Blitz-64:

Getting Started

Download Instructions

Harry H. Porter III

HHPorter3@gmail.com

14 December 2023

System Requirements

Apple MacOS with **Xcode**.

Xcode is Apple's Integrated Development Environment (IDE) for macOS. It is free, unless you wish to upload to the App Store.

All Blitz-64 development is done with traditional **Unix command-line tools in a shell**. This can be done within the **Terminal** app on the Mac.

Xcode is not used. Xcode is only required because it includes a number of Unix tools, such as the C compiler.

As for editing, I prefer Apple's **TextEdit**, but you can used whatever you prefer.

I have not yet attempted to run Blitz-64 on **Linux** or **PCs**.

Step 1: Download the File

Go to **blitz64.org** and download the file **blitz.tar.gz**.

Step 2: Uncompress the Tarball

In MacOS, clicking on the file **blitz.tar.gz** will create a directory (i.e., folder) called **blitz**. You could use the following commands instead. (The "**%**" stands for your shell prompt.)

```
% gunzip blitz.tar.gz
% tar -xvf blitz.tar
```

Put the **blitz** folder wherever you want it. Move **blitz.tar** to the trash.

Directory Organization

Here is the directory structure of the files you just created:

```
blitz/
   tools/
      src/
         makefile
         ...C source code for emulator, assembler, linker, etc...
         kpl/
            ...C++ source code for the KPL compiler...
      bin/
         ...executables for all C/C++ tools...
      testing/
         ...files used for regression testing the tools...
   pgms/
      makefile
      ...Various Blitz example programs...
      asm2/
         ...files for the KPL version of the assembler...
      kpl2/
         ...files for the KPL version of the KPL compiler...
```

Step 3: Compile the Tools

Execute the following commands:

```
% cd blitz/tools/src
% make
```

This should invoke the C and C++ compiler about 20 times, printing a line for each. Hopefully, there are no error messages printed, but if there are… well… don't you hate that?!?!?

Step 4: Modify Your PATH Variable

Modify your search path, so that when you type in the name of a Blitz tool on the command line, your shell will find and execute the program. Use this to view your current search path:

```
% echo $PATH
```

You'll want to change a start up file, so you'll get the modified path every time you login. How you do this will depend on which shell you use.

For **bash**, update your shell's **rc** file (e.g. **~/.bashrc**) to add the appropriate directory to **$PATH**.

```
PATH=/Users/harry/blitz/tools/bin:$PATH
```

For **csh**, something like this might work:

```
set path = ( /Users/harry/blitz/tools/bin $path )
```

Put the Blitz tool directory first so the Blitz tools will be used instead of similarly named Unix tools.[1]

[1] Two of the Blitz tools have name identical to Unix tools: **hexdump** and **link**. You'll want to get the Blitz versions. I prefer the **hexdump** I wrote to the Unix **hexdump** command. The Unix command **link** is a version of the **ln** command; if you want the Unix versions, you can type **/bin/link** or **/usr/bin/hexdump**.

Step 5: Verify the Tools Work

Type **blitz** on the command line. You should see the Blitz-64 emulator's welcome message:

```
% blitz
=============================================
=====                                   =====
=====  The Blitz-64 Machine Emulator    =====
=====     by Harry H. Porter III        =====
=====        15 October 2022            =====
=====                                   =====
=============================================

Enter a command at the prompt.  Type 'quit' to exit or 'help' for
info about commands.
E>
```

If you get "command not found", then check that your path was set correctly.

Type "q" to quit.

Invoke the KPL compiler with the command **kpl** and nothing else. You should see this:

```
% kpl
*****  FATAL ERROR: Output file name is required if input comes
from stdin; use "-o outputFile".  *****
%
```

Step 6: Compile the Example Blitz Programs

Execute the following commands:

```
% cd blitz/pgms
% make
```

You should see a bunch of invocations of **kpl**, **asm**, and **link** to compile the following programs:

```
boot0
ISAValidator
Hello
hexdump
ed0
chess
CacheTest
ListTester
ListTester23
BitMap
Quicksort
TicTacToe
Sudoku
ExamGrader
RelaySim
NumberTest
asm2
kpl2
```

Step 7: Run the Hello Program

Type the following command:

```
% blitz -g -nowarn Hello.exe
```

You should see:

```
Hello, world
i = 123
```

Step 8: Experiment with Errors and Debugging

Use your text editor to look at the following files:

```
blitz/pgms/Hello.h
blitz/pgms/Hello.c
```

Try modifications and use **make** to recompile.

