CSCE 230 Project Part II

Datapath implementation

In general, you are allowed to use block diagrams, VHDL, or a mix of both to complete any of these assignments. Each will come with its own advantages/disadvantages. Block diagrams are easier to realize but more simple mistakes, i.e. wire connections. VHDL may be harder to implement but less simple mistakes. If you copy any code from other sources, internet, or books you must cite your source or receive a zero for the project. Each team member must work together and should only turn in one assignment/report/check off by the TA.

Overview

For the last four weeks you have built an ALU, and Register File, and a control unit. This week you will build a processor. Most of the processor (see figure 1) has been created by you. You are asked to insert your ALU, Register File, and control unit into that processor and implement the connections. Eventually, by adding MUX's, registers, and a few other circuits, you will have a basic processor.

Your objective for this part of the project is to get a working processor with the instruction set limited to the five R-type instructions you made your processor implement in lab 9. The design flow will consist of five steps:

1. Block-level datapath design (or VHDL design)
2. Design of individual Components
3. Datapath integration
4. Control Unit Design
5. System integration and validation

Steps 1-3

For the basic processor design, we follow the scheme from the textbook, chapter 5. There are three types of components:

- Designed: The register file, ALU, and control unit were all designed by you
- Provided: The Instruction Address generator, I/O memory interface, Memory interface, and Immediate block are all of the components that will be provided to you. The only component you will be receiving this week is the immediate block. You must design the rest.
- Remaining: Any required components, such as registers, multiplexors, constants, Instruction Register(IR), and Processor Status Register(PS) should be created by you or from the built in megafunctions provided in Quartus II.

The datapath alone is shown in figure 2. You must connect the control circuitry, see figure 3, with the datapath into the processor that is shown in figure 1.

Step 4

You have already created the control unit for this part. This will be the object that you will modify in later parts so that you can do more than just the 5 R-type instructions you implemented.

Step 5

A **MAJOR** component of each part in the project is the last step. **Make sure you reserve sufficient time for this step**

as your validation results will help us understand the success of your implementation.

The integration of the datapath and control circuitry requires that you understand the interactions between the components well and use the correct and consistent signal names between the components. The instruction encoding and the names of the

control signals are **NOT** the same for our project as they are for the book. Refer to the project overview for correct instruction encoding and refer to your control unit for the correct names of the signals.

After you have integrated the datapath and the control circuitry, you will need to test your implementation thoroughly for correctness and timing performance.

Assignment

Using the components you have created, along with any provided components, you are to implement a processer, see figure 1, that is able to execute each of the 5 R-type instructions your control unit simulated.

Understanding

This section of the document will talk about testing and understanding the processor you are building.

I/O Pins

Until you add I/O components in a later part of the project, the input pins you need for this part will be the clock, reset, and the instruction. The instruction input will connect directly to your IR for now, as we have no way of loading instruction otherwise without a memory interface. For the processor to run it does not technically need any output pins. However, for debugging and demonstration purposes, you should add as many outputs as you deem necessary so you can observe the operation of your processor. These outputs could be RA-RZ or the ALU output, or the control flag outputs.

Registers and Muxes

There are many additional Muxes and registers in the processor. Some of the registers simply act as buffers between stages. Other, like IR/PC/PS, are important for operation. You should be able to either create your own registers or use the megawizard to create them. There are also many Muxes in the processor to help control the flow of data. Each of these select signals for the Muxes come from the processor. For example, the mux that chooses between RB and the immediate value has its select be b select.

Tasks you must perform

- Download the provided component(s) from Piazza
- Look over and develop an understanding any provided component(s)
- Combine the ALU, Register File, control unit, provided component(s), and additional Muxes/registers to create a basic processor
- Hand-assemble an instruction of each type and test your design
- Create a test script(s) (.do file(s)) and run a simulation(s) of the processor to make sure you created everything. The .do file should only control the instruction, the clock, and the reset. Everything else should just be outputs.

Testing and simulation

The basic processor design should be able to carry out add, subtract, and, or, and xor operations correctly. These operations should be able to calculate their value, produce the correct NCVZ flags, and update the destination register with the calculated value in 5 stage.

Demonstrate the correctness of your design by hand-assembling an instruction of each type. To do this you must initialize the IR with an instruction, cycle through the five stages of execution, and verify that the instruction operates correctly. Make sure to try each instruction with a variety of registers for more comprehensive testing. **You will need to hand assemble your instructions based off of the ISA provided in the overview document**

With your testing, strive to make sure your processor is operating at the fastest possible speed. This means to make sure your clock is as fast as possible. **You should show tests in your report** about how fast your are able to make your processor.

