Organization of Digital Computer Lab
EECS 112L/CSE 132L

Lab 2 - Single-cycle MIPS Datapath and Control

University of California, Irvine

January, 11, 2015

Due on January. 24. Note: this is a two-week assignment

1 Goal

In this lab, you will implement the basic processor that runs a subset of the MIPS ISA.

1.1 Lab Description

In this lab you will build a simplified MIPS single-cycle processor using VHDL/SystemVerilog. Then you will load a test program and confirm that the system works. Next, you will implement two new instructions, and then write a new test program that confirms the new instructions work as well. By the end of this lab, you should thoroughly understand the internal operation of the MIPS single-cycle processor. It is so simple, in fact, that it does not even have a branch instruction, so it cannot execute most programs. The processor does, however, have all the parts that a real processor has: A fetch unit, decode logic, functional units, a register file, IO support (not now) and access to memory.

All of you will be implementing the same datapath, so you will all run into similar problems. Learn from your classmates, and then go apply what you learn to your own design.

Before starting this lab, you should be very familiar with the single-cycle implementation of the MIPS processor described in the reference book. The simplified single-cycle processor schematic, shown in Figure 1.1, from the text is repeated here for your convenience. This version of the MIPS single-cycle processor can execute the following instructions: ***add***, ***sub***, ***and***, ***nor***, ***xor***, ***or***, ***slt***, ***lw***, ***sw***. Both the R-Type and I-type instruction should be supported.

Our model of the single-cycle MIPS processor divides the machine into two major units: the control and the datapath. Each unit is constructed from various functional blocks. For example, as shown in Figure 1.1, the datapath contains the 32-bit ALU, the register file, the sign extension logic, and five multiplexers to choose appropriate operands.

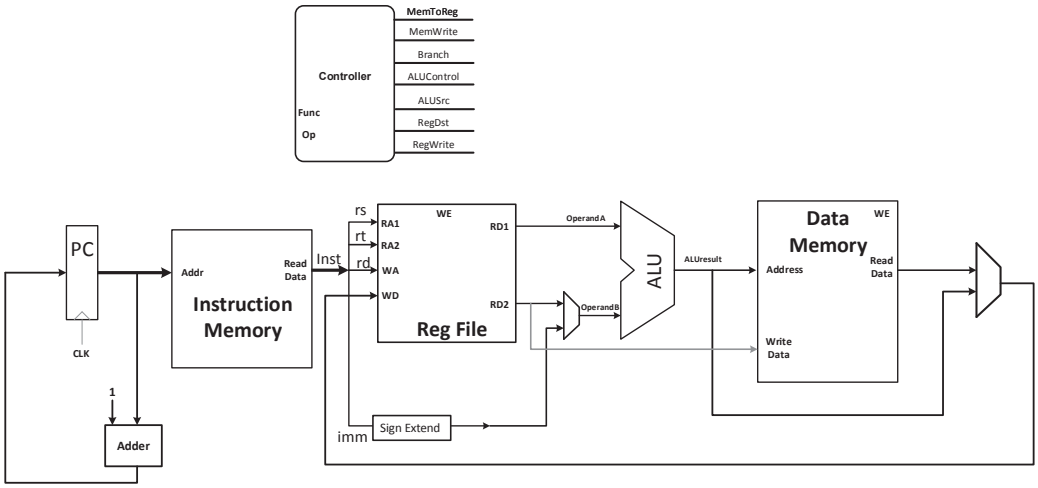

Figure 1: Simplified Single-cycle MIPS processor architecture

1.1.1 Processor

It is assumed in this lab that the program instructions are already preloaded in the instruction memory. Later, we will extend the processor to load the memory contents from outside RAM.

Code 1: Entity of MIPS Processor in VHDL

```
ENTITY processor IS
 PORT(
      ref_clk   : OUT std_logic;
      reset     : OUT std_logic;
 );
END processor;
```

Code 2: Entity of a ALU in VHDL

```
ENTITY alu IS
 PORT(
      Func_in       : IN    std_logic_vector(5 DOWNTO 0);
      A_in          : IN    std_logic_vector(31 DOWNTO 0);
      B_in          : IN    std_logic_vector(31 DOWNTO 0);
      O_out         : OUT   std_logic_vector(31 DOWNTO 0);
      Branch_out    : OUT std_logic;
      Jump_out      : OUT std_logic;
 );
END alu;
```

1.1.2 Instruction Memory

This memory for now is a implemented as a ROM. The instruction rom provides a read-only memory which your processor will get its instructions from.

