

### 3.5.1 Integer constants

*Integer constants* can be specified in decimal, hexadecimal, octal, or binary format.

There are two forms to express integer constants. The first form is a simple decimal number, which shall be specified as a sequence of digits 0 through 9, optionally starting with a plus or minus unary operator. The second form specifies a *based constant*, which shall be composed of up to three tokens—an optional size [constant specification](#), an apostrophe character (' , ASCII 0x27) followed by a base format character, and the digits representing the value of the number. [It shall be legal to macro substitute these three tokens.](#)

The first token, a size [constant specification](#), shall specify the size of the constant in terms of its exact number of bits. It shall be specified as a non-zero unsigned decimal number. For example, the size specification for two hexadecimal digits is 8, because one hexadecimal digit requires 4 bits.

The second token, a `base_format`, shall consist of a case insensitive letter specifying the base for the number, optionally preceded by the single character `s` (or `S`) to indicate a signed quantity, preceded by the apostrophe character. Legal base specifications are `d`, `D`, `h`, `H`, `o`, `O`, `b`, or `B`, for the bases decimal, hexadecimal, octal, and binary respectively.

The apostrophe character and the base format character shall not be separated by any white space.

The third token, an unsigned number, shall consist of digits that are legal for the specified base format. The unsigned number token shall immediately follow the base format, optionally preceded by white space. The hexadecimal digits `a` to `f` shall be case insensitive.

Simple decimal numbers without the size [specification](#) and the base format shall be treated as *signed integers*, whereas the numbers specified with the base format shall be treated as signed integers if the `s` designator is included or as *unsigned integers* if the base format only is used. The `s` designator does not affect the bit pattern specified, only its interpretation.

A plus or minus operator preceding the size [constant specification](#) is a unary plus or minus operator. A plus or minus operator between the base format and the number is an illegal syntax.

*Negative numbers* shall be represented in 2's complement form.

An `x` represents the *unknown value* in hexadecimal, octal, and binary constants . A `z` represents the *high-impedance value*. See [4.1](#) for a discussion of the Verilog HDL value set. An `x` shall set 4 bits to unknown in the hexadecimal base, 3 bits in the octal base, and 1 bit in the binary base. Similarly, a `z` shall set 4 bits, 3 bits, and 1 bit, respectively, to the high-impedance value.

~~If the size of the unsigned number is smaller than the size specified for the constant, the unsigned number shall be padded to the left with zeros. If the leftmost bit in the unsigned number is an `x` or a `z`, then an `x` or a `z` shall be used to pad to the left respectively. If the size of the unsigned number is larger than the size specified for the constant, the unsigned number shall be truncated from the left.~~

[If a sized constant has an unsigned number \(i.e., third token\) with a value that has fewer bits than are specified by its size specification \(i.e., first token\), the value of the unsigned number shall be extended to the specified number of bits by padding it to the left with zero \(even if the constant is signed\), unless the leftmost bit in the unsigned number is `x` or `z`, in which case it shall be padded to the left with `x` or `z` respectively. If the size of the unsigned number is larger than the size specified for the constant, the unsigned number shall be truncated from the left.](#)

[The number of bits that make up an unsized integer constant \(which is a simple decimal number or a based constant without a size specification\) shall be at least 32.](#) Unsized unsigned constants where the high order bit is unknown (`X` or `x`) or three-state (`Z` or `z`) shall be extended to the size of the expression containing the constant.

NOTE—In IEEE Std 1364-1995, in unsized constants where the high order bit is unknown or three-state, the x or z was only extended to 32 bits.

The use of x and z in defining the value of a number is case insensitive.

When used in a number, the question-mark ( ? ) character is a Verilog HDL alternative for the z character. It sets 4 bits to the high-impedance value in hexadecimal numbers, 3 bits in octal, and 1 bit in binary. The question mark can be used to enhance readability in cases where the high-impedance value is a don't-care condition. See the discussion of **casez** and **casex** in [9.5.1](#). The question-mark character is also used in user-defined primitive state tables. See [8.1.6, Table 8-1](#).

In a decimal constant, the unsigned number token shall not contain any x, z, or ? digits, unless there is exactly one digit in the token, indicating that every bit in the decimal constant is x or z.

The underscore character ( \_ ) shall be legal anywhere in a number except as the first character. The underscore character is ignored. This feature can be used to break up long numbers for readability purposes.

*Examples:*

*Example 1—Unsized constant numbers*

```
659          // is a decimal number
'h 837FF     // is a hexadecimal number
'o7460      // is an octal number
4af         // is illegal (hexadecimal format requires 'h)
```

*Example 2—Sized constant numbers*

```
4'b1001     // is a 4-bit binary number
5 'D 3      // is a 5-bit decimal number
3'b01x      // is a 3-bit number with the least
             // significant bit unknown
12'hx       // is a 12-bit unknown number
16'hz       // is a 16-bit high-impedance number
```

*Example 3—Using sign with constant numbers*

```
8 'd -6     // this is illegal syntax
-8 'd 6     // this defines the two's complement of 6,
             // held in 8 bits—equivalent to -(8'd 6)
4 'shf      // this denotes the 4-bit number '1111', to
             // be interpreted as a 2's complement number,
             // or '-1'. This is equivalent to -4'h 1
-4 'sd15    // this is equivalent to -(-4'd 1), or '0001'
16'sd?      // the same as 16'sbz
```

*Example 4—Automatic left padding*

```
reg [11:0] a, b, c, d ;
reg [31:0] e, f, g, h, i ;
reg signed [31:0] j, k ;

initial begin
    a = 'h x;          // a gets 12'h xxx
    b = 'h 3x;        // b gets 12'h 03x
    c = 'h z3;        // c gets 12'h zz3
    d = 'h 0z3;       // d gets 12'h 0z3

    e = 'h 5;         // e gets 32'h 0000_0005
    f = 'h x;         // f gets 32'h xxxx_xxxx
    g = 'h z;         // g gets 32'h zzzz_zzzz

    h = 4'h x ;      // h gets 32'h 0000_000x
    i = 16'o x ;     // i gets 32'h 0000_xxxx
    j = 16'sb 110;   // j gets 32'h 0000_0006
    k = 3'sb 110 ;   // k gets 32'h ffff_fffe
end
```

*Example 5—Using underscore character in numbers*

```
27_195_000
16'b0011_0101_0001_1111
32 'h 12ab_f001
```

NOTES:

1) Sized negative constant numbers and sized signed constant numbers are sign-extended when assigned to a **reg** data type, regardless of whether the **reg** itself is signed or not.

~~2) Each of the three tokens for specifying a number may be macro substituted.~~

~~3) The number of bits that make up an unsized number (which is a simple decimal number or a number without the size specification) shall be at least 32.~~