CprE 381 – Computer Organization and Assembly Level Programming

Mini-Project B, Version 1.0
Last update: 10/21/2013

This is a **three-week project**. You will design and implement two versions of a **single-cycle processor (SCP), whose datapath and control are shown in Figure 1**. The project involves substantial design, implementation, integration, and test tasks. **Due to the complexity of the assignment, this document is subject to minor changes between now and the due date**.

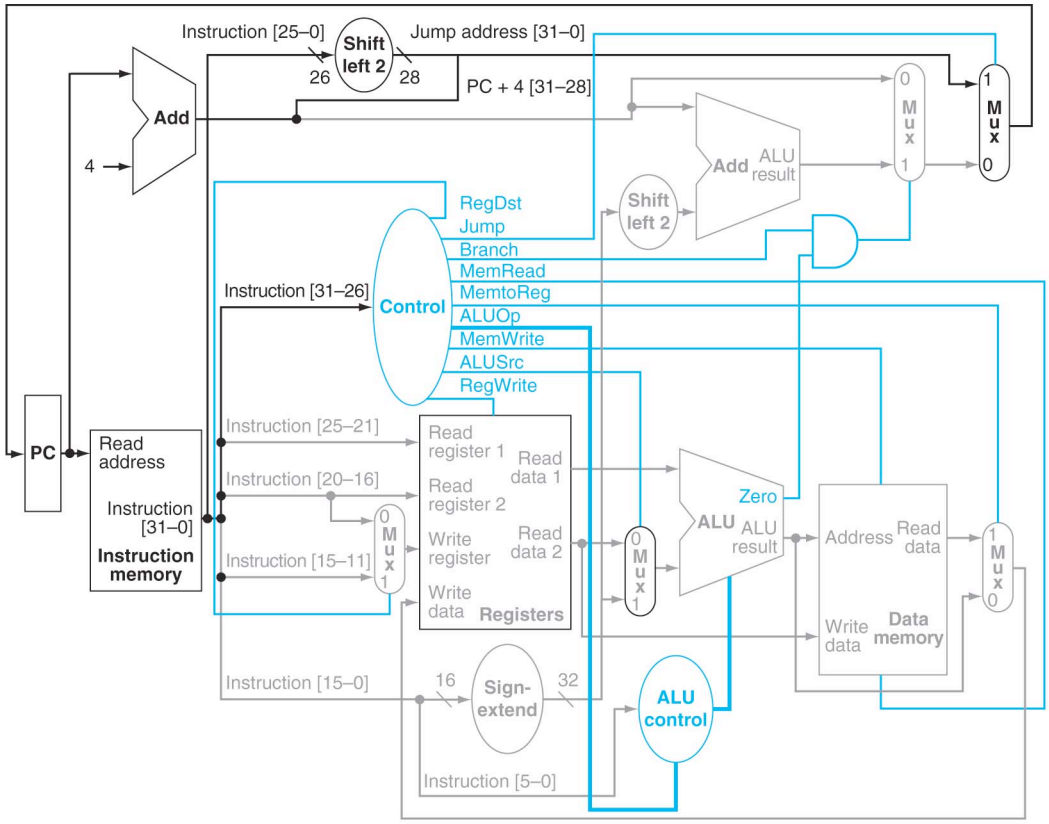

Figure 1. The control and datapath for implementing SCP-V1. It is from the textbook, Figure 4.24.

Deadline Extension to Mini-Project A Late Completion: If you will not have completed Mini-Project A by your lab time this week, demo those parts that are working and start working on Mini-Project B as soon as possible. You may demo the rest of Mini-Project *A* with your Mini-Project B (see Mini-Project B, Part 3). There is 20% penalty that applies to the parts being late, but you have three extra weeks.

Prelab [Not graded] Write down a complete list of the instructions that your SCP has to support to run the bubble sort program. You and your partner shall have the answer from homework assignment 5, problem 1. Consolidate the lists of you and your partner.

