

Computer Organization

5-stage RISCv32I Processor

Assignment 1: Getting to know Cudasip Studio

Spring 2023

Objective: Through this assignment, you will be introduced to Cudasip Studio in the Cloud, a single unified Integrated Development Environment (IDE) for processor design. Studio can generate a Hardware Development Kit (HDK) which includes tools to specify a processor's Instruction Set Architecture (ISA), its hardware Cycle Accurate (CA) design, and output RTL such as Verilog for FPGA or ASIC synthesis. Along with the HDK, Studio can generate a Software Development Kit (SDK) which can be used by the application developer to profile, debug, simulate, and optimize the application.

Cudasip Integrated Toolchain

Programming tools (generated):

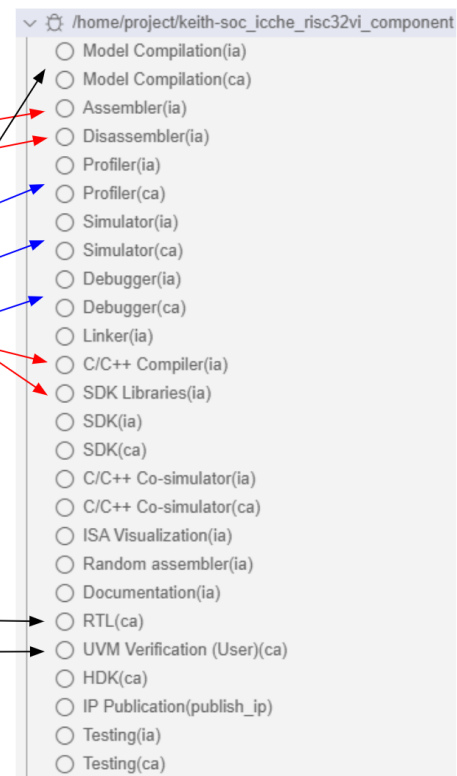
- Assembler
- Disassembler
- C-compiler & libraries

Development tools:

- Profiler
- Simulator
- Debugger

Hardware development tools:

- Processor Modeling
- RTL generation
- UVM Verification support



Tasks are aware of each other, just like a makefile software build, only required tasks are built. For example, if you modify a file that changes the Instruction Accurate (IA) model and it is shared with the CA model, all the tasks for both the IA and CA model will be marked as not built. Upon requesting a higher-level task to be built, all lower dependent tasks that are not built will be built first. A good example is the SDK (ia) task. If this task is built, all dependent tasks such as Model Compilation, Assembler, Disassembler, Profiler, Simulator, Debugger, C/C++ compiler, and SDK libraries will be built if necessary. No script development or toolchain maintenance required.

Key Learning Outcomes of this assignment:

Building Instruction Accurate (IA) models: In this assignment, you will import an IA model and once built, you can assemble, simulate, and debug assembly programs. The IA model is the processor's architectural model and not a processor's hardware representation. In a later assignment, you will be building the Cycle Accurate (CA) model which describes the hardware implementation and can output Verilog for FPGA or ASIC synthesis.

Simulating an assembly or c-program: To validate both processor architecture and hardware models, you will need to simulate. In this assignment, you will make a minor change to an assembly routine, build the assembly program, and simulate the assembly program while using the IA's debugger.

Due Dates:

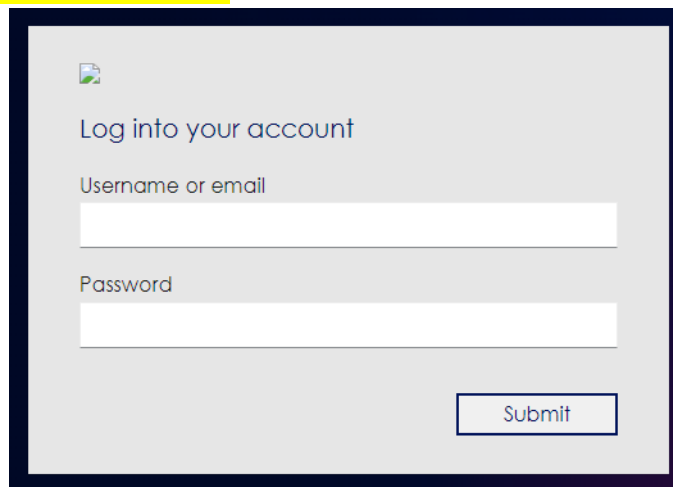
- Due date: See Canvas.

Assignment highlights:

- Yellow highlights signify that you will perform an activity in either your processor, assembly, or c-program projects
- Light green highlights signify important theoretical information that you may be tested upon

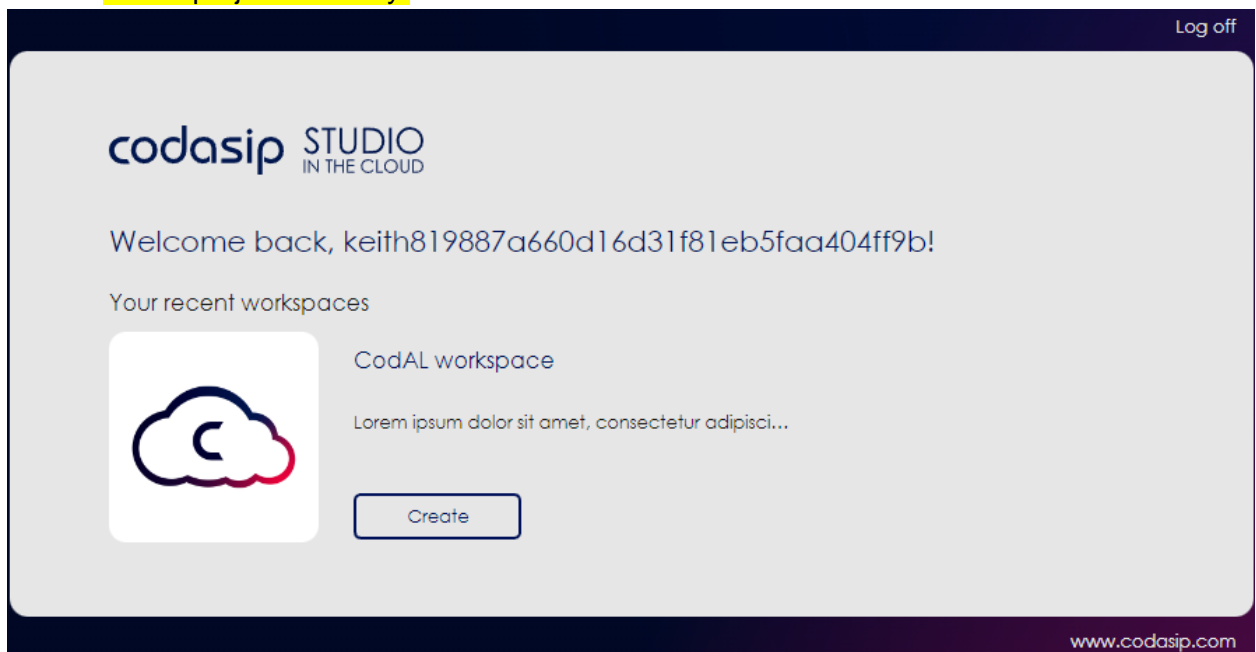
Instructions:

