Title: ISO/IEC TR 18037 issues
Status: For discussion at the WG14 Redmond meeting, October 2004
Source: TR 18037 editor

Note: the clause numbering used in this document is the numbering used in the official approved version of TR 18037; in this official version, the fixed-point arithmetic is in clause 4 (was clause 2), the named address spaces and named-register storage classes are in clause 5 (was clause 3) and the basic I/O hardware addressing is in clause 6 (was clause 4).

The issues presented in this document were reported earlier by email; parts of the email discussion is added here as well. The order of the issues is roughly the order in which they appear in TR 18037.

**Issue 1: Name conflict for strtok**
Description: by selecting the letter 'k' as suffix for the accum type, the the function to convert a string to an accum type gets the name strtok; this name is already in use in the C standard.

Possible solutions:
- use a different letter
  - 'q' as proposed originally, conflicts with quadruple precision format
  - 'y' the only reasonable choice of the 'free' letters (others are 'b', 'm', 'v' and 'w');
- change the names of all "strto" functions to "strto_"; that is: "strtok" becomes "strto_k", "strtoht" becomes "strto_hr", etc. Easy to do but a little bit inconsistent with the C library;
- use the letter 'q' only for the functionnames, leave the 'k' everywhere else.

**Issue 2: Order in which overflow handling and rounding is done**
Description: the current text requires that overflow handling is done before rounding; this is counter intuitive, and not what is done for floating-point overflow/rounding. As it happens, the order in which overflow and rounding are done has no effect on the result when saturation is used for overflow handling.
Hence, if we change the order, the result remains the same for saturation. However, with the current required order, mod_wrap as overflow handling (which is allowed but not required) cannot (reasonably) be implemented. It is therefore proposed to change the order.

Proposed solution: change the required order of overflow handling and rounding.

**Issue 3: Typo in 4.1.6.2.1 - Binary arithmetic operations**
Description: 4.1.6.2.1 (Binary arithmetic operations), last para, the text on the divi fuctions has 'yielding a fixed-point type result'; this must be 'yielding an integer type result'.

Proposed solution: obvious

**Issue 4: Type generic macro's**
Description: The type-generic macro definition sections (2.1.7.6 and 7.18a.6.7) are incomplete and possibly wrong.

Incomplete because there are no rules on which function should be called when a type-generic macro is called with a non fixed-point type argument. Possibly wrong, because it is not clear what the name of the type-generic macro's should be: 2.1.7.6 suggest to call the type-generic abs function 'absfx', in 7.18a.6.7 the 'fx' part of the name is in italic which might suggest that the name should really be 'abs'. So, do we want to have 'abs', 'round' and 'countls', or 'absfx', 'roundfx' and 'countls'?

Proposed solution:

**Issue 5: Typo in new text for 6.2.6.3**
Description: the new text for clause 6.2.6.3, 3rd para, last sentence: replace 'integer types' by 'fixed-point types' twice.

Proposed solution: obvious

**Issue 6: fp arithmetic support functions do not specify what happens if an integer result overflows**
Description: 7.18a.6.1 (fp arithmetic support functions) does not specify what happens if an integer result overflows.

Proposed solution: Isn't there a blanket statement to the effect that when a specified result is not representable in the type, the behavior is undefined? If not, there should be.

**Issue 7: Error in 7.18.a.6.3**
Description: the rounding functions in 7.18a.6.3 require that

> Fractional bits beyond the rounding point are set to zero in the result.

This should not apply when saturation has occurred.

Proposed solution: replace the offending text by:

> When saturation has not occurred, fractional bits beyond the rounding point are set to zero in the result.

**Issue 8: Diagnostic required on named-register constraint violation?**
Description: consider

```
// file 1
register REG_A int reg_a;

// file 2
extern int reg_a;
```

```
int main() { return reg_a; }
```

According to the new constraints for 6.7.1 this is not allowed:

> If an object is declared with a named-register storage-class specifier, every declaration of that object shall include the same named-register storage-class specifier.

The 'shall' implies that a diagnostic is required here. However, so far C compilers have not been required to diagnose such issues across translation units. Is this really the intention?

Proposed solution:

**Issue 9: Effective type definition (aka US-40)**

Description: a couple of the statements concerning effective types in 6.5 of the C Standard are not exactly correct in the presence of address-space qualifiers. The easiest fix is probably to modify the concept of *effective type* in the C Standard so as not to include an address-space qualifier. (The whole concept of *effective type* is used only in 6.5 and in one footnote elsewhere in the C Standard.)

Prossible solution: add the following section to clause 5.3 of TR 18037:

> **Clause 6.5 - Expressions**, replace the first two sentences of paragraph 6 with:
>
> The *effective type* of an object for an access to its stored value is the declared type of the object, if any, without any address-space qualifier that the declared type may have.[72)] If a value is stored into an object having no declared type through an lvalue having a type that is not a character type, then the type of the lvalue, without any address-space qualifier, becomes the effective type of the object for that access and for subsequent accesses that do not modify the stored value.

and remove the notion *additionally access-qualified version* from TR 18037 (first replacement paragraph of paragraph 26 of 6.2.5, replacement text for paragraph 7 of 6.5).

**Issue 10: The relationship beween named-registers and external object definitions**

Description: the relationship beween named-registers and 6.9.2 (external object definitions) need to specified: when there is no initializer in the named-register declaration, the declaration is not an external definition for the identifier. The declaration is however also not a tentative definition, because it has a storage-class specifier. Then, what is it?

Proposed solution: That's one problem with using existing syntactic categories for new facilities. A workaround would be to change storage-class-specifier to storage-class-specifier|whatever in the grammar, where "whatever" is some new category, and then adjust the text that constrains storage-class-specifier to say the right thing (just storage-class-specifier or storage-class-specifier|whatever, depending). Then the construct would not have a storage-class-specifier and thus would be a tentative definition.

**Issue 11: Initialization of global named-registers**
Description: the current specification allows global named-registers to be initialized:

```
register REG_A int reg_a = 32;
```

It is however unclear when, and by whom this initialization should be done (one could imagine that the register storage onto which the variable maps does not really exist until some device is initialized by some user code).

Proposed solution: disallow initializers on named-register variables.

**Issue 12: Address space qualifier in specifier-qualifier-list**
Description: in the new text for 6.7.2.1, the TR adds a constraint:

> The *specifier-qualifier-list* in the declaration of a member of a structure or union shall not include an address space qualifier.

This is a mistake, because it keeps us from declaring something innocuous like

```
struct onePointer { _X int *pX; };
```

As written, the constraint would make the member declaration invalid, whereas we only intended to prohibit declarations such as this:

```
struct oneInteger { _X int iX; };
```

Proposed solution: change the constraint to be:

> Within a structure or union specifier, the type of a member shall not be qualified by an address space qualifier.

**Email crossreference list:**

| Item | Embedded-c emailnumber |
|---|---|
| 1: | 188, 189, 190, 191, 192, 194, 196, 198, 200, 201, 203, 204, 207 |
| 2: | 208 |
| 3: | 206 |
| 4: | 211 |
| 5: | 206 |
| 6: | 208, 210 |
| 7: | 208 |
| 8: | 208, 210 |
| 9: | 184, 195, 199, 207 |
| 10: | 208, 210 |
| 11: | 208, 209, 210 |
| 12: | 207 |