Part 1. SCPv1 Prototyping

Starting Point. In this part, you will build the MIPS Single-Cycle Processor, Version 2 (**SCPv1**) with a fast prototyping strategy. SCPv1 only supports nine MIPS instructions:

1. Memory reference: `lw`, `sw`
2. Arithmetic/logical: `add`, `sub`, `and`, `or`, `slt`
3. Control transfer: `beq`, `j`

You are provided with coarse-level VHDL modeling of the following datapath elements. You are required to use them as they are[1].

a. mips32.vhd: A VHDL package of pre-defined constants and VHDL data types. *You are required to only use those types in your implementation (contact Dr. Zhang if any type you needed is not provided).*
b. regfile.vhd: The register file
c. reg.vhd: A single register, intended for modeling PC (and pipeline registers in Mini-Project C)
d. alu.vhd: The ALU module
e. adder.vhd: The adder for calculating next sequential PC (PC+4) and branch target
f. mem.vhd: The memory, for both instruction memory and data memory

The VHDL programs have been tested. However, you are expected to test them using your own test bench programs before using them.

For this project, the instruction memory and data memory are treated as *external datapath elements* to the CPU, which means in a real implementatino they are components outside the CPU. The two memory instances shall be declared outside cpu.vhd. See "**Demonstration**" for detail and a provided file named tb_cpu.vhd for an example.

To help you get started, the following VHDL programs are provided as partial samples and samples. They are from a working prototype of an SCP implementation. The partial samples have most code lines removed.

a. cpu.vhd (partial sample): The CPU composition of the internal datapath elements and control units.
b. control.vhd (partial sample): The main control unit.
c. tb_cpu.vhd: A test-bench program of the CPU, the instruction memory, and the data memory.
d. imem.txt: The MIPS binary code used as the MIF (memory initialization file) for the instruction memory. It uses LW, SW, BEQ and ADD instructions. You may extend it, or use extra MIPS binary code, to test all the nine instructions.
e. dmem.txt: The initial memory content as the MIF for the data memory.
f. tb_regfile.vhd: A test-bench program for register file.

Before you start, list all components that you must have before you start to build the CPU. Those include all internal datapath elements (not excluding the memories which are outside of the CPU), the main control unit, and the ALU control unit. Note that the textbook provides all the details of all datapath elements, the main control unit, and the ALU control unit for the nine-instruction ISA (except some details relate to the Jump instruction).

[1] Contact Dr. Zhang if you think there is any problem with those programs.

VHDL Modeling Requirements. You must have a VHDL program file with name **cpu.vhd** for CPU composition, i.e. to build the CPU from the datapath elements, the main control, and the ALU control. The cpu.vhd program must be **strongly structural**, with the following requirements:

a. No behavior modeling can be used. In particular, no process statement can be used.
b. Limited dataflow modeling, with only three forms acceptable:
   1) Signal copying/splitting, e.g.
      ```
      opcode <= inst(31 downto 26);
      ```
   2) Signal Merging, e.g.
      ```
      j_target <= PC(31 downto 28) & j_offset & "00";
      ```
   3) One-level of basic logic gates, e.g.
      ```
      taken_branch <= branch AND zero;
      ```
c. All entities that are used in cpu.vhd must be declared as component. Only *component instantiation* can be used; no entity instantiation may be used. See the partial sample cpu.vhd for an example.

However, you may use any modeling style and techniques to model the datapath elements and the control units. The provided regfile.vhd, alu.vhd and adder.vhd all use behavior modeling.

****Demonstration****. Your design should include a test-bench program named tb_cpu.vhd (a sample has been provided). Demonstrate that the CPU works correctly by 1) tracing the program execution, and 2) and inspecting the register and memory contents at the end of execution. Explain to your lab TA why the CPU works correctly.

The test-bench program should create three instances for CPU, instruction memory, and data memory, respectively, and connect the three instances properly. The CPU should be "reset" in the beginning, in the sense that the register file is cleared and the PC is set to some predetermined initialization address.

Every of the nine instructions must be tested in your demo. Your may use multiple MIPS binary codes (i.e. multiple MIFs for the instruction memory). See the provided imem.txt for an example. You may write the binary code manually (which is tedious but can be fun), or use a tool to generate the binary. You may also use multiple MIFs for the data memory. See the provided dmem.txt for an example.

