**RISC-V Class Project Phase 4 – ADD + ADDI Schematic**

Phase 4 of the Class Project is the beginning of the creation of a true Cycle Accurate model of the RISC-V processor. This first phase will build a schematic of a very simple version of the processor, which executes only three instructions – ADD, ADDI and HALT (which is not a true RISC-V instruction but necessary for our testing). The schematic is a diagram showing all of the hardware connections in the processor, which will then be used to create the Codal language description of the processor which can be built and simulated in Codasip. This and subsequent schematics will be based on the simplified block diagrams presented in the textbook.

The schematic will be created using a free web-based tool from Google called app.diagram.net. The following steps should be implemented to produce the first schematic.

1. **Copy the Generic Schematic and Rename It**

Copy the schematic Phase4\_orig.xml from the Class Project -> Phase 4 area in Canvas. Rename it to standardname4.xml so that it can be correctly submitted. Alternatively named schematics will not be accepted.

1. **Open app.diagram.net and the Schematic**

Open a browser and go to [https://www.app.diagram.net](https://www.draw.io). Select Device as the save location the first time you start app.diagrams.net. Select “Open Existing Diagram” from the Device (the Device is your local computer). Select the standardname4.xml file imported in step 1. At this point you will have a set of 6 schematic pages (IF Stage through Library) selected by tabs at the bottom of the drawing. The first 5 pages each contain one of the five pipeline stages of the RISC-V processor (IF, ID, EX, ME and WB). the Library page contains the component library which is where all components required for the schematics are kept.

Each schematic has a Title Block in the lower right corner. In each Title Block change the “Created by:” section to your name, and the date to the appropriate date. The Library page should not be modified.

1. **Examine the Component Library**

The Component Library in the Library page contains a number of components used to create the schematic. Each component is described below. They are divided into sections based on the component type.

* 1. **Signal Labels**

Every signal connection in Codasip must be named, and the Signal Labels group provides elements for this. Codasip connections are of two types:

1. A Signal, which is the output of a combinational block. Signals are always named in the format s\_STG\_SIG, where STG is the stage where the signal is used (if, id, ex, me or wb) and SIG is the specific signal name. An example is s\_ex\_alu, which is the output of the ALU in the EX stage. It is recommended that SIG names be all lower case, short and not include any special characters except underscore. In many cases a specific signal name will be required.
2. A Register output, which is the output of a register or state element. Registers are always named in the format r\_STG\_SIG as described above. An example is r\_id\_pc, which is the Program Counter (PC) pipelined to the ID stage.

Registers must be the output of a pipeline register, created with a Register symbol as described in Section 3.2. Signals are the output of a combinational block symbol, and signals and registers may both be the input to any block.

The Signal Labels group has three types of labels, which all appear to be the same but are not. The “Internal” symbol is used to name signals running between blocks within a schematic page. The “Signals In” symbol is used to define signals which are inputs to a schematic page but come from another page, and therefore have a connection point on the right. The Signals Out symbol is used to define a signal (not a register) which goes to another schematic page, and therefore has a connection point on the left. The differences will be clarified in examples below.

* 1. **Register**

One of the key elements in the design is a register, which consists of one or more flip-flops which are state elements. All state elements are clocked by the same clock, which is therefore not shown in the schematic. Because a register may include several different signals and groups, the register of the schematic is comprised of three parts:

1. The first is the rectangle with an R in it, indicating the overall register.
2. Individual elements of the registers are indicated by the solid rectangle, which has an input and an output which must be a register type (r\_STG\_SIG). Several individual elements may be stacked under the overall register symbol indicating that they are all clocked together.
3. For clarity in the schematic, it is desirable to separate register elements vertically. The dashed rectangle may be used to tie all of the elements together visually.

The example in section 5.2 below, especially Figure 10, shows how these three parts are used.

