QL Assembly Language Book

This book was written over many years and has evolved from the original articles printed in QL Today magazine. The source code etc for the book, might prove useful to someone, so here it is.

The book has been successfully built on Windows and Linux - so take your pick. Mac users, I'm sorry. You will have to work it out for yourself, I haven't had the urge to touch any Apple kit since the Apple 2E all those years ago!

The Source Code

The top directory is where you should be when you are trying to build, or edit the source for the book. It has a number of files present - or otherwise, depending on whether a build has taken place or not - and these are:

- `buildPdf.sh` - runs the commands necessary, see below, to build and style the final book with table of contents, indexes etc.
- `AssemblyLanguage.tex` - The top level source file for the book. This file includes all the chapters, images etc. The book's template (how it looks and how it will be built) is defined in this file too.
- `structure.tex` - Part of the book's template structure. Defines numerous 'things' such as layouts, colours etc.
- `StyleInd.ist` - defines how the index section of the book will be styled. Don't touch!
- `HTMLColours.tex` - probably redundant, but defines all the standard HTML colours in a *LaTeX* manner. This is included so that the overall colours used in the book can be easily changed. See later under *Customisation* for details.
- `bibliography.bib` - A *LaTeX* bibliography file. Not used in this book. Part of the template, so best left alone. Unless you feel the need to add a bibliography of course.

In addition to the files above, there are also some sub-directories:

- `Content` - Contains all the separate chapters in `*.tex` files. The chapter naming should be easy to follow! You can edit any of the `*.tex` files, but leave any of the others that may be present untouched!

- `Content/images` - Contains the `*.dot` files that are used with `graphviz` to produce the various images that are used in various parts of the book, plus, there are some images that are imported directly into the chapters.
- `images` - Used on an older version of the book, no longer required, but sentimentality means that it is still around!
- `Pictures` - The image files used by the book template. In here you will find the cover image (`background.pdf`), the half page images used for the various chapter heading pages and a place holder for missing images. These are the original files that were supplied with the book template.

Customisation

Colours

The colour used throughout the book is defined in the file `structure.tex` as *ocre* and has the definition:

```
% Standard colour for the whole book.
\definecolor{ocre}{RGB}{243,102,25}
```

You can change the books entire colour layout by changing the definition. For example, if you wanted a dark green colour scheme, the following would work:

```
% Standard colour for the whole book.
\definecolor{ocre}{named}{wwwDarkGreen}
```

The www colours are to be found in the file `HTMLColours.tex` and *letter case is significant*. Rebuilding the book will change the colour scheme.

If you want to use a bright red colour, by specifying the RGB settings, then the code would be:

```
% Standard colour for the whole book.
\definecolor{ocre}{RGB}{255,0,0}
```

But that *might* be a tad worrying on the eyes!

Summary

- Use uppercase `{RGB}{R,G,B}` to define a colour by its RGB code.
- Use `{named}{colour_name}` to define colours that are defined elsewhere by name. The `named` part must be lower case while the colour name is case sensitive.

Indexing

Adding a word, phrase etc, to an index is simple. I have indexes set up to include programs, MC6800x Instructions and Addressing Modes, as well as separate global indexes.

Programs and Applications

If you mention a program or application name, you do it like this, mentioning the program name first, then the index detail afterwards:

```
... the program QMON\program{QMON} can be used to debug ....
```

This adds the current page to the index entry for QMON, under the main index entry of *Programs and applications* - this way, all mentioned program in the book can be looked up under the *programs and applications* entry.

Assembly Instructions

If a machine code instruction is mentioned in the main text, then it's name shall be entered into the main index's *MC6800x Instructions entry* as follows:

```
... the instruction \opcode{NOP}\mc6800x{NOP} has little or
no effect other than to waste around 4 clock cycles. ....
```

Addressing Modes

For an addressing mode's entry in the global index's *Addressing Modes* entry, we do the following:

```
\subsection{Register Direct}\address{Register Direct}
```

These are a little different as only the sections explaining the addressing modes is indexed. Mentions of different addressing modes in the main text are not indexed.

Vectored Utilities

Any mention of a vectored utility, `UT_MTEXT` for example, would be written as follows in the main body of the book:

```
... you can use the vectored utility \vector{UT_MTEXT} to ....
```

This format will add the page number to the global index entry for all the mentions of any vectored utility - Vectored Utilities.

Trap Calls

Any mention of a trap call utility, `IO_OPEN` for example, would be written as follows in the main body of the book:

```
... you can use the trap call \trap{IO_OPEN} to ....
```

This format will add the page number to the global index entry for all the mentions of any trap call - Trap Calls.

Pointer Environment Vectors

Any mention of a PE vectored utility, `WM_SETUP` for example, would be written as follows in the main body of the book:

```
... you can use the vectored utility \pe{WM_SETUP} to ....
```

This format will add the page number to the global index entry for all the mentions of any PE vectored utility - Pointer Environment Vectors.

Opcodes

Not strictly an indexing command, but as it was used above, I though it should be mentioned. All opcodes in the book are printed in fixed format text. To do this is simple, you just wrap the instruction in the `\opcode` macro, as follows:

```
... the instruction \opcode{NOP}\mc6800x{NOP} has little or
no effect other than to waste around 4 clock cycles. ....
```

Building

Building is *interesting*. You need to build the pdf a few times to be sure that everything is done correctly:

- Running `pdflatex AssemblyLanguage.tex` builds an initial version of the book, but no indexes or contents page numbers are to be found. The cover will not have any text on it.
- Running `pdflatex AssemblyLanguage.tex` again, sorts out the contents pages and will add text to the cover. The index is nowhere to be seen yet!
- Running `makeindex -s StyleInd.ist  AssemblyLanguage.idx` builds the AssemblyLanguage.ind file, which explains how the index is to be put together, and the entries in each section of the index.
- Finally, running `pdflatex AssemblyLanguage.tex` again, adds and styles the index.

The above process is exactly what the `buildPdf.sh` script does. :-)

Linux

To build the pdf in Linux is relatively simple. You need to install a utility named `TeXStudio` for editing, if you want a fully fledged *LaTeX* IDE, plus the supporting *LaTeX* requirements.

```
cd <top_level_directory>
./buildPdf.sh
```

When this finishes, and there will be a lot of text scrolling up the screen, the finished book will be found in the file `AssemblyLanguage.pdf`.

Windows

To build the pdf in Linux is relatively simple. You need to install a utility named `MikTeX` for editing, if you want a fully fledged *LaTeX* IDE, plus the supporting *LaTeX* requirements. Installing MikTeX will pretty much take care of this for you.

When you run a build, then any missing *LaTeX* packages will be installed either manually, or you will be prompted to install them individually as the build progresses. It's best to set up the application to automatically install missing packages.

The build process is .....

TBA.