gcc-amigaos

COLLABORATORS			
	TITLE : gcc-amigaos		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 31, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

gcc-amigaos

1.1 gcc-amigaos.guide

This document describes the AmigaOS-only features of the GNU CC compiler.

Last updated Oct 25th, 1997.

1.2 gcc-amigaos.guide/Introduction

Introduction
****

This document is supposed to be an addendum to the baseline GCC documentation.

It focuses on the features that are visible by users and are important to them. It is not supposed to document the internals of the AmigaOS port of GCC.

It describes features implemented in the Geek Gadgets GCC port. As of this writing, this is version 2.7.2.1, Geek Gadgets snapshot 970728. For more information about Geek Gadgets, please refer to:

http://www.ninemoons.com/ADE/ADE.html
ftp://ftp.ninemoons.com/pub/geekgadgets/README

This document also describes some features that are not yet part of the Geek Gadgets GCC port, but which should be there soon. Such features are marked with [EXPERIMENTAL]. If you have GCC from a snapshot later than specified above, it's possible that these features are available in it. Some of these features might also be available in BETA GCC releases available on Kamil Iskra's WWW page:

http://student.uci.agh.edu.pl/~iskra/ade.html

This document focuses on GCC. It does not describe the AmigaOS-only features of other GNU packages, such as binutils, unless they are very closely connected to GCC.

This means, that, unless stated otherwise, when we talk about the "compiler", we mean the gcc, cpp and cc1 executables, i.e., the executables that convert C source code to assembly source code. The assembler and linker are generally beyond the scope of this document.

The primary source of information used to create this document was the GCC source code. Some parts of this document are based on:

* The LibNIX manual, written by Matthias Fleischer and Gunther Nikl:

  fleischr@izfm.uni-stuttgart.de
  gnikl@informatik.uni-rostock.de

* The A2IXLibrary manual, written by Hans Verkuil:

  hans@wyst.hobby.nl

* The README file, maintained by Rask Ingemann Lambertsen:

  gc948374@gbar.dtu.dk
  http://www.gbar.dtu.dk/~c948374/GNU/

* The Geek Gadgets FAQ, maintained by Lynn Winebarger:

  owinebar@indiana.edu
  http://nickel.ucs.indiana.edu/~owinebar/interests/amiga/amiga.html

* The FAQ for g++ and libg++, written by Joe Buck:

  jbuck@synopsys.com
  http://www.cygnus.com/misc/g++FAQ_toc.html

* Discussions on various Geek Gadgets mailing lists:

  gg@ninemoons.com
  gg-gcc@ninemoons.com
  gg-ixemul@ninemoons.com

This document was created by Kamil Iskra. Please email any questions, suggestions etc. to <iskra@student.uci.agh.edu.pl> or, even better, to the <gg-gcc@ninemoons.com> mailing list.

The author would like to thank Kriton Kyrimis <kyrimis@cti.gr> and Lars Hecking <lhecking@nmrc.ucc.ie> for correcting an awful lot of

language mistakes in this document.

1.3 gcc-amigaos.guide/Invocation

Invocation
****

The AmigaOS port of GCC supports the following non-standard command line options:

-noixemul	Link with LibNIX.
-fbaserel	Produce a4-relative data.
-resident	Produce a pure executable.
-fbaserel32	Produce a4-relative data with no size limits.
-resident32	Produce a pure executable with no size limits.
-msmall-code	Produce PC-relative code.
-mstackcheck	Produce stack-checking code.
-mstackextend	Produce stack-extending code.
-mfixedstack	Produce plain code.
-mrestore-a4	Reload a4 in public functions.
-malways-restore-a4	Reload a4 in all functions.
-mregparm	Pass function arguments in registers.
-frepo	Enable C++ Template Repository.

Accordingly, the AmigaOS port of GCC supports several flavors of linker libraries. See
Relation between library flavors and compile-time options.

1.4 gcc-amigaos.guide/-noixemul

-noixemul
=========

By default, the executables created with GCC require ixemul.library to run. This has its advantages (easy porting of UN*X programs, resource tracking, debugging, profiling, etc) and disadvantages (UN*X-style pathnames, large shared library, etc).