- Logging into Cudasip Studio in the Cloud for the first time
 - In either a Chrome or Firefox browser, type in studioin.cloud in the browser's address bar and then return



- **Note:** Safari and Microsoft Edge browsers are not supported
- Log into your account using your username and default password, codasip
 - The username is based on your email address.
 - email address: firstname.lastname@university.edu
 - username: [firstname.lastname](#)

- If your email address has a middle initial, the user name would be `firstname.middleinitial.lastname`
- If your email has special characters such as a "-", the user name would be as if the special character was not present
 - email address: `firstname.lastname-1@university.edu`
 - username: `firstname.username1`
- You will then be asked to create your "pod" which is your work environment. Click the "Create" button. This "pod" is dedicated to you and will store your files. It's Linux based, so in the instructions in 2 pages when it says to launch a terminal, that's launching a terminal in a Linux machine and put you into the `/home/project` directory.



- In the process of creating your workspace, you will be taken to a screen which includes the terms of the license agreement to use CodaSip Studio in the Cloud. Please read the terms and conditions carefully and click on "accept" only if you agree to the software license agreement

Codasip Cloud Studio - Click-through License

PLEASE READ CAREFULLY BEFORE ACCESSING ANY SOFTWARE FROM THIS WEBSITE:

This end-user license agreement (License) is a legal agreement between you (Licensee or you) and Codasip s.r.o. of Technická 2935/23, Královo Pole, 616 00 Brno, Czechia (Licensor, us or we) for:

Codasip Studio in the Cloud computer software (Software); and
any accompanying printed materials or online electronic documents (Documents).

You must agree to all of the terms of this License in order to use the Software or Documents. This License Agreement does not constitute a sale or any transfer of ownership of the Software or Documents to you. We remain the owners of the Software and Documents at all times.

IMPORTANT NOTICE TO ALL USERS:

BY CLICKING ON THE "ACCEPT" BUTTON BELOW YOU AGREE TO (AND/OR BY ACCESSING THE SOFTWARE AND/OR MATERIALS YOU ARE DEEMED TO HAVE AGREED TO) THE TERMS OF OUR PRIVACY POLICY

REJECT

ACCEPT

- If you select "accept," you will be taken to a screen that will ask you to either "open" your workspace or to "Teardown." Do not click on "Teardown" unless specifically asked to do so. "Teardown" will delete all your files in the current workspace. Click "Open" to enter your workspace.

Log off

codasip STUDIO
IN THE CLOUD

Welcome back, keith819887a660d16d31f81eb5faa404ff9b!

Your recent workspaces



CodAL workspace

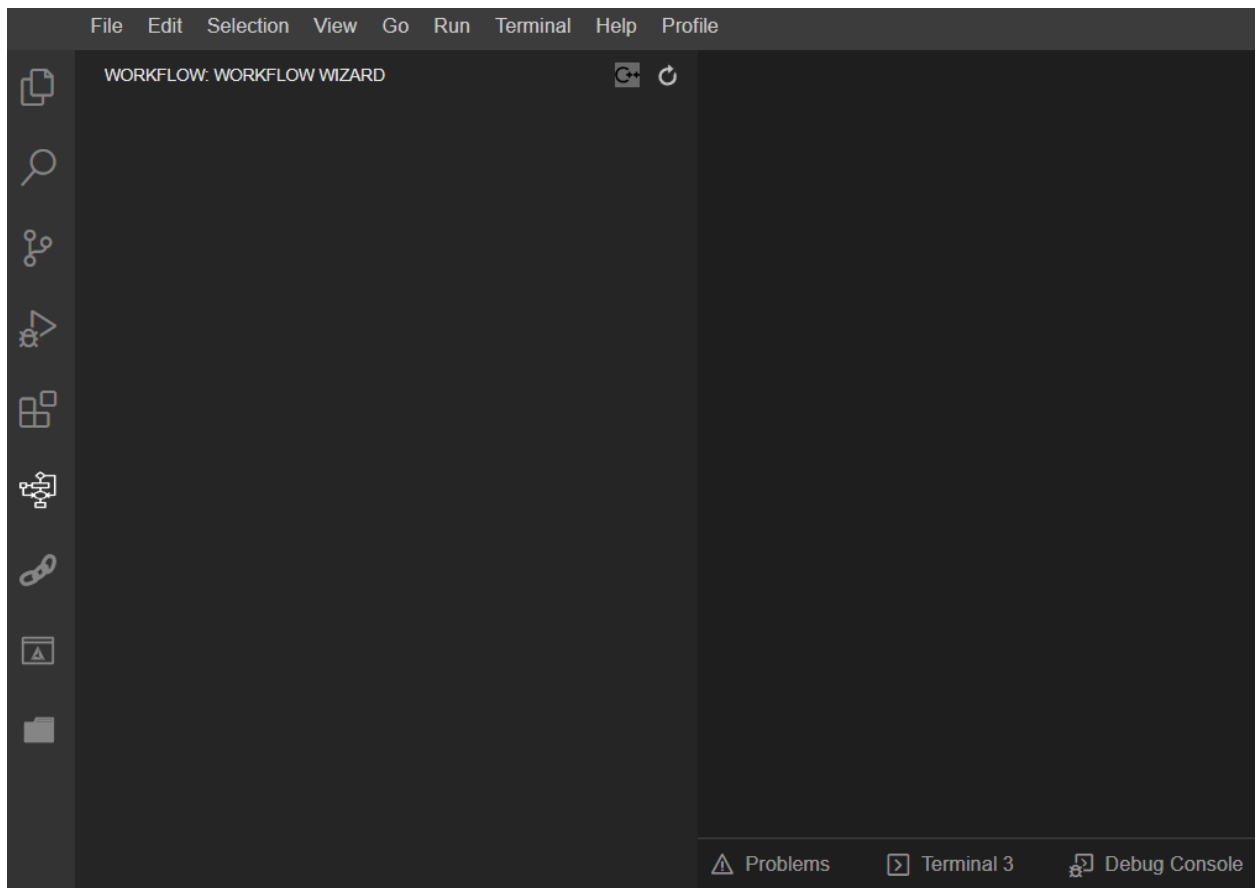
Lorem ipsum dolor sit amet, consectetur adipiscing...