Insert the following code, and run the program.

```
while true
  printIntVar ("i = ", i)
  i = i + 1
endWhile
```

- *What is the effect of hitting Control-C?*
- *What is the effect of hitting Control-C several times?*

Uncomment the following line (by removing "--") and rerun the program.

```
-- i = i + 0x7fff_ffff_ffff_ffff
```

The addition will overflow and the will be an Arithmetic Exception error. Notice that you can see with functions (and methods) are currently active and where in each function execution is stopped.

- *What line number and from what file did the error occur?*

When prompted with

```
...How many more frames would you like to see?
```

type **99**.

- *What is the current value of variable "i" at the moment of the error?*

Try the following commands:

r	Display the registers
s	Single step execution
<enter>	See the current location
h	Display list of all emulator commands
q	Quit the emulator

Uncomment the following line and rerun the program.

```
-- debug "point 1 reached"
```

Try the following commands:

`stack`	Display the call stack
`g`	Resume execution

• *What is the current value of the Program Counter (PC)?*

Try the following command:

`dis`	Disassemble instructions

Type in the current value of the PC. To see more instructions, use this command

`d`	Continue disassembly from this point

Try the following command. Enter the value of the PC.

`dm`	Display memory contents

If the PC is not doubleword aligned, round the last digit down to **`0`** or **`8`**. For example:

`005f6b2ec → 005f6b2e8`

• *What form is memory displayed in, besides hex?*

Try the following commands:

`globals`	Display memory contents
`hex`	Convert hex values to other forms
`dec`	Convert decimal values to other forms
`ascii`	Convert ascii characters to numeric forms

• *What decimal value of 0x1234?*
• *What is the ASCII character for decimal 107?*
• *What is the ASCII hex code for 'a'?*

Try the following command:

`parm`	Display emulation constants
`cores`	Display the status of all cores

- *What is the size of the main memory?*
- *How many cores exist?*

Step 9: Explore Other Programs

The following example programs are included in the download and you can run and modify them. They are located in **blitz/pgms** and will be compiled by the **makefile** in that directory.

`TicTacToe`	*A simple program to demonstrate the preferred KPL coding style; Also demonstrates input functions*
`Quicksort`	*Sorts a small array, which is hardcoded into the program*
`Sudoku`	*Solves a puzzle; Several puzzles are hardcoded in the code; You can edit and recompile to change puzzles*
`ed0`	*A simple text editor; It mimics a screen-based editor using only line-based output*
`hexdump`	*A KPL translation of the Blitz hexdump tool in the C language*
`RelaySim`	*A circuit simulator designed to model a computer I built out of relays; It can also generate a 64x64 multiply unit out with more relays than anyone has time to solder*
`CacheTest`	*Program to generate test vectors for a Verilog cache circuit*
`ListTester`	*Menu-based program to test and exercise the List package*
`ListTester23`	*Menu-based program to test and exercise the List2 and List3 packages*
`RBTester`	*Menu-based program to test and exercise the red-black tree code, which is in the RBTree package*
`BitMap`	*Code for a BitMap class, along with code to test and exercise the functionality*
`NumberTest`	*Program to exhaustively test the code in the Number package*
`chess`	*The min-max algorithm applied to chess; The level of play is not particularly good; This program could use more attention*

- *Can you beat the **TicTacToe** program? Is the code understandable without having to learn KPL first?*
- *Does the **ed0** editor program work for you? Is it intuitive and usable?*

ISAValidator.s is an assembly language program that runs stand-alone. This program is used to verify that a Blitz-64 core is functioning correctly.

When it is executed using the **blitz** emulator, you should see this:

```
% blitz ISAValidator.exe -g -nowarn -fp

****  A DEBUG machine instruction was executed  *****

Next instruction to execute:
   COMMENT (line 0)
   ----------

====================
====================  SUCCESS: ALL TESTS WERE PASSED
====================

      00000c9d4: 00340000       debug

Entering machine-level debugger...
=========================================
=====                               =====
=====  The Blitz-64 Machine Emulator  =====
=====     by Harry H. Porter III      =====
=====        15 October 2022          =====
=====                               =====
=========================================

Enter a command at the prompt.  Type 'quit' to exit or 'help' for info
                  about commands.
E>
```

Note the word “SUCCESS”. But if you see the following, then something about the emulator is broken:

```
-->-->-->-->-->-->-->
-->-->-->-->-->-->-->   FAILURE: Check reg s0 to see which test failed
-->-->-->-->-->-->-->
```

The **ISAValidator.s** program is useful for anyone designing a new implementation of the Blitz-64 core. The program is over 13,000 lines of assembler and quite tricky in places, but getting to “success” exercises the core’s functionality and requires the core to conform to the ISA specification.

Step 10: Explore the Documentation

There is additional documentation at **blitz64.org**.