If -noixemul is specified on the GCC command line, the executable created will not require ixemul.library -- it will use the static linker library LibNIX instead. This library is very Amiga-like and SAS/C-like, so it is convenient for the AmigaOS-specific development.

Note: There is no great mystery about the -noixemul option. It has

absolutely no effect on the code generated by the compiler, only instructing the gcc driver to pass different options to the linker and preprocessor (see Options information, See Library flavors).

This option has no negative form.

For more information, please refer to the LibNIX documentation.

1.5 gcc-amigaos.guide/-fbaserel

```
-fbaserel
=========
```

By default, the code generated by GCC references data using 32-bit, absolute addressing.

The -fbaserel option will make GCC generate code that references data with 16 bit offsets relative to the a4 address register. This makes executables smaller and faster. Unfortunately, the size of the data section cannot exceed 64 KB, so this option cannot be used for large programs, like GCC itself.

Note: For a base-relative executable, -fbaserel needs to be specified for compiling and linking. Base-relative programs require special startup code and special versions of linker libraries. Since not all linker libraries are available in both plain and base relative versions, the usefulness of this option is limited. It is important to note that when the base-relative library is missing, the linker will attempt to use the plain one. This might result in strange link-time or even run-time errors.

This option is the AmigaOS equivalent of the standard GCC option -fpic, which is not supported by the AmigaOS port. -fpic generates code that references data indirectly, through a global offset table. The special addressing modes available on the m68k processor family allow for a much more efficient implementation with -fbaserel.

The negative form of -fbaserel is -fno-baserel, and is on by default.

For more information, please refer to the LibNIX documentation.

1.6 gcc-amigaos.guide/-resident

```
-resident
=========
```

Executables produced with the -resident option are pure, so they can be made resident using the AmigaShell resident command. resident

executables are loaded to memory just once, and several concurrent instances share the code section.

Note: The compiler generates the same code for -resident as for -fbaserel (see -fbaserel). Only the linking stage is different (special startup code is linked).

This option has no negative form.

For more information, please refer to the LibNIX documentation.

1.7 gcc-amigaos.guide/-fbaserel32

-fbaserel32
==========

The difference between the -fbaserel32 and -fbaserel options (see -fbaserel) is the same as between the standard GCC options -fPIC and -fpic.

Code generated with -fbaserel32 references data with 32 bit offsets relative to the a4 address register. In contrast to the -fbaserel (see -fbaserel) option, there is no 64 KB size limit. Unfortunately, the addressing modes with 32 bit offsets are only available on 68020 and higher processors. Therefore, it is necessary to specify -m68020 or higher to use this option.

Note: This option used to be called -flarge-baserel before Geek Gadgets snapshot 970109. Since it was not functional then, this should not cause any compatibility problems.

The negative form of -fbaserel32 is -fno-baserel32, and is on by default.

1.8 gcc-amigaos.guide/-resident32

-resident32
==========

This option is an improved version of -resident (see -resident) -- it does not impose any limits on data section size. Unfortunately, just like -fbaserel32 (see -fbaserel32), it is only available for 68020 or higher processors. Therefore, it is necessary to specify -m68020 or higher to use this option.

Note: This option was first made available in the GCC 2.7.2.1, Geek Gadgets snapshot 970109.

This option has no negative form.

1.9 gcc-amigaos.guide/-msmall-code

```
-msmall-code
============
```

By default, the code generated by the compiler references functions using 32-bit, absolute addressing.

Code generated by GCC with the -msmall-code option references symbols in the code section with 16 bit offsets, relative to the PC (program counter). This makes executables smaller and faster. Unfortunately, the size of the code section is generally limited to 32 KB, so this option can only be used for relatively small programs.