Open

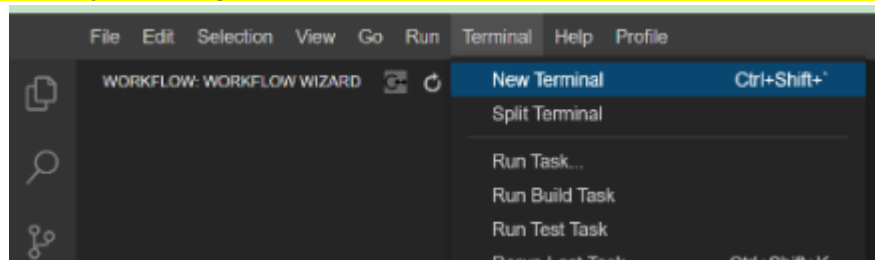
Teardown

www.codasip.com

- You are now in your workspace




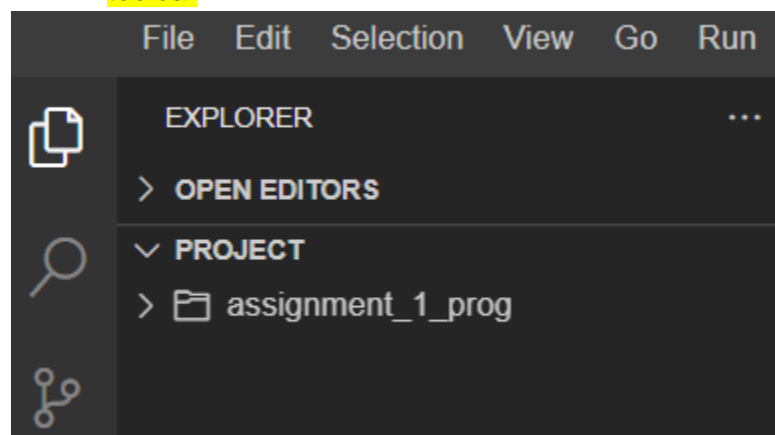
- You will need to import two repositories for this assignment from a GitHub repository.
 - [assignment_1_prg](#) : An assembly program project that will be used to learn how to generate Software Development Kits (SDKs), compile, debug, and export an assignment
 - [ia_risc32i](#) : An Instruction Accurate (IA) model of a 32-bit integer only RISC-V processor
 - You will need a terminal window in the bottom middle of your screen to import the files from GitHub. If you do not have a terminal window, you can open a terminal window by selecting “Terminal” in the top toolbar and then select “New Terminal”.




- To import a file from GitHub, you will be using the “git clone” command in the terminal window.
 - Type in (or copy and paste) the following command in the Terminal Window to import [assignment_1_prg](#)
 - `git clone https://github.com/CompOrg-RISCV/assignment_1_prog.git`

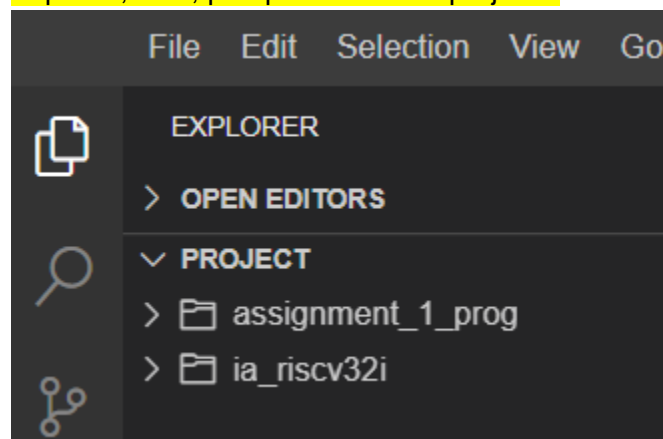
```
Problems Terminal 0 x
root@a8c28db64-371d-47d2-82c0-827592e106eb-deployment-eric-kellgp4r2:/home/project# git clone https://github.com/CompOrg-RISCV/assignment_1_prog.git
```


- After you “Enter” the command, `assignment_1_prog` should appear in your Explorer, , perspective under projects. You can get to the Explorer perspective by clicking on it in the left hand toolbar

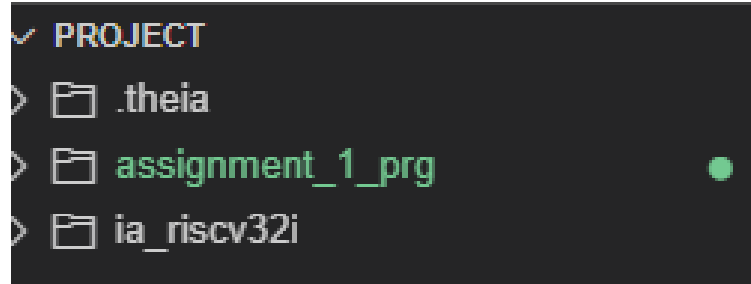


- Next, type in (or copy and paste) the following command in the Terminal Window to import `ia_riscv32i`

- `git clone https://github.com/CompOrg-RISCV/ia_riscv32i`
- After you “Enter” the command, `ia_riscv32i` should appear in your Explorer, , perspective under projects.



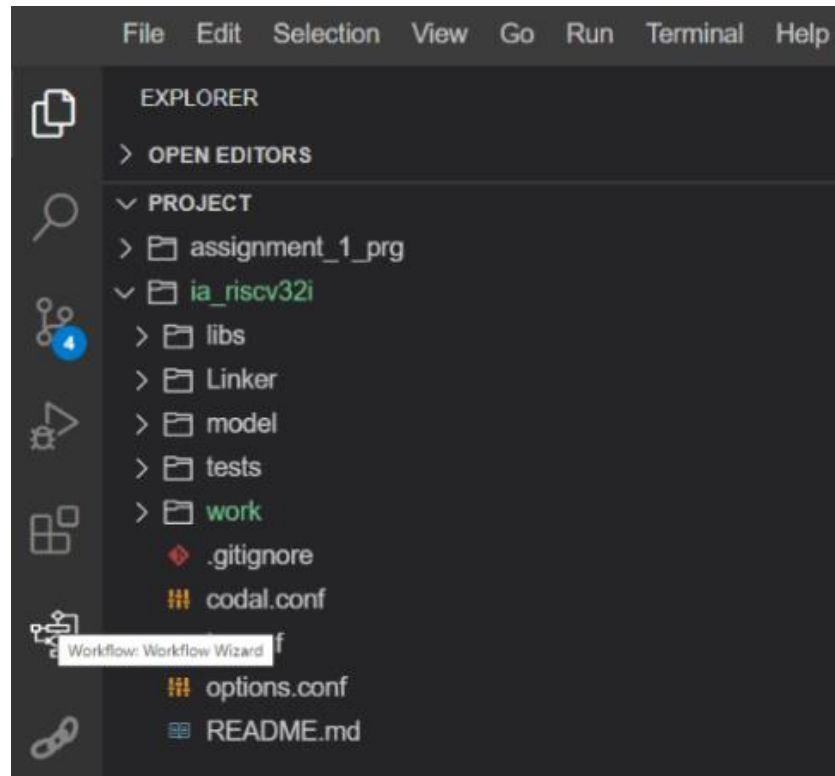
- Once both repos have been imported from GitHub, navigate to the Explorer tab  in the upper lefthand corner of your browser. When you enter the Explorer perspective, find the project banner. Within it, there should be two projects that have been imported.



