OBJECTIVE

The objective of this lab is to continue the design of an elementary central processing unit (CPU) which was started in Lab 6 and Lab 8. You are to add a program counter (PC) and a memory module to store the instructions of a “program” to be executed by the CPU. The ASM diagram for controller has to be changed accordingly.

INTRODUCTION

The main difference between Lab 8 and Lab 9 is as follows. In Lab 8, you input the op codes (i.e., 00, 01, 10, or 11) and data manually. The op codes and inputs were entered between every active clock transition with the switches at INPUT3:0. In Lab 9, the op code and data will be stored in memory a ROM. Your controller will control the signals in such a manner that the op code and data are automatically fetched from memory (using the program counter PC) and input on INPUT3:0.

SPECIFICATIONS

No changes will be made to MUXA, MUXB, MUXC, REGA or REGB from Lab #8.

1. As shown in Figure 1, a 2K X 8 ROM is added. The instructions and data will be stored in this ROM starting in location $200 (What do you need to do to the address lines to accomplish this?).

2. A program counter (PC) is added. PC is a 4-bit up-counter with a synchronous count enable signal (PC_INC). If PC_INC is TRUE, the counter will increment by 1 at the next active clock transition. If PC_INC is FALSE, the counter will hold its current value. The count after 1111 is 0000. Another synchronous signal will be used to load the counter from the INPUT bus. This signal is called PC_LD. Requirement: The PC should be designed from D-Flops to zero before beginning your testing. (You should design every state machine so that you can start it in a known state. For this lab, the known state has state bits of all zeros.)

3. The instruction register is increased to 3 bits (IR2:0), with the following definition:

<table>
<thead>
<tr>
<th>IR2:0 Op code</th>
<th>Instruction Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>TAB</td>
<td>A copied to B</td>
</tr>
<tr>
<td>110</td>
<td>LDAA #data</td>
<td>load A with input data</td>
</tr>
<tr>
<td>101</td>
<td>SAR</td>
<td>shift A right 1 bit, store in A</td>
</tr>
<tr>
<td>100</td>
<td>ABA</td>
<td>load A with A plus B</td>
</tr>
<tr>
<td>011</td>
<td>SAL</td>
<td>shift A left 1 bit, store in A</td>
</tr>
<tr>
<td>010</td>
<td>LDAB #data</td>
<td>load B with input data</td>
</tr>
<tr>
<td>001</td>
<td>Not used</td>
<td>create your own instruction</td>
</tr>
<tr>
<td>000</td>
<td>JMP Addr</td>
<td>load PC with input address</td>
</tr>
</tbody>
</table>

Changes to the ASM diagram:

1. All the manual switching you did in Lab 8 (e.g., setting the INPUT = next op code or data) are now done by incrementing the address on the ROM. This is accomplished by incrementing the PC register or “inc PC” in the Figure 2.

2. There is an instruction called “JMP Addr”. JMP Addr consists of two “nibbles” where the first nibble (4 bits) is the opcode and the second is an address (operand). This instruction loads the PC with a 4-bit address read from memory (2nd nibble).

PRE-LAB REQUIREMENTS

1. Transform the flowchart shown in Figure 2 into an ASM diagram; i.e., put in the actual signals to control the PC, IR, and MUXes in the ASM diagram, and complete the ASM for the other two op codes.

2. Create a next state table and logic equations for the controller shown in Figure 1.

3. Add an active low asynchronous RESET signal to all registers and counters. You should use this to initialize all flip-flops to zero before beginning your testing. (You should design every state machine so that you can start it in a known state. For this lab, the known state has state bits of all zeros.)

4. Hand-assemble the following program into the machine code of your CPU (in hex).

   LDAA #7
   LS: LDAB #3
   SAL
   ABA
   TAB
   SAR
   JMP LS

5. Store the machine code in Task (4) into the ROM, starting in location $200. Simulate the program counter (PC) and the ROM by themselves. Increment the PC to show the contents of the locations of the ROM which contain the machine code. (See a tutorial on the class Web site for the use of LPM_ROM in Quartus 2.)

6. Simulate the entire CPU, showing the execution of the above program. Your simulation outputs should include (at least) IR2:0 (opcode), MSA, MSB, MSC, REGA, REGB, and OUTPUT.

IN-LAB REQUIREMENTS

You will show your TA the designs and simulations that you did for the pre-lab (40%). Note, for maximum credit, you have to able to fully explain your work and execute a simple program given to you by your TA (30%) in Quartus by setting the appropriate inputs & viewing the outputs.

Connect the ROM to your design and run the Reg A & Reg B outputs to LEDs (30%). It will also be useful to see the PC output and data coming out of the ROM as well as the current state in your controller. If you have enough LEDs, you can also watch the output from the IR.
**LAB 9: Elementary CPU Design: Fetching Instructions From a ROM**

**Figure 1. System Hardware Block Diagram**

**Note1:** PC also has a clock input.
**Note2:** Add a **RESET.L signal** to reset the controller and all registers to an initial state.

**Figure 2. Controller Flow Chart**

(fill in ?’s and then convert to **ASM Diagram**)

**Note:** When not specified, the default actions for each state is to “hold” REGA and REGB and OUT = REGA.