> Note: Actually, the compiler always generates 32-bit code references. If the assembler can calculate the offset between the referencing instruction and the referenced symbol (in other words, if the referenced symbol is in the same source file), it replaces the 32-bit reference with the PC-relative one. External references are left intact, unless -msmall-code is used, in which case the assembler generates PC-relative references, and the exact offsets are calculated by the linker.
>
> This option has no negative form.

For more information, please refer to the LibNIX documentation.

1.10 gcc-amigaos.guide/-mstackcheck

```
-mstackcheck
============
```

By default, the code generated by GCC does not check if there is enough stack available before performing stack-consuming operations. This is generally not necessary on UN*X systems, where the stack is extended automagically whenever needed.

Unfortunately, the AmigaOS provides tasks with a static, fixed size stack.

However, if a program is compiled with -mstackcheck, it will check if there is enough stack available before performing any stack-hungry operations. If there is a danger of stack overflow, the program will abort and the user will be notified.

Needless to say, stack checking increases the executable size and the execution time.

> Note: Stack checking cannot be used for functions that might be called from outside your task. This includes interrupt handlers, shared library functions, hooks etc. In such cases, you should either avoid using -mstackcheck for files containing such functions, or use __attribute__((interrupt)) (see interrupt).

It is safe to call a function that performs stack checking from one that does not, and vice versa.

The negative form of -mstackcheck is -mno-stackcheck, and is on by default.

Warning: -mno-stackcheck used to be called -mnostackcheck before Geek Gadgets snapshot 961012.

For more information, please refer to the LibNIX documentation.

1.11 gcc-amigaos.guide/-mstackextend

-mstackextend
=============

-mstackextend is very similar to -mstackcheck (see -mstackcheck).

The main difference is that when a program runs out of stack, it is not aborted, but a new stack area is allocated and the program continues to run.

Note: Stack extension can slow programs down significantly. It is advised that programs are written in such a way that they do not require too much stack. This can generally be achieved by explicitly allocating memory for large structures and arrays using functions like malloc() or AllocMem(), instead of creating them as local variables. Another method is replacing recursion with iteration. In addition, it might be considered to use stack extension only for selected, "dangerous" functions (see stackext), not for all functions in a given program.

The negative form of -mstackextend is -mno-stackextend, and is on by default.

Warning: -mno-stackextend used to be called -mnostackextend before Geek Gadgets snapshot 961012.

For more information, please refer to the LibNIX documentation.

1.12 gcc-amigaos.guide/-mfixedstack

-mfixedstack
============

This option makes GCC generate plain code, that does neither stack checking nor extension. Since this is the default, there is generally no need to use this option.

Note: This option has no negative form.

1.13 gcc-amigaos.guide/-mrestore-a4

-mrestore-a4
============

This option is used to create the IXEmul shared libraries (those *.ixlibrary files).

It sets a4 to the appropriate value in the prologues of all public functions (i.e., functions with external linkage). This is necessary if these functions are called from the code of application.

Note: This option should not be used except for the creation of an IXEmul shared library.

This option was first made available in the GCC 2.7.2, Geek Gadgets snapshot 960902. It used to be called -frestore-a4, and was relabeled to its current name in the GCC 2.7.2.1, Geek Gadgets snapshot 961012.

The negative form of -mrestore-a4 is -mno-restore-a4, and is on by default.

For more information, please refer to the A2IXLibrary documentation.

1.14 gcc-amigaos.guide/-malways-restore-a4

-malways-restore-a4
===================

This option is very similar to -mrestore-a4 (see -mrestore-a4).

The only difference is that it sets a4 in all functions, including private ones (i.e., functions with internal linkage, static). This is safer than -mrestore-a4 (see -mrestore-a4), but is also slower.

Note: This option should not be used except for the creation of an IXEmul shared library.

This option was first made available in the GCC 2.7.2, Geek Gadgets snapshot 960902. It used to be called -falways-restore-a4, and was relabeled to its current name in the GCC 2.7.2.1, Geek Gadgets snapshot 961012.

The negative form of -malways-restore-a4 is -mno-always-restore-a4, and is on by default.

For more information, please refer to the A2IXLibrary documentation.

