Building a cross compiler.

Jason Miller
Spring '10
Source: Your PC

Nerd.  Pretty Boy.
Target: Linksys WRT54G

- 216 MHz MIPS-32 processor
- 16 MB RAM
- 4 MB Flash Memory
- 1 802.11B/G wireless interface
- 5 100Mbs ethernet ports
# The processors

<table>
<thead>
<tr>
<th></th>
<th>X86</th>
<th>MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designer</td>
<td>Intel / AMD</td>
<td>MIPS Computer Systems</td>
</tr>
<tr>
<td>Design</td>
<td>CISC</td>
<td>RISC</td>
</tr>
<tr>
<td>Bits</td>
<td>16,32,64</td>
<td>32,64</td>
</tr>
<tr>
<td>Endian</td>
<td>Little</td>
<td>Bi *</td>
</tr>
<tr>
<td>Type</td>
<td>Register-Memory</td>
<td>Register-Register</td>
</tr>
</tbody>
</table>
The operating systems

- Fedora Core 9  Linux
- Kernel 2.6.27.25-78.2.56.fc9.x86_64 #1 SMP

- OpenWrt Linux
- Kernel 2.6.25.20 #4
The compiler

- The **Gnu Compiler Collection**
- Includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and associated libraries
- A 5-stage compiler (preprocessing, parsing, translation, assembly, linkage)
What do we need?

• The obvious answer is that we need a version of the GCC C compiler that can run on x86 hardware and can output machine code for the MIPS hardware.

• The not-so-obvious addendum to that is that, since we are writing software for an embedded device, we need to keep the machine code as small as possible.

• Having said that, we decide to use Newlib (http://sourceware.org/newlib/). It is a drop-in replacement for libc (the C standard library) that is specifically designed to provide a functional subset of capability on platforms with extremely limited resources. This includes many different embedded devices as well as software environments such as Cygwin.
The conundrum

• In order to build our required version of GCC, we need to have an already-built version of Newlib to link against.

• However, in order to build Newlib, we need a working version of our compiler that can output code for our target platform.

What ever shall we do?
The givens

A natively-built **x86 C compiler** (provided by the GCC package as part of the Fedora linux distribution)

A natively-built **x86 assembler** (provided by the GCC package as part of the Fedora linux distribution)

The source code for the GCC **C compiler** (from http://gcc.gnu.org)

The source code for the **binutils assembler** (from http://www.gnu.org/software/binutils/)
The bootstrap

- Configured before compilation

Building the MIPS assembler.

Building the MIPS compiler.

* Configured before compilation
Newlib

Our compiler/assembler from the last step.
We need to re-build GCC and have it link against the Newlib that we just built.
The code

main.c

/*
 * include <stdio.h>
 */

void foo();
void bar();

int main(int argc, char **argv) {
    printf("Built %s - %s\n", __TIME__, __DATE__);
    foo();
    bar();
    return(0);
} /* end main() */

foo.c

/*
 * printf (__const char *__restrict __format, ...);
 */

void foo()
{
    printf("foo() called\n");
} /* end foo() */

bar.c

int printf (__const char *__restrict __format, ...);

void bar()
{
    printf("bar() called\n");
} /* end bar() */
The Makefile

* SDK_DIR=/home/mips-sdk/staging_dir/toolchain-mipsel_gcc4.1.2
  MIPS_CC=${SDK_DIR}/bin/mipsel-linux-gcc
  X86_64_CC=/usr/bin/gcc

all: main-mips main-x86_64

deploy: main-mips
  scp main-mips root@10.0.0.23:/tmp

clean:
  rm -rf *.
  rm *.pp *.tu main-mips main-x86_64

# Let's build a mips binary
main-mips: main.c foo-mips.o bar-mips.o
  ${MIPS_CC} -Wall -S $? -o @$s 2>/dev/null
  ${MIPS_CC} -Wall $? -o @$

foo-mips.o: foo.c
  ${MIPS_CC} -Wall -S $? -o @$s
  ${MIPS_CC} -Wall -c $? -o @$

bar-mips.o: bar.c
  ${MIPS_CC} -Wall -S $? -o @$s
  ${MIPS_CC} -Wall -c $? -o @$

# Let's build an x86_64 binary
main-x86_64: main.c foo-x86_64.o bar-x86_64.o
  ${X86_64_CC} -Wall -E main.c -o @$.pp
  ${X86_64_CC} -Wall -S $? -o @$s 2>/dev/null
  ${X86_64_CC} -Wall -fdump-translation-unit $? -o @$

foo-x86_64.o: foo.c
  ${X86_64_CC} -Wall -S $? -o @$s
  ${X86_64_CC} -Wall -c $? -o @$

bar-x86_64.o: bar.c
  ${X86_64_CC} -Wall -S $? -o @$s
  ${X86_64_CC} -Wall -c $? -o @$
Demo time

Things to do:

- Build the test program
- What can 'file' tell us about the programs?
- Run them both on x86. What happens?
- Deploy to and run on the WRT
- Look at the intermediate output files