SimpleScalar 3.0 installation instructions

ECE/CS 466 Advanced Computer Architecture
University of Illinois at Chicago
TAYUAN(henry)

Introduction to SimpleScalar

SimpleScalar is an open source computer architecture simulator created by Professor Todd Austin while he studied at the University of Wisconsin Madison.

SimpleScalar is written using 'C' programming language. It can be used to show whether Machine A is better than Machine B **without** really building either Machine A or Machine B.

SimpleScalar includes a set of tools modeling a virtual computer system with *CPU*, *Cache* and *Memory Hierarchy*. Using the **SimpleScalar** tools, widely used for research and instruction, users can build modeling applications to simulate real programs running on a range of modern processors and systems. The tool set includes sample simulators, such as a fast functional simulator, a dynamically scheduled processor model that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction. In addition to simulators, the tool set also includes performance visualization tools, statistical analysis resources, and debug and verification infrastructure.

Which Simulators are available in the SimpleScalar toolset?

The tool set contains a collection of microarchitecture simulators that emulate the microprocessor at different levels of detail .Some important simulators are illustrated as follows:

sim-fast: fast instruction interpreter, optimized for speed. This simulator does not account for the behavior of pipelines, caches, or any other part of the microarchitecture. It performs only functional simulation using in-order execution of the instructions (i.e. they are executed in the order they appear in the program).
sim-safe: slightly slower instruction interpreter, as it checks for memory alignment and memory access permission on all memory operations. This simulator can be used if the simulated program causes sim-fast to crash without explanation.
sim-profile: instruction interpreter and profiler. This simulator keeps track of and reports dynamic instruction counts, instruction class counts, usage of address modes, and profiles of the text and data segments.
sim-cache: memory system simulator. This simulator can emulate a system with multiple levels of instruction and data caches, each of which can be configured for different sizes and organizations. This simulator is ideal for fast cache simulation if the effect of cache performance on execution time is not needed.

sim-bpred: branch predictor simulator. This tool can simulate difference branch prediction schemes and reports results such as prediction hit and miss rates. Like sim-cache, this does not simulate accurately the effect of branch prediction on execution time.
sim-outorder: detailed microarchitectural simulator. This tool models in detail and out-of-order microprocessor with all of the bells and whistles, including branch prediction, caches, and external memory. This simulator is highly parameterized and can emulate machines of varying numbers of execution units.

SimpleScalar should port easily to any 32- or 64-bit flavor of UNIX. If you have a real LINUX-based system computer or you want to directly install LINUX on a compute with no operating system, you can skip **step 1** and **step 2**, and get started with **step 3**. If you have only windows-based system computers, you need to begin with **step1**. My platform is

Laptop OS: *Windows XP*
Virtual Machine: *VMware Player 6.0*
Virtual OS: *Ubuntu 12.04*
Simplescalar Version: *3.0*
Note that ‘grey colors’ mark those instructions you need to run.

Step 1 : Install a universal virtual machine on *Windows* systems (e.g., *VMware Player 6.0*)

VMware Player 6.0 is free. You can download it from the following link
https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/6_0
You can reference the following video to learn how to install *VMware Player* on Windows
VMware Player tutorial

Step 2 : Install a *Linux* system on the above virtual machine or a real hardware (e.g., *Ubuntu 12.04*)

Ubuntu 12.04 is free. You can download it from the following link
http://www.ubuntu.com/download/desktop
You can reference the following video about how to install Ubuntu 12.04 on VMware Player
https://www.youtube.com/watch?v=_WnFlCCqUtQ (with HD version)

Step 3 : Install *Simplescalar 3.0* on the above Linux system

1. Necessary Files(Download the necessary Source code files:)

Download the following two files from (www.simplescalar.com/index.html)

- simpletools-2v0.tgz
- simplesim-3v0d.tgz (for Project 1, you should download *simplesim-3v0e.tgz* from blackboard, which includes two extra files *equake.ss* and *equake.in* you need to run)

Download

- simpleutils-990811.tar.gz `(If you insist on using the simpleutils-2v0.tgz on the official page of simplescalar, stop here and try this installation guide instead)`

Download

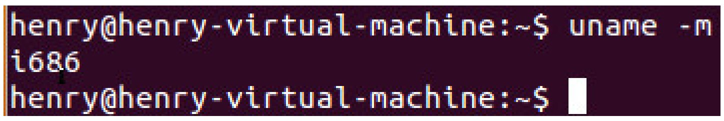
- Cross compiler gcc-2.7.2.3.ss.tar.gz

---

2. Setting up the Environment:

- Find out the machine hardware name by running the following command on the terminal:

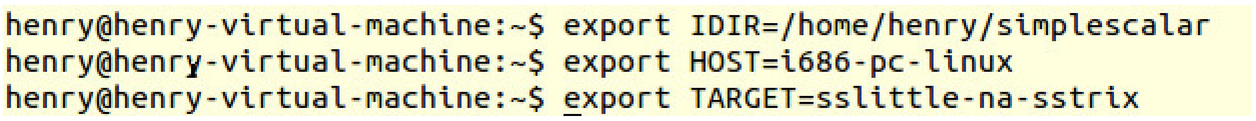
  uname -m



```
henry@henry-virtual-machine:~$ uname -m
i686
henry@henry-virtual-machine:~$ 
```

- Use this to set the HOST environment variable as i686-pc-linux or i386-pc-linux etc. Even if you have a 64 bit machine (machine name x86_64) use i686 itself. We will be using it as 32 bit installation even on the 64 bit machine.
  Now set the following three environmental variables:

  export IDIR=<Installation_directory_for_simplescalar, for instance /home/kapil/simplescalar>
  export HOST=FROM_ABOVE_OPTION
  export TARGET=sslittle-na-sstrix



