1 Getting Ready to Use PennSim

1.1 Make sure that you have a CAE account

You will need to use the simulator in class, and the easiest way is by having a CAE account so that you can log in and access your CAE network storage (which is where you can keep the simulator and your LC-3 programs and data files) from WisCEL or CAE computers. If you don't have a CAE account, take a moment and create one online at:

`https://my.cae.wisc.edu/tools/public/newuser`

If you have problems with obtaining or using your CAE account, you should contact CAE Technical Support.

If you have trouble obtaining or using PennSim on a WisCEL or CAE Windows machine, you can ask the 252 staff for help. It is also possible to use a personal machine instead of (or in conjunction with) a CAE account, but we will NOT provide technical support for this. If you decide to use PennSim on your personal machine, then you do so at your own risk and you'll have to troubleshoot any problems on your own. Please note that in the past, PennSim has been somewhat buggy on Macs.

Except for Step 1.3 below, the rest of this document assumes that you will be using a CAE account on a CAE or WisCEL Windows machine.

1.2 Find the location of resources related to PennSim on the class website

The simulator and documentation are available on the class website near the top of the page.

1.3 Install the Java Run Time Environment (JRE).

If you are using a CAE or WisCEL Windows machine, the JRE is already installed. This step only applies if you plan to use a personal machine for LC-3 programming assignments (remember, we do not guarantee it works or provide technical support for running PennSim anywhere but on a CAE or WisCEL Windows machine). You can download the JRE from here: http://java.com/en/download/index.jsp

1.4 Download the simulator and OS

Create a directory named "PennSim" (without the quotes) in your CAE network storage.

Download the LC-3 Simulator (**`PennSim.jar`**) from the links given on the course website and put it in your PennSim directory. (right-click the link and select **Save Link As…**) You may also wish to save this document, the PennSim Guide, and the PennSim Manual in this same directory for easy reference (all of these are available on the course website).

You will now be able to access the simulator and its files in WisCEL or on any CAE Windows machine by logging in to your CAE account.

1.5 Know what a "text editor" is

If you don't already know, read the beginning (up to but not including "History") of this Wikipedia article: http://en.wikipedia.org/wiki/Text_editor

If you're using Windows, you can use **Notepad** (or WordPad) as a text editor. Just make sure you save your files as "**plain text**". However, there are other text editors out there too, such as Notepad++ and TextPad.

2 Using PennSim

The LC-3 simulator is a program that *simulates* the LC-3 processor. In other words, it lets you execute LC-3 instructions on a "pretend" LC-3 processor and see the results. It also lets you see what is going on *inside* the LC-3 processor as it executes instructions because the simulator shows you the contents of memory, the register file, and the value of the program counter (**PC**).

