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:

`cpp.exe`	C pre-processor
`lcc.exe`	the compiler proper
`lcc.inc`	VASM include file needed by lcc
`lcc.doc`	Microsoft Word 95 documentation
`CPYRIGHT`	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`

`-N`	Ignore standard include directories
`-I`*`includedir`*	Add include directory
`-D`*`macro`*`[=`*`def`*`]`	Predefine macro, value is optional
`-U`*`macro`*	Undefine macro

`lcc`

`-A`	"fussy" option, may be given more than once
`-d`	debugging output
`-P`	print ANSI-style function prototypes
`-target=`*`name`*	Select target architecture. Interesting targets include **`pili`** and **`symbolic`**.

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:

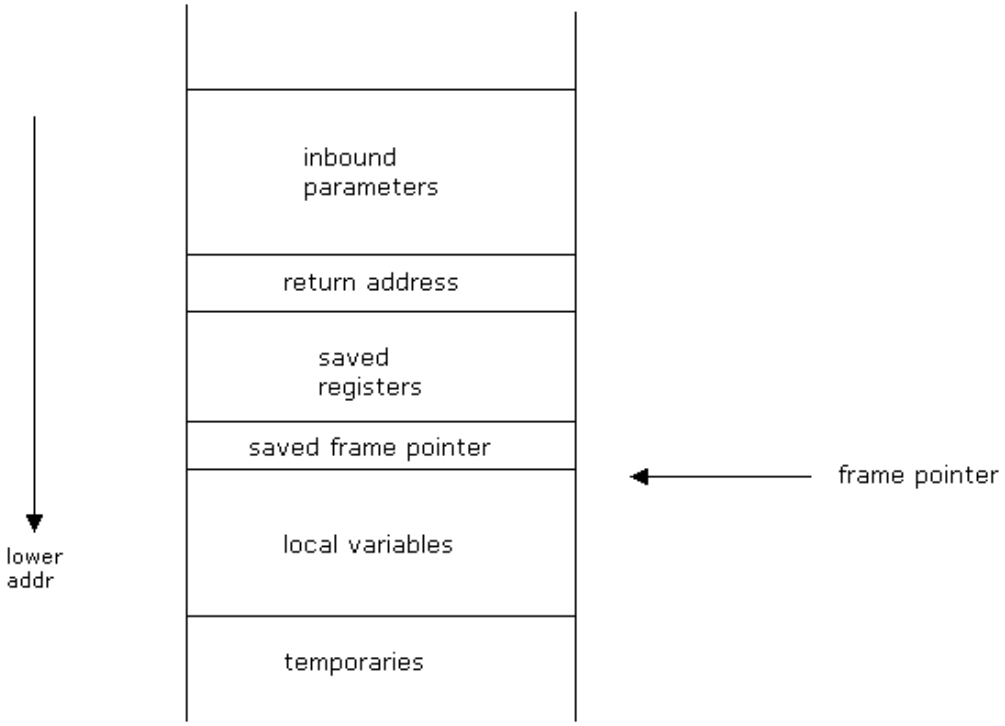
`r0`	integral function return value
`r1` .. `r7`	reserved for parameter passing
`r8` .. `r15`	register variables
`r16` .. `r30`	scratch register
`r31`	frame pointer

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 arg`$_n$
…
`PUSH arg`$_1$
; 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.