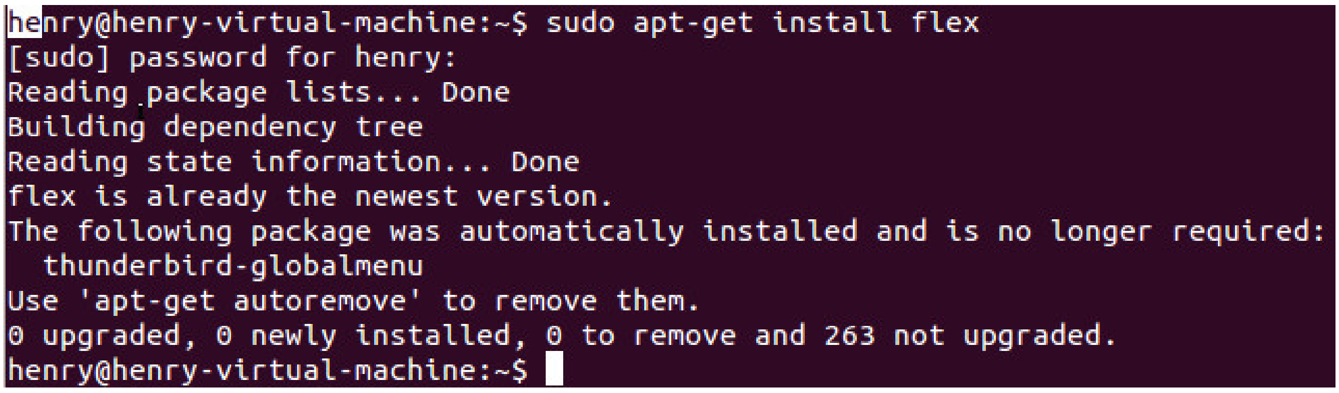
```
henry@henry-virtual-machine:~$ export IDIR=/home/henry/simplescalar
henry@henry-virtual-machine:~$ export HOST=i686-pc-linux
henry@henry-virtual-machine:~$ export TARGET=sslittle-na-sstrix
```

(Assuming your host is also little endian. It could be ssbig-na-sstrix. If this is the case, all the following appeared "sslittle-na-sstrix" should be replaced by "ssbig-na-sstrix".)

- Create the installation directory, i.e. $IDIR and copy All the installations files downloaded into the same
  mkdir $IDIR

  Make sure you have the following packages installed on your system.
  flex
  bison
  build-essential

  You can install these packages using :
  sudo apt-get install <PACKAGE NAME>



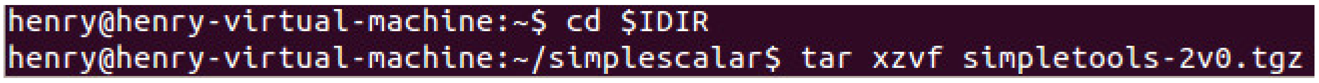
```
henry@henry-virtual-machine:~$ sudo apt-get install flex
[sudo] password for henry:
Reading package lists... Done
Building dependency tree
Reading state information... Done
flex is already the newest version.
The following package was automatically installed and is no longer required:
  thunderbird-globalmenu
Use 'apt-get autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 263 not upgraded.
henry@henry-virtual-machine:~$
```

3. Installing *simpletools*:

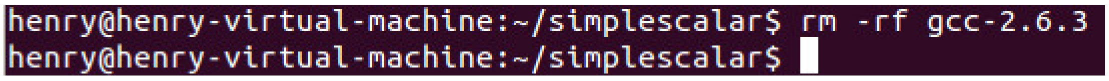
(Basically this package contains the sources of a gcc(GNU C Compiler) and a glibc ported to simplescalar architecture. And according to [7], building the glibc is non-trivial, thus this Simpletools package contains pre-compiled libraries in both ssbig-na-sstrix/ and sslittle-na-sstrix/ folder)

- Unpack the compressed file and remove the gcc folder(older version, we'll be using a newer one)

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



```
henry@henry-virtual-machine:~$ cd $IDIR
henry@henry-virtual-machine:~/simplescalar$ tar xzvf simpletools-2v0.tgz
```



```
henry@henry-virtual-machine:~/simplescalar$ rm -rf gcc-2.6.3
henry@henry-virtual-machine:~/simplescalar$
```

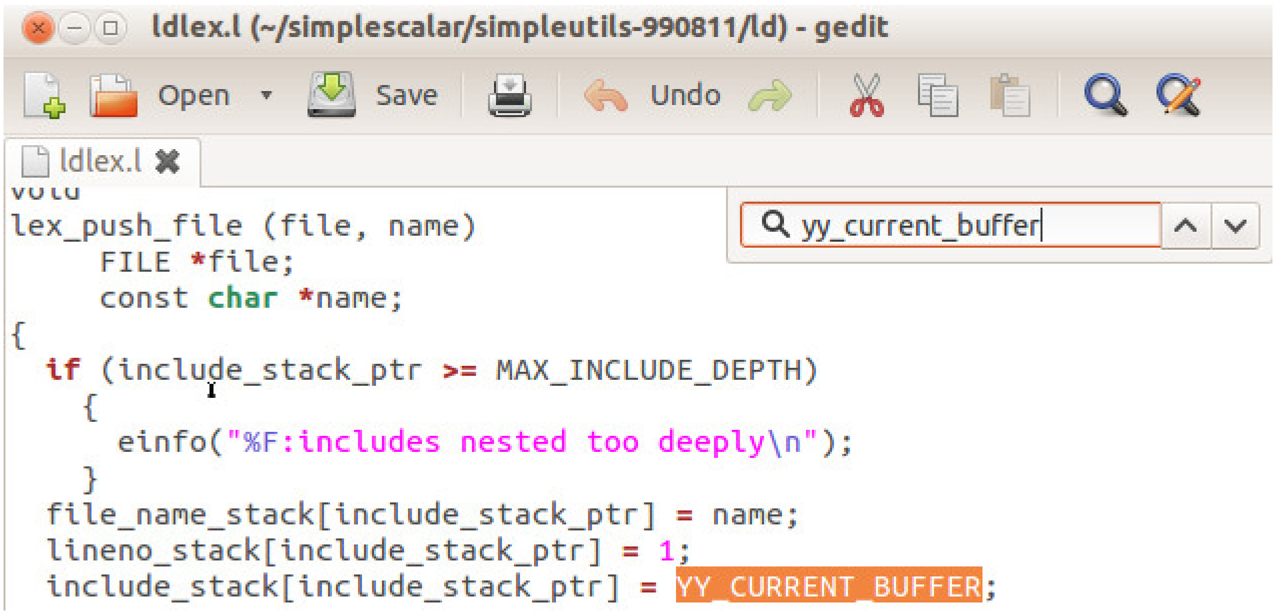
`There is a gcc-2.6.3 folder, but you can remove it because later we will use the newer version gcc-2.7.2.3(also a cross-compiler) instead.`