Test your CPU thoroughly. **Your TA may test your CPU using a different set of binary code and data memory contents.**

Debugging. You may have learned a number of debugging skills on ModelSim simulation. The following may be useful in this project:

- Inspecting signals: You may pause the simulation at any time and then inspect the signal values inside any component instance.
- Inspecting memory structure: You may pause the simulation at any time and then inspect the contents of any memory structure. Those include the contents of the CPU register file and the data memory.
- Be selective with waveform signals: The CPU has many signals ongoing. For a debugging purpose, select the relevant signals into the waveform. For example, when you monitor the CPU program execution for a sequence of instructions, you may want to include only the registers and memory words to be changed by your program, not all registers and memory words.

- Pause simulation and inspect signals at the right time: You may use 100ns as the default clock cycle. When inspecting signals, you may want to stop right before a rising clock edge. There can be uncertainty if you inspect exactly at the time of the rising clock edge. For example, if the rising clock edge occurs at 100ns, 200ns, 300ns and so on, you may want to inspect signals at 90ns, 190ns, 290ns, respectively. Alternatively, you may make the rising clock edge occur at 110ns, 210ns, 310ns and so on, and inspect signals at 100ns, 200ns, 300ns, respectively.

VHDL Programming Style. You are required to use a good VHDL programming style. The code should be well indented and formatted, use meaningful names, and be sufficiently commented. You and your partner must use a consistent programming style, but it doesn't have to be consistent with the provided VHDL program files. **This requirement applies to all parts in this mini-project.**

Use a separate file for an entity or multiple entities of one type, e.g. alu_ctrl.vhd for the ALU control unit, and mux.vhd for 2-to-1 mux, 3-to-1-mux and so on. Use VHDL generic when appropriate.

Project Report. In your project report, include for Part 1:

a. Give a brief description of your testing cases and testing method.

Part 2. SCPv2 Prototyping (SCPv2a)

SCPv1 is very limited, so you will design MIPS Single-Cycle Processor, Version 2 (**SCPv2**) that extends SCPv1. Your SCPv2 should be able to execute the code of the bubble sort example in the textbook. That means SCPv2 has to support all MIPS instructions used by the bubble sort example. It should continue to support the nine instructions from SCPv1.

The MIPS codes for the `swap()` and `sort()` function are given in Figure 2.25 and Figure 2.27, respectively. Alternatively, Dr. Zhang provides a simpler version of bubble sort in the slides for week 5 (http://class.ee.iastate.edu/cpre381/lectures/lecture-week5.pptx). You may decide to use either version.

You'll have to revise the **datapath**, the **main control unit**, and the **ALU control**. Re-read P&H 4.4 if necessary. Recall in the class we have discussed how to add the support for ADDI, SLL, BNE and SLL on top of SCPv1, and more importantly we have discussed the thought process behind it.
See the slides of week 8 at http://class.ee.iastate.edu/cpre381/lectures/lecture-week8.pptx.

****Demonstration**.** Demonstrate that the CPU works correctly by 1) partially tracing the program execution, and 2) inspecting the register and memory contents at the end of execution. The MIF for the instruction memory should contain the binary code of **the bubble sort program**, and the MIF for the data memory should contain an array to be sorted. You should test your CPU using multiple data arrays and thus multiple data memory MIFs. **You should also test SCPv2a with your test cases from Part 1**.

Test your CPU thoroughly. **Your TA may test your CPU using a different data array.**

Project Report. In your project report, include for Part 2:

a. A datapath diagram of your design, with a mark on each control signal. To keep the diagram clear, you don't have to include the main control and the ALU control and their

connections to the control signals. You may use a software tool to draw the diagram. Alternatively, you may hand-draw the diagram, take a photo, and include the photo in the report. In the latter case, make the diagram as readable as possible.

b. For each instruction beyond SCPv1, include a short description of the changes to the datapath and control.
c. Give a brief description of your testing cases and testing method.

