

SMOK Familiarization, Datapaths, and Control Signals

You may work in pairs for this lab

You will be building a series of components and saving some of them in SMOK containers. We will later connect these to one another to build a working simplified SRC machine which will be able to execute a reduced set of SRC instructions.

Lab Objectives:

In this lab you will:

1. Familiarize yourself with the SMOK tool. The following links have supporting documentation for the version of the tool that we are using:
 - a. **Component descriptions:** <http://www.cs.virginia.edu/~cs333/SMOKsetup/>
 - b. **Basic tutorial :** <http://www.cs.virginia.edu/~cs333/SMOKsetup/SMOKintroduction.html>
2. Reinforce the datapath and control signal concepts covered in Heuring Chapter 4

Deliverables:

1. **Inlab.** Before leaving lab, **complete the tables and checkoffs listed on the last page and submit them to the TA.** Since there are many of you and few TAs, please get several checkoffs at one time. If the TAs are busy helping others, keep making progress on the lab until a TA can get to you to sign your checkoffs.
2. **Postlab.**
 - a. **Connect the components** that you built in lab to one another in the following way and save it as **postlab2.smok**. (You should test it to make sure that it behaves as you expect.)
 - i. Connect the IR Decode to the register file so that register selected when the appropriate control signals are asserted allows data to be written or read from the register.
 - ii. Connect the IR to the memory register (MD). For now, your design should be able to allow the user to input a memory address to fetch from memory and the data at that address should be placed in the IR.
 - iii. Connect the register file R0 and R1 outputs to the ALU structure that you designed.
 - b. **Electronically submit by your next lab session.** (One week):
 - i. **All your .smok and .smokcont files** used for the lab
 - ii. **Postlab2.smok**
 - iii. A **text file** that contains the following information:
 1. Name (and Partner Name)
 2. Email ID (and Partner Email ID)
 3. Text description of the problems you encountered during lab
 4. (If applicable) Questions that you have as part of performing this lab exercise.

1. Familiarization with SMOK Tool.

- a. User interface
- b. Some available components we will work with
 - i. Adders
 - ii. ALUs
 - iii. Containers
 - iv. Debug input, Debug output
 - v. DeMux
 - vi. Registers
 - vii. Register Files
 - viii. Memory
 - ix. Memory Interface

2. Datapath and control signals

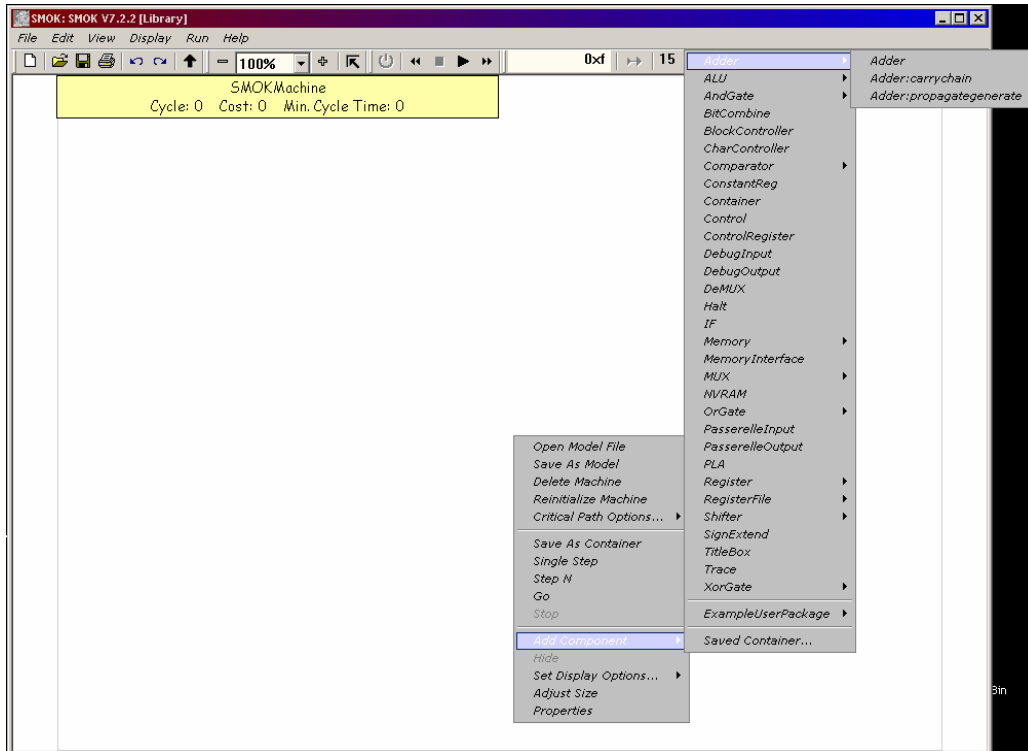
- a. The **datapath** consists of the functional units and resources necessary to perform computation on data given to the machine.
- b. Each functional unit and storage resource (registers and memory) must be given **control signals** to specify and control the direction of the data being transferred (into or out of the register/memory; in other words, read or write).

