Application Note for BE8 Toolchain

MOVING FORWARD
FASTER

Table of Contents

1. APPLICATION NOTE FOR BE8 TOOLCHAIN

1.1 BE32 and BE8 format in toolchain

After compiling and assembling, the big endian object files are all BE32 format (the code and data are all big endian)

After linking, if the --be8 linker option is applied, the output object will be BE8 format (the code is transformed to little endian, the data still remains big endian)

Otherwise, without --be8 linker option, the output object is still BE32 format.

The BE8 format is supported in ARMv7 architecture only.

1.2 Final linking and partial linking (incremental linking) for BE8 format

The BE8 format is only for final linking. If your output objects are still partial linking, please do not apply --be8 linker option.

In the EABI specification, only executable and dynamic (shared) objects can be BE8 format.

The only exception is for kernel modules. Although kernel modules are relocatable objects, they are a kind of "final linking" in the meaning.

So the key concept is that BE8 format is only for the final output objects. It means:

a) You can only apply --be8 linker option in the final step to output the final object.

b) If object files will be input objects for linking (with -r linker option), these input object can not be BE8 format. The BE8 format objects can not be input objects for linker to output a new object.

1.3 How to build BE8 applications

To build BE8 applications with multiple .c files and multiple .o files (these .o files should be BE32 format), please use gcc -c to compile the .c files into .o files.

Then use gcc -Wl,--be8 or ld --be8 to link all the .o files to generate the BE8 format application.

If you have multiple existing .o files and need to merge these .o files into a single .o file and then link with your main.c to generate your application, you can do something like this:

a) gcc -c main.c –o main.o

b) gcc -r a.o b.o c.o -o d.o (these input .o files should be BE32 format)

c) gcc -Wl,--be8 main.o d.o -o myapp.exe (or ld --be8 main.o d.o –o myapp.exe)

1.4 How to build BE8 shared libraries

The concept is the same as building BE8 application.

If you have multiple existing .o files and need to merge these .o files into a single .o file and then link with your main.c to generate your shared library, you can do something like this:

a) gcc -c -fpic main.c -o main.o

b) gcc -r a.o b.o c.o -o d.o (these input .o files should be BE32 format and -fpic compiled)

c) gcc -Wl,--be8 -shared main.o d.o -o myshareobj.so (or ld --be8 -shared main.o d.o –o myshareobj.so)

1.5 How to build BE8 kernel modules

The concept is the same as building BE8 application.

If you have multiple existing .o files and need to merge these .o files into a single .o file and then link with your main.c to generate your kernel module, you can do something like this:

a) gcc -c main.c –o main.o

b) gcc -r a.o b.o c.o -o d.o (these input .o files should be BE32 format)

c) gcc -Wl,--be8 -r main.o d.o -o mymod.ko (or ld --be8 -r main.o d.o –o mymod.ko)

1.6 How to build BE8 static libraries

Since static libraries consist of relocatable objects, and they will be input objects for linker to generate another output object, static libraries can not be BE8 format.

1.7 How to build arm BE8 linux kernel

To build arm linux kernel as BE8 format, you have to add linker option –be8 to ld related flags:

arch/arm/Makefile:LDFLAGS_vmlinux += --be8

arch/arm/boot/compressed/Makefile:LDFLAGS_vmlinux += --be8

1.8 Default support --be8 option

New Marvell GCC 4.4 Toolchain will automatically apply --be8 option for linker when you set mcpu=marvell-fv7 or mcpu=marvell-pj4.