Table of Contents

Overview....2
Setup....3
General Usage....4
- Assembly Files....4
- As2obj....4
- Lc3runner....5
- Comp....5
- Complx....6
  - Command line parameters....6
  - Getting Started....7
  - Interface tour....8
  - Running Programs....10
  - Debugging....12
  - Testing....13
  - Other features of complx....14
    - Console Input....14
    - Memory View....14
    - True Traps Mode....14
    - Interrupts Mode....14
    - Call Stack Viewer....14
    - Expressions....15
- Assembly Files Extended....16
  - Debugging Comments....16
  - Subroutine Annotations....18
  - Plugins....19
- Plugin Interface....20

Credits....21

Overview

complx-tools is a suite of tools for learning lc3 assembly. It includes both a gui and cli based simulator (named complx and comp respectively), an assembler (as2obj), a very simple program that runs lc3 assembly files and spits out whats printed to the console (lc3runner), and a framework for testing lc3 assembly code (lc3test). Complx-tools also be extended with plugins that add additional functionality to the LC3. The tools also come with a C++ interface to the LC3 (liblc3).

These tools are mainly used in CS2110 at Georgia Tech.

Setup

Manual steps to compile the program are as follows

From the root directory of where you have the source code...

1. Ensure you have a C++ compiler (sudo apt-get install build-essential)
2. Install dependency wxWidgets 3.0 (sudo apt-get install libwxgtk3.0-dev)
3. Build the program (make)
4. Install the program (sudo make install)
5. Run ldconfig (sudo ldconfig)

If you'd rather a script to setup the program for you see the following files

To use this make sure you are in the root directory of the source code...

1. Make sure the script is executable (chmod u+x *filename.sh*)
2. Run the script (sudo ./*filename.sh*)

install.sh - Performs the above operations

install_script_fedora_21.sh – Install script specifically for Fedora, thanks Thomas Coe

install_script_ubuntu_13_10.sh - Install script for Ubuntu 13.10 and below, thanks Abhijit Murthy

TODO Talk about Arch Linux here...

Compiling on Windows and Mac is not supported and untested.

General Usage

Assembly Files

An assembly file (with the .asm extension) is just a normal text document. You can create .asm files in any text editor of your choosing. I recommend gedit which should be preinstalled on your linux machine, other text editors are emacs, vim, and nano if you are into those. As a sample assembly program to get you started copy and paste this into a file named helloworld.asm

```
.orig x3000
     LEA R0, HELLOWORLD
     PUTS
     HALT
     ; Hello this is a comment
     HELLOWORLD .stringz “Hello World”
.end
```

As2obj

This program just assembles files and produces an object file (.obj) and a symbol table file (.sym). I will explain the formats of these two files later. Invoking the assembler is easy (note the []'s indicate optional parameters)

as2obj *filename.asm* [-all_errors] [-disable_plugins] [outfile]

an explanation of the command line parameters

- -all_errors Report all errors encountered by the assembler if possible
- -disable_plugins Disable use of all lc3 plugins
- outfile output filename

Lc3runner

This program is a simple command line program that just runs your assembly program until it halts spitting out any output that is written to the console. Running this program is very simple, but it doesn't have many options you can't view the contents of any address unless your program prints it out.

lc3runner filename.asm [-zeroed] [-input=some_text_file] [-count=num]

an explanation of the parameters

- -zeroed if passed in will zero out all memory if not passed in then all memory is randomized.
- -input=some_text_file use this file as stdin any reads from keyboard will read from the file instead
- -count=num from forr num instructions the default is run until HALT is encountered

Comp

Explain this once its finalized

Complx

This program is the main program you will probably want to run. It is an lc-3 simulator, that allows you to play through instructions, undo instructions, and modify the state of the lc3 while a program is running. Also supports a plethora of debugging tools. To start the program you can either find it in the “Start Menu” (Should be under programming), or start it through the terminal,

complx filename.asm

Command line parameters

The command line parameters as shown below are all optional.