Report

You must create a two page (minimum, not including images) report detailing the processor you have created so far. Include its current abilities (which instructions it can perform), a speed overview (how fast it can run), and the components inside of it. You should also briefly talk about your groups experiences with connecting everything. Use normal margins, size 11, and normal formatting. This report will be mostly a check of your teams understanding of what they have done so far. **Make sure to have a title page**

Grading and TA check-off

The grading breaks down from the table below:

Points	Part
15	Design File (.qar)
10	Simulation
20	Check off and demonstration
30	Technical report

The design file, simulation, and report are due by Midnight of April 1st-2nd depending on your lab. The Check off is due by the time Lab ends. If your project is not checked off by the end of the week it is due, April 3rd is the last day, you will not receive any points for the check off. If you are unable to finish this part in time, **still hand in what you have done so far for partial points**. Late submissions will get zero points so make sure to meet the deadlines. Each part of this project is incremental so being late on a deadline will make you late on the next one.

Submit

You must submit the files electronically to webhandin by 11:59PM on April 1st/2nd. The naming convention should be as follows:

- Project"team-number" Part2.qar for the project archive file. The simulation file should be located inside of the archive. E.g. Team2 part2.qar
- Project"-team-number" Part2.(.doc/.tex) for the project report. This report should follow the best practices of writing technical reports.