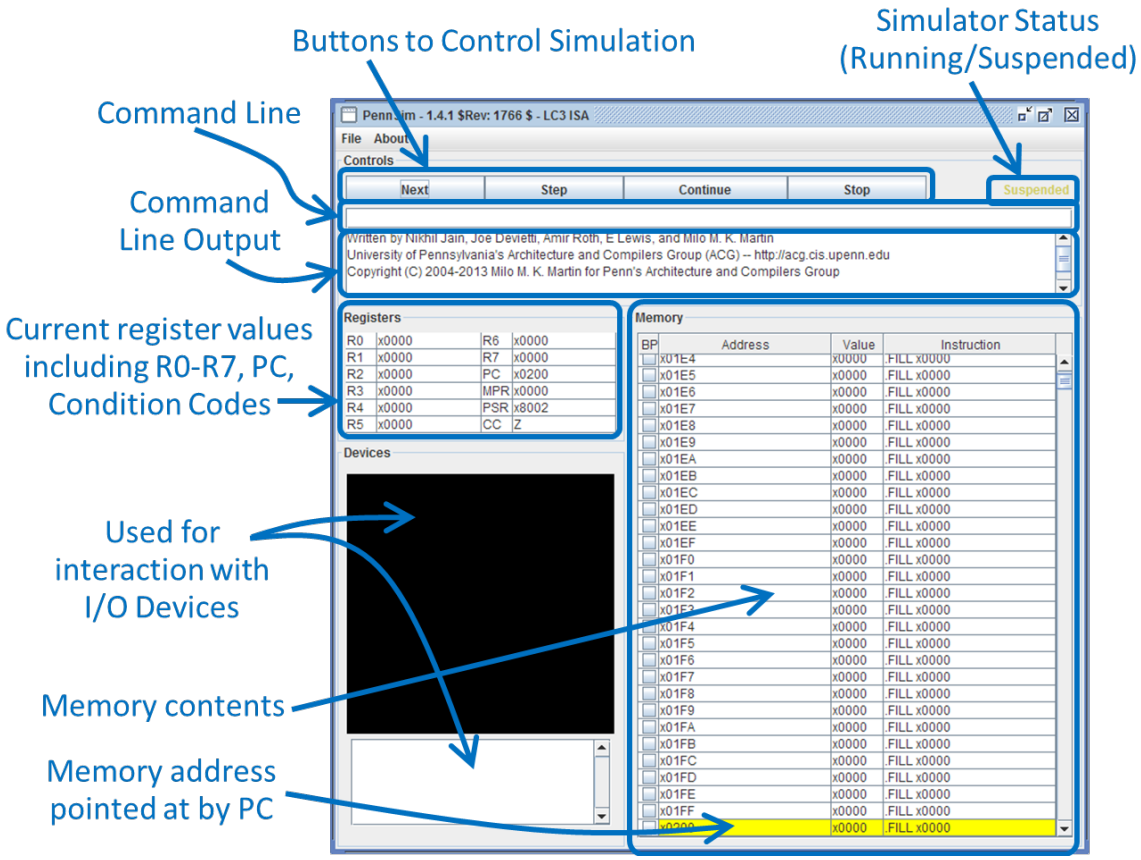
Below is an annotated image of the simulator interface. The registers in the register file (**R0** through **R7**) are shown in the area labelled "Registers" on the left above the Graphic Display area. This part of the simulator window also shows the value of the Program Counter (**PC**), and which of the three condition codes (**CC**) is currently true (in the image, the **Z** condition code is true, and **N** and **P** are both false).

The contents of memory are shown on the right in a scrollable window. For each location, it indicates the value stored in memory (in hex format). The simulator often tries to interpret the contents of each location as an instruction, lists what that would be (whether or not it is actually *intended* to be an instruction!). Other times in that column it will simply say **`.FILL`** followed by the value in memory.

To scroll within the memory window, **DO NOT** drag the "handle" (the rectangle that slides up and down in the scrollbar) unless you need to scroll a **long** distance. Also, **DO NOT** just click the arrows on the scroll bar – that will scroll one pixel at a time, which is **frustrating and slow**. To scroll, use the arrow keys on the keyboard or click in the space between the handle and the arrow for the direction you wish to scroll. Or, use the "list" command that will be explained later.

The data word size of the LC-3 processor is 16 bits, and it uses 16-bit memory addressing, so all values stored in registers and memory, as well as all memory addresses, are given as 4-digit hexadecimal values – they are easier for humans to read than 16-bit binary values!

Unfortunately, PennSim always shows hexadecimal numbers with just an "x" in front of them, instead of the more common "0x". When you type hexadecimal values in your programs and in the simulator, you will also need to just use "x". For example, you would represent the number $CAFE_{16}$ as **xCAFE**.



2.1 Launching the Simulator

If using a Windows (or Mac) machine, you can usually just double-click the **PennSim.jar** file. If this doesn't work, right-click and "Open with…" the Java Platform. A window similar to the one in the image should open.