Code 3: Entity of a ROM in VHDL

```
 ENTITY rom IS
    port (
         addr     : IN std_logic_vector(31 DOWNTO 0));
         dataIO   : INOUT std_logic_vector(31 DOWNTO 0));
  END rom;
```

1.1.3 Register file

Code 4: Entity of a register file in VHDL

```
ENTITY regfile IS
 GENERIC (NBIT: INTEGER := 32;
          NSEL: INTEGER := 3);
 PORT(
      clk     : IN std_logic;
      rst_s   : IN std_logic;   -- synchronous reset
      we    : IN std_logic;     -- write enable
      raddr_1 : IN     std_logic_vector(NSEL-1 DOWNTO 0);  -- read address 1
      raddr_2 : IN     std_logic_vector(NSEL-1 DOWNTO 0);  -- read address 2
      waddr   : IN     std_logic_vector(NSEL-1 DOWNTO 0);  -- write address
      rdata_1 : OUT std_logic_vector(NBIT-1 DOWNTO 0);   -- read data 1
      rdata_2 : OUT std_logic_vector(NBIT-1 DOWNTO 0);   -- read data 2
      wdata   : IN     std_logic_vector(NBIT-1 DOWNTO 0)    -- write data 1
 );
END regfile;
```

1.1.4 Data Memory

The data memory should be a 512-word×32-bit array for now. Design the memory in a way that the size can be reconfigured a instantiation time.

Code 5: Entity of a data memory in VHDL

```
 ENTITY ram IS
    port (
         clk    : IN std_logic;
         we     : IN std_logic;
         addr   : IN std_logic_vector(31 DOWNTO 0));
         dataI  : IN std_logic_vector(31 DOWNTO 0));
         dataO  : OUT std_logic_vector(31 DOWNTO 0));
  END ram;
```

1.1.5 ALU

Table 1.1.5 lists the operations it can perform and what value of *Function* each corresponds to. You might not need all these information now, but it will be helpful in later labs. Its included here for completeness.

Table 1: ALU supported operations

Func	Operation	Output value	Branch out	Jump out
100000	ADD	A+B	0	0
100001	ADD	A+B	0	0
100010	SUB	A-B	0	0
100011	SUB	A-B	0	0
100100	AND	A AND B	0	0
100101	OR	A OR B	0	0
100110	XOR	A XOR B	0	0
100111	NOR	A NOR B	0	0
101000	Set-Less-Than Signed	(signed(A) < signed(B))	0	0
101001	Set-Less-Than Unsigned	(A < B)	0	0
111000	Branch Less Than Zero	A	(A < 0)	0
111001	Branch Greater Than or Equal to Zero	A	(A >= 0)	0
111010	Jump	A	0	1
111011	Jump	A	0	1
111100	Branch Equal	A	(A == B)	0
111101	Branch Not Equal	A	(A ! = B)	0
111110	Branch Less Than or Equal to Zero	A	(A <= 0)	0
111111	Branch Greater Than Zero	A	(A > 0)	0

1.2 Assignment Deliverables

Your submission should include the following:

- Block designs in VHDL/SystemVerilog
- SystemVerilog Testbench capable of preloading memories and veriyfing the correct functionality of the processor using a simple test MIPS program.
- A comprehensive report of the Design and testbench architecture. In the report, include the information on how you have make sure the processor works fine, and if you have found any bugs in your design.

IMPORTANT: only **ONE** submission per group is required and the group leader should be the submitter.

Note1: Compress all your files in "zip" or "tar" format and then submit the compressed file.

Note2: The compressed file should include **design, sim, verif, doc** directories and the corresponding files inside each directory. Your report's tex and pdf files are expected to be inside **doc** directory.

Note3: Remember to include the group ID and the name and student ID of each group member in the report. When in doubt about your group ID, please ask TAs.

Note4: Assuming that the parent directory that you are working under that is named `Lab-2`, the following command will compress it for you:

```
$  tar cvf lab-2.tar lab-2
```