- --unsigned=bool Sets if decimal representations are displayed in unsigned (default 0)
- --disassemble=num Sets the disassemble level 0: basic 1: normal 2: high level (default 1)
- --stack_size=num Sets the Undo stack size (default 65536 instructions)
- --call_stack_size=num Sets the Call stack size (default 10000 subroutine/trap calls)
- --address=hex Sets the PC's starting address (default 0x3000)
- --true_traps=num Enable true traps (see section on True Traps) (default 0)
- --interrupts_enable=num Enable interrupts (default 0)
- --highlight=num Enable instruction highlighting (default 1)

Getting Started

As stated above you can start complx via the command line to load a file immediately by just passing in the path to the assembly file you want to load.

To load a file via the GUI you can use one of the menu options under the File menu

- Randomize and Load – Ctrl + Alt + O
- Randomize and Reload – Ctrl + Alt + R
- Load – Ctrl + O
- Reload – Ctrl + R
- Load Over – Ctrl + Shift + O
- Reload Over – Ctrl + Shift + R

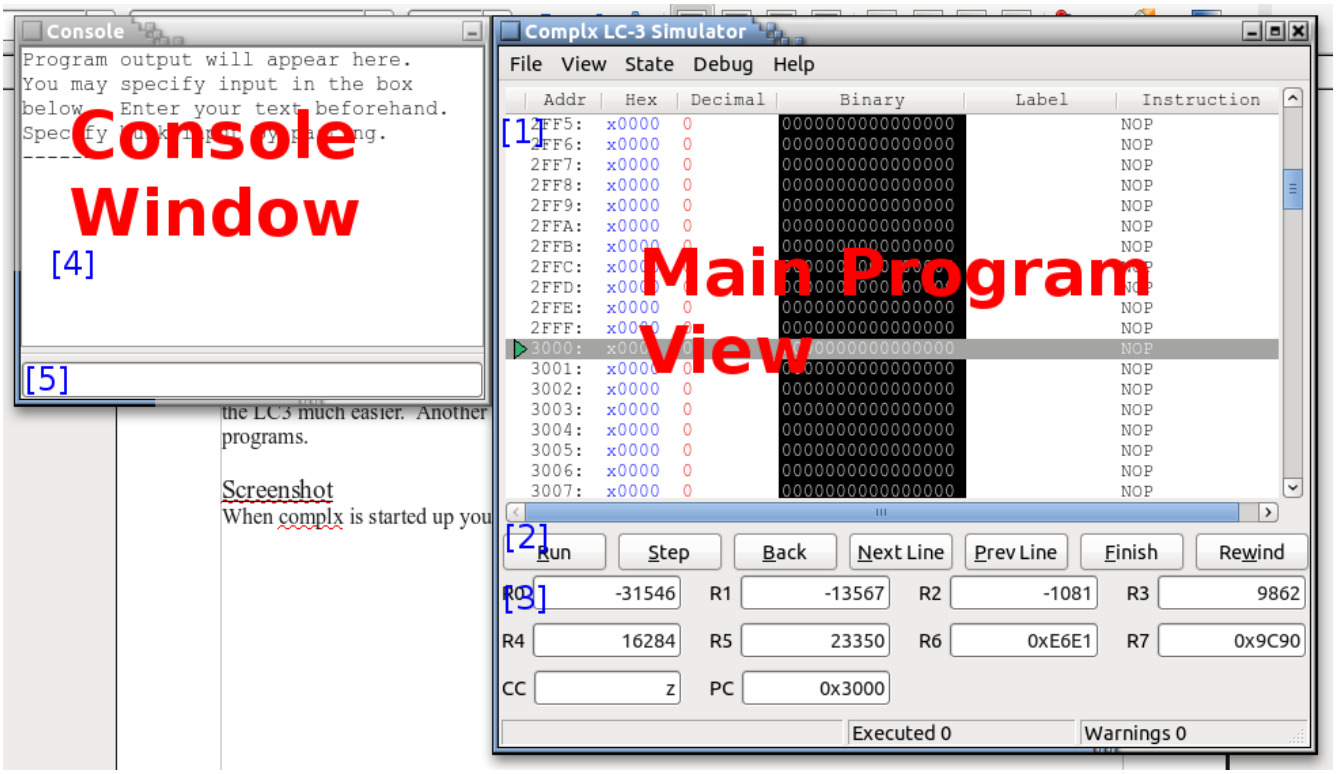
Randomize and Load does the following randomizes every address, then loads your assembly file over the randomized memory, that is, any addresses modified by your program will overwrite memory.

Load does the following cleans out the memory bring the lc3 back to the initial state, then loads your assembly file.