`(after this step you got ssbig-na-sstrix and sslittle-na-sstrix folders under the $IDIR, and they both contains an include folder and a lib folder)`

---

4. Installing *simpleutils*:

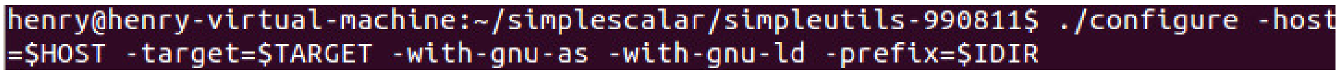
- Unpack the package:
  tar xzvf simpleutils-990811.tar.gz
  cd simpleutils-990811
- Now one change in the code needs to be done:
- In the file **ld/ldlex.l** change all occurrences of yy_current_buffer with YY_CURRENT_BUFFER `(You can to use "vim" or "gedit" to fix it.)`



```
void
lex_push_file (file, name)
     FILE *file;
     const char *name;
{
  if (include_stack_ptr >= MAX_INCLUDE_DEPTH)
    {
      einfo("%F:includes nested too deeply\n");
    }
  file_name_stack[include_stack_ptr] = name;
  lineno_stack[include_stack_ptr] = 1;
  include_stack[include_stack_ptr] = YY_CURRENT_BUFFER;
```

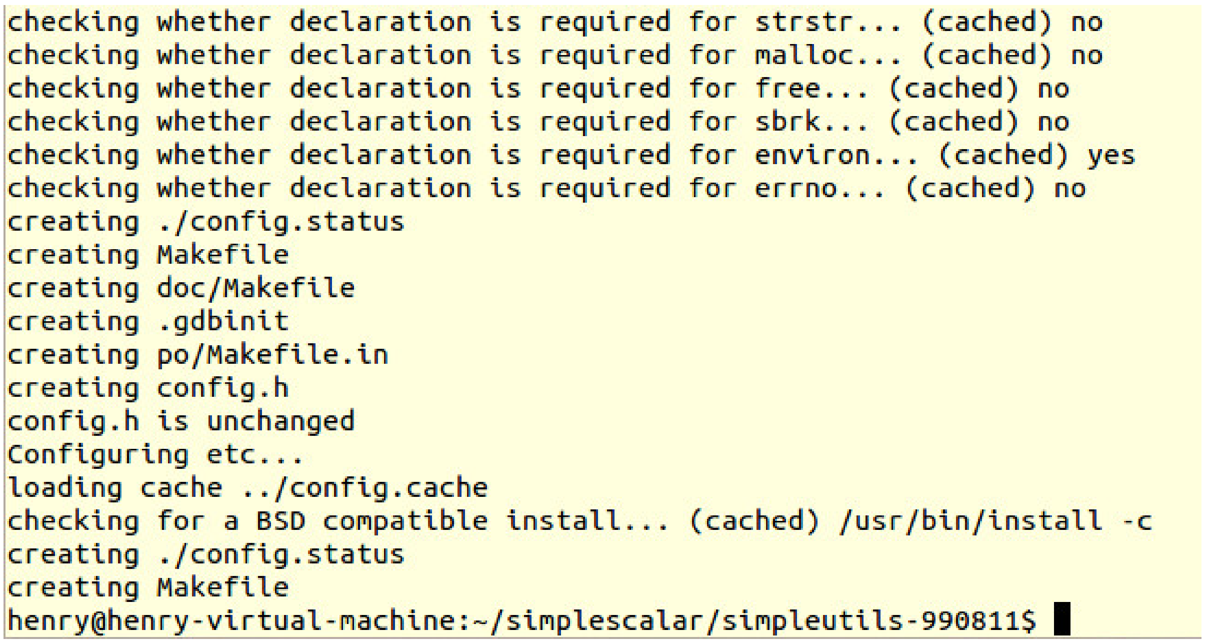
- Install using following commands:

./configure -host=$HOST -target=$TARGET -with-gnu-as -with-gnu-ld -prefix=$IDIR



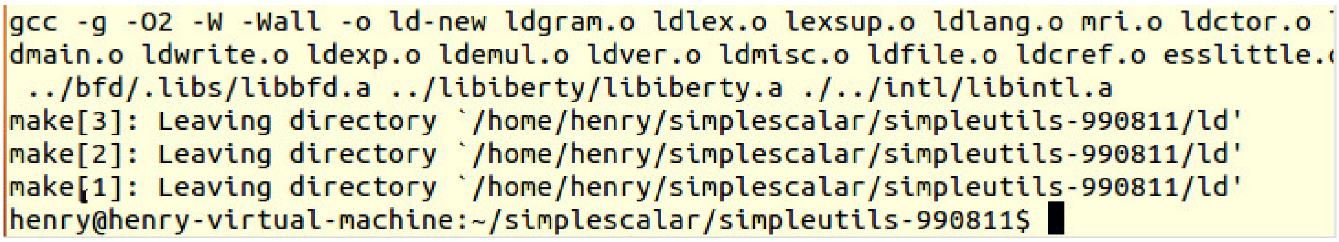
```
henry@henry-virtual-machine:~/simplescalar/simpleutils-990811$ ./configure -host
=$HOST -target=$TARGET -with-gnu-as -with-gnu-ld -prefix=$IDIR
```

*(If you want to know the meaning of these settings, refer to the [1]. Here because we are to build a "**cross**" GNU binutils, both **HOST** and **TARGET**should be specified) (It should be successful configured. If not, check whether you follow all the above instructions before you ask someone. This is also strongly suggested for all the following steps.)*



