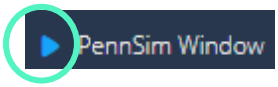
Once you've logged into Codio via Coursera, follow these instructions to get PennSim started within Codio. The assigned problems follow this brief tutorial.

Starting PennSim in Codio

1) Opening up the Codio **X-server**:
   a. Along the top menu, click on the blue "Play" icon next to "PennSim Window"

   b. If things work, you'll get a blank black window, this window is called an "Xserver"
      - *It can display any "graphical" output of your Codio virtual machine*
   c. If you see an error # 502: contact the course staff right away! 😊 Don't attempt to solve this on your own.

2) Opening up the Codio **Terminal Window**:
   a. On the left hand side of the screen, you will see what's called the **File Tree.** This is a listing of all the files on your virtual Codio computer.
   b. Click on the small computer icon:

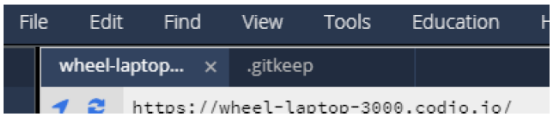
   

   c. This will bring up a Linux terminal window, where you can manually type in commands to your computer.
   d. **Type in** this command to start the LC4 Simulator (PennSim):

      java -jar PennSim.jar

      (DO NOT COPY & PASTE THIS – type it in manually)

      This command uses Java to start our PennSim program
   e. In the "Xserver" window, PennSim (the java app) will open up:
      - Note that the Xserver window will be a randomly named tab (ex: wheel-laptop)

   f. If PennSim freezes and refreshing the page doesn't help, go back to the terminal and use CTRL-C to terminate PennSim.

      Then type in the command: **pkill -f PennSim.jar** and hit enter. Then type the above command **java -jar PennSim.jar** to re-start PennSim.

Running Multiply.ASM in PennSim

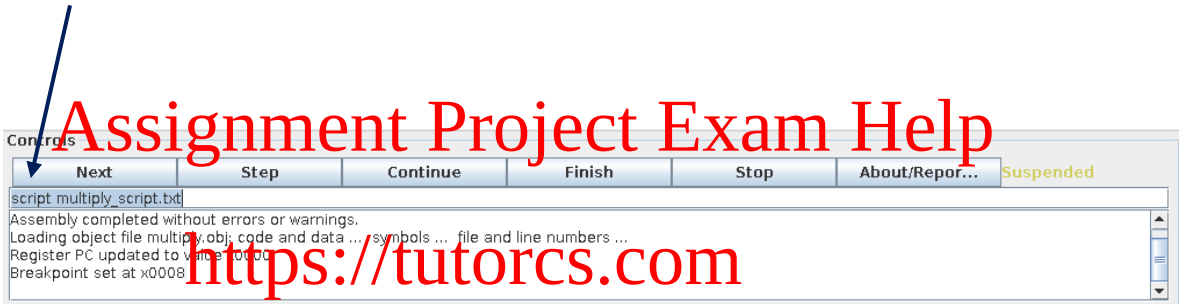
1) From the **File Tree** click on the file: **multiply.asm**
   - Examine the contents of the file. Notice it is the multiply algorithm from lecture!

2) From the File Tree, click on the file: **multiply_script.txt**
   - This file shows all of the commands that must be entered into PennSim to assemble and load the file, examine it line by line

3) Return back to the "PennSim" window you opened in the last section
   - In PennSim's "***Controls***" section, type in:

     script multiply_script.txt

     Then press enter:



4) Press the "**Step**" button to run the program line by line. Carefully examine the register file, the program counter, and the state of the machine as you run it line by line!

Assigned Problems (to be done individually, NOT group work):

***Make certain you've set up Codio and learned PennSim before attempting these problems!**

1) WHILE LOOPS IN ASSEMBLY

The pseudo-code below describes the mathematical operation known as a factorial. When the algorithm below is completed, the variable B will contain A! For the given positive value of A, the factorial is defined as 5 x 4 x 3 x 2 x 1 = 120.

Here is the pseudo-code for the factorial algorithm:

```
A = 5 ;  // example to do 5!
B = A ;  // B=A! when while loop completes

while (A > 1) {
   A = A - 1 ;
   B = B * A ;
}
```


Implement the given algorithm using LC4-Assembly. Use R0 to hold variable A, and R1 to hold variable B. In your Codio "File Tree" you'll see a file I created for you: `factorial.asm`. Open the factorial.asm file on Codio and implement the factorial algorithm above within it. Test it using the other file I've provided for you: `factorial_script.txt`. You may hard code A to have the value 5, but when we grade your assignment, we will try out different #s to ensure your algorithm is working. So be certain to run your program in PennSim and make certain it is working for different values of A.

