Lec-1-tools-1

COSC-120-01, Computer Hardware Fundamentals

What we will do (sort of)

---- build a CPU
---- program at the hardware level
---- understand abstraction/models
---- acquire some math tools
---- learn low-level design tools
---- use logic gates
---- learn to handle complexity by using
 layers of abstraction (hierarchy)
---- become familiar w/ a particular
 CPU micro-architecture (LC3)
---- understand basis of current hardware
 and where industry is going
---- think and program in parallel, using
 hardware simulation language
---- acquire familiarity w/ some unix tools

Typical commandline tools we need.

vi / emacs: editors
make / sh: shell commands, build dependencies
grep: pattern matching in files
sed: stream editing
awk: stream editing w/ more complexity
m4/cpp: pre-processors

typical unix commandline stuff:

```
man, info
ls
pwd
cd
rm
mv
cp
exit

mkdir
rmdir
alias
set, which, whereis
jobs, ctl-z, fg, %2, &
ps -ex, kill -6 (-9)
echo
cat
>
>>
|
<
gzip, gunzip, compress, uncompress (.z)
tar
```

Things unix

shell, login shell, processes, child processes, inherited properties, environment variables, open files, stdin, stdout

```
%> set              #--- see all environment variables
%> echo $PATH  #--- see the PATH variables content (a string w/ ":" separators for sub-strings)

%> cd                                    #--- your home directory in unix/cygwin environments
%> vi .bash_profile
      export VISUAL="vi"          #--- needed for "svn ci" to edit log comments

%> ls                       #--- see files in current directory
%> cd foo; cd ..         #--- move in file system tree
%> mkdir; rmdir        #--- add/subtract sub-tree
%> rm                     #--- remove file, forever
%> pwd                  #--- see shell's idea of current position in file system
%> exit                    #--- kills current shell, returns to parent process
%> tar -xvf foo.tar      #--- unpack a tree
%> gunzip foo.tar.z   #--- uncompress a packed tree or file
%> man ls                 #--- see how to use the "ls" program
%> info ls                #--- also see "ls" usage (more complete?)
%> alias l "ls -F"       #--- make shorthand for a command
%> ps -ex              #--- see all running/sleeping processes
%> kill -9 12345    #--- send a signal to process 12345 that kills it
%> jobs               #-- see current jobs that are asleep
%> ^z             #--- put current jobs to sleep (e.g., current editing session), return to parent process
%> fg             #--- wake up most recently slept process
%> %2            #-- wake up job 2
%> cat foo        #--- dump file content to stdout
%> cat foo bar > foobar  #---send content to file "foobar"
%> cat foo bar | grep "who"  #--- send content to grep via stdin for subprocess
%> less foobar   #--- see content a screenfull at a time
%> more foobar  #--- ditto
%> make target   #--- read Makefile, find target, execute shell commands
%> cat foobar | sed 's/Hi/hi/'   #--- stream editing, by lines
%> awk                                 #-- more stream editing, by fields per line
%> m4           #--- input stream macro expansion
%> cpp           #--- input stream macro expansion, part of C compiler (ala "#define", e.g.)
%> svn co https://svn.cs..../svn/projects2/120-2011/CourseDocuments #-- get working copy
%> svn add  foo #--- mark file or directory "foo" to be added to repository
%> svn ci   #--- send changes (additions, deletions, edits) to repository
%> svn up #--- get updates/changes download from repository to working copy
%> svn status  #--- see state of working copy ("?" not svn-add'ed, "M" modified, "A" add, "D" delete)
                      #--- ( "!" missing, "C" overlapped edits with prior checked in changes, conflict)
%> rm -rf workDir  #--- destroy/remove entire tree, included ".svn" sub-trees
%> cp foo ../bar   #--- copy file or dir
```

See projects/LC3-trunk/docs

Lec-1-tools-1

(0) Web browser to access our repository:
- --- https://svn.cs.georgetown.edu/svn/projects (a base repository we will work from.)
- --- https://svn.cs.georgetown.edu/svn/projects2 (this will have one branch per student.)