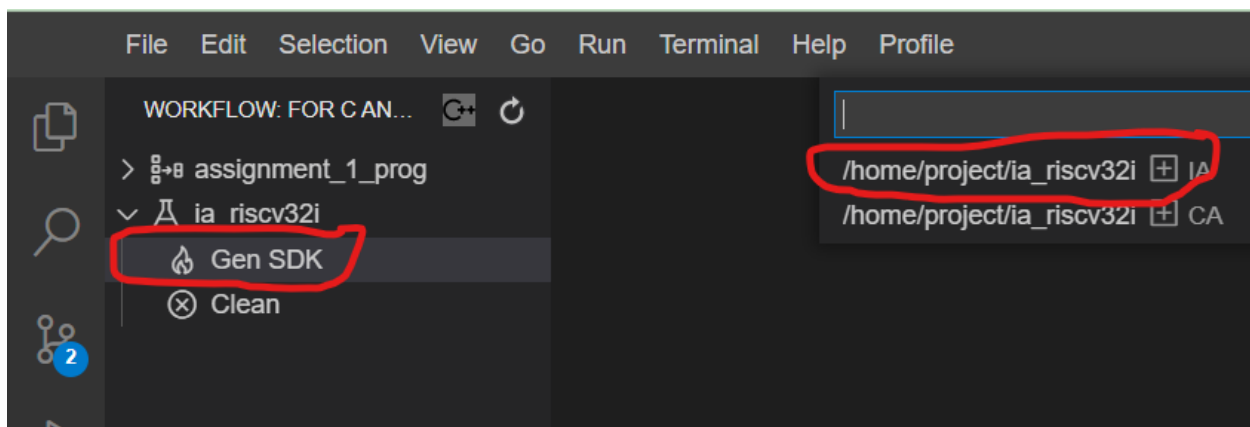
- Why is RISC-V becoming a universal Instruction Set Architecture (ISA)? (From the RISC-V Reader: An Open Architecture Atlas" by David Patterson and Andrew Waterman)
 - Suit all sizes of processors from tiny embedded to High Performance
 - Work well with a variety of popular software languages and stacks
 - Accommodate all implementation technologies from Field Programmable Gate Arrays (FPGAs) to full-custom Application Specific Integrated Circuits (ASICs)
 - Efficient for all microarchitecture styles including microcoded, hard-wired, in-order, out-of-order, dual issue, etc.
 - Support extensive specialization for customized accelerators
 - Stable base ISA that does not change

Checkpoint 1: Modifying and debugging an assembly program


- You will use the [assignment_1_prg](#) project provided with this assignment for **Checkpoint 1**. It is a simple assembly routine to demonstrate how to modify a program, build the project, and use the debugger to validate correct operation of the Instruction Accurate (IA) model.
- Before you can run the assembly routine, you will need to build your CodAL project.
 - Switch to the "Workflow: Workflow Wizard" tab on the left hand side of the browser window. If you hover the cursor over the different tabs, the name will be highlighted.



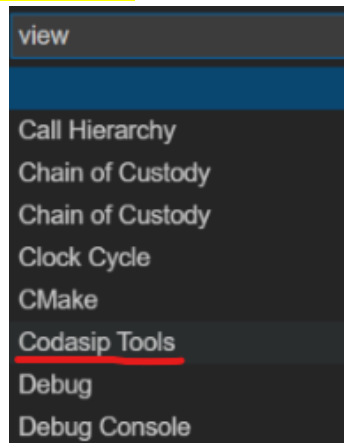
- You will first need to build the tools for the Instruction Accurate (IA) model so that the assembly program can be compiled.
- There are two ways that this can be done and they will be outlined below.
 - The first way to do this is through the “Workflow: Workflow Wizard” perspective.
 - Within the expanded project, locate “Gen SDK” and click on it. A banner will appear at the top of the screen and you will want to click on the IA model.




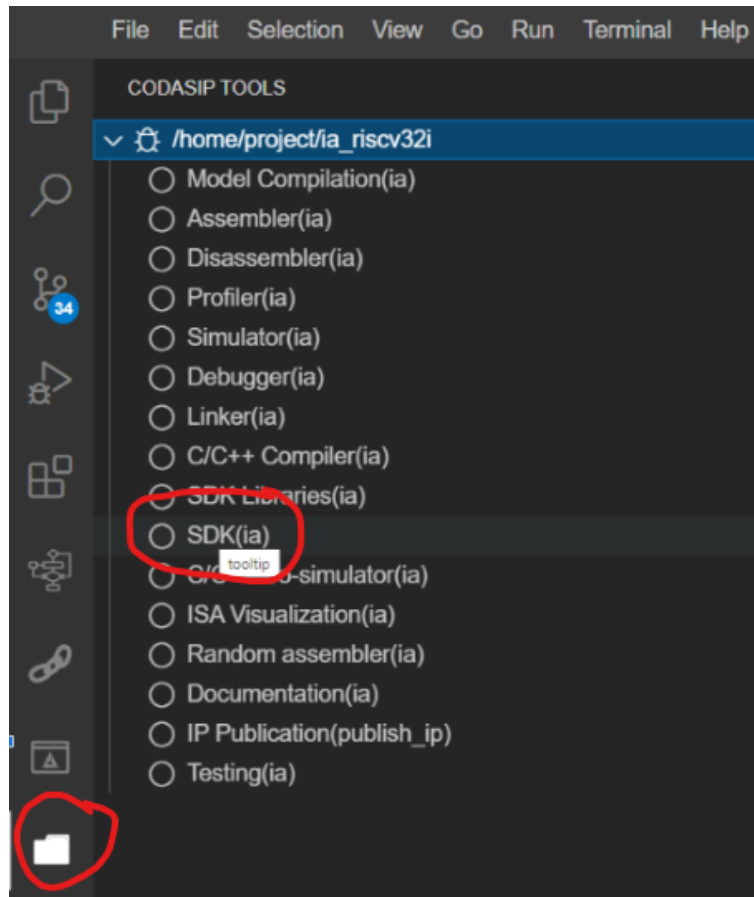
- That will build the IA SDK for the project. It will take approximately 2 - 5 minutes to build and you can monitor its progress through the “Task: Generate SDK” tab in the lower middle of the browser window.