In your program and in your script file be certain to set a breakpoint labeled "END." (see multiply.asm for an example of this). This will ensure you program ends, instead of requiring an infinite loop to stop execution. Be certain to comment your code to help us understand the flow of your program as we grade.

POINTS WILL BE DEDUCTED IF YOUR CODE IS NOT COMMENTED!

For this problem you will edit and change the 2 files: **factorial.asm** and **factorial_script.asm**

2) SUBROUTINES IN ASSEMBLY

For this problem, you'll convert your factorial program from problem #1 into a subroutine and **call it using JSR.**

File Setup: After you've completed problem #1, copy the file: `factorial.asm` in the Codio "File Tree" by right-clicking on the file, clicking "copy" and then right clicking once again in the file tree and pressing "paste." Rename this copied file as: `factorial_sub.asm`.
Next, copy and paste the file: `factorial_script.txt`. Rename the copy as: `factorial_sub_script.txt`. Next, you'll need to open and edit this new file (factorial_sub_script.txt) to ensure it assembles and loads "factorial_sub.asm" instead of "factorial.asm"

What to do for this problem:
Add the label: SUB_FACTORIAL to the top of your factorial program. Remove any "CONST" instructions you may have that would set the variable A (register R0) inside your subroutine. We want "A" to be an argument to your subroutine, so its value must be set before the subroutine is called. Inside the subroutine, replace the END label with a RET statement. Your program should END after the subroutine returns, with B holding the return value of the subroutine.

Above your subroutine code, implement the following pseudocode to call your subroutine:

```
MAIN
  A = 6 ;
  B = sub_factorial (A) ;

  // your sub_factorial subroutine goes here

END
```

After you return from the subroutine, make certain to "jump" over your subroutine to a new END label, so that your subroutine isn't executed twice! Make certain to set END as a breakpoint in your script file.

*Next, a*dd an "If/else" statement to the start of your subroutine to ensure A is a positive # and is <= the largest number your assembly can work with. If A is <= 0 or > the largest number your algorithm can work with, set B = -1 and return from the subroutine without attempting to find the factorial. *Hint: With these constraints, first determine if we as programmers should consider A as signed or unsigned.*

POINTS WILL BE DEDUCTED IF YOUR CODE IS NOT COMMENTED!

POINTS WILL ALSO BE DEDUCTED IF YOU DON'T CALL THE SUBROUTINE USING A JSR and RETURN FROM IT USING AN RET!

For this problem you will have created 2 files: **factorial_sub.asm** and **factorial_sub_script.txt**

3) WORKING WITH DATA MEMORY IN ASSEMBLY

For this problem, you will have to review the example of working with data memory in lecture. You may also find the "sum_numbers.asm" example helpful in the PennSimStartGuide Manual.

File setup:
After you've completed problem #2, copy the file: `factorial_sub.asm` and paste it with the new name: `dmem_fact.asm`. Then copy: `factorial_sub_script.txt` and paste it with the new name: `dmem_fact_script.txt`. Next, update `dmem_fact_script.txt` to ensure assembles and loads "dmem_fact.asm" instead of "factorial_sub.asm"

What to do for this problem:
Recall the ".FILL" directives mentioned in lecture, and also in the sum_numbers.asm example from the PennSimStartGuide. Use the .FILL directive to populate 5 rows of data memory starting address x4020 with the numbers: 6, 5, 8, 10, -5. Write a short assembly program that will load each of the 5 rows of data memory that you've populated and call the subroutine you've created in problem #2 on each of those rows. After the factorial subroutine is run on each row, you should then store the #'s factorial back to data memory overwriting the original #. As an example of how the first row of data memory should look after your program completes, address x4020 should have the number #720.

POINTS WILL BE DEDUCTED IF YOUR CODE IS NOT COMMENTED!

For this part you should generate the 2 files: **dmem_fact.asm** and **dmem_fact_script.txt**

***Extra Credit (5 points)*:** In addition to the program above, create a new program in `dmem_fact_ec.asm` (and its script dmem_fact_ec_script.txt), that allows your subroutine SUB_FACTORIAL to take in a data memory address (instead of a value) in R0 as its only argument. The new SUB_FACTORIAL should then load the value from data memory, specified by the argument, find its factorial, and store the result back in data memory (instead of returning a value). Update your code from problem #3 to call this subroutine properly.

Important Note on Plagiarism:

- We will scan your HW files for plagiarism using an automatic plagiarism detection tool.
- If you are unaware of the plagiarism policy, make certain to check the syllabus.