Experiment 2: Designing BURP

1.0 Description

In this experiment FPGA is used to implement a simple RISC processor. BURP stands for Basic Undergraduate RISC Processor. Please refer to the website of the UCSB/ECE DigiLab FPGA Board (http://vader.ece.ucsb.edu/digilab-fpga/) for more information concerning programming and using the UCSB/ECE DigiLab FPGA Board.

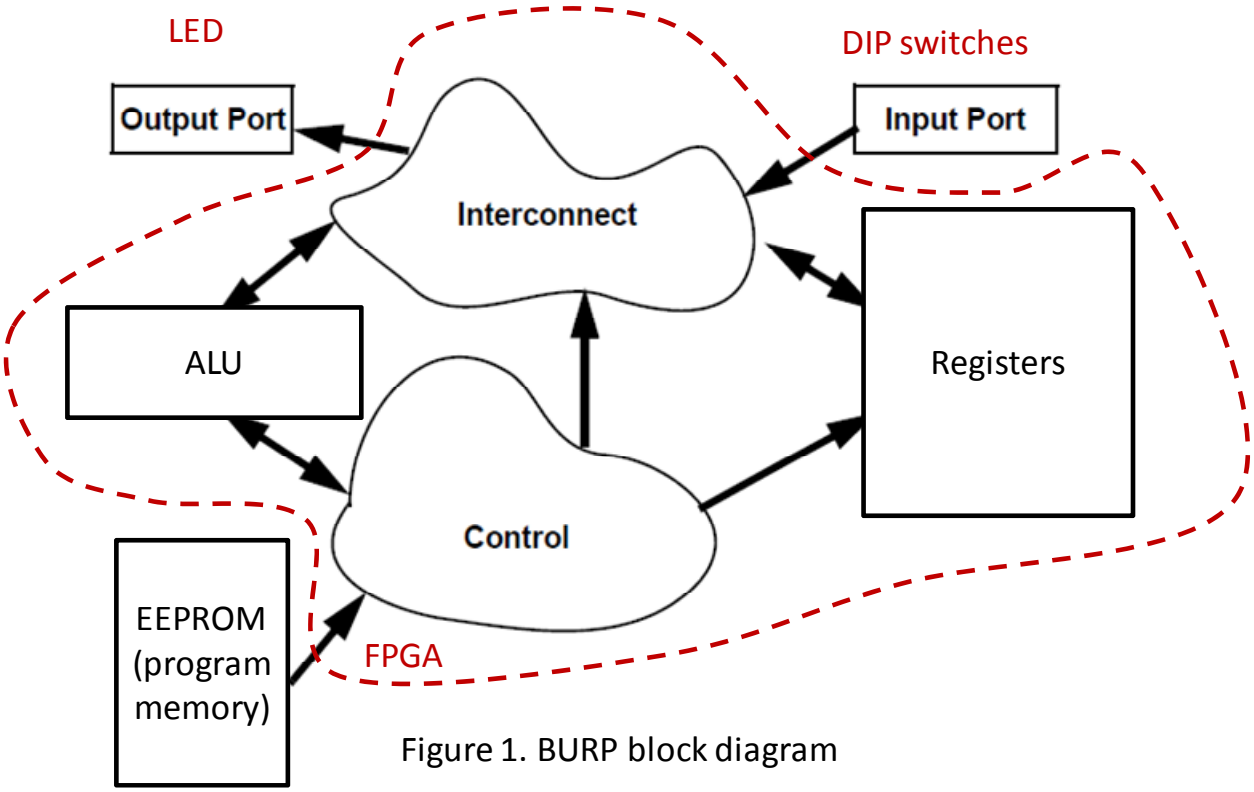

Figure 1. BURP block diagram

Figure 1 shows a block diagram of the system. The two blobs in the block diagram are what you must design to make a functioning processor. The Interconnect blob refers to the busses required for communication between the ALU, the register file, and the controller. The arrows from the Interconnect to the ALU and register file are two way since data must be written to and read from both devices. All such data connections are four bits wide. The connection from the control unit to the interconnect is one way to show that the controller is used to determine how the Register File and ALU are connected at any one time. The interconnect busses can be implemented using multiplexers or tri-state buffers.

1.1 Control Unit

The control unit is the heart of the processor. It reads instructions from ROM where the program is stored, then sequences the appropriate control signals to the ALU, interconnect, and Register File (all implemented with FPGA) to execute the instruction. Figure 2 shows a block diagram of the control unit. Contained within the control unit are the Instruction Register (IR) and Program Counter (PC). Your controller must also maintain a carry flag, which may be encoded as a state in the controller.