1.15 gcc-amigaos.guide/-mregparm

```
-mregparm
=========

   On the m68k architecture, GCC passes function arguments on the stack
by default.

   -mregparm allows for passing arguments in registers. This can be
slightly faster than the standard method of passing arguments on the
stack.

   The full syntax of this option is:

-mregparm[=<value>]

   value should be an integer ranging from 1 to 4. If no value is
provided, 2 will be used.

   Four types of function arguments are recognized:

Integer
     Integer numbers (this includes enumerations, small structures and
     bool in C++, but excludes long long, which is too large). They are
     passed in data registers, starting from d0.

Pointer
     Pointers to objects or functions (this includes C++ references and
     the implicit this argument). They are passed in address registers,
     starting from a0.

Float
     Floating point numbers. If the floating point code generation is
     enabled, they are passed in floating point registers, starting from
     fp0. Otherwise, they are handled like the next type.

Other
     All the other types of arguments, like large structures, pointers
     to class methods in C++, etc. They are always passed on the stack.

   The value given for -mregparm indicates how many arguments of each
of the above first three types should be passed in registers.

   Example: GCC is invoked with -mregparm (without any value, so 2 will
be used) to compile a source containing the function:

     void fun(int a, char *str, char b, int c);

   a and b will be passed in d0 and d1, respectively, str will be
passed in a0, and c will be passed on the stack.

     Note: To use this option properly, it is very important that all
     sources are fully prototyped. There may be very serious problems
```

```
if they are not, since GCC will have to "guess" where to put
arguments, potentially making a wrong decission. Example:

     [in file1.c]
     void f(void)
     {
        g(0); /* Call to a function with no prototype.  The argument
                 will be put in d0, since it is an integer.  */
     }

     [in file2.c]
     void g(char *a) /* The argument is expected in a0, since it is
                        a pointer.  */
     {
     }

-Wimplicit -Wstrict-prototypes should be used to ensure that there
are no prototypes missing.

In case of stdargs functions, such as printf, all arguments are
passed on the stack.

As of this writing, -mregparm is supported by neither IXEmul nor
LibNIX, so its usefulness is very limited.

This option was first made available in the GCC 2.7.2.1, Geek
Gadgets snapshot 961012.

The negative form of -mregparm is -mno-regparm, and is on by
default.
```

1.16 gcc-amigaos.guide/-frepo

```
-frepo
======

   The AmigaOS port of GCC includes C++ Template Repository patch,
so-called repo patch.

   In order to activate it, please compile C++ source files with
-frepo. The compiler will not generate unnecessary template code, and
will create .rpo files that contain information about template symbols
used in each source file.  Afterwards, during linking stage, a special
tool called collect2 will make sure that every required instantiation of
each template is linked into the executable, recompiling some source
files if necessary.

     Note: This option was first made available in the GCC 2.7.2.1,
     Geek Gadgets snapshot 970109.

     This option is not specific to the AmigaOS port of GCC,
     nevertheless it is not fully supported in the baseline sources.

     This patch has been created in Cygnus Support, a company that is a
```

major contributor to the GNU project. It has not been integrated into the baseline sources due to design disagreements.

The negative form of -frepo is -fno-repo, and is on by default.

For more information, please refer to the G++ FAQ.

1.17 gcc-amigaos.guide/Attributes

```
Attributes
**********

   The following non-standard attributes are available in the AmigaOS
port of GCC:


Variable attributes:
  chip       Put object in chip memory.

Function attributes:
  saveds     Reload a4.
  interrupt   Do not mess with the stack.
  stackext   Generate stack extension.
  regparm    Pass arguments in registers.
  stkparm    Pass arguments on the stack.
```

1.18 gcc-amigaos.guide/chip

```
chip
====

   Amiga hardware requires some data to be located in chip memory.

   Typically, if an initialized buffer is required (containing a picture
bitmap, for example), a plain, statically initialized buffer is used,
and the data is copied into a dynamically allocated MEMF_CHIP buffer.

   This is not necessary with the chip attribute. If this attribute is
specified for an initialized, static variable, it will be allocated in
chip memory automagically by the AmigaOS.

   A small example:

     UWORD __attribute__((chip)) bitmap1[] = { ... };

     Note: For compatibility with other AmigaOS C compilers, a
     preprocessor symbol __chip is available, which expands to
     __attribute__((chip)) (see Keyword macros).

     All the chip attribute does is specifying that data should go to a
```

section called .datachip. Therefore, the standard GCC feature __attribute__((section(".datachip"))) can be used instead.