You can view/download the current contents of a repository (the latest version) this way. This is a good way to get materials from the repository tree to play with and read. Reading documentation is the first thing you will want to do this way. NB--Access through the web interface does not create a working directory on your local machine: you cannot check-in/commit changes, "svn ci", nor get updates, "svn up". You will want to read the "README"s in the various sub-directories of the base repository.

(1) A Subversion (SVN) client:
--- http://subversion.apache.org/
This is the home page of the Subversion site. You can find downloads for clients there. Subversion consists of two parts, a server, and a client. You only need a client. Most downloads will include a server, but you do not need to set it up. You may already have an svn client installed as part of your system's operating system. If you need to get a client, see if there is an executable binary available rather than downloading the source code.

--- Mac OSX 10.5 and later: use the terminal app, and the commandline client "svn". See links for binaries. MacPorts has it, too.

--- Windows: there are links to binaries for several different gui svn clients on the subversion web site. However, you will need a unix interface to windows anyway for iverilog; so, you should install cygwin:
--- http://www.cygwin.com/
The setup.exe will give you a window with lots of selections you can make to include various unix tools in your cygwin installation; for instance, text editors such as vim. Here's a list of things to get (you can do some of this piecemeal at a later time):
--- Base: gzip, grep, sed, tar, which
--- Devel: gdb, make, subversion
--- Editors: emacs, vim
--- Net: openssl

(2) electricBinary.jar:
This is in the repository: projects/LC3-tools. Along with PennSim.jar (an LC3 simulator). Also, see LC3trunk/docs/README-Electric.html. Current version is 9.02.
(see http://www.staticfreesoft.com , navigate to Products, download, and "binary release")

(3) Verilog:
See /docs/verilog/README-verilog.html. The software is here:

--- http:/iverilog.icarus.com/eda/
But see this,
--- http://iverilog.wikia.com/wiki/ , especially the installation guide (see details for your system)

Mac builds require Apple XCode development tools. Cygwin builds require unix development tools. You will often find it already installed or as part of a distribution archive. Current source is in projects/LC3-tools/.

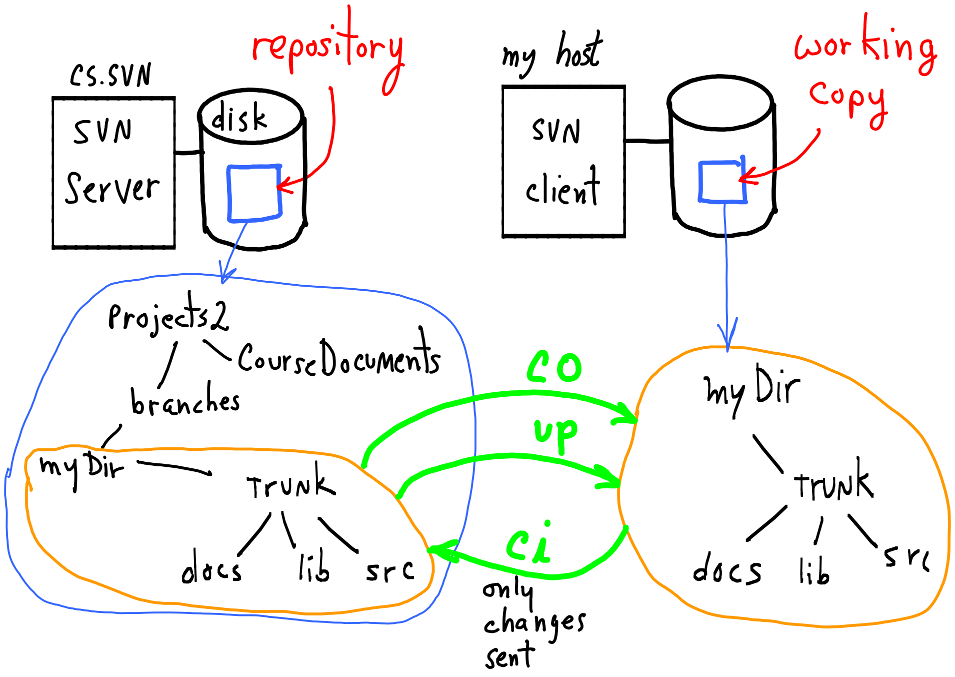
how to unbundle source (after which, see readme):
```
--- gunzip the .gz file:     gunzip *.gz
--- untar the .tar file:     tar -xvf *.tar
```