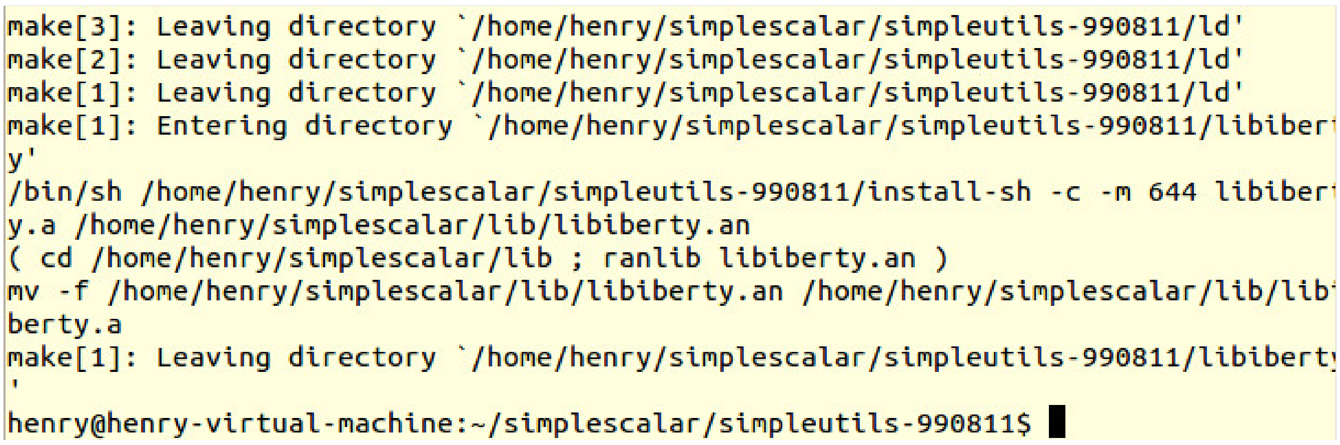
```
checking whether declaration is required for strstr... (cached) no
checking whether declaration is required for malloc... (cached) no
checking whether declaration is required for free... (cached) no
checking whether declaration is required for sbrk... (cached) no
checking whether declaration is required for environ... (cached) yes
checking whether declaration is required for errno... (cached) no
creating ./config.status
creating Makefile
creating doc/Makefile
creating .gdbinit
creating po/Makefile.in
creating config.h
config.h is unchanged
Configuring etc...
loading cache ../config.cache
checking for a BSD compatible install... (cached) /usr/bin/install -c
creating ./config.status
creating Makefile
henry@henry-virtual-machine:~/simplescalar/simpleutils-990811$
```

Make



```
gcc -g -O2 -W -Wall -o ld-new ldgram.o ldlex.o lexsup.o ldlang.o mri.o ldctor.o
dmain.o ldwrite.o ldexp.o ldemul.o ldver.o ldmisc.o ldfile.o ldcref.o esslittle.
 ../bfd/.libs/libbfd.a ../libiberty/libiberty.a ./../intl/libintl.a
make[3]: Leaving directory `/home/henry/simplescalar/simpleutils-990811/ld'
make[2]: Leaving directory `/home/henry/simplescalar/simpleutils-990811/ld'
make[1]: Leaving directory `/home/henry/simplescalar/simpleutils-990811/ld'
henry@henry-virtual-machine:~/simplescalar/simpleutils-990811$
```

make install

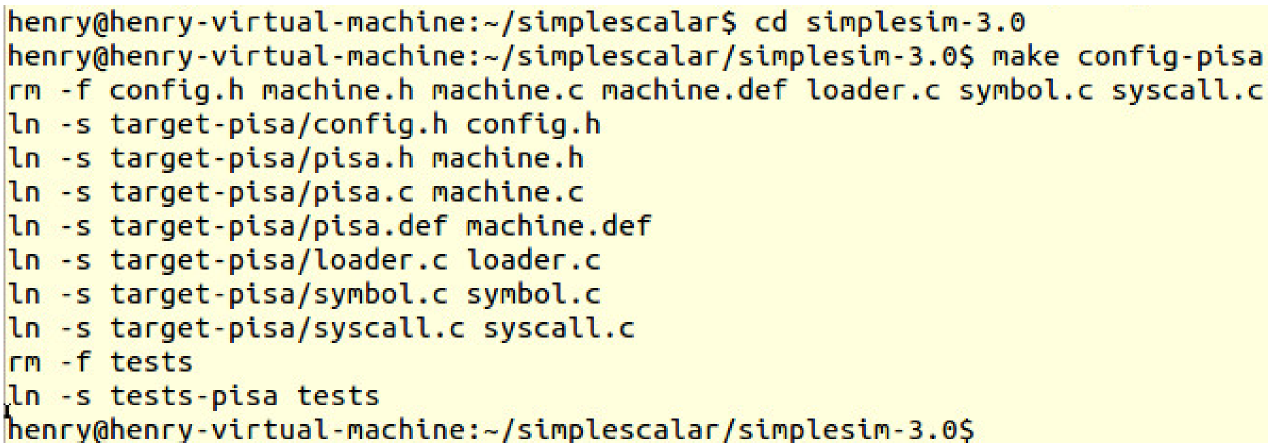


```
make[3]: Leaving directory `/home/henry/simplescalar/simpleutils-990811/ld'
make[2]: Leaving directory `/home/henry/simplescalar/simpleutils-990811/ld'
make[1]: Leaving directory `/home/henry/simplescalar/simpleutils-990811/ld'
make[1]: Entering directory `/home/henry/simplescalar/simpleutils-990811/libiber
y'
/bin/sh /home/henry/simplescalar/simpleutils-990811/install-sh -c -m 644 libiber
y.a /home/henry/simplescalar/lib/libiberty.an
( cd /home/henry/simplescalar/lib ; ranlib libiberty.an )
mv -f /home/henry/simplescalar/lib/libiberty.an /home/henry/simplescalar/lib/lib
berty.a
make[1]: Leaving directory `/home/henry/simplescalar/simpleutils-990811/libibert
'
henry@henry-virtual-machine:~/simplescalar/simpleutils-990811$
```

5. Installing *simulator*: :

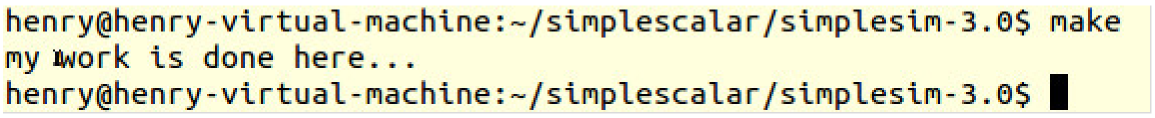
- Unpack the simulator:

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



```
henry@henry-virtual-machine:~/simplescalar$ cd simplesim-3.0
henry@henry-virtual-machine:~/simplescalar/simplesim-3.0$ make config-pisa
rm -f config.h machine.h machine.c machine.def loader.c symbol.c syscall.c
ln -s target-pisa/config.h config.h
ln -s target-pisa/pisa.h machine.h
ln -s target-pisa/pisa.c machine.c
ln -s target-pisa/pisa.def machine.def
ln -s target-pisa/loader.c loader.c
ln -s target-pisa/symbol.c symbol.c
ln -s target-pisa/syscall.c syscall.c
rm -f tests
ln -s tests-pisa tests
henry@henry-virtual-machine:~/simplescalar/simplesim-3.0$
```

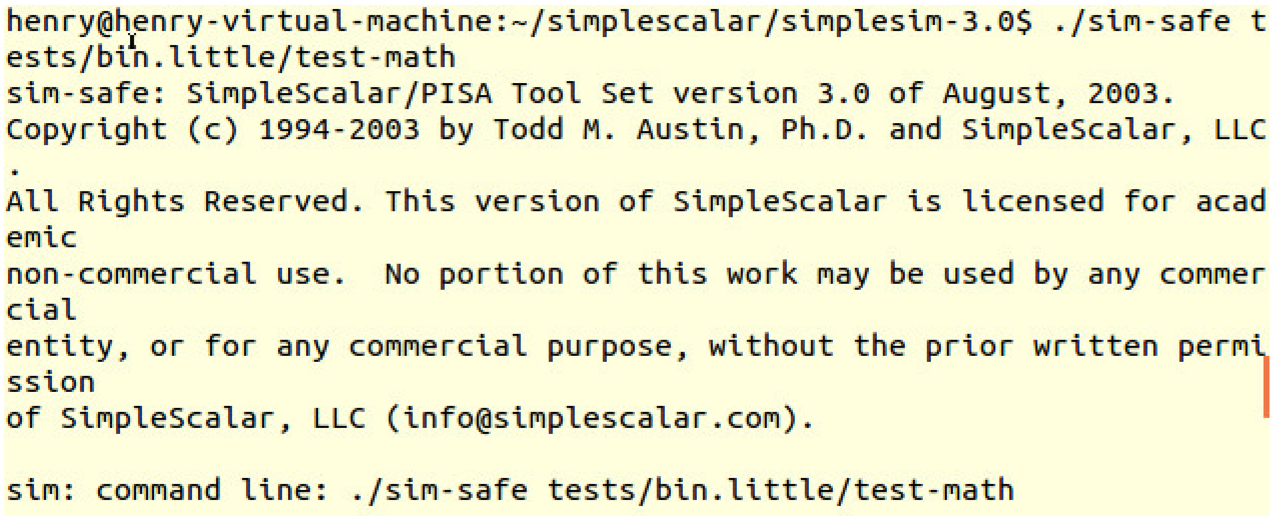
make



```
henry@henry-virtual-machine:~/simplescalar/simplesim-3.0$ make
my work is done here...
henry@henry-virtual-machine:~/simplescalar/simplesim-3.0$
```

Congratulations! You have installed the simulator. The battle is half won.

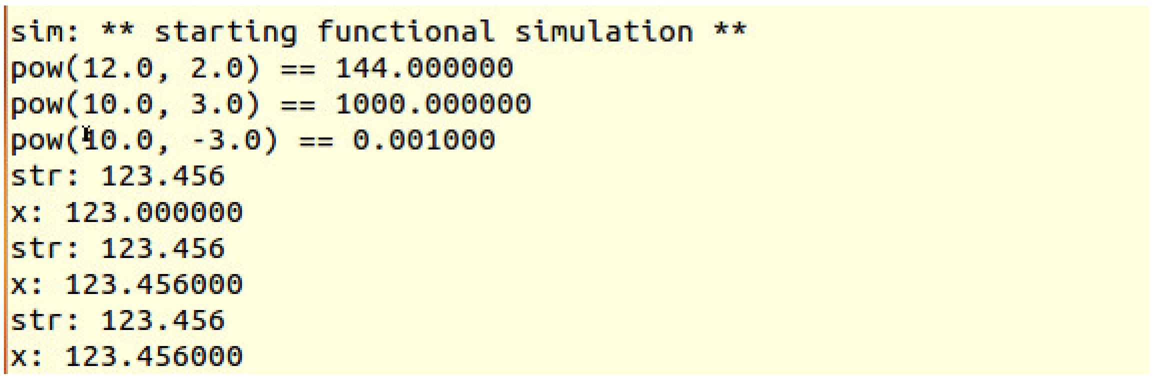
- Test your installation by running a sample program:
  ./sim-safe tests/bin.little/test-math



```
henry@henry-virtual-machine:~/simplescalar/simplesim-3.0$ ./sim-safe t
ests/bin.little/test-math
sim-safe: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC
.
All Rights Reserved. This version of SimpleScalar is licensed for acad
emic
non-commercial use.  No portion of this work may be used by any commer
cial
entity, or for any commercial purpose, without the prior written permi
ssion
of SimpleScalar, LLC (info@simplescalar.com).

sim: command line: ./sim-safe tests/bin.little/test-math
```

(You will see something like

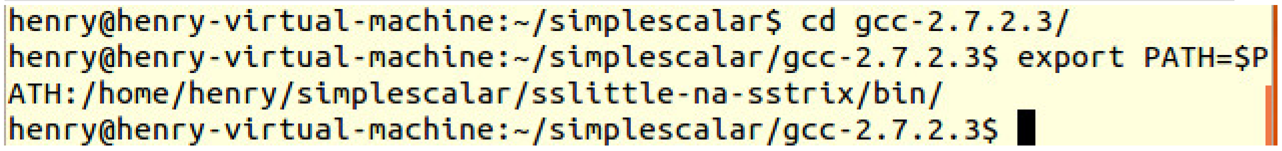


```
sim: ** starting functional simulation **
pow(12.0, 2.0) == 144.000000
pow(10.0, 3.0) == 1000.000000
pow(10.0, -3.0) == 0.001000
str: 123.456
x: 123.000000
str: 123.456
x: 123.456000
str: 123.456
x: 123.456000
```