Lab Exercise:

Part I: Intro to SMOK Basics

1. **Start the SMOK tool.** The location of this tool depends on the lab you are in, but a shortcut likely resides on the Desktop or under the Start->Programs menu. A TA will be able to help you locate the tool if you are encountering difficulty. The executable is named **smok.exe**. This software is also available in the ITC computing labs. If you wish to install a version of SMOK on your own computer, a newer version (7.2.2.) is available at: <http://www.cs.washington.edu/homes/zahorjan/homepage/Tools/SMOK/index.shtml>

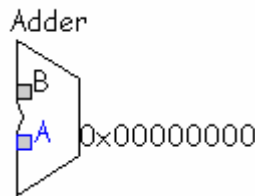
This *newer version has some differences* from the version available in the labs. **Your work will be graded on its ability to work in SMOK 7.1.6** which is what is installed in all the labs.



- a. SMOK will open a window with an empty canvas that contains only a title bar. The **properties** of this canvas can be changed. Right click and select **Properties** and change the **Name** to Adder.

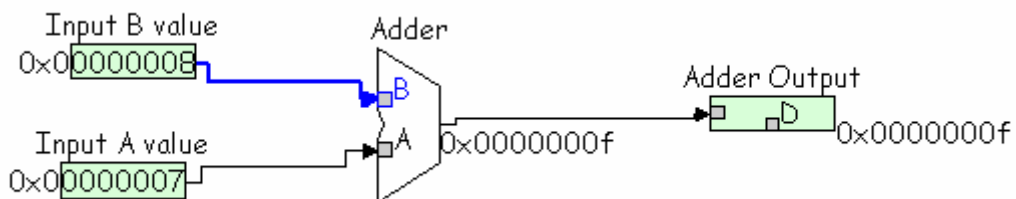
2. **Add components.**

- a. Right click and select **Add Component -> Adder**. An adder will appear on the canvas.

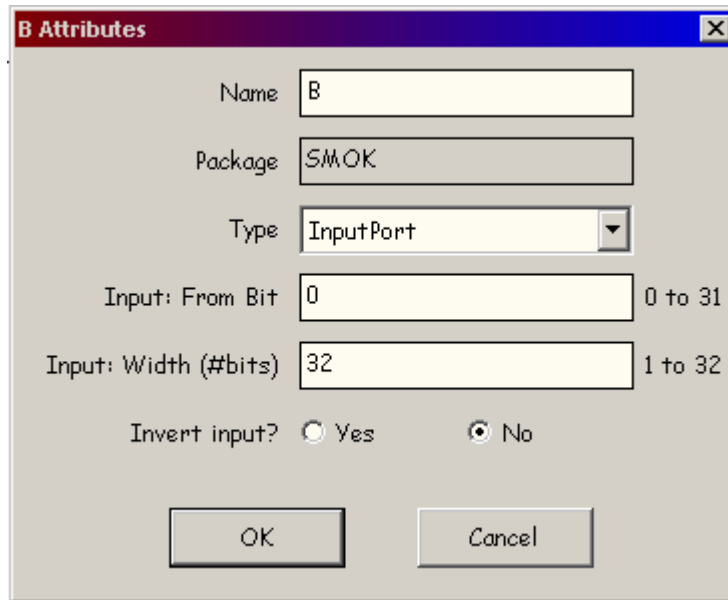


The two grey boxes represent the inputs to the adder. You can name the inputs by selecting an input, then right clicking and selecting **Properties**. Name the Adder inputs **A** and **B**. Components can be resized by selecting the component, then right-clicking and selecting **Adjust size**.

- b. Add two **DebugInput** components and a **DebugOutput** component. Rename them **Input A**, **Input B**, and **Adder Output**.



Notice that the starting input bit and the number of input bits (width) can be specified in the properties. For now, leave the input width specified as 32 bits.