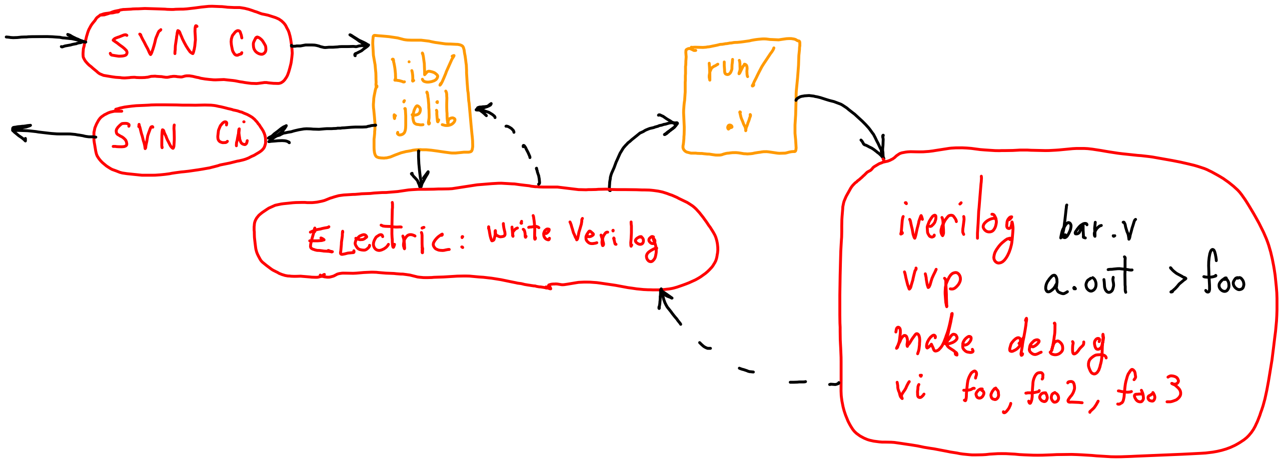
Subversion

-- repository sits on a server.
-- client on local host gets a working copy:
"**svn co** https://..../myDir
-- client sends changes to repository:
"**svn ci**" (in myDir)
-- client gets changes (from other's "svn ci"):
"**svn up**" (in myDir)

NB--SVN commands only apply to portion of tree **at and below current directory**.

--See history of changes:
"**svn -v log**"





--- There can be many working copies checked out.

--- Changes go to the repository via "svn ci".

--- Delete working copy w/o harm (provided changes were saved).

--- Checkout a prior version ("svn co -r123 URL/myDir   myDir123") name local copy

--- Start a new development branch ("svn copy")

--- Find out if there are changes in your working copy ("svn status")

--- See history of changes ("svn -v log")

General workflow:
--- **Electric**.File.OpenLibrary "myDir/trunk/lib/foo.jelib"   ===> make changes to lib files.
--- **in terminal window**: "cd myDir", "svn ci" (write good comments in commit window.)
--- **Electric**.Tools.Simulation.WriteVerilogDeck:
creates verilog file from design, e.g., "myDir/trunk/run/foo.v").
--- **in terminal window**: "cd myDir/trunk/run"
compile verilog and simulate ("iverilog foo.v", "vvp a.out > foo_output")
--- **in terminal window:** check for errors ("vi foo_output").
--- go back to Electric, revise design.

