Build Amiga GCC 3.3.3 Cross Compiler for Linux or Windows

by Troy Silvey

This tutorial will help build an Amiga GCC 3.3.3 cross compiler on Linux or Windows XP. I've sucessfully tested this doc on RedHat Linux and Windows XP. Other tutorials used GCC 2.95 and usually ended in failed builds for me, so I decided to do a new tutorial. Please send comments if you attempt a build or find an error. I also added a small bit on building a library and allocating chip memory with GCC.

Please read this doc through a couple of times before attempting to use it. I may save some time and irritation later. Consider this guide a beta document. It's only been used successfully a few of times on specific hardware. You should backup any important data on your system before continuing. I recommend starting with a new clean system that you can afford to erase and start over if things don't go well. But that would never happen, right? :)

This guide will help you build a development box based on Linux or WindowsXP. It uses the Cygwin Linux environment on WindowsXP to make and use a GCC cross compiler that cranks out Amiga 68k code for OS3.x or OS4.x. (Maybe even 2.x). It should be possible to follow this document and do a PPC compiler also using a TARGET=powerpc-amigaos, but I don't have PPC yet to test. Add UAE (http://www.winuae.net) or (www.amigaforever.com) and you have a computer that can create, test and use Windows, Linux and Amiga apps. This guide will help you build the Amiga cross compiler on a Linux based system also if you prefer. Just ignore the Cygwin install section if you're working on Linux.

Requirments

I've written this to help the newbie, but it expects you to know or learn many basics for the different OS's, programming and compilers we're using. All of that would be impossible to cover in these few pages. If you are a newbie, make sure to follow the instructions closely. You need to execute commands from the proper folders and have correct files in those folders to succeed. Miss a file or a directory change (cd) command and the build will probably fail. As the build progresses, many warning messages appear. These errors do not affect the build and are probably results of the "patched" amiga code that appears in some of the files. If you encounter a true error message however, that indicates the build has probably failed. There are a couple of error exceptions that I'll mention as we go that don't seem to harm the build.

So… you're already an expert on Amiga, you grew up using Windows and you already know the basics of Linux and it's text based editors. If that's not the case, this project may be a bit over your head. But you can still learn a lot by trying it out. And there's always plenty of help on the Amiga forums, web sites and other books. If you build on Windows, you can use cut and paste editors like notepad making changes to the Cygwin Linux files easier if you've never used Linux text editors like Vi or Ed.

You'll need a fast internet connection to install all the components. A slow connection will work, but may take hours to download Cygwin. You will need a fast PC with WindowsXP installed, 2gig free harddrive space and 256meg memory. (Win9x or Millieum might work but aren't tested). I started this project with a PII-350/128meg memory. It worked but was very very slow. A slower PC will be ok for GCC, but if you plan to install

UAE, I recommend the fastest PC you can get. I've moved to a P4 1.6gig and that runs very nicely. This also takes a huge amount of HD space. Once the build was complete it used 1.7 gig. Your install make need more space. Some of that can probably be cleaned up later, but I haven't bothered yet. Much of that spaced is used up by the Linux environment. If you build on a Linux system instead, you won't need as much free harddrive space.

Last you will need several Amiga specific files that we can get from Aminet, geekgadgets and other links. It's important as you follow the steps to make sure that the folders and files go into the proper locations. When it's time to do the "makes" and compiles, the system will count on files being in certain locations. If your build fails, you might try backing up and making sure that all files and folders have been created moved and copied into the necessary locations.

Important – If for any reason you want to start over from scratch, you can delete the work folders while saving all of your downloaded files. Delete this list of folders, then start the instructions over again bypassing the download instructions. (I've done this 30-40 times while writing this doc

```
# rm -R ~/amigaos/build-binutils
# rm -R ~/amigaos/binutils-2.14
# rm -R ~/amigaos/build-gcc
# rm -R ~/amigaos/gcc-3.3.3
# rm -R ~/usr/local/amigatools
```

Getting Started

1a (Start here if you're building on a WindowsXP system)