- The second way to do Generate the SDK is through “Cudasip Tools” that you can navigate to via the left hand toolbar, the one that looks like a folder, 

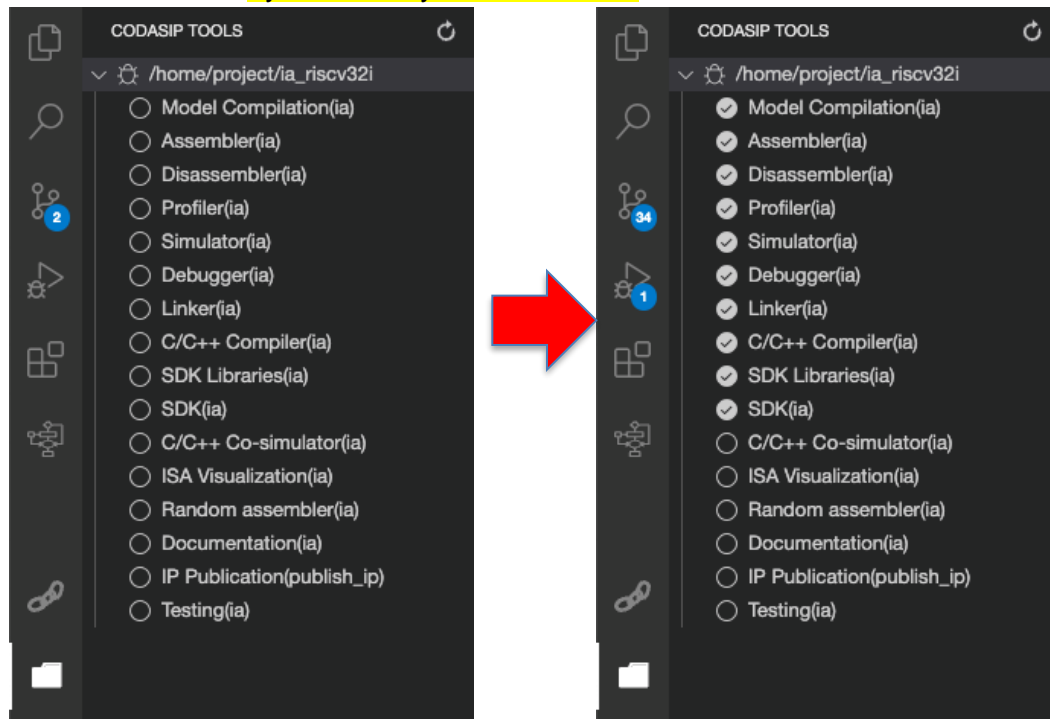
- If the “Cudasip Tools” is not available on the left hand toolbar, you can add it by navigating to “View” on the top toolbar, select “Open View...” and then select Cudasip Tools. That will add the tab on the left side of the screen.



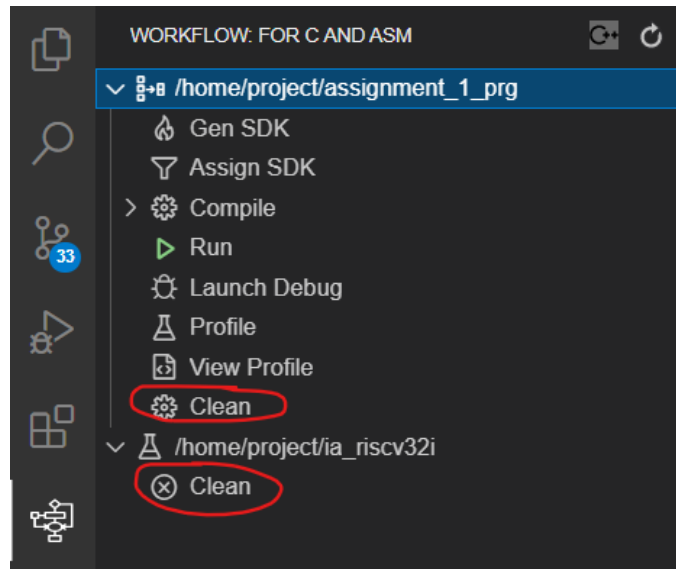
- Now that it's been added, click on the “Cudasip Tools” tab . Within the tab, select/click SDK(ia). This will build the SDK for the IA model as we did above. After it has been built, click on Refresh Data and you will notice what tools have been built.



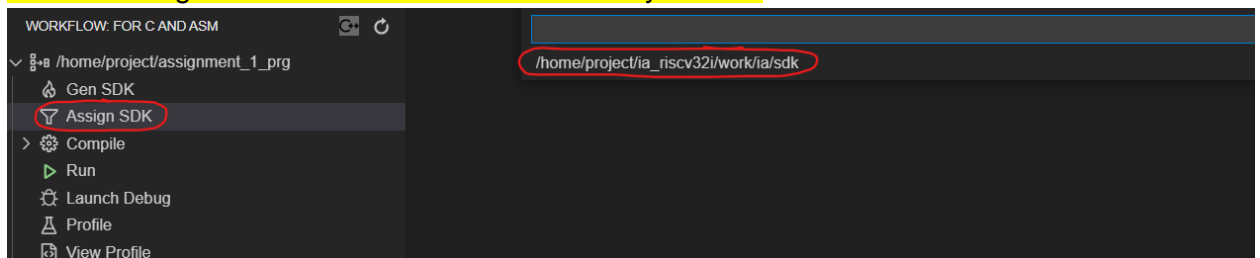
- NOTE: If your project does not appear in the Codasip Tools tab, try to refresh your browser tab



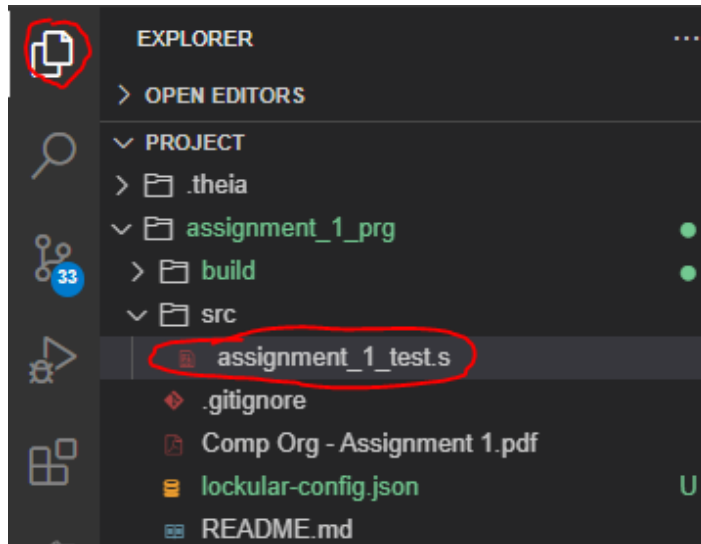
- If all the IA tasks are successfully built, they will be filled in with a check mark. If any IA tasks failed to build, debug the issue or request assistance. The imported [ia_riscv32i](#) project should build successfully.
- Debug Note: If you get an error with the Cmake file, try to clean the project. That can be done in “Workflow: Workflow Wizard” by clicking the clean icon as shown below.



- With the Software Development Kit (SDK) built, you need to assign the SDK to [assignment_1_prg](#) to use it. To do this, navigate back to “Workflow: Workflow Wizard.” Click on Assign SDK and then select the IA SDK you built.



