

LCC's PILI Target

by Markus Weber
101530.10@compuserve.com

1. About LCC

LCC is a retargetable ANSI C compiler. For further information (including the source code), see <http://www.cs.princeton.edu/software/lcc/index.html>

If you plan to work with the sources, buy the textbook describing it:

Christopher W. Fraser & David R. Hanson
A Retargetable C Compiler: Design and Implementation
The Benjamin/Cummings Publishing Company, Inc.
ISBN 0-8053-1670-1

2. Packing List

The archive should include the following files:

<code>cpp.exe</code>	C pre-processor
<code>lcc.exe</code>	the compiler proper
<code>lcc.inc</code>	VASM include file needed by lcc
<code>lcc.doc</code>	Microsoft Word 95 documentation
<code>CPYRIGHT</code>	LCC's copyright

Please note: The executables are compiled as WIN32 console applications, i.e. they should run on Windows 95/NT systems only. Native DOS and OS/2 versions are planned, but not yet available.

3. Usage

I don't have a compiler front end—yet. I've included the `cpp` and `lcc` executables, which you'll have to run manually for the time being. Command line options include:

`cpp`

<code>-N</code>	Ignore standard include directories
<code>-Iincludedir</code>	Add include directory
<code>-Dmacro [=def]</code>	Predefine macro, value is optional
<code>-Umacro</code>	Undefine macro

`lcc`

<code>-A</code>	“fussy” option, may be given more than once
<code>-d</code>	debugging output
<code>-P</code>	print ANSI-style function prototypes
<code>-target=name</code> <code>symbolic.</code>	Select target architecture. Interesting targets include <code>pili</code> and <code>symbolic</code> .

4. Machine Model

I'm using a subset of PILI's native registers to create an abstract machine. In particular, I'm using 32 general purpose registers, r0 to r31. Some of these registers are dedicated to special functions:

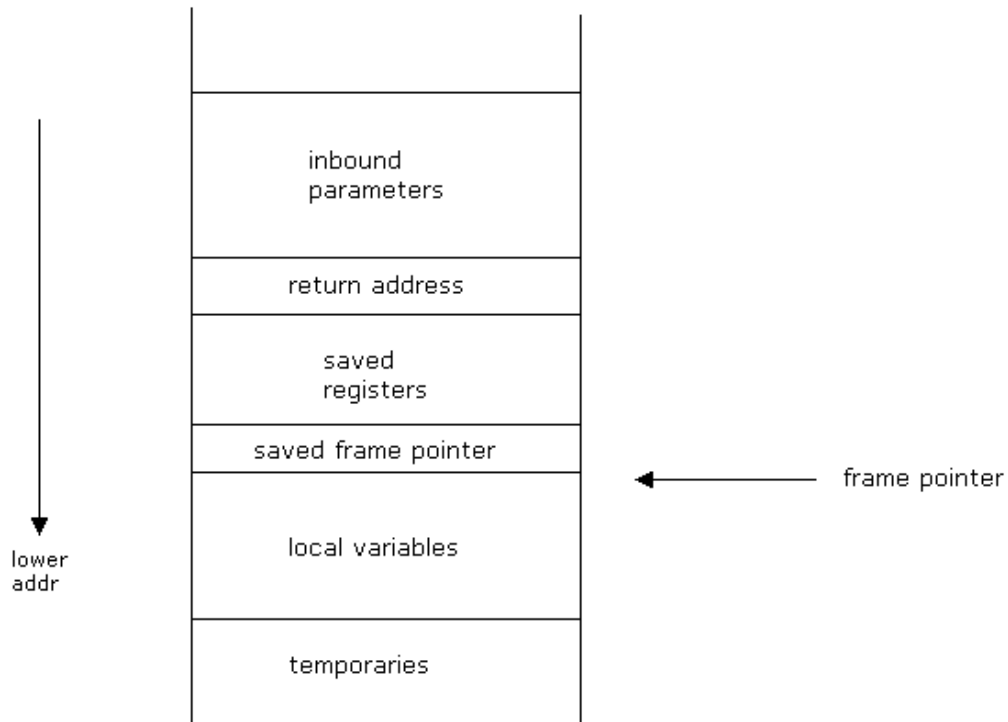
r0	integral function return value
r1 .. r7	reserved for parameter passing
r8 .. r15	register variables
r16 .. r30	scratch register
r31	frame pointer

Given that floats are limited to 32 bits, I'd like to share integral and floating point registers. However, LCC doesn't allow that. I've added 16 floating point registers, f0 to f15. The compiler uses these registers as follows:

f0	double function return value
f1 .. f3	reserved for parameter passing
f4 .. f7	scratch registers
f8 .. f15	register variables

Please note: The register model presented here is under revision. I'm even considering to allow several compile-time register models.