Load Over just loads your assembly file over the current state of the memory

The reload versions do exactly the same as above except it does not query you for a file. It will use the last file loaded (or if there isn't one it will query you for one.

Interface tour



As a brief tour of the interface

[1] Is the main memory view. This shows the value each address contains in the LC-3 interpreted in hexadecimal, decimal, binary, and as an LC-3 Instruction. Remember that all instructions can be represented as a 16 bit value.

So if I had the instruction ADD R0, R0, R0 then I should see the following:

- Hex column I should see x1000
- Decimal column I should see 4096
- Binary column I should see 0001000000000000
- Instruction column I should see ADD R0, R0, R0 (if the disassemble level is set to normal).

[2] The main control buttons. You will use these buttons to run your program (Run) or step through your program one instruction at a time (Step), you may also undo instructions using Back or rewind the entire program undoing everything in the undo stack (Rewind).

[3] The current state of all registers. You can enter values in binary, hexadecimal, or decimal. You can also change the base the registers contents is displayed in by either double clicking the text box or right clicking and selecting a new base for the register. By default R5-R7 are displayed in hexadecimal since these registers usually contain an address.

[4] Console window output will be displayed here. Also any warnings generated from your program will also be spit out here.

[5] Console window input will be typed in here. Its best to type your text before the program is ran.

Running Programs

The control buttons from the previous section should be used to run your assembly program. I will explain what each button does in this section

Step – F2 Executes exactly one instruction

Back – Shift + F2 Undoes exactly one instruction

Next Line – F3 Performs one instruction, if used on a subroutine or trap it will execute the entirety of it, that is, if used on a statement the PC shall point to the next line in the program.

Prev Line – Shift + F3 Undoes one instruction, but if backed into a subroutine or trap will undo the entirety of it, that is, if used on a statement the PC shall point to the previous line in the program

Run – F4 Runs your program until it HALTs or you stop it manually.

Run For... – Ctrl + F4 Runs for X instructions (will always query you for X)

Rewind – Shift + F4 Repeatedly undoes instructions until the undo stack is exhausted

Finish – Shift + F5 Executes enough instructions to step out of the current subroutine or trap you are in

Run Again – (only available as a menu option) Ctrl + Space Same as Run For but will not query you for X after the first time it is used

Example

This program does nothing interesting, but is designed to show where the PC will end up if you use each of these buttons. The end result is that 2 should be loaded into R0 by the time HALT is executed

```
.orig x3000
JSR FAKE_SUBR                ;x3000 [1]
HALT                         ;x3001 [2]
FAKE_SUBR ST R7, SAVE        ;x3002 [3]
AND R0, R0, 0                ;x3003
JSR ADD2                     ;x3004
LD R7, SAVE                  ;x3005 [4]
RET                          ;x3006
ADD2 ADD R0, R0, 2           ;x3007 [5]
RET                          ;x3008
SAVE .blkw 1                 ;x3009
.end
```

Note the flow of this program is the following instructions at these addresses get executed

x3000, x3002, x3003, x3004, x3007, x3008, x3005, x3006, x3001

control \ point	[1]	[2]	[3]	[4]	[5]
step	x3002	x3001	x3003	x3006	x3008
back	x3000	x3008	x3000	x3004	x3004
next line	x3001	x3001	x3003	x3006	x3008
prev line	x3000	x3000	x3000	x3004	x3004
run	x3001	x3001	x3001	x3001	x3001
finish	Error	Error	x3001	x3001	x3005

To read this table look at the instruction with the comment [#] in the header of the column the address in the cell is the result of using that operation. From the flow of the program above look where the PC ended up relative to the starting address.

For example see using Finish at point [5] in the program.

Point [5] is address x3007 and using Finish at this point will go to x3005

Therefore the instructions at x3007 and x3008 get executed to perform the Finish operation.

Debugging

So what tools does complx provide for people struggling to get a program working. At a basic level the step and back buttons along with viewing the state of the memory and registers would suffice for simple programs; however, for bigger programs stepping one instruction at a time may not be enough or too time consuming. So here are more ways to get more control over a running program.