Ok you have Windows installed, you're connected to the internet and ready to go. Cygwin installation is first on the list. You may want to read a bit about Cygwin and see what you're installing. Check the home page at http://www.cygwin.com/ . Once you're ready, this site …(Installing Cygwin) has a great guide for installing Cygwin that chooses all of the same features that we need, so we'll follow their lead. Follow the instructions on their page and especially pay attention to the instructions for installing the "Devel", "Doc" and "Editor" options. You may want to print out the instructions so you can have them handy for quick reference as you work. It's also important that we include BZIP2 from the Utils options and WGET from the Web options. Make sure to include them. We use these two utilities to download and install some of our software. Once the Cygwin installation is finished, you will have a new icon on your Windows desktop ready to run the Linux environment on your PC. If you did not install Cygwin on your C: drive, you will need to edit the file INSTALLDRIVE:\cygwin\cygwin.bat and change the line that reads 'C:' to your actual drive letter, or the batch file short cut won't launch Cygwin. Click on your new Cygwin icon and we're at a text prompt ready for step 2.

1b (Start here if you're building on a Linux system)

This guide expects you to be working as a user other than root. Root normally does not have a home folder that we will be using to do much of our work. If you wish to work as root, then create a base folder to work from and adjust the commands in this tutorial as needed. While working as a normal user, there will be times we have to execute commands as Root or SU to root because a normal user does not have permissions to certain files and folders. The instructions will guide you when it's time to SU to root or run a command as root. Each time we use root you will be prompted for the root password so be prepared to enter it.

If you have not already installed the development tools and libraries on your Linux system, you will need to do so. We will need the dev tools to build our Amiga cross compiler. For my system I used GCC 3.3.3. the GCC tools that come with your Linux are probably a different release. Try to stick with 3.x if you can. I have no way to tell if using a different version will cause a problem for you. I have seen posts where others have successfully built compilers using GCC 2.7, 2.95, 3.1, 3.2, 3.4 and EGCS 1.1.2, so you should be ok. You'll need these basics installed…. binutils - gcc - gdb - make – mktemp - bison and the utilities bzip2 and wget. (If you're building on a Macintosh, WGET can be found at http://www.statusq.org/images/wget.zip).

2. (Steps from here forward are identical for either Windows or Linux)

Next we'll create work folders in our home directory and download the needed source files. ~/ is equal to your home folder. In the following lines '#' is your command line prompt so don't type it. Your character may differ.

```
# mkdir -p ~/amigaos/build-binutils/m68k-amigaos
# mkdir -p ~/amigaos/build-gcc/gcc/include
# mkdir ~/amigaos/gg-archives
# cd ~/amigaos
```

You should still be in the ~/amigaos folder, now download these files with Wget

```
# wget http://adsl-065-006-130-241.sip.asm.bellsouth.net/files/binutils-2.14-src.tar.bz2
# wget http://adsl-065-006-130-241.sip.asm.bellsouth.net/files/gcc-3.3.3-m68k-src.tar.bz2
```

Now unarchive them.

```
# bzip2 -dc binutils-2.14-src.tar.bz2 | tar -xf -
# bzip2 -dc gcc-3.3.3-m68k-src.tar.bz2 | tar -xf -
```

Download the following files from geekgadgets to the gg-archives folder:

```
# cd ~/amigaos/gg-archives
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/fd2inline-1.11-bin.tgz
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/ixemul-48.0-bin.tgz
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/ixemul-48.0-env-bin.tgz
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/ixemul-48.0-inc-bin.tgz
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/libamiga-bin.tgz
# wget http://gg.portail-e.com/geekgadgets/amiga/m68k/snapshots/current/bin/libnix-1.2-bin.tgz
```

Now they will be extracted them to the dir ~/amigaos/build-binutils/m68k-amigaos

```
# cd ~/amigaos/build-binutils/m68k-amigaos
# tar -xzf ../../gg-archives/fd2inline-1.11-bin.tgz
# tar -xzf ../../gg-archives/libamiga-bin.tgz
# tar -xzf ../../gg-archives/libnix-1.2-bin.tgz
# tar -xzf ../../gg-archives/ixemul-48.0-env-bin.tgz
# tar -xzf ../../gg-archives/ixemul-48.0-inc-bin.tgz
# tar -xzf ../../gg-archives/ixemul-48.0-bin.tgz
```