Note: any assembly programs you write need to be in the ***same directory*** as **PennSim.jar**, or PennSim will not be able to find them! **DO NOT RUN PENNSIM DIRECTLY FROM A BROWSER WINDOW**. Although at first it will appear to work (PennSim will open), it not be able to find your source code to assemble it!

2.2 Using the Simulator

To use the simulator, you will type commands into the box labelled "Command Line" in the figure above to assemble and load your program. Once you load a program, you can then use the buttons above the Command Line ("**Next**", "**Step**", "**Continue**", and "**Stop**") to execute your program. These buttons will be discussed more later.

2.2.1 An Example LC-3 Program

Download the **tutorial.asm** file and place it in the same directory as **PennSim.jar**. The **.asm** extension tells you that it is an assembly code file. Open **tutorial.asm** in a text editor so you can look at its source code.

The top of the program has a comment telling what the program is for. Generally, this statement should provide more information about what the program does, but here the point of the program really is to introduce you to the PennSim simulator.

The **.ORIG** assembler directive at the start of the program tells the assembler that when the program is loaded, it will need to start at memory address **x0200**, which is important, because that is where the LC-3 will start executing instructions when it first starts or after it is reset.

Next in the program is the word **START**. This is a label that lets us refer to this location in the program. If you look at the end of the program, there is an instruction **BR START**. That instruction simply tells the simulated LC-3 processor to update the PC with the address of the instruction immediately after the **START** label, which happens to be the first instruction in the program. This means that when we run this program, it will repeat forever until we tell the simulator to stop. Looping over the code is a useful debugging technique, because it makes it easy to step through the program, instruction-by-instruction, more than once.

Next is a sequence of instructions that perform logical operations. Note that the comments in this program are a very BAD example of commenting code – they are worthless for helping you figure out the purpose of the instructions since they just repeat what the instruction does, which is already obvious. This is intentional in this case. However, in the future, your code must have comments that address **why** the instruction is in the code – anyone can easily figure out **what** an instruction does.

Finally, after the **BR START** instruction (which was discussed above) is the **.END** assembler directive, which just tells the assembler not to look any further in the file for code or data to assemble.

To run this program, we will first assemble it to translate it into binary instructions that the simulated LC-3 processor will execute, and then we will load the resulting binary program into the memory of the simulated processor that will run it.

2.2.2 Assembling and Loading a Program

PennSim has a built-in assembler; you can assemble this program by typing the following command into the Command Line box of the simulator:

```
as tutorial.asm
```

You should see the below response in the Command Line Output area:

```
Assembly of 'tutorial.asm' completed without errors or warnings.
```

You should **ALWAYS** check the response anytime you enter a simulator command. Don't just assume that everything worked the way you assumed it should or wanted it to!

The assembler will create two new files (**`tutorial.obj`** and **`tutorial.sym`**) in the same directory as **`tutorial.asm`**.

An **`.obj`** file contains the machine code for the program (the instruction binary). It only contains the starting address for the program and the instructions that make up the program. A **`.sym`** file is a "symbol file". The **`.sym`** file lists all of the labels and the memory addresses associated with them. This tells the simulator, for example, to show the label **START** at memory address **`x0200`** for the tutorial program. The **`.sym`** file is not used by the processor—only by the **simulator** to make it easier for you to look at your assembled code.

Next, load the assembled program into the LC-3 processor by clicking **`Open .obj File`** in the **`File`** menu, and selecting **`tutorial.obj`**. You should see the below response in the Command Line Output area:

```
Loaded binary object file 'tutorial.obj'
Loaded symbol file 'tutorial.sym'
```

NOTE: there is a **`load`** command that you can type to load an object file, but unfortunately PennSim has a bug where sometimes it only partially "refreshes" the memory window to show what has changed when the program was loaded. For example, it may not show labels and may not show the new values in memory (though it will show the instructions they represent. If you do load an object file that way, you can scroll so that the program is not in the range of addresses shown, and then scroll back, which forces a re-rendering of the window… **Or, just use the File menu (which avoids this problem).**

After you load the program, the contents in some memory locations will be different (because the program's instructions will occupy those locations). Although you only loaded the **`.obj`** file, the simulator also automatically looks for and loads a **`.sym`** file with the same name when you load an **`.obj`** file.