Altering SVN Tree

NO: SVN IMPORT!!!

- remove → svn rm
- move → svn mv
- add → svn add
- svn status
- svn help –

NO: ADDING TEMPORARIES, Working Copies

rm: if you delete a file w/o using svn, svn will think the file is missing and will get a copy when you "svn up".

mv: if you rename a file w/o using svn, svn will think it is new (and the old one missing). NB--**mv** will appear as an D-A pair.

add: if you want something to become part of your repository, "svn add".

Do "svn status" before doing a commit:

"?" file (or dir) is unknown, nothing will be done.
"M" file is modified, changes will be sent.
"A" file is queued to be added to repository
"D" file is queued to be deleted from repository
"C" Conflict: you tried to commit changes that overlapped
with other changes already committed.

If your local copy is confused, you can completely erase it locally,

```
% /bin/rm -rf myDir
```

then re-checkout. If you have altered files, put them in a safe place first, then do **rm**, then move them into your new working copy.

- Use Web access for downloading anything not in your own subtree of repository.
  — Safest
- Checkout tree, but never checkin except in your subtree
  — Handy: you get updates to docs, etc.
- **myDir/.svn** **~/.subversion** – Subversion keeps track of
  - repository address
  - authentification
  - files/dirs changes/status

- Using Electric

projects/LC3-trunk/examples/ElectricTutorial/
projects/LC3-tools/electricBinary.jar

→ see 0AAA-ReadMe

→ see text in cells

- reg{sch} , regUsage{sch}

→ Electric.tools.simulation.(write Verilog deck)

— use tab "Explorer" to see library contents — also, see (Place Cell)

— Cell creation: Cell.NewCell library, type=schematic

- Blackbox + Components.schematic.Misc.VerilogCode (edit code box) Edit.Properties

Electric's names versus Verilog's names

- schematic cell vs. icon cell

Design is in a schematic cell: foo{sch}
Icon for design is in icon cell: foo{ic}
Hierarchy: place icon foo{ic} in bar{sch}

Electric		Verilog
schematic cell	⟷	class/module def
schematic exports	⟷	module parameters
icon instance	⟷	module instance
wires to icon exports	⟷	instance's args

- Electric: schematic cell "sch" has a name.
- Verilog: module uses cell's name
- Electric: each icon instance has its own name
- Verilog: each module instance has icon instance name.
- Hierarchy: Electric's Exports = Verilog's args list (ports)

Verilog

modules = classes (except top-level module from top-level cell) ← like Java main class or C's "main()", aka "testbench"

- port connections: selecting ports, placing wire

  - creating ports: - characteristic [input/output]
    - name

Place and select a pin to create port:

Export.CreateExport

select export text (not pin) to see export properties (Text has a highlighted X across it when selected; pin shows as a highlighted square).

Edit.Properties.ObjectProperties

wires go from pin to pin.
Busses are collections of wires.
Busses go from bus-pin to bus-pin