---

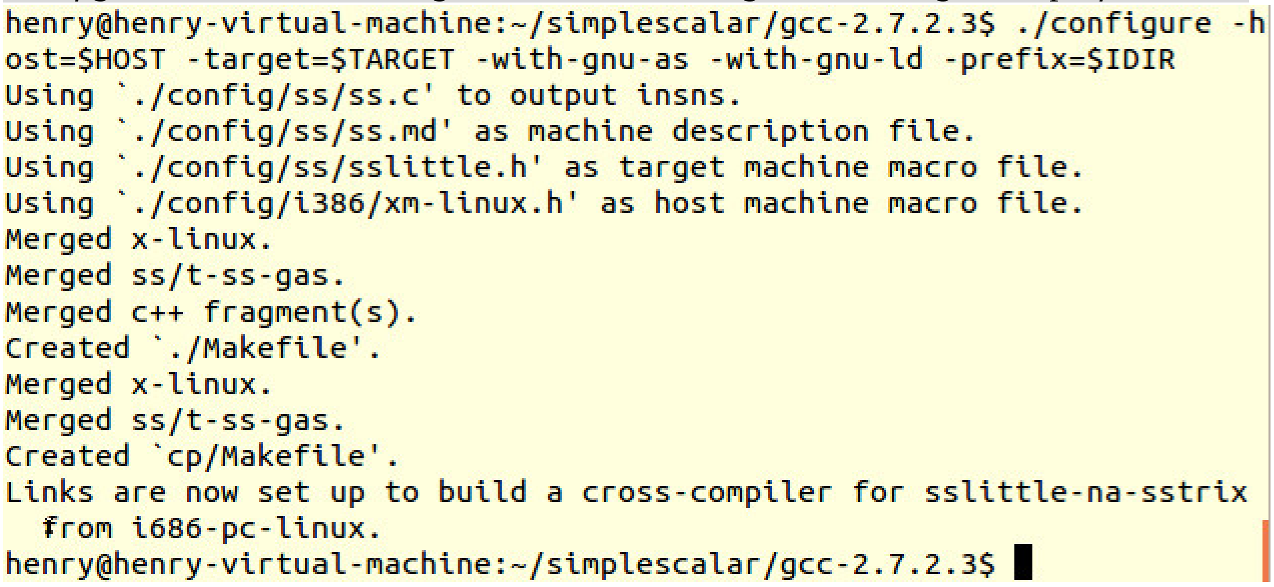
6. Installing the cross compiler:

- Unpack the compiler and append the path to sslittle-na-sstrix/bin to the environmental variable PATH
  This is the most error-prone part for the whole installation. As always, unpack the file first:
  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



```
henry@henry-virtual-machine:~/simplescalar$ cd gcc-2.7.2.3/
henry@henry-virtual-machine:~/simplescalar/gcc-2.7.2.3$ export PATH=$P
ATH:/home/henry/simplescalar/sslittle-na-sstrix/bin/
henry@henry-virtual-machine:~/simplescalar/gcc-2.7.2.3$ █
```

./configure -host=$HOST -target=$TARGET -with-gnu-as -with-gnu-ld -prefix=$IDIR



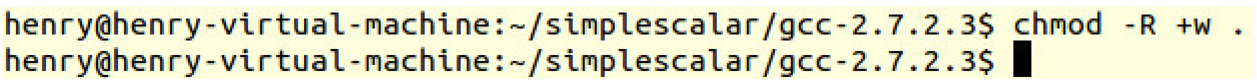
```
henry@henry-virtual-machine:~/simplescalar/gcc-2.7.2.3$ ./configure -h
ost=$HOST -target=$TARGET -with-gnu-as -with-gnu-ld -prefix=$IDIR
Using `./config/ss/ss.c' to output insns.
Using `./config/ss/ss.md' as machine description file.
Using `./config/ss/sslittle.h' as target machine macro file.
Using `./config/i386/xm-linux.h' as host machine macro file.
Merged x-linux.
Merged ss/t-ss-gas.
Merged c++ fragment(s).
Created `./Makefile'.
Merged x-linux.
Merged ss/t-ss-gas.
Created `cp/Makefile'.
Links are now set up to build a cross-compiler for sslittle-na-sstrix
  from i686-pc-linux.
henry@henry-virtual-machine:~/simplescalar/gcc-2.7.2.3$
```

(so far so good! But we cannot proceed with a "make" here because there will be many errors because of various incompatibilities. To get over these, we need to fix some of the sources:)

- Change permissions to edit the files:

  (Firstly, for our convenience, do the following to get ourselves with write access to the current directory. *Don't forget about the little "." at the end of this command!*)

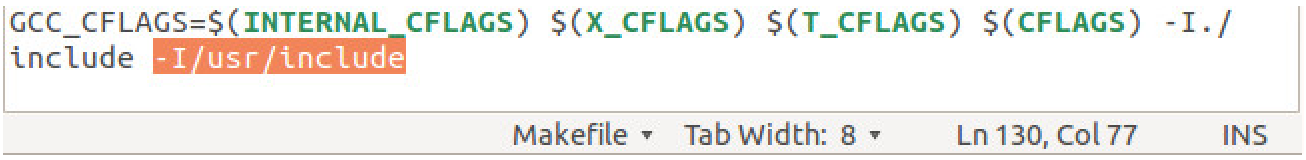
  chmod -R +w .



```
henry@henry-virtual-machine:~/simplescalar/gcc-2.7.2.3$ chmod -R +w .
henry@henry-virtual-machine:~/simplescalar/gcc-2.7.2.3$
```

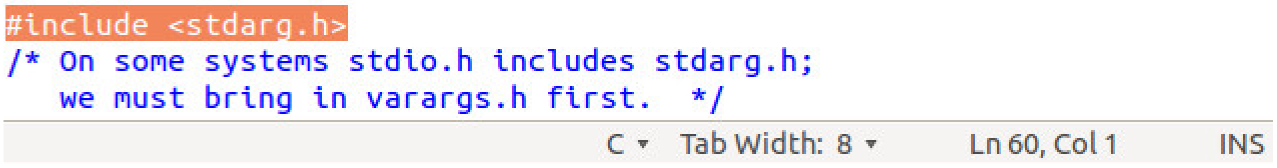
- Next, you need to do some changes in the code:

  1) Appending "-I/usr/include" at the end of the 130 line of the **Makefile**



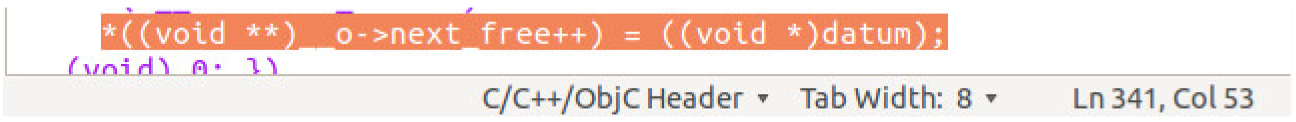
```
GCC_CFLAGS=$(INTERNAL_CFLAGS) $(X_CFLAGS) $(T_CFLAGS) $(CFLAGS) -I./
include -I/usr/include
```

  2) Replace <varargs.h> with <stdarg.h> in the 60 line of **protoize.c**



```
#include <stdarg.h>
/* On some systems stdio.h includes stdarg.h;
   we must bring in varargs.h first.  */
```

3 In the 341 line of **obstack.h**, change
*((void **)__o->next_free)++ to be *((void **)__o->next_free++)



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

C/C++/ObjC Header ▾ Tab Width: 8 ▾ Ln 341, Col 53

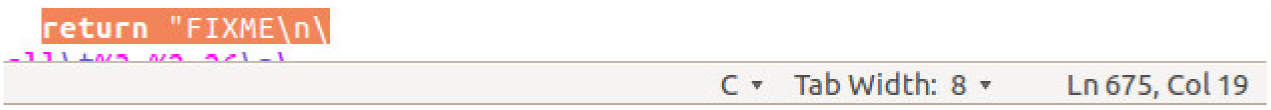
4 Now copy the following files as shown:

```
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/
```

5 Then build it

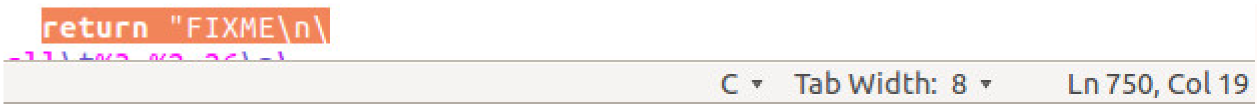
```
make LANGUAGES="c c++" CFLAGS="-O" CC="gcc"
```

Then you will come across with errors regarding to **insn-output.c**, so we fix it by appending "\" at the end of the 675, 750 and 823 line of the file insn-output.c. (i.e., Change them from `return "FIXME\n` to `return "FIXME\n\`)



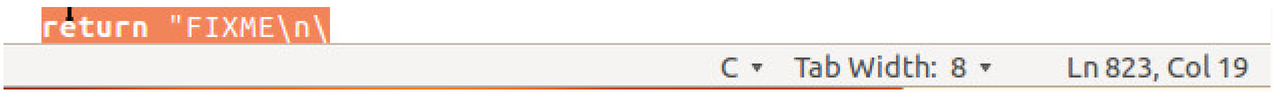
```
return "FIXME\n\
```

C ▾ Tab Width: 8 ▾ Ln 675, Col 19



```
return "FIXME\n\
```

C ▾ Tab Width: 8 ▾ Ln 750, Col 19



```
return "FIXME\n\
```

C ▾ Tab Width: 8 ▾ Ln 823, Col 19

Then make again by:

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

Then you will come across something like
"*** buffer overflow detected ***: /home/USER_NAME/simplescalar/sslittle-na-sstrix/bin/ar terminated"
At this point you might encounter a "buffer overflow" if you use Ubuntu 8.10 or later. If so,download the following files and put them in $IDIR/sslittle-na-sstrix/bin:

- http://www.ict.kth.se/courses/IS2202/ar
- http://www.ict.kth.se/courses/IS2202/ranlib

(Then do,

cd $IDIR/sslittle-na-sstrix/bin/
chmod +x ar ranlib)

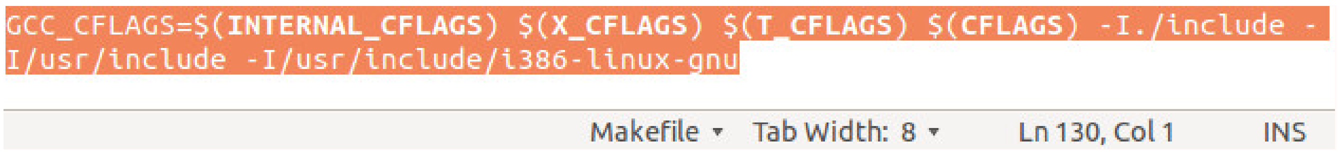
Now go back to your gcc-2.7.2.3 directory, after doing so and you can make once again

make LANGUAGES="c c++" CFLAGS="-O" CC="gcc "

Then you may come across something like

```
"In file included from /usr/include/features.h:389,
                from /usr/include/limits.h:27,
                from include/limits.h:112,
                from include/syslimits.h:7,
                from include/limits.h:11,
                from ./libgcc2.c:1121:
/usr/include/gnu/stubs.h:7: gnu/stubs-32.h: No such file or directory"
```

It can be solved by appending "-I/usr/include/i386-linux-gnu" at the end of the **line 130 of Makefile.**
(So now the 130 line of Makefile becomes:
GCC_CFLAGS=$(INTERNAL_CFLAGS) $(X_CFLAGS) $(T_CFLAGS) $(CFLAGS) -I./include -I/usr/include -I/usr/include/i386-linux-gnu)



```
GCC_CFLAGS=$(INTERNAL_CFLAGS) $(X_CFLAGS) $(T_CFLAGS) $(CFLAGS) -I./include -I/usr/include -I/usr/include/i386-linux-gnu
```

Then you make again:

make LANGUAGES="c c++" CFLAGS="-O" CC="gcc"

And still there is error regarding to the generated file cxxmain.c. Now edit the generated cxxmain.c file and remove line number 2978-2979

i.e. Remove
*char * malloc ()*
*char * realloc ()*

Then you make again:

make LANGUAGES="c c++" CFLAGS="-O" CC="gcc"

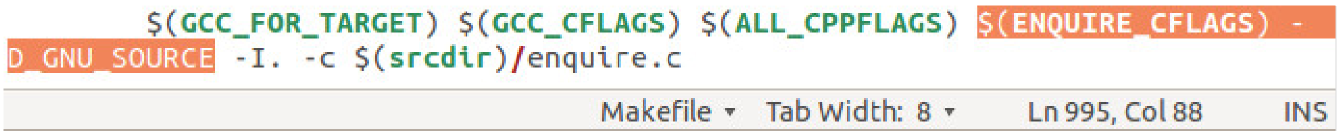
There should be no error this time.