Test your CPU thoroughly. **Your TA may test your CPU using a different data memory MIF.**

Part 3. SCPv2 Detailed Implementation (SCPv2b)

In the first two parts, the modeling for datapath elements is coarse-level: Behavior modeling is used for ALU, adder, and register file. In this section, you will use detailed modeling for those three components. The requirements are as follows:

a. Extend the provided regfile.vhd, alu.vhd, adder.vhd by adding another architecture body with your preferred name, e.g. "structural" or "detailed". *Do not remove the current architecture body.*
b. Use your code from Labs 1-4 and Mini-Project A to implement the new architecture bodies. You may revise your code.
c. Your final code should be strongly structural. The general guideline is that that code should be detailed enough for a VHDL synthesis tool to synthesize efficient circuits. If you are not sure if your code would meet the requirement, consult your lab TA.

****Demonstration**.** Repeat your demonstration for Part 1 and Part 2, but this time, test your CPU with the new architecture bodies for register file, ALU and adders.

In VHDL programing, create a new tb_cpu.vhd with a VHDL *configuration* for your CPU, and bind the new architecture bodies with the entities for those three components, respectively. Here is a link for an example of VHDL configuration.
http://www.sigasi.com/content/advanced-vhdl-configurations-tying-component-unrelated-entity.

Note that it is common in hardware design that designers first model the structure/organization of a complex component, and then work out the detailed modeling of the sub-components.

Notes on Demonstration:

a. Part 3 demo may not substitute Part 2 demo.
b. In case your Part 2 demo doesn't work fully, you may demo Part 3 based on your demo for Part 1. (In this case you would get a partial credit for Part 2.)
c. To get the full credits for *both* Part 2 and Part 3, you have to demo Part 3 using the bubble sort code.
d. If you can demo Part 3 using the bubble sort code, you don't have to repeat the testing from Part 1 demo.
e. If you haven't yet done a full demo of your Mini-Project A and you can demo Part 3 successfully, you don't have to demo for Mini-Project A. You will get the credit for the late parts of Mini-Project A with 20% late penalty.

Project Report. In your project report, include for Part 3:

a. Give a brief description of your testing cases and testing method.

EVALUATION FORM AND SUBMISSION:

- Print out a hard copy of the evaluation from (*Lab1-Eval.pdf* in *Lab-01.zip*).
- Complete the demos to your TA and ask your TA to sign on the evaluation form.
- Create a zip file *Project-A-submit.zip*, including the completed code and document from the four lab parts and the lab report. Put code for parts 1, 2, 3 under directories /P1, /P2, and /P3.
- Include a project report named *Project-A-Report.doc*. See the description of project report contents in Parts 1, 2 and 3.
- The file names in your zip file should be self-explained.
- Submit the zip file on BlackBoard Learn under "Project A" assignment.

Bonus Project, Part 1: Green MIPS SCP (SCP-G)

This is the first part of the Bonus Project. You will extend SCPv2 to support **all *integer* instructions** listed on the green sheet (both sides) of the textbook. The second part, to be done with Mini-Project C, is for pipelined CPU implementation to support those instructions.

****Demonstration**.** Create appropriate testing for your SCPv3 implementation, demonstrate it to you lab TA and explain why it works. You may use multiple tests, each of a set of MIPS binary code (instruction MIF) and data memory contents (data MIF). To get the full credit, your tests must collectively cover all those instructions.

Partial Credit. If your SCP-G supports M instructions, M < N where N is the number of all instructions that are supposed to be supported, the partial credit is given by the following formula:

% of partial credit = (M – K) / (N – K)

The number K represents roughly the number of instructions that have to be supported in the SCPv2.

Demo Deadline. The deadline for the demos of the Bonus Project, for both Part 1 and Part 2, is your last lab this semester. You may demo the parts individually and before than the deadline (which is encouraged).

Project Report. In your project report, include:

a. A datapath diagram of your design, with a mark on each control signal. To keep the diagram clear, you don't have to include the main control and the ALU control and their connections to the control signals. You may use a software tool to draw the diagram, or hand-draw the diagram and include a picture. In the latter case, make the diagram as readable as possible.
b. For each instruction beyond SCPv2, include a short description of the changes to the datapath and control.
c. Give a brief description of your testing cases and testing method.

EVALUATION FORM AND SUBMISSION: The information will be released soon.