In the simulator, only the actual instructions have been loaded into memory—this is because **`.ORIG`** and **`.END`** are **not** LC-3 instructions. They are ***assembler directives*** that provide information to the assembler. Notice that the **START** label is shown next to memory address **`x0200`** – the LC-3 simulator knew to put that word there based on the contents of the **`tutorial.sym`** file. (If the **START** label is not visible, then please read the note in the box above.

At this point, you can answer the tutorial-related questions in the homework.
The remainder of the tutorial will be completed during an in-class exercise.
In class, you will be provided with a printout of the remaining part of the tutorial.

2.2.3 Running the Program

As part of this tutorial, you will be answering questions in an in-class exercise.

After assembling and loading the tutorial program, you are ready to run it. If you have not yet assembled and loaded the program, do so now. Assemble the program using the command **`as tutorial.asm`**, and load the object file using **`File→Open .obj File`** in the Simulator menu.

The first instruction of the program, which should now be in memory at address **`x0200`**, is highlighted in yellow. PennSim highlights the next instruction that will be (but has not yet been) executed. Remember – the yellow highlight shows the next instruction that will execute.

> **TUTORIAL QUESTION 1:** This question in the Moodle exercise lists a memory address. At this point of the tutorial, what value does PennSim show is in memory at that address?

> **TUTORIAL QUESTION 2:** This question in the Moodle exercise lists a memory address. At this point of the tutorial, what value does PennSim show is in memory at that address?

In the ***Registers*** window, you can change a register's value by double-clicking on its value and typing a new value, then hitting "Enter" on the keyboard. Don't forget to hit Enter or the value isn't changed! This can be very useful when testing your code.

Get the values for R0 and R1 from TUTORIAL QUESTION 3 in the exercise, but do not yet answer that question.

Double-click in the box next to **`R0`**. Change its value to the value indicated in Moodle and press the Enter key on the keyboard. Also set **`R1`** to its assigned value from Moodle. Be sure to include the "x" (but don't use "0x") when you type in the value (so that the simulator knows you are typing a hexadecimal number), and don't forget to press Enter each time!

Press **STEP** once now. The **STEP** button tells the simulator to execute **only** the highlighted instruction, and then pause again.

The highlight should have moved to the next location. Also, the value in register **R2** has been changed by the instruction that just executed. It should now contain the bitwise complement of the value in **`R0`**.

Press the **STEP** button two more times (so that you have pressed it three times total) Memory address **`x0203`** should now be highlighted with yellow. If you accidentally step too far, keep pressing **STEP** until you get there again.

> **TUTORIAL QUESTION 3:** What is the value in register R3 at this point of the tutorial?

Press **STEP** button until the last instruction in the program (**`BR START`**) is highlighted, which is at memory address **`x0208`**. If you accidentally step too far, keep pressing **STEP** until you get there again.

Note that the simulator does not show what happens during the different phases of the instruction cycle, it only shows the results of each instruction. At this point, we have just finished executing the last NOT instruction, and are about to execute the instruction highlighted in yellow.

> **TUTORIAL QUESTION 4:** What is the value in register R2 at this point of the tutorial?

Press **STEP** one more time.

> **TUTORIAL QUESTION 5:** What happened after you pressed STEP this time? Choose the best answer.

Next, you will set a "breakpoint" at memory address **`x0208`**, which is the address of the **`BR START`** instruction. There are two ways you can do this. One is to click on the box to the left of its memory address (**`x0208`**). The other is to enter the command:

```
break set x0208
```

Either method should turn that line red in the memory window, indicating that a breakpoint is set. Creating a breakpoint does not change the program; it just tells the simulator that when the **PC** is incremented to that address, the simulation should pause (you can think of it as "take a break") until you tell it to continue.