- Now that the SDK has been assigned, when you actually execute the code, it will use the assigned SDK. For this assignment you are using the ia model, however from assignment 5 forward, you will be using the ca model's SDK.
- Navigate to the Explorer tab on the left side of the screen, by default it is the top tab. Next, expand the [assignment_1_prg](#) project. Locate the src folder and expand it, you should see [assignment_1_test.s](#). Open the file by double clicking it. The file extension .s signifies that the file is an assembly routine and not a c-program.



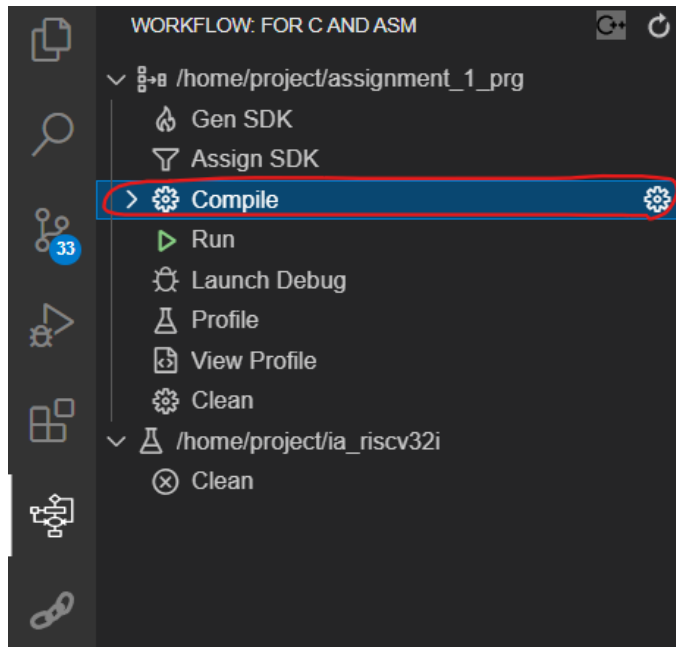
- Replace the Author's name with your name
- Replace the 'k' and 'g' with your initials within the the single quotes in lines 14 and 16


```

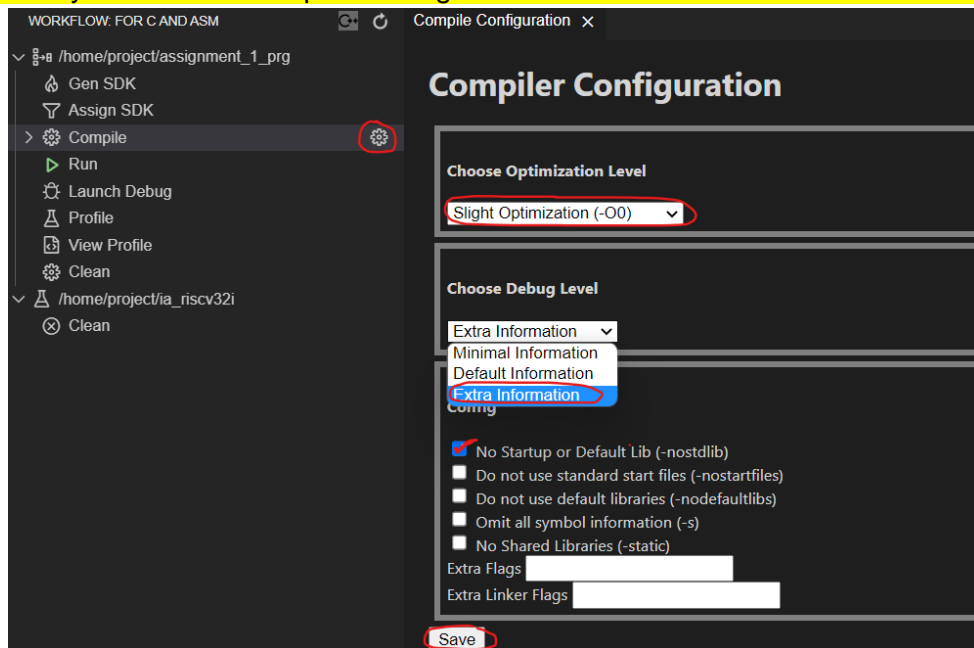
4  * Created on: April 18th, 2022
5  * Author: Keith Graham
6  */
7
8  // Section .crt0 is always placed from address 0
9  .section .crt0, "ax"
10
11 _start:
12     .global _start
13
14     addi    x4, x0, 'k'           // place your first initial, 'lower case', between the single quotes such as 'k'
15     slli    x4, x4, 8
16     addi    x4, x4, 'g'           // place your last initial, 'lower case', between the single quotes such as 'g'
17 TEST_LABEL:
18     slli    x4, x4, 8
19     addi    x3, x0, 0xa5          // load 0xa5 into x3
20     nop                                           // use the register view and record the value in register x3
21     or      x5, x4, x3
22     nop                                           // use the register view and record the value in register x5
23     nop
24     halt

```

- Now that the project has been properly edited, it's time to build it. To do this, navigate back to "Workflow: Workflow Wizard."
- Before you begin to compile your `assignment_1_prg`, you will want to enable support for C and assembly debugging as well as CodAL debugging. CodAL debugging enables you to set breakpoints within your hardware implementation pipeline .codal files and to step through your CA model. In later assignments, you will be implementing a 5-stage RISC-V32I pipelined processor. Within the "Workflow: Workflow Wizard" tab, expand the `assignment_1_prg` and locate compile

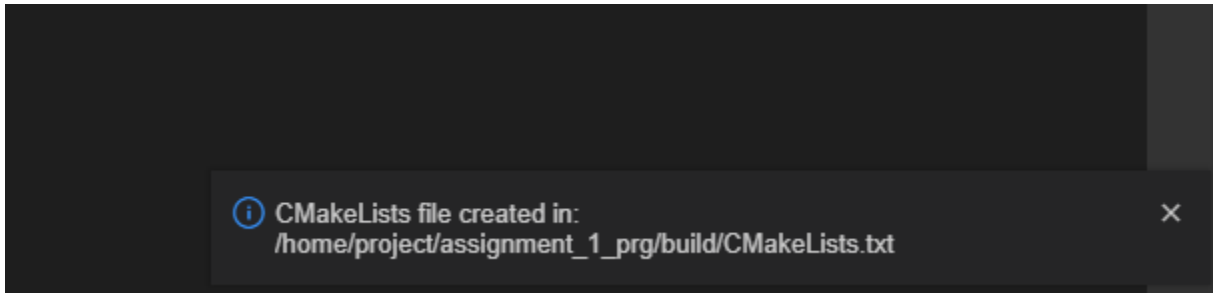


- Hover the cursor over compile and click the cog, , on the right hand side. That should pull up the Compiler Configurations window. Within Compiler Configurations, make sure the optimization level is -O0, the debug level is Extra Information and “No Startup or Default Lib” is checked. Finally **click save** which will exit you from the Compiler Configuration screen when the save has occurred.