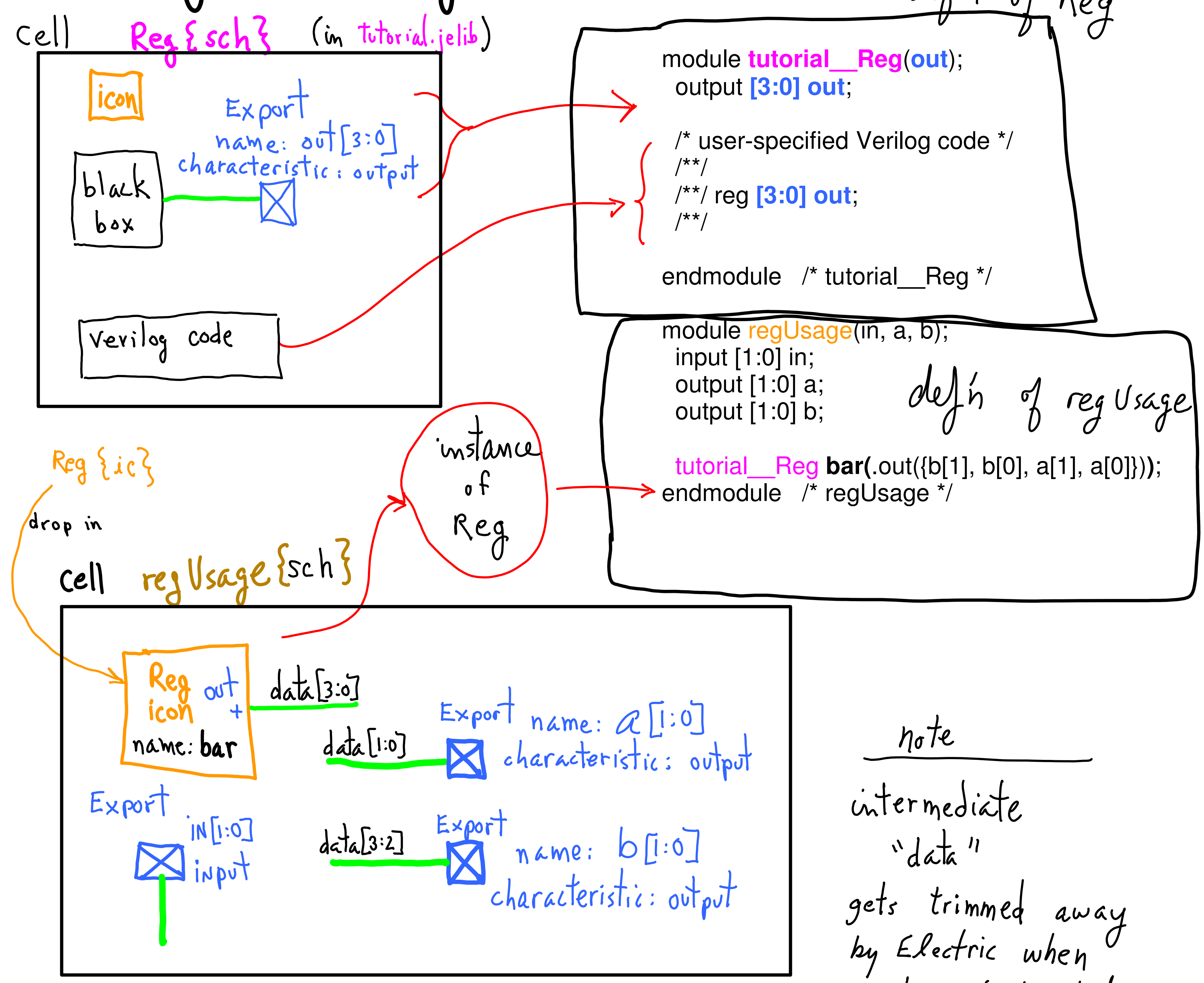
Electric.Components.{wire pin}
" " .{wire}
" " .{bus pin}
" " .{bus}

- Wires use pins, wire from selected pin to {cell area}^ or "+"^
- busses
  - concatenation examples Tutorial.jelib/RegUsage
  - sub-bus connections
  - connect via naming
  - Bus naming: foo[3:0], 4-wire bus

Verilog
- wire naming and connections in module instantiation arg list.
  - connect by position in arg list: and and_0(Y, A, B);
  - or by name: and and_1(.in0(A), .in1(B), .out(Z));

- wire, reg, input, output [designations] (size parameters)