* 1. **Multiplexors**

One of the most common elements is a multiplexor, which allows a signal to be driven from several sources. Two, three and four input versions are supplied. The input signals are connected to the (0, 1, 2, 3) inputs, the output comes from the O connection, and the input is selected by the connection on the top. Note that for our purposes all of the inputs are equivalent, and simply indicate they are selected by different values of the select input (i.e. input 0 isn’t necessarily selected when select == 0).

* 1. **Arithmetic Elements**

There are two types of arithmetic elements used. The ALU operates on two inputs (1 and 2) based on the OP connection, and produces the output of the operation defined by the OP input on the OUT connection and also sets the Z (zero) output to 1 if the OUT output is 0, and sets the Z output to 0 otherwise. The Adder is a simpler function which just produces the sum of the 1 and 2 inputs.

* 1. **Decoder**

The processor includes a single Decoder element, which receives a portion of the instruction being executed and produces a number of control signals which are used in many areas of the design. The top symbol in this group includes the instruction input connection and one of the output connections. If more output connection (i.e. more control signals) are required, copies of the second element should be stacked below the basic Decoder element. Maintaining correct alignment is useful to help keep the schematics readable.

* 1. **Control (Combinational) Blocks**

Control blocks are used to define combinational functions, in which some number of input signals are used to generate some number of output signals. As with the Decoder, the first element creates the basic control block, with one input and one output, and the CTRL name should be changed to a name for the specific control function. Additional inputs and outputs are added by stacking one or more of the elements below the basic element.

* 1. **Register File**

This element is used to define the RISC-V general purpose Register File. Although the Register File is a single element, it is read in one pipeline stage (ID) and written in a different stage (WB). In order to make the behavior clearer in the schematic, the Register File is defined with two different symbols, one for the read connections and one for the write connections. This also matches the Codal implementation.

* 1. **Memory**

This element is used to define the RISC-V Instruction and Data Memories. Although these are each a single element, the Codasip memory modules used have the address and operation defined in one pipeline stage and the data read or write operation defined in the next stage. As with the Register File, these memories are defined with two different symbols, one for the address operation and one for the data operation. This also matches the Codal implementation.

1. **Creating a Schematic Basics**

Creating a schematic includes two functions – copying the required components from the library to the desired schematic page, and adding the signal connections between the components.

* 1. **Adding a Component**

Components are added by copying them from the component library on the Library page. Each component is a set of symbols which are grouped using the app.diagram.net grouping function. This does not work as well as it could, so some care must be taken when dealing with the components. The main issue is that the individual elements of a group can still be selected and moved independently, which should be avoided.

Normally, left clicking close to the body of the component will select the group. When this happens, a single rectangle with blue dots will be highlighted as shown in Figure 1.

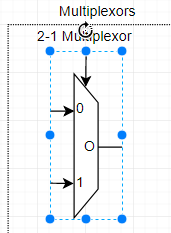


Figure 1

Clicking within the element body may select only an element, which will be highlighted by a dashed blue box as shown in Figure 2. If this happens, move outside the body and click again. Note that in some cases the rectangle with blue dots will appear to be larger than the component, but this is simply an issue with app.diagram.net and can be ignored.

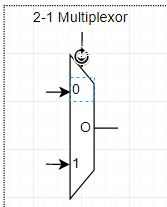
******

Figure 2

Once a component is selected, copy it with CTRL-c (or right click and select Copy). Go to the desired schematic page and add the component and paste it with CRTL-v (or right clock and select paste). The component will be placed in the location matching its location on the Library page, and it can then be moved into the desired location. Moving the component with the mouse will keep it on the desired grid and is recommended, as moving with the arrow keys can get it off the grid and complicate future placements. As with selecting a component in the Library, be careful that you have selected the entire component and not an element. Ctrl-z can be used to undo an incorrect operation.

One minor issue with app.diagram.net is that when a component is copied (for example, from the Component Library) to another page, it is placed in the same position on the destination page as it occupies on the source page. This requires moving each component after placement.

