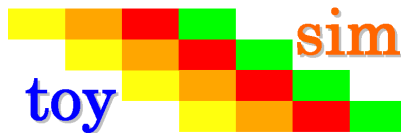


# toysim user manual



<b>Title</b>	toysim (ArchC functional simulator for the Princeton TOY machine)
<b>Author</b>	Nikolaos Kavvadias 2010, 2011, 2012, 2013, 2014, 2015, 2016
<b>Contact</b>	<a href="mailto:nikos@nkavvadias.com">nikos@nkavvadias.com</a>
<b>Website</b>	<a href="http://www.nkavvadias.com">http://www.nkavvadias.com</a>
<b>Release Date</b>	11 August 2016
<b>Version</b>	0.0.5
<b>Rev. history</b>	
<b>v0.0.5</b>	2016-08-11 Update for 2016.
<b>v0.0.4</b>	2014-12-02 Added project logo in README.
<b>v0.0.3</b>	2014-11-02 Documentation corrections.
<b>v0.0.2</b>	2014-10-30 Project cleaned-up and updated for Github.
<b>v0.0.1</b>	2010-12-11 First public version.

## 1. Introduction

This is the ArchC (<http://www.archc.org>) functional simulator model for the Princeton TOY processor. The Princeton TOY machine is a 16-bit educational RISC processor with only two orthogonal encodings. A description of the basic ISA is available in the form of the [TOY reference card](#).

This model has the system call emulation functions implemented, so it is a good idea to turn on the ABI option. It should be noted that this capability is currently untested.

## 2. File listing

The `toysim` distribution includes the following files:

<code>/toysim</code>	Top-level directory
<code>AUTHORS</code>	List of <code>toysim</code> authors.
<code>LICENSE</code>	The modified BSD license governs <code>toysim</code> .
<code>README.html</code>	HTML version of README.
<code>README.pdf</code>	PDF version of README.
<code>README.rst</code>	This file.
<code>VERSION</code>	Current version of the project sources.
<code>defines_gdb</code>	Macro definitions for GDB integration.
<code>modifiers</code>	Instruction encoding and decoding modifiers.
<code>rst2docs.sh</code>	Bash script for generating the HTML and PDF versions of the documentation (README).
<code>run_tools.sh</code>	Script for automating the build of the simulator and the associated binary utilities ( <code>binutils</code> ) port.
<code>toy.ac</code>	Register, memory and cache model for TOY.
<code>toy_gdb_funcs.cpp</code>	GDB support for the TOY simulator.
<code>toy_isa.ac</code>	Instruction encodings and assembly formats.
<code>toy_isa.cpp</code>	Instruction behaviors.
<code>toy_syscall.cpp</code>	OS call emulation support for TOY (untested).
<code>toysim.png</code>	PNG image for the <code>toysim</code> project logo.
<code>/tests</code>	Tests subdirectory
<code>run-tests.sh</code>	Run a selected benchmark.
<code>/fibo</code>	Fibonacci series benchmark directory
<code>Makefile</code>	Makefile for building the benchmark.
<code>ac_start.s</code>	Startup file (prior <code>main()</code> ) for TOY.
<code>fibo.asm</code>	Fibonacci benchmark using the alternative Princeton TOY assembly syntax (defined for the ArchC model).
<code>/popcount</code>	Population count benchmark directory
<code>Makefile</code>	Makefile for building the benchmark.
<code>popcount.asm</code>	Population count benchmark using the original assembly syntax (needs to be converted).

## 3. Usage

To generate the interpreted simulator, the `acsim` executable is ran:

```
$ acsim toy.ac [-g -abi -gdb]          # (create the simulator)
$ make -f Makefile.archc              # (compile)
$ ./toy.x --load=<file-path> [args]   # (run an application)
```

To generate the compiled application simulator, the `accsim` executable is ran:

```

$ accsim toy.ac <file-path>          # (create specialized simulator)
$ make -f Makefile.archc             # (compile)
$ ./toy.x [args]                     # (run the application)

```

The [args] are optional arguments for the application.

There are two formats recognized for application <file-path>:

- ELF binary matching ArchC specifications
- hexadecimal text file for ArchC

In order to generate the binary utilities port (binutils port), the acbingen.sh driver script must be used. This should be called as follows:

```
$ acbingen.sh -atoy -i`pwd`/../toysim-tools/ toy.ac
```

for generating the binutils port executables. This includes the following tools:

- addr2line
- ar
- as
- c++filt
- gdb (the GDB port is also generated in the same directory)
- gdbtui
- ld
- nm
- objcopy
- objdump
- ranlib
- readelf
- size
- strings
- strip

## 4. Notes

The assembly instruction syntax followed by the ArchC-based simulator for TOY is quite different than the original syntax. The following table summarizes the differences of the two syntax variations.

Original syntax	ArchC-compatible syntax
-----------------	-------------------------

<code>R[d] &lt;- imm8</code>	<code>lda rd, imm8</code>
<code>R[d] &lt;- mem[imm8]</code>	<code>ld rd, imm8</code>
<code>R[d] -&gt; mem[imm8]</code>	<code>st rd, imm8</code>
<code>R[d] &lt;- mem[R[t]]</code>	<code>ldi rd, rt</code>
<code>mem[R[t]] &lt;- R[d]</code>	<code>sti rd, rt</code>
<code>R[d] &lt;- R[s] + R[t]</code>	<code>add rd, rs, rt</code>
<code>R[d] &lt;- R[s] - R[t]</code>	<code>sub rd, rs, rt</code>
<code>R[d] &lt;- R[s] &amp; R[t]</code>	<code>and rd, rs, rt</code>
<code>R[d] &lt;- R[s] ^ R[t]</code>	<code>xor rd, rs, rt</code>
<code>R[d] &lt;- R[s] &lt;&lt; R[t]</code>	<code>shl rd, rs, rt</code>
<code>R[d] &lt;- R[s] &gt;&gt; R[t]</code>	<code>shr rd, rs, rt</code>
<code>R[d] &lt;- pc; pc &lt;- imm8</code>	<code>jal rd, imm8</code>
<code>pc &lt;- R[d]</code>	<code>jr rd</code>
<code>if (R[d] == 0) pc &lt;- imm8</code>	<code>jz rd, imm8</code>
<code>if (R[d] &gt; 0) pc &lt;- imm8</code>	<code>jp rd, imm8</code>
<code>pc &lt;- pc</code>	<code>halt</code>

Supported pseudo-instructions include:

- `nop` (no operation)
- `move` (move register)
- `neg` (negate)
- `li` (load immediate)
- `la` (load address)

## 5. Prerequisites

- ArchC installation (tested on Cygwin/Win7-64bit and Linux)
- Standard UNIX-based tools: `make`, `gcc`.