Exports / verilog module args
Heirarchy ~ connectivity



```
module tutorial__Reg(out);
  output [3:0] out;

  /* user-specified Verilog code */
  /**/
  /**/ reg [3:0] out;
  /**/

endmodule   /* tutorial__Reg */
```

```
module regUsage(in, a, b);
  input [1:0] in;
  output [1:0] a;
  output [1:0] b;

  tutorial__Reg bar(.out({b[1], b[0], a[1], a[0]}));
endmodule   /* regUsage */
```

note

intermediate "data" gets trimmed away by Electric when creating Verilog deck

in Electric	equivalent in Verilog
Reg{sch}	module tutorial_**Reg**(**out**)
Export, **out**[3:0], **output**	**output** [3:0] **out**
instance of Reg{ic} named **bar**	tutorial_Reg **bar**( )
bar.**out**[1:0]-to-**a**[1:0] connection	**.out**(**{** ..., **a**[1], **a**[0] **}**)
equivalent syntax	..., .out[1](a[1]), .out[0](a[0])

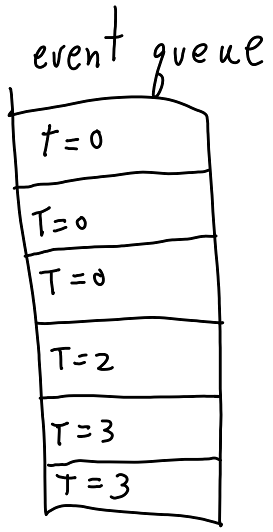
Bus Concatenation

The connections between levels in a hierarchy are expressed as "Exports" in Electric and as args in Verilog. Electric trims away redundant wires; so, the busses dissappeared in the Verilog code.

Seeing Verilog results

- Verilog compilation + simulation [discrete event]

```
Electric.Tools.Simulation(Verilog).WriteVerilogDeck   ⇒ save as foobar.v
iverilog foobar.v
vvp a.out > foo   ⇒ saves simulation output
make debug   ⇒ see contents of foo, foo2, foo3
     or
(just look at foo w/ editor)
```



Least time-stamp simulation event pulled from queue, executed, new events posted to queue.

- Verilog
  - getting something to happen -
    - initial begin . . . end
    - always begin . . . end (NB 0-delay → ∞ loop)
  - stopping:
    - $finish;
  - Seeing what happened:
    - $display("%d", $time);
    - $write("hi"); ← no eoln

    when? delay to see results.
  - documentation: see

    docs/verilog

    (or google "verilog tutorial")

LC3 system

- projects/trunk/
- " lib/system.jelib.top{sch} — Top-level cell of LC3
- " " test.jelib.top_rtL_test — Top-level cell testbench has instance of system.top, a "main" for simulation.
- " " system.jelib.top.showRegs — a "task" can be called in verilog code.

Task defin

task f; begin ... end endTask

Task invocation

f;

Plan

1) See docs/LC3-uArch-PPappendC : machine states, datapath

2) See docs/LC3-uArch-controlStates : control signals

3) See Lec - LC3-vonNeuman : simplified instruction phases

See test.jelib.test-rtl-allInstr — use for getting debug output for machine states and controls

4) See docs/LC3-instructions...

Using projects/LC3-trunk

Read Lec-1-HW-2-tutorial.html.

Below is just an overview, and is inaccurate.

0.) Checkout your branch, and add a trunk/ to it.

1.) Copy contents of src/ to your branch.
   -- We may do a nested "svn co" so you can get updated files. Don't "svn add".

2.) run src/'s Makefile,

```
    cd src/
    make
```

3.) follow instructions to set up your branch's environment

  -- creates: trunk/bin/ and trunk/run/ (temporaries, not svn added.)

  -- creates: trunk/src2/ and adds it to your branch. Contains source code and build tools.

4.) Build tools and set PATH.

5.) Copy .jelib files from projects/LC3-trunk/ to your trunk/lib.

6.) Your assembly language source code will be in trunk/src2/. Which is part of your branch.