* 1. **Adding a Connection**

Once some components have been placed, adding signals to the connection points is straightforward. Hover the mouse near the desired connection point, until a green disc appears. There will be either an arrow cursor or a four arrow cursor – either is fine to make the connection. Once the green disc appears, hold down the left mouse button and drag to the other desired connection point. Make sure to select a connection point on the component (another green disc). Once the mouse button is released, the connection will be made with rectangular segments. Each segment may be dragged horizontally or vertically to optimally position the connection. Connections should not run over or under any components. If the end point of the connection (a small X in a blue circle) is not correctly connected, it can be dragged into the proper position.

To connect a component to an existing line, start the connection with the component and then simply drag to the desired point on the existing line. Note that there will not be a green connection disc in this case – the lines will simply touch.

* 1. **Labeling a Connection**

Each signal connection should be labelled. For signals which are inputs to the schematic page, the correct label (signal or register) should be copied from the Signal In part of the Signal Labels group in the component library to the schematic page, to the left side of the schematic. For signals which are outputs from the schematic page, the signal label should be copied from the Signal Out part of the Signal Labels group in the component library to the schematic page, to the right side of the schematic. Note that these labels should be added prior to making a connection, as each label has a connection point in the proper place. Also note that register outputs are included in the register symbol itself.

For signals created between components on the schematic, copy the signal label from the Signals Internal part of the Signal Labels group, and position it slightly above the connection.

As the labels are added, double click them and create the desired name. Signals must begin with “s\_” and register outputs must begin with “r\_”. The correct stage must then be added, followed by an underscore and the desired signal name.

* 1. **Navigating the Schematic**

There are several handy shortcuts which facilitate moving around in a schematic:

1. The mouse wheel scrolls the schematic up and down.
2. Shift plus the mouse wheel or a mouse wheel horizontal motion (if available) scrolls the schematic left or right.
3. Control plus the mouse wheel zooms the schematic in or out.
4. Right click and drag moves the schematic.
5. **Create the Real Schematics**

Using the operations described in Section 4, we will now create the full set of schematics for the ADD + ADDI phase of the RISC-V design. The operations on each page will be described in individual sections.

Figure 3 shows some rough sketches of what the 5 schematic pages should look like. Following this arrangement will make it easier to add components in future Phases.

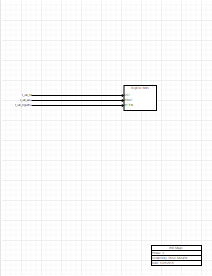
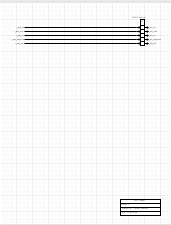
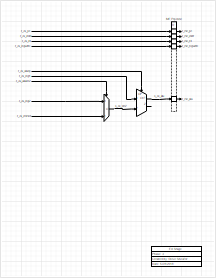
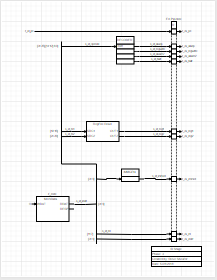
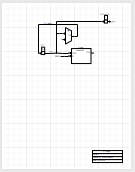


Figure 3

The basic RISC-V processor block diagram is shown in Figure 4.33 in the textbook, shown below in Figure 4. Since only the ADD and ADDI instructions are required for Phase 4, a number of elements of this diagram, particularly the logic for branches and the Data Memory, are not required and can be eliminated as shown by the red deletions in Figure 5.