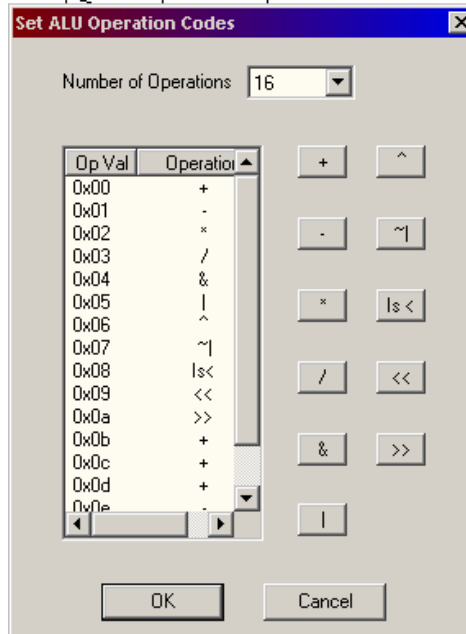


- c. **Create connections** to the adder inputs by selecting the destination, holding the CTRL key down and then selecting the data source. For example, for the adder's B input, select it, hold down CTRL and then click on the Input B DebugInput component. (See figure above).
- d. **Run** inputs through the design and watch the outputs. Select **Run**→**Single Step**. Prompt windows will appear to ask for input. Enter some values (assumed hexadecimal) and watch the Adder Output component to see the resulting value coming out of the adder.
- e. **Save your design**. Select **Save**. SMOK design files will have a **.smok** extension.

Part II: SRC Datapath Components and Control Signals

1. **ALU**. Using your existing Adder design. Delete the adder and replace it with an ALU component.
 - a. Change the title box to read: **SRC ALU**.
 - b. Go to the [online documentation](#) and read about the inputs and outputs of the ALU, particularly **Z**, **C**, and **Op** inputs. The outputs are distinguished from inputs by being colored yellow.
 - c. To match the SRC model in the book (Figure 4.7), rename the 2 inputs: **A** and **B** respectively.
 - d. Rename the ALU **C** output to **Carry** (which represents the carry bit of the ALU)
 - e. Name the output of the ALU result to **C** (to match the SRC model).
 - i. Connect a **Register** component to the ALU output port **C**.

1. **Notice** that there are 2 inputs to the register. Examine the properties to determine which is the one-bit control signal and name that control input **C In**
 - ii. Connect another Register component to the **A** input. Name the one-bit control signal input: **A In**
- f. Add and connect **ConstantRegs** and **DebugOutputs** to the various remaining inputs and outputs of the ALU. Rename them with descriptive names.
 - i. Notice that the **Op** input is only 4 bits wide. Adjust the DebugInput component that you connect to it to also be 4 bits wide.
 - ii. Adjust any other DebugInputs to the appropriate width to match the inputs they are connected to.
- g. Right-click on the ALU and select **Edit ALUops**. You will see a list of hexadecimal operation values which control the operation performed by the ALU. These operation mappings can be changed by selecting the Op Val and then selecting the operation to be mapped to that Op control signal.



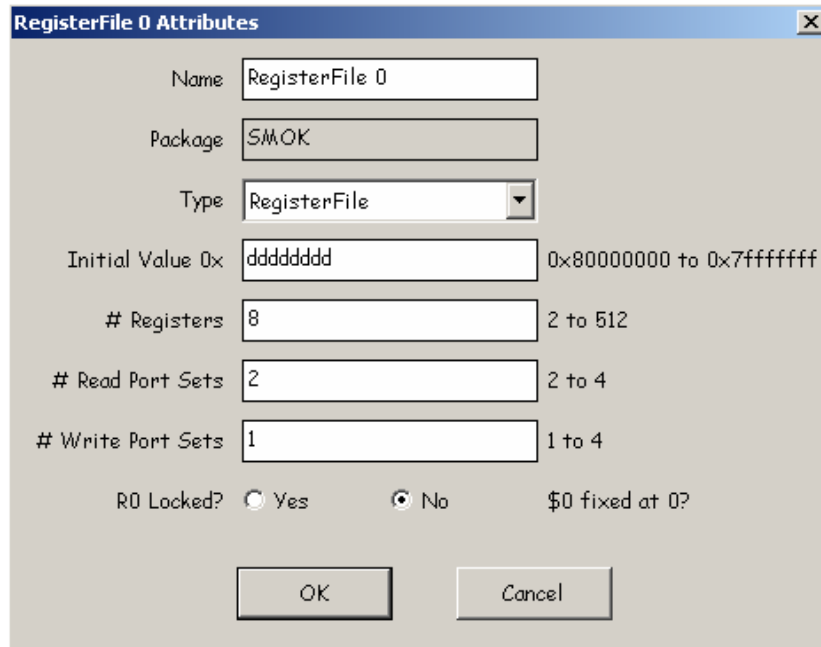
- h. **Run** your design with different A and B input values and complete the **ALU Results Table** on the last pages by changing the **A_{In}** and **C_{In}** values as specified in the table.
- i. **Save your ALU design as a Container.** Name it src.alu. SMOK containers have **.smokcont** as their file extension. A container basically allows a previous design to be included in another design. We will use it again soon.