Make a folder where the finished executables will be created. You need root permission to create folders in the /usr/local folder. SU to root (su – root) if necessary for these steps.

```
# su – root (Skip this step if building on Windows/Cygwin)
# mkdir -p /usr/local/amigatools/lib
# mkdir -p /usr/local/amigatools/m68k-amigaos/sys-include
# mkdir -p /usr/local/amigatools/m68k-amigaos/include/dos
# chmod -R 755 /usr/local/amigatools
```

If you had to SU to root for the last 4 steps, exit back to your normal account.

```
# exit (Skip this step if building on Windows/Cygwin)
```

Now we'll set our PREFIX and TARGET variables used to build the utilities and compiler.

```
# export PREFIX=/usr/local/amigatools
# export TARGET=m68k-amigaos
# cd ~/amigaos/build-binutils
```

Now we can begin to create the 680XX Binary Utilities by running the configuration script. This command is shown on 2 lines, but is actually one single command. The '\' (the continuation mark) says to continue my command typing on the next line. It makes code easier to read some times. You can cut and paste these lines into your command line directly from here, one line at a time. Leave out the # and be careful to stop at the '\' and hit return after each line paste. If you paste the blank space after the '\', it will enter an unwanted carriage return at the end and cause the script to execute wrong. If you make a mistake, just run it again.

```
# ../binutils-2.14/configure --target=$TARGET --prefix=$PREFIX --disable-nls \
--with-gnu-as --with-gnu-ld —with-headers=$PREFIX/m68k-amigaos/include
```

Configuration is done, now it's time to build the source code for binutils. If you are building on a Linux system, make sure you have rights to install to your chosen directory (PREFIX). If you are installing on Linux, you may need to su to root to run the next commands if your current account doesn't have sufficient rights. This will take 10-30 mins or more if you have a slower system.

```
# make (took about 3 minutes on my 1.8ghz AMD)
# make install (su –c 'make install' If working in Linux)
```

We're ready to build the GCC cross compiler. We add the new binutils to the search path, and change to the build folder.

```
# export PATH=$PREFIX/bin:$PATH
# cd ~/amigaos/build-gcc
```

Run the configuration script for the GCC compiler. Once again you can cut and paste. Just follow the same guide lines as above with binutils.

```
# ../gcc-3.3.3/configure --target=$TARGET --prefix=$PREFIX --enable-languages=c,c++ \
--with-headers=$PREFIX/m68k-amigaos/include --with-gnu-as —with-gnu-ld
```

Now we need to edit the file ~/amigaos/gcc-3.3.3/gcc/config/m68k/amigaos.h This command assumes you know how to use the editor vi. If you're using windows, you can use notepad/wordpad to edit the file. In windows the path of the file will be… c:\cygwin\home\YOURHOMEFOLDER\amigaos\gcc-3.3.3\gcc\config\m68k\amigaos.h

```
# vi ~/amigaos/gcc-3.3.3/gcc/config/m68k/amigaos.h
```

In amigaos.h locate STARTFILE_SPEC section. This is where we find the line we need to edit. (Use the search function in vi …. Esc /STARTFILE_SPEC enter) (If you make a mistake, you can exit with out saving with Esc q! enter)

```
#define STARTFILE_SPEC
"%{!noixemul:"
"%{fbaserel:%{!resident:bcrt0.o%s}}"
"%{resident:rcrt0.o%s}"
"%{fbaserel32:%{!resident32:lcrt0.o%s}}"
"%{resident32:scrt0.o%s}"
"%{!resident:%{!fbaserel:%{!resident32:%{!fbaserel32:"
"%{pg:gcrt0.o%s}%{!pg:%{p:mcrt0.o%s}%{!p:/crt0.o%s}}}}}}}"
"%{noixemul:"
"%{resident:libnix/nrcrt0.o%s} "
"%{!resident:%{fbaserel:libnix/nbcrt0.o%s}%{!fbaserel:libnix/ncrt0.o%s}}}"
```

We need to add our PREFIX/lib path to the /crt0.o entry. Locate and change this line…

```
"%{pg:gcrt0.o%s}%{!pg:%{p:mcrt0.o%s}%{!p:/crt0.o%s}}}}}}}"
```

When you are finished, the edited line should read …

```
"%{pg:gcrt0.o%s}%{!pg:%{p:mcrt0.o%s}%{!p:/usr/local/amigatools/lib/crt0.o%s}}}}}}}"
```

When you have the PREFIX path inserted, save the file and exit vi. (vi save and exit - Esc wq!) Now we need to move the crt0.o file into place.

```
# cp ~/amigaos/build-binutils/m68k-amigaos/lib/crt0.o /usr/local/amigatools/lib
```

Time to build the Amiga GCC executable with ‘make’ and install it with ‘make install’. This part is tricky. Make is going to FAIL! the first time through. It is going to be looking for files that are not yet present. Once make fails, we will copy the needed files into place and rerun make. If we move the files in to place before the first pass, It’s possible they will get over written and cause make to exit with an error anyway. So we have to wait for the process to fail then put them in place.