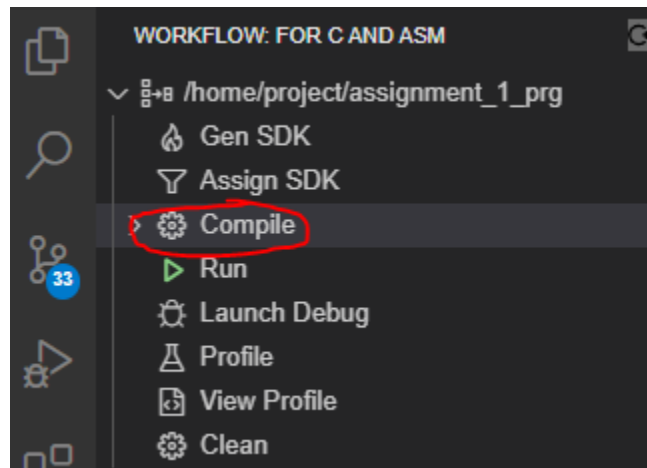


- It may take 5 to 15 seconds to save
- Note: Whenever you make changes, there will be messages that pop up in the bottom right hand corner like the screenshot below. This is nothing

to be worried about, it's simply part of cloud studio informing you that a task has completed.



- With the Compiler Configuration saved, it is time to click Compile to build your assembly routine.



- Note: The program should compile without warnings. However, there may be warnings that are related to Cloud Studios. If you see these warnings, you can simply ignore them and move on. Below is what a successful build looks like (with the acceptable warnings).

```
> Executing task: cd /home/project/assignment_1_prg/build && cmake . && make <

-- Configuring done
-- Generating done
-- Build files have been written to: /home/project/assignment_1_prg/build
Scanning dependencies of target main.xexe
[ 50%] Building ASM object CMakeFiles/main.xexe.dir/home/project/assignment_1_prg/src/assignment_1_test.s.o
tl_assignment1-ia-clang: warning: argument unused during compilation: '-D CODASIP_STUDIO_VERSION_MAJOR=9' [-Wunused-command-line-argument]
tl_assignment1-ia-clang: warning: argument unused during compilation: '-D CODASIP_STUDIO_VERSION_MINOR=1' [-Wunused-command-line-argument]
tl_assignment1-ia-clang: warning: argument unused during compilation: '-D CODASIP_STUDIO_VERSION_PATCH=1' [-Wunused-command-line-argument]
tl_assignment1-ia-clang: warning: argument unused during compilation: '-fdwarf-exceptions' [-Wunused-command-line-argument]
tl_assignment1-ia-clang: warning: argument unused during compilation: '-O1' [-Wunused-command-line-argument]
[100%] Linking C executable main.xexe
tl_assignment1-ia-clang: warning: argument unused during compilation: '-rdynamic' [-Wunused-command-line-argument]
[100%] Built target main.xexe

Terminal will be reused by tasks.
```

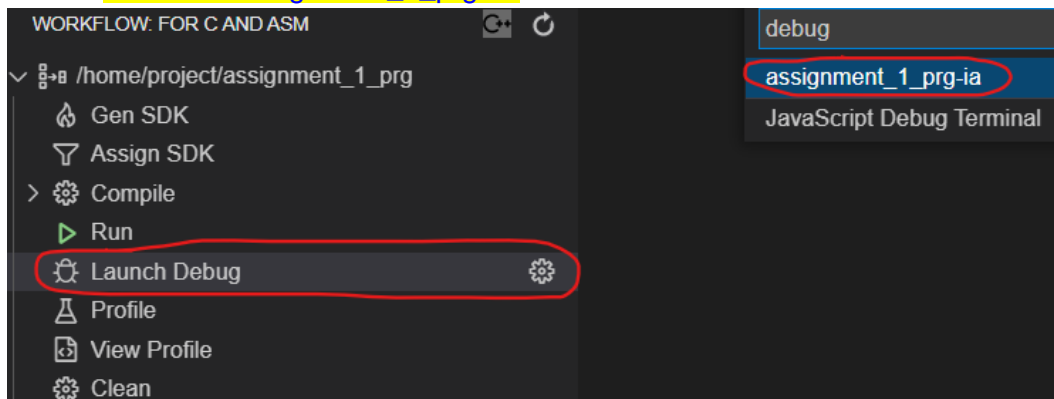
- Before you debug your assembly project, place two breakpoints in your assembly project. Go back into your `assignment_1_prg.s` file, and place two breakpoints by clicking to the left of line numbers 15 and 17. Before you click, you can see a time red dot indicating that you can place a breakpoint at this location.

```

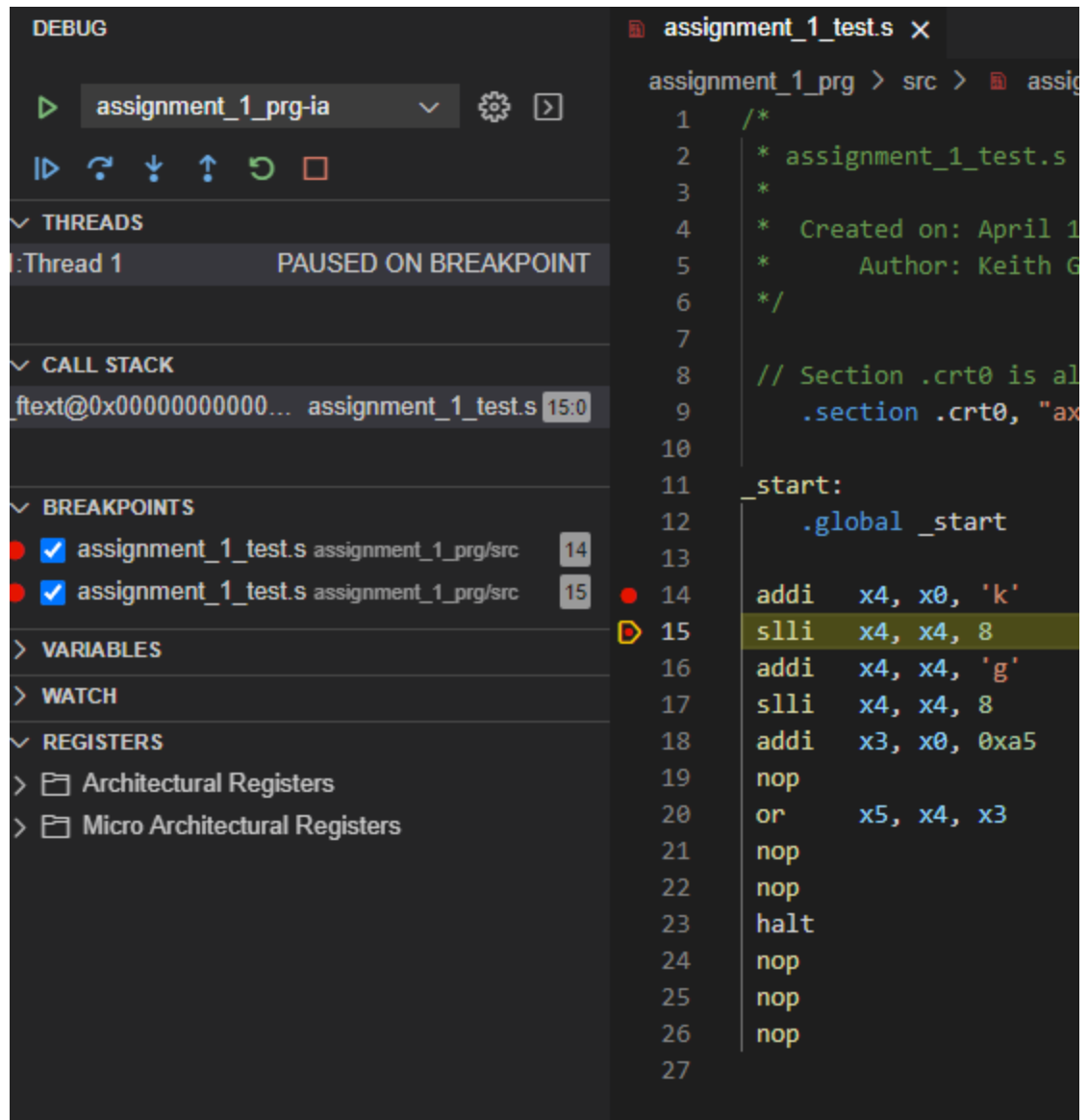
11  _start:
12      .global _start
13
14      addi    x4, x0, 'k'
15      slli    x4, x4, 8
16      addi    x4, x4, 'g'
17      slli    x4, x4, 8
18      addi    x3, x0, 0xa5
19      nop
20      or      x5, x4, x3
21      nop
22      nop
23      halt

