Quick Start Guide

GNU 68K/Coldfire C/C++ Compiler Toolkit

The GNU Toolkit includes the GNU C compiler, assembler, linker and assorted object file utilities. The ASH WARE Inc. distribution's binaries are configured as a 680x0/CPU32 and Coldfire cross compiler that runs under Windows 95, 98, NT and 2000. It is based upon the Experimental GNU Compiler System (EGCS) V1.1.2 source distribution. The GNU compiler supports both ANSI and "Classic Kernighan & Ritchie" C. ASH WARE distributes this Toolkit. without warranty.

GNU is a project of the Free Software Foundation; see www.gnu.org. It is subject to the terms of the GNU Public License (GPL) found in file c:\GccAshWare\license.txt.

This quick start guide gives short overviews and usage of the major components of the GNU C compiler & toolkit, with special attention to their use with ASH WARE's MtDt and the demonstration code found in the directory listed below. To obtain this demonstration code you must install a demonstration version of the ASH WARE 683xx Hardware Debugger.

```
c:\ASH WARE\Demos683xx\68332HardwareDebugger
```

Installation

When installing from the ASH WARE CDROM run setup.exe found in the root directory and select the GNU compiler from the list of installable software. When installing from the ASH WARE website, download and run the self-extracting executable file.

In addition, the following path and environment variable modifications are required. Note that the PATH can conflict with other installed compilers such as Borland as certain executables have the same name.

```
set PATH=c:\GccAshWare\Bin\;%path%
set PATH=c:\GccAshWare\gcc-m68k-ashware\bin\;%path%
```

The GNU compiler consists of well over a thousand files. The following is a list of the most important ones.

`C:\GccAshWare\docs\index.html`	On-line help and documentation
`C:\GccAshWare\gcc-m68k-ashware\bin\make.exe`	The GNU make utility

`C:\GccAshWare\gcc-m68k-ashware\bin\m68k-coff-as.exe`
`C:\GccAshWare\gcc-m68k-ashware\bin\m68k-coff-gcc.exe`
`C:\GccAshWare\gcc-m68k-ashware\bin\m68k-coff-gcc.exe`
	The GNU Assembler, Compiler, and linker.

The Makefile

The use of the make utility and a makefile allows you to modify your source-code and then re-compile only those files that are affected by these modifications. Run the following batch file to rebuild the supplied demonstration program.

```
C:\ASH WARE\Demos683xx\68332HardwareDebugger\Mk.bat
```

The GNU make utility supports a number of command line options. The following is a list of the most important ones.

`-f makefile.gnu`	Instructs the make utility to use file **makefile.gnu.** Default file is **makefile**
`-h`	Print a help screen. This option is common for most GNU programs!

The GNU make utility does not automatically generate the required file dependency information. Instead, you specify this in a dependency file such as the following.

```
C:\ASH WARE\Demos683xx\68332HardwareDebugger\makefile.gnu
```

The make utility and associated makefile are incredibly powerful and flexible and a detailed explanation is beyond the scope of this quick-start guide. The makefile listed above provides detailed descriptions of a typical makefiles primary components.

Another feature of the makefile is the ability to multiple tasks. For example, the following command deletes various intermediate files, thereby forcing a complete rebuild on the next make.

```
Make –f makefile.gnu clean
```

GNU Assembler and C/C++ Compiler

When the GNU C compiler (also referred to as gcc) is invoked, it will normally do the preprocessing, compilation, assembly and linking. Generally, it is desirable to do the linking as a separate phase at the end (see the -c option and GNU linker documentation). The following is a list of the most common options.

`-c`	Specifies that the linker **NOT** be automatically invoked after each file is assembled or compiled. Surprisingly, this is not the default.
`-D`***`macro`***	Defines ***macro*** with "1" as its definition for use during preprocessing.
`-g`	Produces debug information in the default format, which in this distribution is Common Object File Format or COFF. Can be enabled regardless of optimization level specified.
`-m`***`cpu32`***	Specifies a cpu32 target. Valid options are -m68020, -m5200 (Coldfire).
`-O2`	Sets level 2 optimization. Valid levels are 0 (disabled) through 4 (maximum). Debugging is simplest when it is disabled. Higher levels cause variables to be optimized out, unusual single-stepping, etc.
`-Wall`	Enables the most useful set of warnings.
`-Wl,`***`opt`*** `-Wa,`***`opt`***	Passes the specified option ***opt*** onto the linker (**-Wl**) or assembler (**-Wa**). Used for options only available in those individual tools.

Although the assembler can be invoked directly, assembly files are more typically passed through gcc. The case of the assembly file's suffix (.s versus .S) is important. If gcc is invoked with an uppercase assembly file suffix (.S) it is passed first to the preprocessor whereas a lowercase file suffix (.s) passes goes directly to the assembler.

Inline Assembly, Interrupts, and Floating Point

Inline assembly is supported with the __asm__() function. References to C objects are supported. . Since there is no special interrupt support at the C level, __asm__() wrappers must be used in C-level interrupt service routines. See the following file.

`C:\ASH WARE\Demos683xx\68332HardwareDebugger\IsrLib.c`

The ASH WARE GNU C/C++ distribution supports floating point, which is not typical. But there is a significant performance penalty due to the CPU32's lack of native-mode floating point support.

GNU Linker

The following is a list of the most important link options.

`-M [mapfile]`	Output a symbol map to the file ***mapfile***.
`-o outputfile`	Place the linker output into file ***outputfile***. Default is a.out.
`-T commandfile`	Causes the linker to read commands from ***commandfile,*** thereby overriding all defaults

The key to linking, particularly for an embedded system, is the linker command file. See this following file.

`C:\ASH WARE\Demos683xx\68332HardwareDebugger\linkDbg3xxTpuRam.ld`

Two following are the most important linker commands.

`MEMORY`	Describes of the location and size of memory blocks.
`SECTIONS`	Controls placement of input sections within output sections and their order
`OUTPUT_FORMAT(`***`srec`***`)`	Specifies that an srec (SRECORD) output file be created. Other formats are available.