This attribute was first made available in the GCC 2.7.2.1, Geek Gadgets snapshot 970328.

For proper operation, this attribute requires a special version of the assembler, which generates standard AmigaOS object files. This version is not yet available in Geek Gadgets in binary form, since support for this object files format is not yet complete.

1.19 gcc-amigaos.guide/saveds

saveds
======

This attribute is ignored, unless base-relative data (see -fbaserel) is compiled.

To improve speed, programs compiled with the AmigaOS port of GCC set the a4 register to the appropriate value only once, in the startup code. Code generated with the standard GCC option -fpic, in contrast, sets the a4 register in every function which references global data.

This is only safe as long as all function calls are performed from within your own code. Things become "tricky" if callback functions, like the AmigaOS hooks, interrupt handlers etc. are used. If global data is referenced in such functions, a4 has to be set properly.

This is exactly what the saveds attribute does: it initializes a4 in the function prologue, and restores it to its original value in the function epilogue.

Note: For compatibility with other AmigaOS C compilers, a preprocessor symbol __saveds is available, which expands to __attribute__((saveds)) (see Keyword macros).

Please do not use this attribute in pure executables (see -resident, see -resident32). This is because several invocations of pure executables can run concurrently, each one having its own data section, and there is no way to find out to which of these sections should a4 be set.

The saveds attribute is not necessary in function declarations (prototypes).

This attribute was first made available in the GCC 2.7.2.1, Geek Gadgets snapshot 961012.

1.20 gcc-amigaos.guide/interrupt

```
interrupt
=========

   This attribute should be used for any kind of callback functions that
can be called from outside your task. This includes interrupt handlers,
shared library functions, etc.

   Most often, the interrupt attribute is only necessary if a program
is compiled with stack checking or extension (see -mstackcheck, see
-mstackextend). It will prevent the compiler from generating stack
checking or extension code for the function it was specified for.

   Additionally, it will set CC (condition codes register) in the
function epilogue to return value, by performing tstl d0.

     Note: For compatibility with other AmigaOS C compilers, a
     preprocessor symbol __interrupt is available, which expands to
     __attribute__((interrupt)) (see Keyword macros).

     The interrupt attribute is mutually exclusive with the stackext
     attribute (see stackext).

     This attribute is not necessary in function declarations
     (prototypes).

   This attribute was first made available in the GCC 2.7.2.1, Geek
Gadgets snapshot 961012.
```

1.21 gcc-amigaos.guide/stackext

```
stackext
========

   This attribute makes GCC generate stack extension code for the
function for which it was used (see -mstackextend). This makes it
possible to use stack extension selectively, only for the "dangerous"
functions -- recursive functions, functions with large local variables,
etc.

     Note: For compatibility with other AmigaOS C compilers, a
     preprocessor symbol __stackext is available, which expands to
     __attribute__((stackext)) (see Keyword macros).

     The stackext attribute is mutually exclusive with the interrupt
     attribute (see interrupt).

     This attribute is not necessary in function declarations
     (prototypes).

   This attribute was first made available in the GCC 2.7.2.1, Geek
Gadgets snapshot 961012.
```