If you see this message about several minutes later (5mins on my 1.8Ghz), then make has failed as expected.

```
/home/tsilvey/amigaos/build-gcc/gcc/include/stddef.h:57:26: machine/ansi.h: No such file or
directory
make[2]: *** [libgcc/./_muldi3.o] Error 1
make[2]: Leaving directory `/home/user1/amigaos/build-gcc/gcc'
make[1]: *** [libgcc.a] Error 2
make[1]: Leaving directory `/home/user1/amigaos/build-gcc/gcc'
make: *** [all-gcc] Error 2
```

Now it’s time to move the needed files in to the proper path and run make again. Make will pick up where it left off so it won’t take as long the second time. If a file already exist when we copy over the new files, you will be asked if you want to over write the current file. Always answer NO!! to the over-write prompt. There sould be 8 files we need to skip over.

```
# cp -ir ~/amigaos/build-binutils/m68k-amigaos/include ~/amigaos/build-gcc/gcc/
# cp ~/amigaos/build-binutils/m68k-amigaos/lib/libamiga.a ~/amigaos/build-gcc/m68k-amigaos/lib
# make
```

The second time through make, it ends with the error message below. It doesn't seem to harm any thing. Our new utilities and cross compiler have been created. (Seems odd to me, but I'm not a guru)

```
configure: error: installation or configuration problem: C
compiler cannot create executables.
make: *** [configure-target-libiberty] Error 1
```

Now it's time to finish the build.

```
# make install (su –c 'make install' If working in Linux)
```

You can look to see if the new compiler is in place now.

```
# ls -l /usr/local/amigatools/m68k-amigaos/bin
```

Run gcc to show it's working and that we have the correct version

```
# /usr/local/amigatools/m68k-amigaos/bin/gcc –v
```

You should find …. Configured with: ../gcc-3.3.3/configure --target=m68k-amigaos …among the information.

Now we need to move these amiga libraries into place so the new cross compiler can find them. If we are building on Linux as a non-root user, we need to fix permissions so average users can get to the new folders.

```
# su - root (Skip if building on Windows/Cygwin)
# chmod -R 755 /usr/local/amigatools (Skip if building on Windows/Cygwin)
# exit (Skip if building on Windows/Cygwin)

# cp ~/amigaos/build-binutils/m68k-amigaos/lib/libc.a  /usr/local/amigatools/lib/gcc-lib/m68k-amigaos/3.3.3
# cp ~/amigaos/build-binutils/m68k-amigaos/lib/libamiga.a  /usr/local/amigatools/lib/gcc-lib/m68k-amigaos/3.3.3
```

If we made it this far with out major errors, then we're ready to work. The new compiler is located at /usr/local/amigatools/bin/m68k-amigaos-gcc.exe I make a shorter symbolic link name for myself.

```
# cd /usr/local/amigatools/m68k-amigaos/bin
# ln -s ./gcc ./gcc68
```

This separates the new GCC 68K cross compiler from the local Linux GCC you just used to build GCC68. (And helps end confusion for me) Now add it to your path

```
# export PATH=$PATH:/usr/local/amigatools/m68k-amigaos/bin
```

You may also want to add this to your users login path by adding it to file .bashrc in your user home folder. That way you don't have to manually enter the path each time you log in.

