low level TEX

registers

Contents

1 Preamble 1
2 TEX primitives 1
3 ε-TEX primitives 4
4 LuaTEX primitives 4
5 LuaMetaTEX primitives 5

1 Preamble

Registers are sets of variables that are accessed by index and a such resemble registers in a processing unit. You can store a quantity in a register, retrieve it, and also manipulate it.

There is hardly any need to use them in ConTEXt so we keep it simple.

2 TEX primitives

There are several categories:

- Integers (int): in order to be portable (at the time it surfaced) there are only integers and no floats. The only place where TEX uses floats internally is when glue gets effective which happens in the backend.
- Dimensions (dimen): internally these are just integers but when they are entered they are sliced into two parts so that we have a fractional part. The internal representation is called a scaled point.
- Glue (skip): these are dimensions with a few additional properties: stretch and shrink. Being a compound entity they are stored differently and thereby a bit less efficient than numbers and dimensions.
- Math glue (muskip): this is the same as glue but with a unit that adapts to the current math style properties. It's best to think about them as being relative measures.
- Token lists (toks): these contain a list of tokens coming from the input or coming from a place where they already have been converted.

The original TEX engine had 256 entries per set. The first ten of each set are normally reserved for scratch purposes: the even ones for local use, and the odd ones for global usage. On top of that macro packages can reserve some for its own use. It was quite

easy to reach the maximum but there were tricks around that. This limitation is no longer present in the variants in use today.

Let's set a few dimension registers:

```
\dimen 0 = 10 pt
\dimen2=10pt
\dimen4 10pt
\scratchdimen 10pt
```

We can serialize them with:

```
\the    \dimen0
\number \dimen2
\meaning\dimen4
\meaning\scratchdimen
```

The results of these operations are:

```
10.0pt
655360
\dimen4
dimension 10.0pt
```

The last two is not really useful but it is what you see when tracing options are set. Here `\scratchdimen` is a shortcut for a register. This is *not* a macro but a defined register. The low level `\dimendef` is used for this but in a macro package you should not use that one but the higher level `\newdimen` macro that uses it.

```
\newdimen\MyDimenA
\def     \MyDimenB{\dimen999}
\dimendef\MyDimenC998

\meaning\MyDimenA
\meaning\MyDimenB
\meaning\MyDimenC
```

Watch the difference:

```
\dimen264
macro:\dimen 999
\dimen998
```

The first definition uses a yet free register so you won't get a clash. The second one is just a shortcut using a macro and the third one too but again direct shortcut. Try to imagine how the second line gets interpreted:

```
\MyDimenA10pt \MyDimenA10.5pt
\MyDimenB10pt \MyDimenB10.5pt
\MyDimenC10pt \MyDimenC10.5pt
```

Also try to imagine what messing around with `\MyDimenC` will do when we also have defined a few hundred extra dimensions with `\newdimen`.

In the case of dimensions the `\number` primitive will make the register serialize as scaled points without unit `sp`.

Next we see some of the other registers being assigned:

```
\count  0 = 100
\skip   0 = 10pt plus 3pt minus 2pt
\skip   0 = 10pt plus 1fill
\muskip 0 = 10mu plus 3mu minus 2mu
\muskip 0 = 10mu minus 1 fil
\toks   0 = {hundred}
```

When a number is expected, you can use for instance this:

```
\scratchcounter\scratchcounterone
```

Here we use a few predefined scratch registers. You can also do this:

```
\scratchcounter\numexpr\scratchcounterone+\scratchcountertwo\relax
```

There are some quantities that also qualify as number:

```
\chardef\MyChar=123 % refers to character 123 (if present)
\scratchcounter\MyChar
```

In the past using `\chardef` was a way to get around the limited number of registers, but it still had (in traditional TEX) a limitation: you could not go beyond 255. The `\mathchardef` could fo higher as it also encodes a family number and class. This limitation has been lifted in LuaTEX.

A character itself can also be interpreted as number, in which case it has to be prefixed with a reverse quote: `, so:

```
\scratchcounter\numexpr`0+5\relax
```

```
\char\scratchcounter
```

produces “5” because the `0 expands into the (ascii and utf8) slot 48 which represents the character zero. In this case the next makes more sense:

```
\char\numexpr`0+5\relax
```

If you want to know more about all these quantities, “TEX By Topic” provides a good summary of what TEX has to offer, and there is no need to repeat it here.

3 ε-TEX primitives

Apart from the ability to use expressions, the contribution to registers that ε-TEX brought was that suddenly we could use upto 65K of them, which is more than enough. The extra registers were not as efficient as the first 256 because they were stored in the hash table, but that was not really a problem. In Omega and later LuaTEX regular arrays were used, at the cost of more memory which in the meantime has become cheap. As ConTEXt moved to ε-TEX rather early its users never had to worry about it.

4 LuaTEX primitives

The LuaTEX engine introduced attributes. These are numeric properties that are bound to the nodes that are the result of typesetting operations. They are basically like integer registers but when set their values get bound and when unset they are kind of invisible.

- Attribute (attribute): a numeric property that when set becomes part of the current attribute list that gets assigned to nodes.

Attributes can be used to communicate properties to Lua callbacks. There are several functions available for setting them and querying them.

```
\attribute999 = 123
```

Using attributes this way is dangerous (of course I can only speak for ConTEXt) because an attribute value might trigger some action in a callback that gives unwanted side effects. For convenience ConTEXt provides:

```
\newattribute\MyAttribute
```

Which currently defines `\MyAttribute` as `integer 1025` and is meant to be used as:[1]

[1] The low level `\attributedef` command is rather useless in the perspective of ConTEXt.

```
\attribute\MyAttribute = 123
```

Just be aware that defining attributes can have an impact on performance. As you cannot access them at the TEX end you seldom need them. If you do you can better use the proper more high level definers (not discussed here).

5 LuaMetaTEX primitives

The fact that scanning stops at a non-number or `\relax` can be sort of unpredictable which is why in LuaMetaTEX we also support the following variant:

```
\scratchdimen\dimexpr  10pt + 3pt \relax
\scratchdimen\dimexpr {10pt + 3pt}
```

At the cost of one more token braces can be used as boundaries instead of the single `\relax` boundary.

An important property of registers is that they can be accessed by a number. This has big consequences for the implementation: they are part of the big memory store and consume dedicated ranges. If we had only named access TEX's memory layout could be a bit leaner. In principle we could make the number of registers smaller because any limit on the amount at some point can be an obstacle. It is for that reason that we also have name-only variants:

```
\dimensiondef \MyDimenA   12pt
\integerdef   \MyIntegerA 12
\gluespecdef  \MyGlueA    12pt + 3pt minus 4pt
\mugluespecdef\MyMuA      12mu + 3mu minus 4mu
```

These are as efficient but not accessible by number but they behave like registers which means that you (can) use `\the`, `\advance`, `\multiply` and `\divide` with them.[2] In case you wonder why there is no alternative for `\toksdef`, there actually are multiple: they are called macros.

todo: expressions

[2] There are also the slightly more efficient `\advanceby`, `\multiplyby` and `\divideby` that don't check for the `by` keyword.

5 Colofon

Author	Hans Hagen
ConTEXt	2023.04.27 17:04
LuaMetaTEX	2.1008
Support	www.pragma-ade.com
	contextgarden.net