- Make enquire

Also under the "$IDIR/simplescalar/gcc-2.7.2.3" directory, do

make enquire

Then it will pop out some error as: "undefined reference to `__isoc99_sscanf", which could be solved by adding "-D_GNU_SOURCE" after the "$(ENQUIRE_CFLAGS)" in the line 995 of the **Makefile** under the directory (namely $IDIR/simplescalar/gcc-2.7.2.3). Then make again



```
	$(GCC_FOR_TARGET) $(GCC_CFLAGS) $(ALL_CPPFLAGS) $(ENQUIRE_CFLAGS) -
D_GNU_SOURCE -I. -c $(srcdir)/enquire.c
```

make enquire

- **Finally we are done!**

Finally we follow the rest of those old references to install the cross compiler:

../simplesim-3.0/sim-safe ./enquire -f > float.h-cross
make LANGUAGES="c c++" CFLAGS="-O" CC="gcc" install

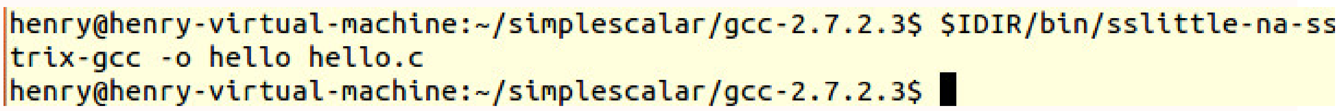
- Hello World!

We can check it with a hello.c program. Use 'gedit' to write the following program called hello.c:

```
#include <stdio.h>
main() {
  printf("Hello World!\n");
}
```

Then compile it with the cross compiler:

$IDIR/bin/sslittle-na-sstrix-gcc –o hello hello.c

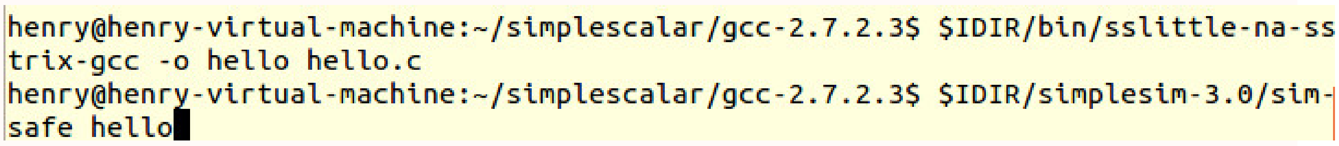


```
henry@henry-virtual-machine:~/simplescalar/gcc-2.7.2.3$ $IDIR/bin/sslittle-na-ss
trix-gcc -o hello hello.c
henry@henry-virtual-machine:~/simplescalar/gcc-2.7.2.3$
```

(If you fail to run this, try entering the above "bin" directory and run "./sslittle-na-sstrix-gcc -o ../hello ../hello.c" assuming that the hello.c program is within the $IDIR directory. But if you still fail, maybe there is some chaos with your environment viable, restart the terminal and try it once again)

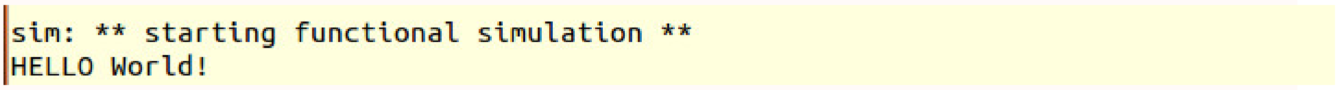
The you can run the output file "hello" with the simulator:

$IDIR/simplesim-3.0/sim-safe hello



```
henry@henry-virtual-machine:~/simplescalar/gcc-2.7.2.3$ $IDIR/bin/sslittle-na-ss
trix-gcc -o hello hello.c
henry@henry-virtual-machine:~/simplescalar/gcc-2.7.2.3$ $IDIR/simplesim-3.0/sim-
safe hello
```

If you see something like this:



```
sim: ** starting functional simulation **
HELLO World!
```

Congratulations!

For testbenches: the tests-alpha/ and tests-pisa/ folders under simplesim-3.0/ contains some benchmarks, also there are some instructor's benchmarkshere.

To remind once again, if you get stuck in any of these steps, check whether you strictly follow all the previous steps first before you ask someone else.

- Last piece of advice, add the cross-compiler bin path permanently to the PATH environmental variable by editing your ~/.bashrc as:

export PATH=$PATH:< simplescalar installation path >/bin
export PATH=$PATH:< simplescalar installation path >/simplesim-3.0

eferences:

[1] http://www.simplescalar.com/docs/install_guide_v2.txt
[2] http://www.ann.ece.ufl.edu/courses/eel5764_10fal/project/1.36445-Simplescalar-installation-instructions.pdf
[3] http://www.ann.ece.ufl.edu/courses/eel5764_10fal/project/simplescalar_installation.pdf

[4] http://gcc.gnu.org/ml/gcc/2012-02/msg00314.html

[5] http://harryscode.blogspot.hk/2008/10/installing-simplescalar.html

[6] http://lists.gnu.org/archive/html/ltib/2011-11/msg00009.html

[7] A User’s and Hacker’s Guide to the SimpleScalar Architectural Research Tool Set (for tool set release 2.0), Todd M. Austin, 1997

Also thanks for the page http://www.ann.ece.ufl.edu/courses/eel5764_10fal/project.html, although I did not reference it! It contains a lot of useful links.

The following link is a very good Lab on **how to use simplescalar**:
http://www.ecs.umass.edu/ece/koren/architecture/Simplescalar/
whose parent link:
http://www.ecs.umass.edu/ece/koren/architecture/
contains a lot of useful tools on understanding computer architecture and organization as well.