Get new values for R0 and R1 from TUTORIAL QUESTION 6
in the Moodle exercise, but do not yet answer that question.
These should be different values than what you used earlier.

Modify the contents of registers **R0** and **R1** based on the new values given to you in the exercise. Then press the **CONTINUE** button. This button executes from the current instruction until it finds a breakpoint. Since you have set one at memory address **x0208**, that location should now be highlighted in yellow. The simulator did not "skip" any instructions; it just executed each of them without pausing for you to look at the result, and stopped when it reached the breakpoint (but before it executes the instruction at that location).

TUTORIAL QUESTION 6: What is the value in register R2 at this point of the tutorial?

2.2.4 Modifying the Tutorial Program

Now you will modify the tutorial program. It currently contains an unconditional branch instruction that makes it repeat the program forever if you kept **STEP**ping through it. Open **tutorial.asm** in a text editor and "comment out" the **BR START** instruction. You can do this by putting a '**;**' character in front of it so that the line looks like:

```
; BR START          ; repeat forever
```

BR START is now a comment instead of an instruction, so the assembler will not create a binary instruction for it. Commenting out code is a good way to "remove" instructions from your program without deleting them (in case you need to put them back later).

Do not modify any of the rest of the code. Save the file. Shortly, we will assemble, load, and run the modified program. But first, reset the simulator by typing:

```
reset
```

The console should output:

```
System reset
```

You should also see that all memory locations have been cleared to **0**, the registers have been cleared (except for the **PC**, which is now set to **x0200** again), and any breakpoints you set have been removed. This ensures that there are no instructions left in memory from the previous version of the program when we run the new version.

Assemble **tutorial.asm**, then load **tutorial.obj** into the simulator. Look at memory location **x0208**. This location used to contain the instruction **BR START**, but you removed that instruction from the program by commenting it out. If you cannot see this address in the window, you can either scroll to it or make the window scroll to it automatically by entering the command:

```
list x0208
```

This command moves range of memory addresses shown in the window to include the indicated address. It also displays the value in that memory location in the Command Window Output area.

TUTORIAL QUESTION 7: What is the value in memory at address x0208?

Set a breakpoint at **x0208** again, and press **CONTINUE**. Press **STEP** once.

TUTORIAL QUESTION 8: What happened after you pressed STEP this time? Choose the best answer.

That completes this tutorial. You will learn about a few more features of PennSim (such as how the **NEXT** and **STEP** buttons differ) later in the course.

Remember, the more you program and use PennSim, the more comfortable you will be with it!

Note: A quick start guide for the simulator is given on the back of the LC-3 Instruction Set sheet.

Summary of Useful PennSim Commands:

- Reboot system (clear all memory and registers): **`reset`**
- Assemble a program: **`as <program_name.asm>`**
  - Example – to assemble tutorial.asm: **`as tutorial.asm`**
- Load an already-assembled program: Select **`File→Open .obj File`** in the PennSim menu, and choose **`<program_name.obj>`**
  - Can also use **`load`** command in the Command Line, but it is somewhat buggy (see note in tutorial), and so is not recommended
- Display a memory location: **`list <address>`**
  - Example – to display location x0200: **`list x0200`**
- Set the value of a register: **`set <register> <value>`**
  - Example – to set R4 to 42_{10}: **`set R4 #42`**
  - Example – to set the PC to x0200: **`set PC x0200`**
- Set a breakpoint: **`break set <address>`**
  - Example – to set a breakpoint at x0200: **`break set x0200`**
  - Can also click on the checkbox at that memory address
- Run one instruction (step through the program): **`step`**
  - Can also click the **`STEP`** button
- Run program until a breakpoint: **`continue`**
  - Can also click the **`CONTINUE`** button
- Stop a program that is running (from using **`CONTINUE`**): Click the **`STOP`** button

More Information:

For more information about the simulator and simulator commands, refer to the PennSim Manual posted on the class website.