2. Instruction Register Decoding.

- a. Start a new canvas and call it **IR Decode**. Using the left side of Figure 4.4 in the book (instruction register to the 5-to-32 decoder). The instruction register contains the bits that specify the operation to be performed and the location of operands that are needed by the instruction.
- b. Create a design which has a 32-bit register named **IR** and performs the logic to decode the 5-bit **ra**, **rb**, **rc** fields and add the control signals **Gra**, **Grb**, **Grc**. For reasons of limitations of the SMOK tool, **select the lower 4 bits from each field**.

The 4 bit register value coming out of the OR gate should be the input to the DeMux. It will cause the DeMux to select a line representing the register number chosen.

- i. Use the starting bit numbers indicated in Figure 4.4. You'll need to add:
 1. A register named **IR**. It should be 32 bits wide. Name its one-bit control signal **IR In**
 2. **AND gates** for each of the 4-bit ra, rb, rc fields. Make sure to label each of the inputs to the gates appropriately. Each AND gate should only select the appropriate 4 bits
 - a. One input should be appropriate 4 bits from the ra, rb, or rc field of the instruction register
 - b. The input to the AND gate should be the **Gra, Grb, or Grc** signal.
 3. An **OR gate** which merges the data coming out of the **AND** gates. Additional inputs to the OR gate can be specified by modifying the # inputs under the Properties for the component.
 4. A **DeMux** to decode the 4-bit input and activate one of the register selection outputs. You can modify the number of inputs and outputs of the DeMux by selecting it and selecting **Properties**.
 - a. The SMOK DeMux is limited to a maximum of 4 bits of input, which is why we will only take 4 bits from each of the ra, rb, rc fields of the instruction register.
 - b. The **S** input will take the 4-bit register number from the instruction register as input.
 - c. The **D** input of the demux is the data that needs to be passed through to the selected output specified by **S**.
 - d. Use a one bit **ConstantReg** component to specify the value that needs to be passed to the selected line specified by **S**.
 - c. Connect enough **ConstantReg** and **DebugOutput** components to test to see that your design correctly decodes values you enter in for the IR register.
 - d. Save this design as a **container** as well. You will be using this container in the next step.
 - e. Before you leave lab, **demo this to the TA for a checkoff** (See checkoff sheet)
- .
3. **Register File.**
 - a. On a new canvas, add a **RegisterFile** component. For now, specify that there are 16 32-bit registers. Any initial value in the registers is fine.
 - b. For this lab, we will only use the **R0#** input and **R0** output (we will not use R1# input and R1 outputs). The **WOE** input controls whether data sent to the **WOD** can be written to the register number specified at R0#. (See online documentation for more details and description.)
 - c. Read through the online documentation to see which attributes of the register file can be set:

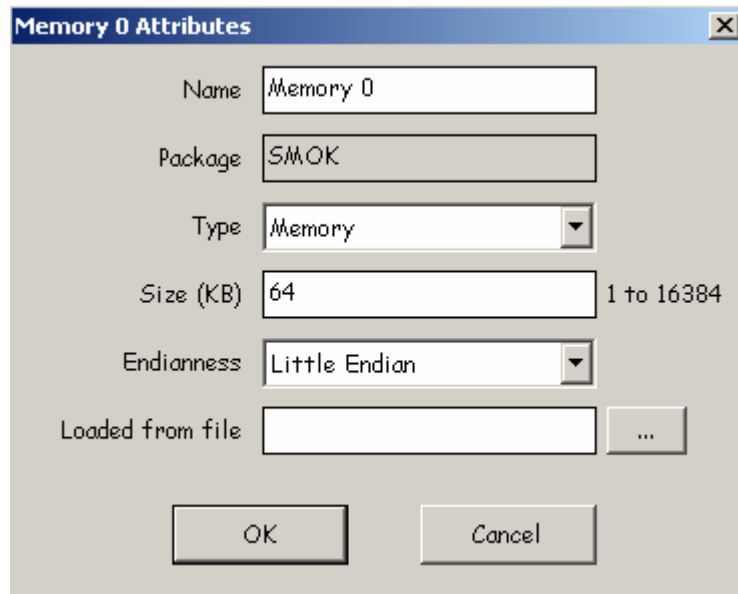


- d. Right-click and select **Display Contents** to view the contents of the register file. The contents of the register file can be edited by selecting a register and hitting the **Edit Contents** button.
- e. Try changing some values in the various registers and **Run** it to see that you can select a register and get the appropriate contents from the register file.
- f. **Add a Saved Container.** Add the **IR Decode** container that you designed earlier in lab to the canvas. Do this by right-clicking on the canvas and selecting: **Add Component**→**Saved Container...** and selecting your IR Decode container that you created.
 - i. Connect the output of the IR Decode container's OR gate (you will have to remove the DeMux) to **R0#**
 - ii. Connect **ConstantReg** and **DebugOutput** components as necessary to test data going into and out of the register file.
- g. **Save this design.** You will be using it for postlab.

4. **Memory and Memory Interface Components.**

SMOK comes with memory and memory interface components which we will use.

- a. Add a **Memory** component. Set the Endianness to Big Endian. As with the register file, you can display and edit memory contents.



- b. Add a **Memory Interface** component and connect it to the memory component.
 - i. Read the online documentation on the interface and hook up any **ConstantReg** and **DebugOutput** components needed to allow you to input information to simulate how the memory interface works to get data from memory and output it.
- c. Add the **MA** and **MD** registers, and rename the input control signals for the registers (**MA In**, **MD In**) Figure 4.6. For now, we will not worry about implementing/using a bus, so just name the control signal for MD as specified above.
 - i. The MA register will provide the request address to the memory interface
 - ii. The MD register will:
 1. Take input from the memory interface's Val output. This will allow our SRC machine to read data from memory.
 2. Serve as input to the Data input of the memory interface. This will allow the SRC machine to write data to memory.
- d. Save this as a container. You will use this for postlab.
- e. Demonstrate your understanding of this design to the TA before leaving lab. (Checkoff on the last sheet)

Part III: Postlab, Due via electronic submission one week from current lab session.

You will take the containers that you designed during lab (ALU, IR Decode, Register File, and Memory+Memory Interface) and combine them into a single SMOK file (**postlab2.smok**) where these containers will pass information to one another. Because we don't have any programs to simulate at the moment, keeping the ConstantReg components mostly attached is fine.

1. A memory address specified in MA will fetch data and place it in MD.
2. The MD for now can write its data to either the IR register or the register file.
3. The IR Decode container should be connected to the register file so that register selected when the appropriate control signals are asserted allows data to be written or read from the register.
 - a. You will need to remove the DeMux from your IR Decode container because the R0# input port takes care of the selection for you.
 - b. Before continuing, test to see that the contents of a register that you specify in the IR can be written or read.
4. The IR should also be connected to the memory register (MD). For now, your design should be able to allow the user to input a memory address to the MA register to specify an address to fetch from memory and the data at that address should be placed in the IR.
5. Connect the register file R0 output to the ALU structure that you designed.
 - a. To test getting data from the register file to the ALU in order to perform an ALU operation, you will have to first get data into the A register attached to the ALU from the register file, then get information to the B input of the ALU.

Electronically submit the following:

1. All the .smok and .smokcont files generated for this lab. Save copies for yourself as well.
2. postlab2.smok
3. A text file containing the following information:
 - a. Name (and if applicable, partner name)
 - b. Email ID (and if applicable, partner email ID)
 - c. Description of difficulties encountered during lab
 - d. Questions about datapath or control signals that arose from this lab (if applicable)

Checkoff Sheet

Name: _____ **Email ID:** _____

Date: _____

ALU Results Table (10 pts):

Complete the table without reinitializing the machine between steps.

Register	A _{In}	C _{In}	Result
A	0	0	
	0	1	
	1	0	
	1	1	
C	0	0	
	0	1	
	1	0	
	1	1	
ALU B input (write in the TA's chosen value)			
ALU A input (write in TA's chosen value)			
Operation	Val:	Desc:	
Z	N/A		
Carry			

_____ (5 pts.) **IR Decode:** Demo to the TA that your IR decode design works.

_____ (5 pts.) **Register File:** Demo working register file setup to the TA.

_____ (5 pts.) **Memory and Memory Interface:** Demo your understanding of how the memory and the memory interface works.