SIMPLESCALAR INSTALLATION MADE SIMPLE

Preface

The following text describes the procedure of installation of Simple Scalar on Linux distribution Ubuntu. It was tested on Ubuntu 9.04 by me, and 8.10. The sources used while writing the post have been listed at the bottom and they hold the highest credit.
Update: [September 23rd 2009] – It has been tested on **Redhat** too with the exception of omitting/avoiding Error #1 of GCC Cross Compiler installation

Necessary Files

Download the necessary Source code files.

Simpletools-2v0.tgz
Simplesim-3v0d.tar.gz
Simpleutils-990811.tar.gz
Gcc-2.7.2.3.ss.tar.gz

Setting up environment

Open up the terminal and type

```
uname -a
```

You will get to know your kernel version and the type of linux installed (i386/i686)
Depending on that change the Host id below as either
HOST=i686-pc-linux or HOST=i386-pc-linux

```
$ export HOST=FROM_ABOVE_OPTION
$ export IDIR=/home/YOUR_USER_NAME/simplescalar
$ export TARGET=sslittle-na-sstrix
```

(If you use tcsh or the like, the only difference should be in how environment variables are set.)
Create the directory “simplescalar” under your home directory and copy all the four tar files into it. To do so, use the following commands.

```
$ mkdir $IDIR
$ cd $IDIR
```

Make sure you have installed the following packages

flex
bison
build-essential

You can use the command “sudo apt-get install <PACKAGE_NAME>” to retrieve and install these packages.

Installing Simple tools

Just un-pack the package file, and remove the old gcc folder. To do so, use:

```
$ cd $IDIR
$ tar xzvf simpletools-2v0.tgz
$ rm -rf gcc-2.6.3
```

Installing SimpleUtils

First un-pack the package file. To do so, use:

```
$ tar xzvf simpleutils-990811.tar.gz
$ cd simpleutils-990811
```

Before building the code, you need to fix some sources of errors. In directory ld find file ldlex.l and replace all instances of
yy_current_buffer with YY_CURRENT_BUFFER.

在这条语句之前，
加入：CC=gcc

You may either do it manually by opening the respected file and renaming it or to make it simple just type this code

```
$ find . -type f -print0 | xargs -0 sed -i -e ’s,yy_current_buffer,YY_CURRENT_BUFFER,g’
$ ./configure –host=$HOST –target=$TARGET –with-gnu-as –with-gnu-ld –prefix=$IDIR
$ make
$ make install
```

Installing Simulator

Un-pack the simulator package.

```
$ cd $IDIR
$ tar xzvf simplesim-3v0d.tgz
$ cd simplesim-3.0
$ make config-pisa
$ make
```

You may test the installation of simplesim by

```
$ ./sim-safe tests/bin.little/test-math
```

Installing GCC Cross-Compiler

This is the important step where most of the newbies (including me) are/were struggling. So please follow these steps carefully.

```
$ cd $IDIR
$ tar xzvf gcc-2.7.2.3.ss.tar.gz
$ cd gcc-2.7.2.3
$ export PATH=$PATH:/home/YOUR_USER_NAME/simplescalar/sslittle-na-sstrix/bin
$ ./configure –host=$HOST –target=$TARGET –with-gnu-as –with-gnu-ld –prefix=$IDIR
```

Now before you proceed ahead, there are quite a few corrections that have to be made in some files:

1.) Change the Makefile at line 130, by appending –I/usr/include to the end of the line
To do so you can use command

```
$ gedit Makefile
```

(This, I feel is the important step which makes this installation process Ubuntu specific. I may be wrong as -I./include should have done the trick for other linux. But still its OK to include it in the usr folder than the current directory)

2.) Edit line 60 of protoize.c, and replace

#include <varargs.h> with #include <stdarg.h>

Todo so you can use command

```
$ chmod +w protoize.c
$ gedit protoize.c
```

3.) Edit obstack.h at line 341 and change

```
*((void **)__o->next_free)++=((void *)datum);\
```

with

```
*((void **)__o->next_free++)=((void *)datum);\
```

To do so you can use the following command

```
$ chmod +w obstack.h
$ gedit obstack.h
```