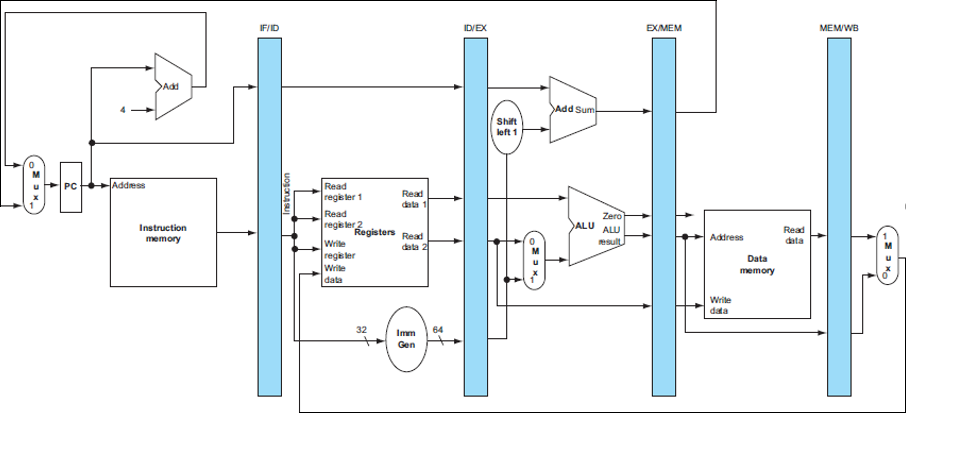


Figure 4

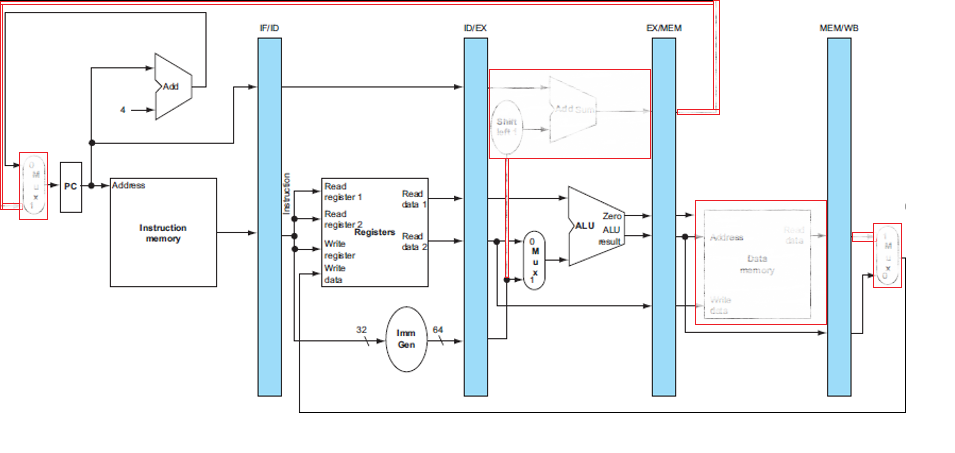


Figure 5

* 1. **IF Stage**

The IF stage contains the Program Counter (PC). The PC itself is included in the Phase4\_orig schematic, since it is handled slightly in a slightly different way from the normal pipeline registers. This stage also includes the ID Stage pipeline register, which has only one register which is a pipelined version of the PC used for debug and in future Phases.

Add the remaining elements of this stage by copying them from the component library as described in Section 4.1.

1. Add the Address part of a memory block as shown in Figure 6. This is the address function of the Instruction Memory. Note that this component does not produce the actual instruction output, which will come from the Data part of the memory in the ID stage. Since the Instruction Memory is read-only and we always read 4-byte instructions, the OP input can be connected to a constant READ4 value (use a Signal In symbol for this). The STAT output can be left unconnected for now. The Instruction Memory needs an identifier, so add an Internal Signal symbol (which is just a text block) named if\_code and place it near the memory symbol.

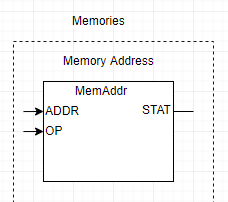


Figure 6

1. Add an Adder component as shown, naming the output signal as s\_if\_pcin.
2. Connect the signals. Note that the address connection to the memory (which is r\_pc) can be made anywhere on the existing connection.
3. Use another Signal In symbol to show the constant “4” which is the second input to the Adder, since we always add 4 to the PC to produce the next sequential address.
   1. **ID Stage**

The ID stage is the most complex part of Phase 4. Several steps are required to create this page. The page in the Phase4\_orig schematic includes the passing of r\_id\_pc to the EX Stage pipeline register. Note that the pipeline register extension box is also included, as other registers will be added to this page.