Once the compiler is built, the paths to it's libs and headers are hard-coded. So don't move things out of the /usr/local/amigatools folder unless you're really sure that you know what doing. To see the default setting and folder paths use the command

```
# gcc68 –v
```

We're done building the executables, but we still have a few things to do to complete our development environment. First let's test our new compiler on the famous helloworld.c I've used vi to create it in my home folder ~/helloworld.c If you're working on a windows platform, you can use anything like notepad to do the same. For our first test we need to put a copy of stdio.h into place.

```
# cp ~/amigaos/build-binutils/m68k-amigaos/include/stdio.h /usr/local/amigatools/m68k-amigaos/include/dos
```

Now let's go back to our home folder, create a test file and compile it.

```
# cd ~/
# vi helloworld.c
```

Once you're in vi editor, type in this sample code and save with the keystrokes Esc wq!

```
#include <dos/stdio.h>
int main(void)
{
  printf("Hello World n");
  return 0;
}
```

Let's see if it works. This will compile the executable and try to optimize it. Trying to optimize this tiny program is not practical, but simply tests the compiler functionality.

```
# gcc68 -O -o helloworld -m68020 ./helloworld.c
```

If the compile fails for any reason, you can use this command to compile with verbose and error output. You can examine the output for clues to what is missing or broken. I used this to find a path that was incorrect.

```
# gcc68 -v ./helloworld.c –m68020 -Wl,--verbose 2> ldlog.txt
# vi ./ldlog.txt
```

If there were no errors, then we are almost ready to run the helloworld file on an Amiga for the final test. One last library file needs to be loaded on your Amiga. You need to download the ixemul library version 48.0 from aminet and put it in your Amiga's Libs folder. You need this file whether you're on a real Amiga or using UAE. The ixemul library makes the Amiga more POSIX compliant by adding additional library functions. This is needed to run the code generated by the POSIX compliant GCC compiler we've just built. The archive from aminet comes with different versions for different CPU's. For our setup we need the 68020 version. Download, uncompress and copy the files into the Amiga LIBS folder. Rename the file from ixemul-020-fpu.library to ixemul.library. Here are a couple of locations to get the file, but you can choose another aminet mirror that works best for you.

- http://main.aminet.net/util/libs/ixemul-48.0.lha
- http://ftp.plig.org/pub/aminet/util/libs/ixemul-48.0.lha

Now copy the helloworld we compiled earlier to your Amiga by putting it in one of your UAE folders, (I chose a test folder on Work) or by using a network connection or floppy to move the file to your real Amiga. Now run it either from command line or by clicking it from your Amiga GUI. You should see a small window displaying the simple line "Hello World". If you got this far, excellent! Now we will move the remaining Amiga Include and Header files to the cross compiler work area so you can start working on real projects.

Last we need to copy all the Amiga specific headers over to our new dev system. You can get these from the Amiga dev CD 2.1. Amiga.org used to host these files at http://www.amiga.com/3.9/download/NDK3.9. lha but removed it with out notice when their site was last updated. It is still available at the Aweb OpenSource project at address http://aweb.sunsite.dk/files/dev/NDK3.9.lzx. You can probably ask some friendly user at your local Amiga Dev or chat site for a copy. Once you get a copy and have it uncompressed to a folder on your Dev system, you need to copy the header files from the /include_h folder from NDK 3.9 over to our work folders. I also moved the old includes out of the build path that were created during the build of gcc68. We still need our old stdio.h file it should remain in the /include/dos folder where we copied it earlier.

```
# cp -R /YOUR_NDK_3.9_FOLDER/Include/include_h/* /usr/local/amigatools/m68k-amigaos/include
# mv /usr/local/amigatools/lib/gcc-lib/m68k-amigaos/3.3.3/include /usr/local/amigatools/lib/gcc-lib/m68k-amigaos/3.3.3/include.not
```

Now test the new library by compiling some real Amiga Code. Read closely the sample code included below. For this cross-compiler, it's necessary to include the matching PROTO header for any Amiga libraries you are using. You must also cast the Amiga libraries properly. Sample code from books, magazines and internet sources written specifically for Amiga probably won't compile properly with out some editing. Notice the lines from the Sample that include the Protos …..

```
#include <proto/dos.h>
#include <proto/exec.h>
#include <proto/intuition.h>
#include <proto/graphics.h>
```

And also notice the casting needed when opening and closing libraries….

```
IntuitionBase =(struct IntuitionBase *)OpenLibrary("intuition.library",37L);
GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",33L);
CloseLibrary((struct Library *)GfxBase);
CloseLibrary((struct Library *)IntuitionBase);
```

You may need to edit sample code that you get from other sources to match these examples, or it will fail to compile with "missing file", "undefined function" or "type mismatch" errors.

