Types

Introductions

Although ZX BASIC was originally designed with ZX Spectrum in mind, it is a three-stage *retargeable* compiler. This means it should not be difficult to hack ZX BASIC SDK to compile for other target machines (E.g. Commodore 64) or current machines (PC) or even virtual machines like Java or .NET. Porting ZX BASIC to other Z80 architectures like Amstrad or MSX should be almost straightforward (only the library.asm should need some work to use different ROM routines).

Once said that, *data types* were designed for the Z80 platform. So, for example 64 bits integers are not implemented.

Types

A data *type* describes how the information of a variable is stored in memory. For example, a variable can hold a floating point number or a string in the legacy Sinclair BASIC. But sometimes this is a waste of time and memory. For example, why don't we use a single byte to store a value we know it's always an integer between 0 and 10?

Now you can: ZX BASIC allows more data types to save memory and achieve higher speed.

There are 3 kinds of types: **integrals** (integer numbers), **decimals** and **strings**

Integral

Integrals are numerical types to store integer values. They can be *unsigned* (their value is always 0 or positive) or *signed* (can take negative values). ZX Basic integer types sizes are 8, 16 and 32 bits. Unsigned types have the prefix *U*.

Type name	Size (bytes)	Signed?	Range	Description
Byte	1	yes	-128..127	8 bits signed integer
UByte	1	no	0..255	8 bits unsigned in
Integer	2	yes	-32768..32767	16 bits signed integer

Type name	Size (bytes)	Signed?	Range	Description
UInteger	2	no	0..65535	16 bits unsigned integer
Long	4	yes	−2,147,483,648 .. +2,147,483,647	32 bit signed integer
ULong	4	yes	0 .. 4,294,967,295	32 bit unsigned integer

Decimals

Decimals, as suggested, stores decimal numeric types. Their sizes are 32 bit for `Fixed` type and 40 bits for `Float` one.

Fixed

32 bit Fixed Point decimal. First 16 bits are the integer part, whilst remaining 16 contains the decimal one. Ranges from -32767.9999847 to 32767.9999847 with a precision of 1 / 2^16 (0.000015 aprox.). Fixed points decimal are less precise than Floating ones, but much faster and requires less space (1 byte less). Also, their range is much limited. They're usually used on screen drawing when Floating point is too slow and decimal calculations are required.

Float

Floating point type is **identical** to the Sinclair BASIC one. It requires 5 bytes (1 byte for exponent, 4 bytes for mantissa). Read the ZX Spectrum manual or here.

> To store the number in the computer, we use five bytes, as follows:
>
> Write the first eight bits of the mantissa in the second byte (we know that the first bit is 1), the second eight bits in the third byte, the third eight bits in the fourth byte and the fourth eight bits in the fifth byte; replace the first bit in the second byte which we know is 1 by the sign: 0 for plus, 1 for minus; write the exponent +128 in the first byte. For instance, suppose our number is 1 / 10, then
> $1 / 10 = (4 / 5) * 2^{(-3)}$

Strings

String types are used to store alphanumerical strings. Strings can contain up to 65535 characters, and they can change its size dynamically so, unlike other data types, their content is stored in a different memory area, called the heap.

In most BASIC dialects, string variables used to have the **$** suffix (also called *sigil*), but suffixes are optional in ZX BASIC (you can omit them).

Classes

At this moment, ZX BASIC is not an OOP compiler. But there are three main *classes*. A class can be considered as a *kind of type*.

Var

Vars are *scalar* variables. Scalar variables are those which store a single value. Almost all variables are scalars:

```
REM A simple scalar variable
DIM a = 3
```

Arrays

Unlike scalars, array variables can hold more than a single value at once. You access a single value within the array container using an integer *index*:

```
REM An array variable
DIM a(1 TO 10) AS UBYTE
LET a(3) = 5: REM pick a(3) cell and store the number 5 in it
```

Labels

Unlike the above, labels are not variables. They refer to memory positions. Line numbers are also treated as labels and they are completely optional:

```
10 REM here '10' is a Label
 5 REM here '5' is another label, so the number order does not matter
   REM Line numbers are optional. So this line is ok either.
   ' This line is also a comment. The character (') is like REM

mylabel:
   REM the above 'mylabel' identifier is also a Label. When not a line number,
   REM they must be ended with a colon (:)

   GOTO mylabel: REM Goes to 'mylabel' line

   GO TO 5: REM Another way to write GOTO. Jumps to line 5
```

Read here for more info about labels.

See Also

- DIM (statement)

- Labels