1. Add the Data part of the Instruction Memory. Note that this is slightly different than the block diagram of Figure 5 because the Codasip memories are split across two stages. Like the Address part in the IF Stage, the Data part should have a label of if\_code. To allow room for other ID stage logic, this memory block should be in the lower left part of the schematic. The WDAT input of this memory can be connected to the constant 0 since the Instruction Memory is never written. The RESP output can be left unconnected for now.
2. The RDAT output of the Instruction Memory is the fetched instruction, which is effectively in the ID Stage pipeline but must be declared a signal because the register is implied within the memory block. Name the signal s\_id\_instr. Several fields will be extracted from this bus based on the instruction formats. To show this in a schematic, we use a heavier line for the bus signals, and make connections to that line with indicators of which bits are connected. Figure 7 shows what this should look like, with the heavy line representing the s\_id\_instr bus. A line is added by selecting the line symbol in the shape area to the left of the drawing, placing it, moving the two ends to the desired locations, and selecting the line weight in the Style tab on the right. 3 pt. is a good choice for this bus. Note that this weight becomes the default, so change it back to 1 pt. after placing the bus. Indicate which bits of the bus are part of each connection with [X:Y] as shown – in this case, the whole bus [31:0] is connected. Use a Signal Internal symbol, and it may be valuable to make it left justified in the Text tab to the right.

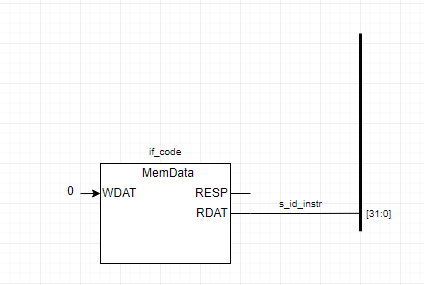
**

Figure 7

1. A major component of the ID Stage is the instruction decoder, which receives the portion of the instruction necessary to decode the operation (the OPCODE, FN3 and FN7 fields) and produces a number of control signals. This function is not shown in the pure data path of Figure 5 because it is a control function, so refer to Figure 4.17 of the textbook. The Decoder is shown as the “Control” function combined with the “ALU Control” function. Add the DECODER block and make a connection from it to the s\_id\_instr bus called s\_id\_opc, which is a subset of the instruction. Figure 8 shows how this should appear.

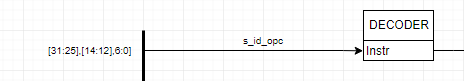
******

Figure 8

1. In Figure 4.17 and the subsequent description in the textbook there are several control functions produced by the Decoder, but for ADD and ADDI only four control signals are required:
   1. The ALU operation must be selected by the signal s\_id\_aluop.
   2. The ALU source 2 input must be selected between a register value (for ADD) and an immediate value (for ADDI) by the signal s\_id\_alusrc2.
   3. The RegisterFile write enable is selected by the signal s\_id\_regwrite.
   4. The HALT instruction decode is selected by the signal s\_id\_halt.

Note that the for ADD and ADDI the ALU operation is always ADD, and the register file is always written. We include s\_id\_aluop and s\_id\_regwrite in Phase 4 to simplify the implementation of future phases.

1. Because we need three control signals, add three additional Decoder outputs as shown in Figure 9. In subsequent phases we will need ~6 additional Decoder output signals, so leave some space below the symbol.

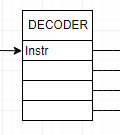


Figure 9

1. As shown in Figure 4.48 of the textbook, the control signals must be carried through the pipeline to the stage where they are used. Each control signal (s\_id\_XXX) must therefore go to a pipeline register (r\_ex\_XXX) in the EX pipeline. Add the three register symbols to the EX Pipeline, modify the names to be correct and make the connections to the decoder as shown in Figure 10.