Figures for reference

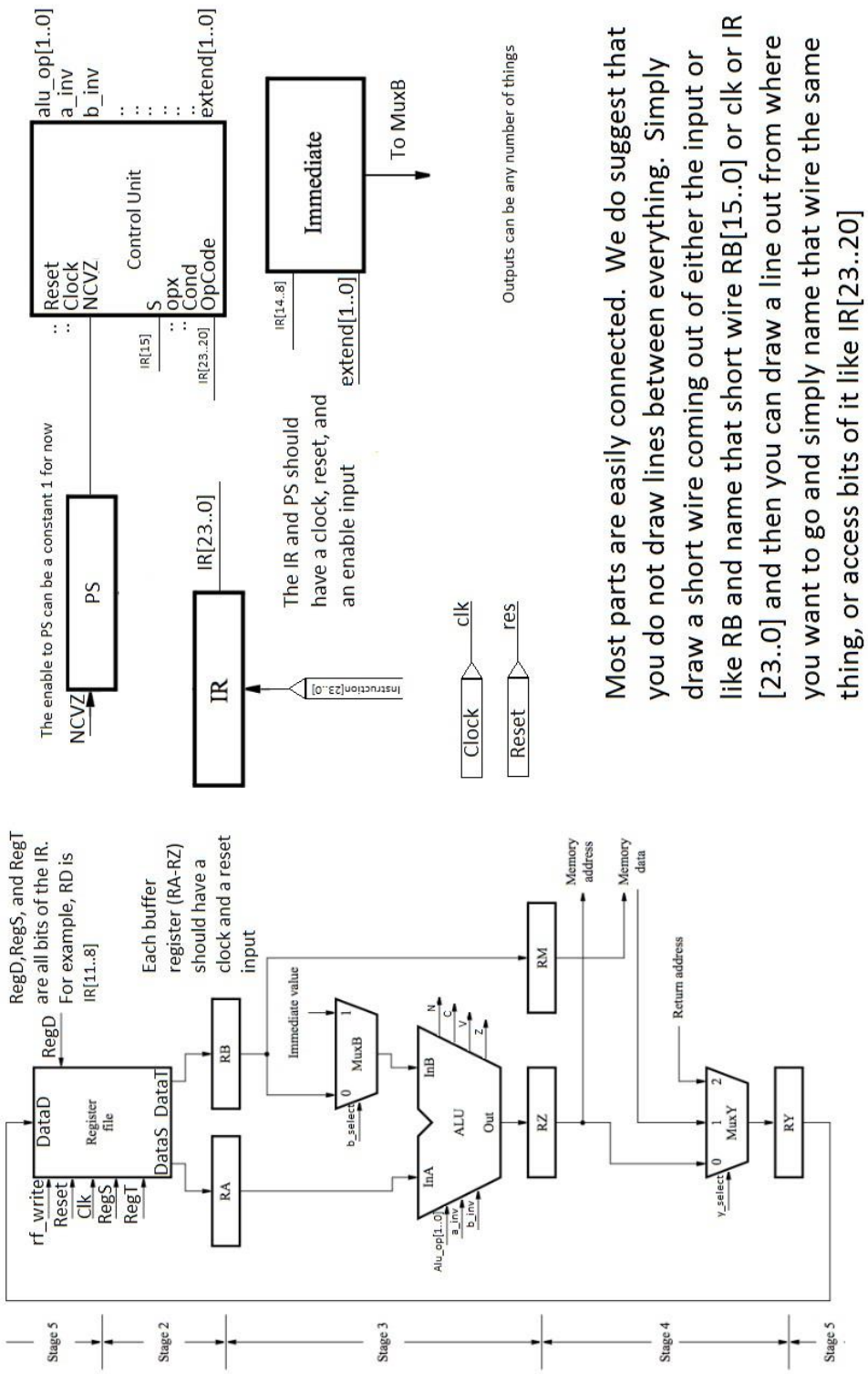

Figure 1: The full processor for Part II

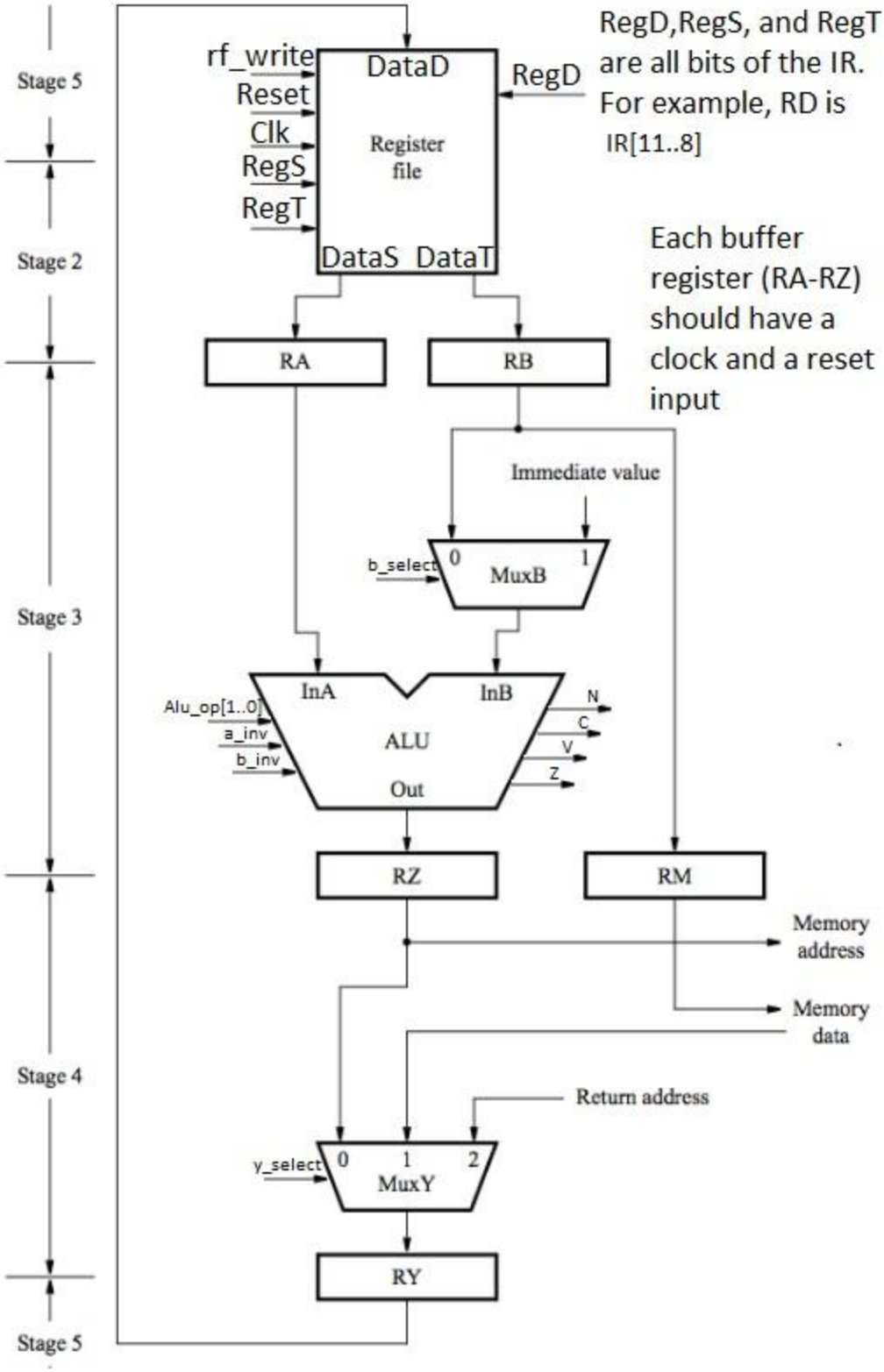
Stage 5
Stage 2
Stage 3
Stage 4
Stage 5
rf_write
Reset
Clk
RegS
RegT
DataD
RegD
Register file
DataS DataT
RegD,RegS, and RegT are all bits of the IR. For example, RD is IR[11..8]
Each buffer register (RA-RZ) should have a clock and a reset input
RA
RB
Immediate value
b_select
0
1
MuxB
InA
InB
Alu_op[1..0]
a_inv
b_inv
ALU
Out
N
C
V
Z
RZ
RM
Memory address
Memory data
Return address
y_select
0
1
2
MuxY
RY

Figure 2: The datapath with correct signal names Figure 3: The control circuitry