1.22 gcc-amigaos.guide/regparm

```
regparm
=======

   The regparm attribute, together with the stkparm attribute (see
stkparm), can be used to fine-tune the way arguments are passed. It
makes GCC pass arguments in registers for the function for which it was
used, regardless of whether the global -mregparm option was used or not
(see -mregparm).

   An optional integer argument ranging from 1 to 4 indicates how many
arguments of each type should be passed in registers (see -mregparm).
The syntax is the following:

     void __attribute__((regparm(3))) fun(int a, char *str, char b, int c);

   This will make GCC pass a, b and c in d0, d1 and d2, respectively,
and str in a0.

   If the argument is not provided, the value given for -mregparm will
be used (or 2 if that option was not specified, see -mregparm).

     Note: There is generally no need to use this attribute unless
     files compiled with different calling conventions are linked
     together.

     For compatibility with other AmigaOS C compilers, a preprocessor
     symbol __regargs is available, which expands to
     __attribute__((regparm)) (see Keyword macros).

     The regparm attribute is mutually exclusive with the stkparm
     attribute (see stkparm).

     This attribute is necessary both in function declarations
     (prototypes) and definitions (function code).

   This attribute was first made available in the GCC 2.7.2.1, Geek
Gadgets snapshot 961012.
```

1.23 gcc-amigaos.guide/stkparm

```
stkparm
=======

   The stkparm attribute, together with the regparm attribute (see
regparm), can be used to fine-tune the way arguments are passed. It
makes GCC pass arguments on stack for the function for which it was
used, regardless of whether the global -mregparm option was used or not
(see -mregparm).

     Note: There is generally no need to use this attribute unless
     files compiled with different calling conventions are linked
```

```
     together.

     For compatibility with other AmigaOS C compilers, a preprocessor
     symbol __stdargs is available, which expands to
     __attribute__((stkparm)) (see Keyword macros).

     The stkparm attribute is mutually exclusive with the regparm
     attribute (see regparm).

     This attribute is necessary both in function declarations
     (prototypes) and definitions (function code).

   This attribute was first made available in the GCC 2.7.2.1, Geek
Gadgets snapshot 961012.
```

1.24 gcc-amigaos.guide/Defines

```
Defines
*******

   The AmigaOS-specific preprocessor symbols available in GCC can be
divided into three groups:


  Identifying machine   What machine is this?
  Options information   Which options have been specified?
  Keyword macros       Compatibility with other compilers.
```

1.25 gcc-amigaos.guide/Identifying machine

```
Symbols identifying machine
===========================

   The following machine-identifying preprocessor symbols are available:

mc68000
     This macro identifies the machine as having a CPU from the Motorola
     68000 family.

amiga

amigaos

amigados
     These macros identify the machine as being an Amiga, running the
     AmigaOS.

AMIGA

MCH_AMIGA
```

These macros are provided for compatibility with other AmigaOS C compilers.

Note: These symbols are available in three groups: plain (as specified above), with two leading underscores, and with two leading and two tailing underscores. The plain ones are not available when compiling with the -ansi option.

The amigados symbol is obsolete and will be removed in future. Please use amigaos, which was first made available in the GCC 2.7.2.1, Geek Gadgets snapshot 961012.

1.26 gcc-amigaos.guide/Options information

Symbols identifying specified options
=====================================

GCC has several options to choose the CPU model that the code should be generated for. The following preprocessor symbols identify which options have been specified on the command line:

mc68020
Either one of -m68020, -mc68020 or -mc68020-40 has been specified.

mc68030
-m68030 has been specified.

mc68040
-m68040 has been specified.

mc68060
-m68060 [EXPERIMENTAL] has been specified.

__HAVE_68881__
-m68881 has been specified.

Note: The symbols beginning with mc are available in three groups: plain (as specified above), with two leading underscores, and with two leading and two tailing underscores. The plain ones are not available when compiling with the -ansi option. The "underscored" ones were first made available in the GCC 2.7.2.1, Geek Gadgets snapshot 970109.

mc68000 is defined regardless of which -m680x0 options have been used.

In addition to the above, a preprocessor symbol ixemul (together with the "underscored" versions) is available when not compiling with -noixemul (see -noixemul) and identifies the runtime environment as IXEmul. This symbol was first made available in the GCC 2.7.2.1, Geek Gadgets snapshot 970328.