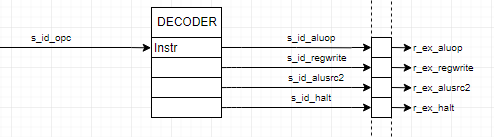


Figure 10

1. The Register File read operation occurs in the ID stage, so add the Register File read symbol. Connect the two source address inputs to the appropriate bits of the s\_id\_instr bus, naming them s\_id\_rs1 and s\_id\_rs2. The two register outputs are s\_id\_reg1 and s\_id\_reg2, which must be connected to pipeline registers r\_ex\_reg1 and r\_ex\_reg2. Leave room between the Register File and the register block to add some components in later stages.
2. The Immediate Generator shown in Figure 5 extracts the immediate value from the full s\_id\_instr bus (bits [31:0]) and produces an immediate value s\_id\_immed which is passed to the EX Pipeline as r\_ex\_immed. This is a general combinational function, so add a Control block and name it IMMGEN. For Phase 4 only one input and one output are required, so the basic control block is adequate. Note that since we use the entire s\_id\_instr bus, we don’t need to create a separate field name for it (although we could).
3. Figure 5 indicates that the write address of the Register File comes directly from the instruction, but that is not correct for a pipelined implementation as shown in Figure 4.39 of the textbook. As a result, add a pipeline register for the rd field (r\_ex\_rd) and connect it to the correct bits of the s\_id\_instr bus.
4. For debug purposes, we will want to have the current instruction available in each pipeline stage, so add registers r\_ex\_instr and connect it to the instruction bus.
   1. **EX Stage**

The EX stage includes only the two elements shown in Figure 5.

1. Add a 2-1 Multiplexor symbol to select the Source 2 value for the ALU. Its output should be a signal s\_ex\_src2. Connect the control and data inputs to the correct r\_ex\_XXX inputs from the ID Stage page.
2. Add an ALU symbol. Its output must also be a signal s\_ex\_alu. The Z output may be left unconnected for now. The OP control input is connected to r\_ex\_aluop.
3. The PC register of the ME pipeline is included in Phase4\_orig. Add any additional required pipeline registers, including r\_me\_instr for debug purposes and r\_me\_alu to carry the data result.
4. Note that the pipelined version of s\_id\_halt (r\_ex\_halt) is not used in the schematics, since it is for a special Codasip function.
   1. **ME Stage**

Since Phase 4 doesn’t utilize the Data Memory, the ME stage only contains the pipeline registers for signals which are needed in the WB stage. Add the necessary pipeline elements (there should be five of them).

* 1. **WB Stage**

Figure 5 indicates that there is nothing in the WB stage at this point, but because the Register File is written in the WB stage we must add the Register File Write symbol here. Connect its inputs to the correct signals from the ME stage.

1. **Saving the Project**

Use the File -> Export -> XML function to save the project to the Device, saving the file as standardname4.xml. Leave the default Compressed selection. Note that the resulting schematic will appear in the Downloads folder for Windows, and in a similar place on a Mac. It is recommended that schematics be moved to a separate area immediately. To open the schematic for further editing, use the File -> Open From -> Device function to select the schematic where it was saved.

1. **Scoring the Project**

The score for this phase will be calculated by deducting points from 100% for any errors in the schematic – primarily missing or incorrect connections and missing or incorrect signal names (~1-3% for each error). Phase 4 has a Target Date of Sunday, February 21 at 10:00 PM, and 1% will be added for each day before this date (up to a maximum of 7%) and 10% will be deducted for each day after this date when the schematic is submitted. Once the schematic is scored and any errors indicated, a correct version must then be submitted within 1 day, with an additional 10% deducted per day after 1 day.

1. **Exporting the Project**

Unlike the Codasip-based project Phases, for Phase 4 the final submission of standardname4.xml will be via a Slack Direct Message to the Professor (Steve Sheafor). Attach the file using the paper clip sign to the right of the message block. DO NOT submit projects to the #phase4 channel.