Testing

Other features of complx

Console Input

Memory View

True Traps Mode

Interrupts Mode

Call Stack Viewer

Expressions

For any prompt requiring a memory address or a symbol you can give it an expression instead. Expressions are also used to determine when a watchpoint stops the program. The condition of a watchpoint (or breakpoint and blackbox for that matter) is an expression that gets evaluated to determine whether it temporarily stops the program. Note that if an expression evaluates to a non zero value it is considered to be true, otherwise it is false.

Here is a list of variables you can use in expressions

Registers - R0-R7

Program Counter – PC

Value in memory address – MEM[ADDR]

Any symbol (will resolve to the address where the symbol lives).

Examples

R0 * 5

MEM[x5000] – MEM[x5001]

MEM[MEM[x7000]]

MEM[R4]

PC == HELLOWORLD

MEM[PARAM] != 25

For a full list of operators that are supported

- Arithmetic (*, /, %, +, -)
- Bitwise operators &, |, ^
- Logical operators (&&, ||, !&, !|). That is logical and, or, nand, and nor.
- Equality operators ==, !=
- Shifts <<, >>
- Relational operators < > <= >=.
- Parenthesis ()

Assembly Files Extended

Debugging Comments

These are special smart comments that will automatically set up debugging breakpoints and watchpoints within any simulator in complx-tools. Note that use of these comments will not affect the testing environment in lc3-test nor any autograders. Please see the debugging section under complx for an overview of what breakpoints, watchpoints, and blackboxes are.

Breakpoints

To specify in a comment to create a breakpoint at a specified address you can use one of two ways to create it.

;@break address=address/symbol/expression name=label condition=1 times=-1

;@break address name condition times

In the first form any parameter can be omitted the default values are given after the equal sign.

In the second the parameters must be given sequentially, that is, if you want to define “times” then you must specify address, name, and condition. However, if you only want to specify the address in the second form you may omit the rest of the parameters.

Default values and expected types for the parameters are as follows:

Parameter	Type	Default
address	Expression, Address, or Symbol	The address below where the comment is placed.
name	String	An empty string
condition	Integer	1 (always break when breakpoint is encountered)
times	Integer	-1 (the breakpoint will never expire)

Watchpoints

To specify a watchpoint as a comment the syntax is very similar to that of a breakpoint.

;@watch target=address condition="0" name=label times=-1

;@watch target condition name times

The parameter condition is required omitting it will cause the watchpoint to never trigger

Default values and expected types for the parameters are as follows:

Parameter	Type	Default
target	Address, Symbol, or Register	The address below where the comment is placed.
condition	Integer	0 (Watchpoint will never trigger)
name	String	An empty string
times	Integer	-1 (the watchpoint will never expire)

Blackboxes

And to specify blackboxes the syntax is again similar.

;@blackbox address=address name=label condition=1

;@blackbox address name condition

Default values and expected types for the parameters are as follows:

Parameter	Type	Default
target	Address or Symbol	The address below where the comment is placed.
name	String	An empty string
condition	Integer	1 (You will always skip over the blackbox)

Subroutine Annotations

You can give the simulator more information about your subroutines (that follow the lc3 calling convention) and this will improve the output of the view call stack function in complx.

The syntax for this is as follows:

;@subroutine address=address name=label num_params=0

;@subroutine address name num_params

Default values and expected types for the parameters are as follows:

Parameter	Type	Default
address	Address or Symbol	The address below where the comment is placed.
name	String	An empty string
num_params	Integer	0 (Subroutine takes no parameters)

Plugins

And lastly you can extend the simulator and assembler through use of plugins. With plugins you may add new devices, traps, and a new instruction for the LC-3.

To include a plugin with your assembly file the following syntax must be used

;@plugin filename=??? vector=??? address=??? interrupt=???

The arguments vary by plugin but for a minimum the filename must be given. For plugins introducing new traps vector must be specified, it will be the entry in the trap vector table where the trap lives. For plugins introducing devices address must be specified. This will be the address where the device register lives. If the plugin generates interrupts then the interrupt parameter must be specified.

Parameter	Type	Description
filename	Filename	Must be specified. Name of file without extension and without lib in the name. So a file named liblc3_udiv.so will be specified here as lc3_udiv
vector	Address	Address to install Trap plugin into trap vector table.
address	Address	Address where Device Register plugin will live.
interrupt	Address	If plugin generates interrupts the interrupt vector that is sent

Plugin Interface

Credits