1.27 gcc-amigaos.guide/Keyword macros

```
"Keyword" macros
================

   Most AmigaOS-specific C compilers have special "custom keywords",
which make the AmigaOS-specific development easier.  Unfortunately, the
idea of "custom keywords" is not available in GCC. However, attributes
are available, and they provide virtually identical functionality. For
compatibility with other AmigaOS C compilers, preprocessor symbols are
provided, which expand to the appropriate attributes (see Attributes).

__chip
     See chip.

__saveds
     See saveds.

__interrupt
     See interrupt.

__stackext
     See stackext.

__regargs
     See regparm.

__stdargs
     See stkparm.

__aligned
     This expands to the standard GCC __attribute__((aligned(4))).

     Note: With SAS/C, these keywords may be specified either before or
     after the type, so the following declaration is correct:

          __saveds void func(void);

     Unfortunately, the syntax rules of GCC 2.7.2.1 do not allow to
     specify the attributes before the type, so the above example must
     be changed to:

          void __saveds func(void);

     This will be fixed in GCC 2.8.0.
```

1.28 gcc-amigaos.guide/Miscellaneous

```
Miscellaneous
*************

   The following "hard to categorize" features are available in the
AmigaOS port of GCC:
```

1.29 gcc-amigaos.guide/Explicit register specification

```
Explicit register specification
===============================
```

In certain situations, like writing callback hooks, "patchers", standard shared libraries, etc., functions have to receive arguments in particular registers.

-mregparm (see -mregparm) is not appropriate in this case, since it does not give the programmer enough control on which registers will be used.

To overcome this problem in the AmigaOS port of GCC, explicit register specification has been extended to be available for function arguments, as well:

```
void myhook(struct Hook* hook __asm("a0"), APTR object __asm("a2"),
            APTR message __asm("a1"))
{
 ...
}
```

Note: This feature is currently not available in G++.

Only the ANSI-style declarations (prototypes) are supported.

Registers have to be specified both in function declarations (prototypes) and definitions (function code).

This feature was first made available in the GCC 2.7.2.1, Geek Gadgets snapshot 961012.

1.30 gcc-amigaos.guide/Case sensitive CPP

```
Case sensitive CPP
==================
```

The preprocessor available in the AmigaOS port of GCC is case sensitive. This means, that the header names provided in the #include directives have to be correct, including upper and lower case letters. This affects only the way the preprocessor works. Currently available native AmigaOS file systems are case insensitive.

Note: This might seem like a horrible hack and a crazy attempt to implement a "ridiculous" UNIX feature on Amiga. However, this feature has been introduced to terminate the endless G++ problems with a standard ANSI C header string.h: under the AmigaOS, a C++ header String.h would be included, instead.

1.31 gcc-amigaos.guide/GCCPRIORITY

GCCPRIORITY
==========

GCC supports one AmigaOS-specific environment variable: GCCPRIORITY.

This variable specifies the exec priority of the compiler. If this variable is not set, the default Shell priority will be used.

Note: By default, the AmigaOS assigns the priority 0 to user tasks. It is thus generally unwise to set GCCPRIORITY higher than 0.

1.32 gcc-amigaos.guide/Library flavors

Library flavors
==========

The AmigaOS port of GCC may use different linker libraries depending upon the options used while invoking the compiler. These libraries reside in subdirectories of the standard locations, such as GG:lib/ or, with GCC 2.7.2.1, GG:lib/gcc-lib/m68k-amigaos/2.7.2.1/.

If you invoke gcc with -v, you'll see the precise flavor of libraries used as a -fl option in the ld invocation. Here is a list of the available flavors (and hence the subdirectories names):

* libb corresponds to the -fbaserel option.

* libb32 corresponds to the -fbaserel32 option.

* libm020 corresponds to the -m68020 (or higher) options.

* libm881 corresponds to the -m68881 option.

* libnix corresponds to the -noixemul option.

More than one flavor can be specified simultaneously. For example, when both -fbaserel and -m68020 are specified, the libraries will be searched in libb/libm020 subdirectory (as well as in libb subdirectory and in the standard location).

1.33 gcc-amigaos.guide/Index

Index
*****