Chapter 1

First Steps with TIGCC

Translated by Spectras ; Many thanks to him/her! - Website: http://www.tiwiki.org/

At the lowest level, all modern computers work by processing and combining elementary informational units. Those units, called bits, represent the presence or the absence of electrical power on a wire. Therefore, it can only take one of two values, 0 or 1. For easier and faster handling, bits are grouped 8 by 8 to make bytes. All processors have one or two internal word lengths, usually powers of two (for instance, 32 for the MC68000, 64 for the new AMD processors, and so on). Thus, each and any of the pieces of information a computer handles is made of bytes. What makes them different is how those bytes are interpreted.

A special kind of information a computer handles is programs. Programs are sequences of bytes the processor can decode and execute to do things. At this fundamental level, the sequences of bytes, called opcodes, are very basic and can only do elementary computations, such as adding a number to another, testing two values for equality, reading values from memory, etc. Though it is possible for a programmer to directly write programs by putting opcodes together, it is very tedious and one never does that anymore. A lowlevel programming tool was introduced long ago : assembly language. Assembly language allows the programmer to do basically the same thing, but it uses human-readable mnemonics and syntax to write instructions. When the assembly program is complete, one runs it through the assembler, which creates the machine-readable program by convertings instructions to opcodes.

However, assembly language keeps a large part of the problems of direct opcode, namely its inability to do sophisticated computations and structured programming. There come programming languages. Programming languages are abstractions of the inner working of the processor, allowing the programmer to write high-level instructions. Therefore, the programmer can spend his time on the program itself rather than on the writing of the instructions. The program source is then processed into assembly language by a compiler, and finally the assembler generates the binaries of the program.

There are a lot of programming languages, which differ by syntax as well as the concepts they introduce. The most widely-used programming language is C. It features a relatively simple syntax, and keeps close to the assembly, allowing almost-complete control over what the compiled program will do, and a very low speed and size overhead. Anyway, C is the only compiled language available for TI-92/TI-89 platforms, TI-BASIC being merely an interpreted script, often buggy and always slow.

To sum it up, one can choose either assembly or C to make efficient programs for TIs, with C trading a bit of speed for a lot of flexibility. One can also mix both to achieve better performance in time-critical code while retaining the power of C in other places [#link].

I:\ Creating a new TIGCC IDE project

A project in TIGCC IDE represents the set of file and settings needed to compiler your program into a binary executable for your calculator. It typically has C and/or assembly source files (usually using .c and .s extensions), and a few text files to store documentation or notes (though some people prefer putting those directly in the source files).

Note : The following instructions will not work with TIGCC IDE versions older than 0.95

To create a new project, select *New* then *Project* from the *File* menu. The new empty project appears. You can then add a first source file. Since this is a C tutorial, we will add a C source file, again from the *File* menu. Give it any name you want, for instance **main** as it will be our main source file (well, our only source file for now). At this point, we have a full project, which you can save using the *Save all* item from the *File* menu.
Please note the the name you give to your project will be the name of the generated TI program.

The default C source file created by TIGCC IDE is quite encumbered, so we would better get a clean start : delete all its content and copy the following lines instead :

```
#include <tigcclib.h>

// Program entry point
void _main(void)
{
    // Place your code here.
}
```

Exemple Complet

For all other code samples in this tutorial, we will assume you have a fresh project with this starting code, using TIGCC IDE 0.95 or newer.

II:\ Compiling the project

Once you are done with your program, you can click the *Make* icon or press [Alt-F9] to let the TIGCC compiler generate the binary. The process should be very fast on any recent computer. In case you made some basic mistakes (which will happen almost everytime), the output panel will show you either errors or warnings (or both). Errors are fatal problems that prevent the compiler from generating the program (for instance if it does not understand something). On the other hand, warnings are a strong indication that you made something unusual that is probably not what you intended, but that the compiler understands anyway. You should always pay attention to warnings, because they almost always mean a bug. You should also know that errors tend to propagate, for instance when a typo confuses the compiler, it can misinterpret all that follows, thus generating many error messages. In such cases, only the first error is meaningful.

III:\ Running your program

Upon successful compilation, a TI-89/92/V200 archive is generated (the file with a .89z, .9xz or .v2z extension). This file can be sent to a calculator and run. However, programs always have bugs in them, and running the program on a calculator is not the easiest way to locate and remove them. One usually prefer using a specialised tool called a debugger, which exists for this purpose. There are two debuggers at the large now: VTI and TI-Emu. You can choose either, but VTI is getting old as it is no longer maintained.

You can send a file to a running VTI directly from TIGCC-IDE by clicking the green arrow icon, or pressing F9. Please also note that the transfer is somewhat buggy so you should avoid doing anything else until the operation completes.

Now we have seen the basics of creating, compiling and running a C program using TIGCC IDE, we will move to a discussion about kernel-mode and nostub-mode programming.