4.) Copy the patched files located in the patched directory to avoid some parse errors while compiling. To do so use the following command.

```
$ cp ./patched/sys/cdefs.h ../sslittle-na-sstrix/include/sys/cdefs.h
$ cp ../sslittle-na-sstrix/lib/libc.a ../lib/
$ cp ../sslittle-na-sstrix/lib/crt0.o ../lib/
```

**If you dont find the patched directory in your browser, you probably didnt unrar it properly. Again unrar the GCC tar file “at a different location” (say your Desktop) and copy from it.*

5.) *** Crucial Step*** Download this file, un-tar it and place its contents .i.e ar & ranlib in $IDIR/sslittle-na-sstrix/bin – FILE

You would also want to confirm that these files have “execution & write permission” You can do so by

```
$ cd $IDIR/sslittle-na-sstrix/bin
$ ls -al
```

If you see each file of this folder with write(w) & execution(x) permission then you are ready to go further. If not then you have to assign them the permission by using

```
chmod +w <filename>
chmod +x <filename>
$ make
```

Again you will face few errors

1.) Now you will get many insn-output.c errors. To solve this you need to add line breaks (‘\’) after each of the three FIXME (line 675, 750 and 823) in the insn-output.c

To open this file, use

```
$ gedit insn-output.c
$ make
```

2.) In objc/sendmsg.c, add the following code at line 35

```
#define STRUCT_VALUE 0
$ cd $IDIR/gcc-2.7.2.3/objc
$ chmod +w sendmsg.c
$ gedit sendmsg.c
$ cd ..
$ make LANGUAGES="c c++" CFLAGS="-O" CC="gcc"
```

3.) The last make command will lead to an error message which requires you to edit cxxmain.c file

To solve this you need to remove lines 2978-2979 in file cxxmain.c .i.e Remove the following lines

char * malloc ();

char * realloc ();

To do so, use this command.

```
$ chmod +w cxxmain.c
$ gedit cxxmain.c
```

Now cross your fingers, because we are about to execute the final error free make command.

```
$ make LANGUAGES="c c++" CFLAGS="-O" CC="gcc"
$ make install  LANGUAGES="c c++" CFLAGS="-O" CC="gcc"
```

Testing

To test the simulator use these commands

```
$ clear
$ cd $IDIR
$ gedit hello.c
```

Now in Gedit you can write your program like this (You may modify it)

```
#include<stdio.h>
main (void)
{
printf("My name is Yogesh Mhatre. I'm not a Coder, but I do like debugging errors.\nI like blogging and sharring my minuscule knowledge to the world through it.\n");
}
```

Once you are done writing the code, use the following command to test it.

```
$ $IDIR/bin/sslittle-na-sstrix-gcc -o hello hello.c
$ $IDIR/simplesim-3.0/sim-safe hello
```

You would "hopefully" get an output similar to this

```
sim: ** starting functional simulation **
My name is Yogesh Mhatre. I'm not a Coder, but I do like debugging errors.
I like blogging and sharing my minuscule knowledge to the world through it.
sim: ** simulation statistics **
sim_num_insn 9242 # total number of instructions executed
sim_num_refs 4328 # total number of loads and stores executed
sim_elapsed_time 1 # total simulation time in seconds
sim_inst_rate 9242.0000 # simulation speed (in insts/sec)
ld_text_base 0×00400000 # program text (code) segment base
ld_text_size 71968 # program text (code) size in bytes
ld_data_base 0×10000000 # program initialized data segment base
ld_data_size 8528 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base 0×7fffc000 # program stack segment base (highest address in stack)
ld_stack_size 16384 # program initial stack size
ld_prog_entry 0×00400140 # program entry point (initial PC)
ld_environ_base 0×7fff8000 # program environment base address address
ld_target_big_endian 0 # target executable endian-ness, non-zero if big endian
mem.page_count 26 # total number of pages allocated
mem.page_mem 104k # total size of memory pages allocated
mem.ptab_misses 26 # total first level page table misses
mem.ptab_accesses 495046 # total page table accesses
mem.ptab_miss_rate 0.0001 # first level page table miss rate
```