5. Calling Conventions



The figure above illustrates the stack frame used by LCC. Here's a timeline of the implementation of function calls:

Caller

```
; push function args on the stack
PUSH argn
...
PUSH arg1
; transfer control
CALL function
```

Callee

```
; save registers
PUSH r8
...
PUSH r15
; save frame pointer
PUSH fp
; set up new frame pointer
PUSHSP
POP fp
; function code
...
; restore frame pointer
POP fp
; restore registers
POP r15
...
POP r8
; return control
RET
```

```
; clear stack
RLD32 lcctmp, paramsize
ADDSP lcctmp
```

The code fragments in the table above aren't complete. The calling conventions are under review.

6. Bugs

What definitely won't work?

- Function calls are currently completely broken, due to my misreading of the PILI assembler syntax. In particular, the generated opcodes and arguments to CALL are in error, and the stack isn't cleaned up. Correct handling of calls requires a special handling.
- Arithmetic right shifts. There's no opcode to do it, although a workaround is possible.
- Type conversions are suspect. I'm assuming FINT returns an integral value, rather than a floating point number. If this assumption doesn't hold, we need either an FTOI opcode, or a special handler.
- **struct** arguments and return values. Need special handling.
- Conditional jumps are suspect.
- There's no guarantee that the compiled code will go past VASM, or will

7. To Do

What needs to be done? (Other than fixing the bugs above)

- Add runtime library for PILI FNCALLs.
- Add general purpose C runtime.
- Make better use of the PILI instruction set.
- Save only registers that will be clobbered by the called function.
- Pass arguments in registers.
- Improve the documentation.
- Compile lots of code samples for regression tests.

8. Help Wanted

- Bug reports, enhancement requests, and suggestions in general are very welcome. I'd like to have a copy of every piece of code that ever miscompiled.
- I need as many volunteers as possible to manually inspect (i.e. proof-read) the generated code.
- The FNCALL runtime library shouldn't be hard to write.
- A peephole optimizer might smooth over some of LCC code generation deficiencies.

9. LCC Copyright

The authors of this software are Christopher W. Fraser and David R. Hanson.

Copyright (c) 1991,1992,1993,1994,1995 by AT&T, Christopher W. Fraser, and David R. Hanson. All Rights Reserved.

Permission to use, copy, modify, and distribute this software for any purpose, subject to the provisions described below, without fee is hereby granted, provided that this entire notice is included in all copies of any software that is or includes a copy or modification of this software and in all copies of the supporting documentation for such software.

THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

lcc is not public-domain software, shareware, and it is not protected by a 'copyleft' agreement, like the code from the Free Software Foundation.

lcc is available free for your personal research and instructional use under the 'fair use' provisions of the copyright law. You may, however, redistribute lcc in whole or in part provided you acknowledge its source and include this COPYRIGHT file. You may, for example, include the distribution in a CDROM of free software, provided you charge only for the media, or mirror the distribution files at your site.

You may not sell lcc or any product derived from it in which it is a significant part of the value of the product. Using the lcc front end to build a C syntax checker is an example of this kind of product.

You may use parts of lcc in products as long as you charge for only those components that are entirely your own and you acknowledge the use of lcc clearly in all product documentation and distribution media. You must state clearly that your product uses or is based on parts of lcc and that lcc is available free of charge. You must also request that bug reports on your product be reported to you. Using the lcc front end to build a C compiler for the Motorola 88000 chip and charging for and distributing only the 88000 code generator is an example of this kind of product.

Using parts of lcc in other products is more problematic. For example, using parts of lcc in a C++ compiler could save substantial time and effort and therefore contribute significantly to the profitability of the product. This kind of use, or any use where others stand to make a profit from what is primarily our work, requires a license agreement with Addison-Wesley. Per-copy and unlimited use licenses are available; for more information, contact

J. Carter Shanklin
Addison Wesley Longman, Inc.
2725 Sand Hill Rd.
Menlo Park, CA 94025
415/854-0300 x2478 FAX: 415/614-2930 jcs@aw.com

Chris Fraser / cwfraser@microsoft.com
David Hanson / drh@cs.princeton.edu
\$Revision: 1.3 \$ \$Date: 1996/09/30 13:55:00 \$