This sample code includes many of the library calls that all Amiga programs will typically use and will be a good test of the functionality of the new cross compiler. If you can compile and run this sample, you should be on your way to developing more serious projects on your new system. I have been able to down load open source Amiga code from aminet, re-write a few lines to make it GCC friendly, compile and run successfully on UAE. So I believe if you have made it this far, your dev system should be reliable.

You can cut, paste and save the following code for your next test as we did with the helloworld example. Save the file as windowtest.c to your preferred work folder, cd to your folder and run the following commands. Remember your search path must be set correctly to run gcc68 without using the complete folder path name. If you haven't already added this path to your login script in .bashrc or .profile, then you will have to execute this export each time you log in

```
# export PATH=$PATH:/usr/local/amigatools/bin (if needed)
# cd (to your project work folder)
# gcc68 -o ./windowtest -m68020 ./windowtest.c –lamiga (Remember to use –lamiga from now on)
```

```
//Amiga Cross-Compiler sample code windowtest.c

#include <dos/stdio.h>
#include <exec/types.h>
#include <exec/exec.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <intuition/screens.h>
#include <graphics/gfxmacros.h>
#include <graphics/gfxbase.h>
#include <proto/dos.h>
#include <proto/exec.h>
#include <proto/intuition.h>
#include <proto/graphics.h>

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct TagItem win_tags[] = {
    {WA_Left,        20},
    {WA_Top,         20},
    {WA_Width,       200},
    {WA_Height,      160},
    {WA_CloseGadget, TRUE},
    {WA_IDCMP,       IDCMP_CLOSEWINDOW},
};

int main(void)
{
    struct Window *win;
    struct RastPort *rastport;
    int x;
    IntuitionBase =(struct IntuitionBase *)OpenLibrary("intuition.library",37L);
    if (IntuitionBase!=NULL)
    {
       GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",33L);
        if (GfxBase!=NULL)
        {
            win=OpenWindowTagList(NULL,win_tags);
            if (win!=NULL)
            {
                rastport=win->RPort;
                for (x=20;x<200;x+=2)
                {
                    Move(rastport,x,150);
                    Draw(rastport,20,20);
                }
                WaitPort(win->UserPort);
                CloseWindow(win);
            }
            CloseLibrary((struct Library *)GfxBase);
        }
        CloseLibrary((struct Library *)IntuitionBase);
    }
    return 0;
}
```

You can check for errors using ….

```
# gcc68 -v ./windowtest.c –m68020 -lamiga -Wl,--verbose 2> ldlog.txt
# vi ./ldlog.txt
```

If the compile completed without errors, it's time to copy the finished executable over to your Amiga or to one of your UAE folders and run it. If it works, then a simple window with diagonal lines will appear on your workbench. You can then close it with the close gadget. If you've made it this far and the test program has worked, then congratulations! It's time to get busy coding your next Amiga project.

Below I've created a couple of extra topics that I though were important. They will help when it comes time to make your own libraries and deal with old code that uses the __chip pre-processor directive. You might also like to check out a nice editor for windows and linux at http://www.scintilla.org/SciTE.html. It does a very nice job, and it's free !

Extra Stuff

Building Your Own libamiga.a file First we need the utility hunk2gcc. This is an Amiga utility that has to be run on a real Amiga or UAE. It is not a Linux or Cygwin executable. Get the utility at http://main.aminet.net/dev/gg/hunk2aout-bin.lha

Make a new folder Ram:amigalib on your Amiga Uncompress the hunk2aout executable file to your Amiga Ram: folder. Now also copy the file amiga.lib from your NDK 3.9 folder or CD to your Amiga Ram: folder Optionally I copied over the reaction.lib file too, but the doc says it will only work with SAS/C. I plan to try it out and see what other libs reaction.lib might add. You can leave this extra file out. From your Ram: folder run

```
> hunk2aout amiga.lib reaction.lib (reaction.lib is optional)
```

This will create an a.out object file for every lib function in the amiga.lib hunk file. (and any other lib files you include in your ram: folder and command line) There will be many with names of obj*.

