736$

  $= 10000000000 - 541.736$

  $= 236.042$

- *Alternatively*, $X'_8 = X'_7 + ulp$

  $= 236.041 + 0000000001$

  $= 236.042$

Example

Find the F's and the 16's complement of the following HEX numbers:

a- 3FA9

b- 9B1.C70

Solution:

a- X = 3FA9 → *n=4,*

- $X'_F = (16^4 - ULP) - 3FA9$

  $= FFFF - 3FA9$

  $= C056$

- $X'_{16} = 16^4 - 3FA9$

  $= 1000000000 - 3FA9$

  $= C057$

- *Alternatively*, $X'_{16} = X'_F + ulp$

  $= C056 + 0001$

  $= C057$

b- $X = 9B1.C70 \rightarrow n=3, \rightarrow m=3$

- $X'_F = (16^3 - ULP) - 9B1.C70$

  $= FFF.FFF - 9B1.C70$

  $= 64E.38F$

- $X'_{16} = 16^3 - 9B1.C70$

  $= 10000000000 - 9B1.C70 = 64E.390$

- *Alternatively*, $X'_{16} = X'_F + ulp$

  $= 64E.38F + 000.001$

  $= 64E.390$

Example

Show how the numbers +53 and -53 are represented in 8-bit registers using signed-

magnitude, 1's complement and 2's complement representations.

	+53	-53
Signed Magnitude	00110101	10110101
1's Complement	00110101	11001010
2's Complement	00110101	11001011

An important point to remember is that In all signed number representation methods, the sign bit indicates the sign of the number with one representing

Note: In all signed number representation methods, the sign bit indicates the sign of the number with 1 representing negative numbers and 0 representing positive numbers.

Quiz:

For the shown 4-bit representations, indicate the corresponding decimal value in the shown representations.

	Unsigned	Signed Magnitude	1's Complement	2's Complement
0000				
0001				
0010				
0011				
0100				
0101				
0110				

0111				
1000				
1001				
1010				
1011				
1100				
1101				
1110				
1111				

Correct Answer

	Unsigned	Signed Magnitude	1's Complement	2's Complement
0000	**0**	**+0**	**+0**	**+0**
0001	**1**	**+1**	**+1**	**+1**
0010	**2**	**+2**	**+2**	**+2**
0011	**3**	**+3**	**+3**	**+3**
0100	**4**	**+4**	**+4**	**+4**
0101	**5**	**+5**	**+5**	**+5**
0110	**6**	**+6**	**+6**	**+6**
0111	**7**	**+7**	**+7**	**+7**
1000	**8**	**-0**	**-7**	**-8**
1001	**9**	**-1**	**-6**	**-7**
1010	**10**	**-2**	**-5**	**-6**
1011	**11**	**-3**	**-4**	**-5**

1100	**12**	**-4**	**-3**	**-4**
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1

Comparison:

	Signed Magnitude	1's Complement	2's Complement
Symmetric	yes	yes	no
No. of Zeros	2	2	1
Largest Value	$+(2^{n-1}-1)$	$+(2^{n-1}-1)$	$+2^{n-1}$
Smallest	$-(2^{n-1}-1)$	$-(2^{n-1}-1)$	$-(2^{n-1}-1)$

Value			

End of Lessons Exercises

1. Find the binary representation in signed magnitude, 1's complement, and 2's complement for the following decimal numbers: +13, -13, +39, -39, +1, -1, +73 and -73. For all numbers, show the required representation for 6-bit and 8-bit registers

2. Indicate the decimal value corresponding to all 5-bit binary patterns if the binary pattern is interpreted as a number in the signed magnitude, 1's complement, and 2's complement representations.