```



- With the IA model built, your test program compiled, and initial breakpoints set, you will now debug your first project in Cudasip Cloud Studio.
 - With the breakpoints added, you can launch the debugger through the “Workflow: Workflow Wizard,” by clicking on launch debug. A banner will appear at the top, select the [assignment_1_prg-ia](#).

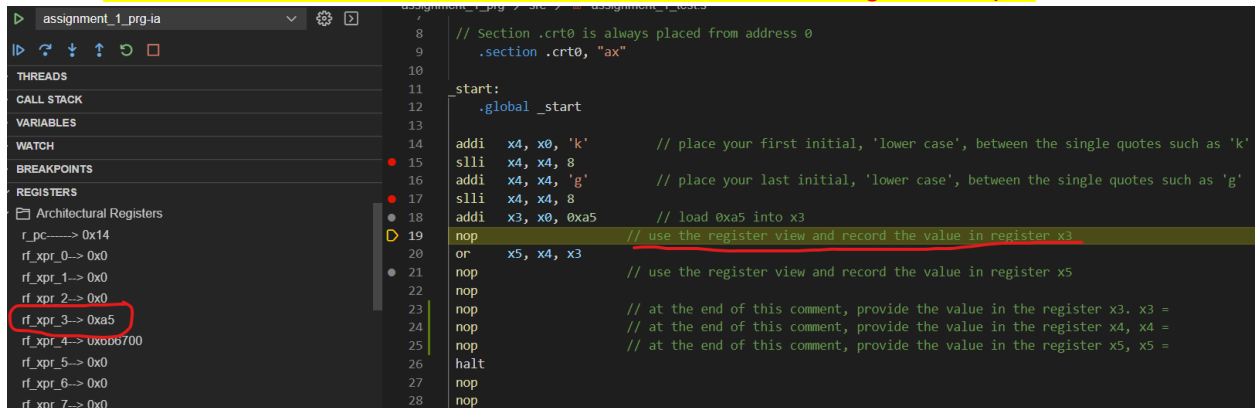


- Debug [assignment_1_prg](#). Now that we have launched debugger, the tab should change to the debug perspective as shown below.



- In the debug tab, locate the registers. Within this window, there are two types of registers
 - Architectural Registers: Registers defined by the RISC-V Instruction Set Architecture (ISA). These include the Program Counter (PC) and the Register File
 - Microarchitectural Registers: These are the registers that have been created to implement the Cycle Accurate (CA) model such as the pipeline registers
 - “In computer engineering, **microarchitecture**, also called computer organization and sometimes abbreviated as **µarch** or **uarch**, is the way a given instruction set architecture is implemented in a particular processor.”
 - Wikipedia: <https://en.wikipedia.org/wiki/Microarchitecture>

- Expand to view the Architectural Registers to view the Register File. This view will enable you to see the Register File change as it is updated by the program.
- Single step the assembly program by clicking on either the “Step Into,”  or “Step Over,”  icons in the menu bar until it points to the first nop instruction on line 19. Per the comment on line 19, record the value in register rf_xpr3



```

8 // Section .crt0 is always placed from address 0
9 .section .crt0, "ax"
10
11 start:
12 .global _start
13
14 addi x4, x0, 'k' // place your first initial, 'lower case', between the single quotes such as 'k'
15 slli x4, x4, 8
16 addi x4, x4, 'g' // place your last initial, 'lower case', between the single quotes such as 'g'
17 slli x4, x4, 8
18 addi x3, x0, 0xa5 // load 0xa5 into x3
19 nop // use the register view and record the value in register x3
20 or x5, x4, x3 // use the register view and record the value in register x5
21 nop
22 nop
23 nop // at the end of this comment, provide the value in the register x3. x3 =
24 nop // at the end of this comment, provide the value in the register x4, x4 =
25 nop // at the end of this comment, provide the value in the register x5, x5 =
26 halt
27 nop
28 nop

```

Architectural Registers

- r_pc → 0x14
- rf_xpr_0 → 0x0
- rf_xpr_1 → 0x0
- rf_xpr_2 → 0x0
- rf_xpr_3 → 0xa5**
- rf_xpr_4 → 0x000700
- rf_xpr_5 → 0x0
- rf_xpr_6 → 0x0
- rf_xpr_7 → 0x0

- Step through the project until the “halt” instruction, line 23, but do not execute the “halt” instruction. Executing “halt” will close out the debugger
- Record the values in rf_xpr3, rf_xpr4, and rf_xpr5 at the end of the comments on lines 23, 24, and 25.

```

22 nop
23 nop // at the end of this comment, provide the value in the register x3. x3 =
24 nop // at the end of this comment, provide the value in the register x4, x4 =
25 nop // at the end of this comment, provide the value in the register x5, x5 =
26 halt

```

- Save your file by going to File in the top toolbar and selecting Save All

- You have now completed the activities for Assignment 1

-
- Complete the assignment
 - Download your assignment_1_test.s assembly program by going to the Explorer perspective and right clicking assignment_1_test.s and selecting Download
 - Once the file is downloaded change the name to the standard name: first initial, last name, phase number (e.g., ekeller1)
 - Submit the .s file to the assignment on the Canvas course website.