If you're working in UAE on your Cygwin system, move all the new obj files from the UAE Ram: folder to a Cygwin folder by using window explorer to move all the obj files to /usr/local/crosstools/newlibamiga (MAJOR CHEAT) If not, you will have to copy the files over to the Cygwin system by net or floppy.

```
# mkdir /usr/local/crosstools/newlibamiga
# cd /usr/local/crosstools/newlibamiga
```

Now convert all the new obj_files into an a.out style library called libamiga.a .

```
# ../m68k-amigaos/bin/m68k-amigaos-ar.exe qc ./libamiga.a obj.*
# ../m68k-amigaos/bin/m68k-amigos-ranlib.exe ./libamiga.a
```

Find and remove all the old libamiga.a files. (Or rename them if you prefer)

```
find /usr/local/crosstools -name libamiga.a –exec rm '{}' \;
```

(Here's the output from my folders)
/usr/local/crosstools/lib/libamiga.a
/usr/local/crosstools/lib/libb/libamiga.a
/usr/local/crosstools/m68k-amigaos/sys-include/lib/libamiga.a
/usr/local/crosstools/m68k-amigaos/sys-include/lib/libb/libamiga.a
/usr/local/crosstools/m68k-amigaos/lib/libamiga.a

Copy the new libamiga.a file in to the GCC environment.

```
# cp /usr/local/crosstools/newlibamiga/libamiga.a_/usr/local/crosstools/m68k-amigaos/lib/
# ../../bin/m68k-amigaos-ranlib.exe ./libamiga.a
# cd /usr/local/crosstools/bin
```

More Extra Stuff on Chip Memory Allocation in GCC

If your are going to be working with existing code or you are used to working with native Amiga compilers that handle chip memory with pre-processor labels like __chip, then you will need to make a few changes to your new code. Here is an example that that I worked with while working on this document.

The original code was written as….

```
__chip UWORD Data[] = {
  0x0, 0x0, 0x0, 0x0, 0x3, 0x8000, 0x7, 0xc000, 0xf,
  0xe000, 0x1f, 0xf000, 0x7f, 0xfc00, 0x1ff, 0xff00, 0x7ff, 0xffc0,
  0x3fff, 0xfff8, 0x7ff, 0xffc0, 0x1ff, 0xff00, 0x7f, 0xfc00, 0x1f,
  0xf000, 0xf, 0xe000, 0x7, 0xc000, 0x3, 0x8000, 0x0, 0x0,
  0x0,  0x0, };
```

The native compiler knows how an where in Amiga memory to allocate the data when it reads __chip. GCC does note understand __chip. This version of GCC can allocate chip memory though if we write the code properly. Here is my example of how to allocate chip memory. My function doesn't include error checking as it should, but you can add that later.

First we change the data struct a bit and add a new pointer.

```
UWORD *Data;
UWORD chipmem_Data[] = {
  0x0, 0x0, 0x0, 0x0, 0x3, 0x8000, 0x7, 0xc000, 0xf,
  0xe000, 0x1f, 0xf000, 0x7f, 0xfc00, 0x1ff, 0xff00, 0x7ff,0xffc0,
  0x3fff, 0xfff8, 0x7ff, 0xffc0, 0x1ff, 0xff00, 0x7f, 0xfc00, 0x1f,
  0xf000, 0xf, 0xe000, 0x7, 0xc000, 0x3, 0x8000, 0x0, 0x0,
  0x0, 0x0, };
```

Then we create a new proto and function .

```
int LoadData(void);
int LoadData(void){
  int j;
  Data = (UWORD *)AllocMem(sizeof(chipmem_Data), MEMF_CHIP);
  for (j=0; j<37;j++)
  Data[j]=chipmem_Data[j];
  return 0;
}
```

Later we call our new function to load our data into chip memory.

```
int main(int argc, char *argv[]) {
  ****
  (Some where after all your libs have been opened)
   LoadData();
   ****
}
```

Props and Acknowledgements

- Maciek Plewa (http://www.mil-sim.net), Par Taz (http://www.guru-meditation.net);
- Nicolas Mendoza for helping me solve the casting and lib problems;
- Nick Gammon (http://www.gammon.com.au);
- Kurt Wall and William von Hagen the authors of “The Definitive Guide to GCC” (http://www.apress.com/DefinitiveGCC);
- Thanks for the knowledge gained in reading their books and guide on the net.

Source :

UtilityBase, Your guide to Amiga development
(http://www.libsdl.org/extras/amigaos/cross/src/article.html)