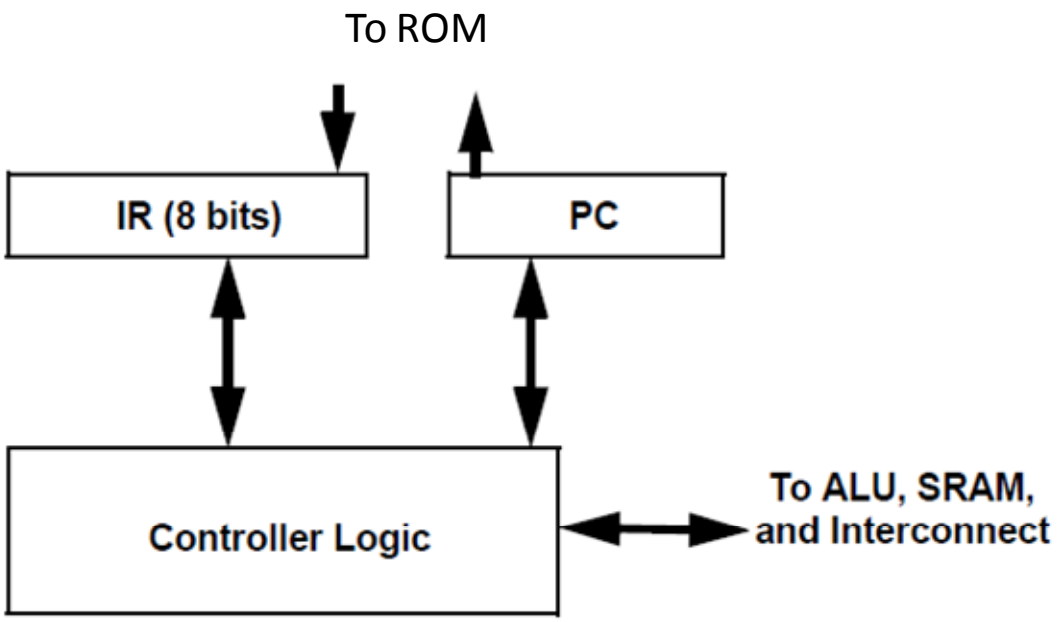

Figure 2. Controller block diagram

The IR can be implemented with either level sensitive or edge triggered latch. The PC counter can implemented using counters and should be capable of addressing at least 8 bits of the program memory address space.

The controller can be implemented in several ways. For example, a counter implementation, like the one discussed in class, could be used with the FPGA implementing the decoding and branching logic. Alternatively, the FPGA can be used to implement either a Moore or Mealy state machine. Both approaches will work and have different strengths and weaknesses. The design and simulation environment for the FPGA is Modelsim.

8.2 Instruction Set

Table 1 lists the instruction set which your controller must be able to interpret. Each instruction must be encoded in exactly 8 bits. The implementation should use pipelining where appropriate.

Table 1:

Mnemonic	RTL	Update Carry?
JC <address>	address + PC -> PC	no
JMP <imm> RA	(imm << 4) + RA -> PC	no
MOV R1, R2	R1 -> R2	no
MVI RA, <imm>	imm -> RA	no
INC R	R + 1 -> R	yes
ADD R1, R2	R1 + R2 + CY -> R1	yes
SUB R1, R2	R1 - R2 + CY -> R1	yes
AND R1, R2	R1 & R2 -> R1	no
OR R1, R2	R1 | R2 -> R1	no
SC	1 -> CY	yes
CC	0 -> CY	yes
PUSH R	R -> Stack	no
POP R	Stack -> R	no
IN R	IN -> R	no
OUT R	R -> OUT	no
NOP		no

In the specifications, R, R1 and R2 do not refer to specific registers but to any of four independently addressible registers (RA, RB, RC and RD) in your design.

The first instruction in the table is JC, or “jump on carry”. The operand to this instruction is an address that can access four bits of the ROM space. This 4-bit address will be an immediate value contained within your instruction. This implies that you will only be able to branch at most

15 instructions ahead of the branch instruction. The carry flag should not be updated. The second instruction in the table is JMP or “jump immediate”. Its destination address is composed of two parts. The upper bits of the address are specified as an operand of the JMP instruction. The lower four bits of the destination address are located in register RA. Therefore, this JMP should be able to address eight bits of the program memory space.

The MOV instruction transfers data between two registers R1 and R2. MVI places a 4-bit value into register RA.

The ALU instructions (INC, ADD, SUB, AND, OR) are similar to those of 74LS181. Registers are specified in the operands, and the RTL explains the actions required. Note that only 4 registers (RA, RB, RC, RD) are required. This allows you to encode instructions like ADD R1, R2 with 8 bits. SC and CC set and clear the carry flag respectively. Unlike the first experiment, SUB should not force a carry. Instead, the programmer should include a SC command prior to the start of a subtraction to get a twos’ complement result.

The PUSH function puts the contents of register R onto the top of a stack. The stack should be implemented in FPGA memory. The POP function moves the contents of the top of the stack into register R. Your implementation should specifically address overflow (pushing to a full stack) and underflow (popping an empty stack) issues.

The IN instruction reads a value from a set of DIP switches and places it into register R. OUT writes the contents of register R to a latch which should be instrumented with LED’s so that its value may be observed. NOP is the “no operation” instruction and, as such, does nothing.

Observe that BURP has no external memory; all program values are maintained within the 16 entry, 4-bit per entry memory. Again, in the specifications, R, R1 and R2 do not refer to specific registers but to any of four independently addressible registers (RA, RB, RC and RD) in your design. All these registers are implemented in FPGA.

8.3 Simulation of the entire BURP

By end of Week 8 of the quarter, you must simulate your entire Lab 2 BURP design with Modelsim and demonstrate that your simulation results are correct before programming FPGA. You should use Verilog for simulation. The Verilog models of the components used for building BURP are available on the class website or will be provided by TAs. You can easily add components, if necessary, to your Verilog. By building these models and simulating your complete BURP design, it should save your debugging time as you will benefit from being able to simulate and debug your design rapidly before implementing it on a bread board. Also note that the simulator will simulate program memory for you! Please refer to the online simulator help for more information.

9.0 Demonstration and report

As you implement your design, you should develop a program which completely tests each instruction. Before the demonstration, you will be provided with a test program as a benchmark for your project. If your system runs correctly, no problems should be encountered. You may choose to present additional programs which demonstrate any special features of your design. Besides a working test program, grading criteria will include design, efficiency, performance, and degree of pipelining. Your report should discuss the following areas:

1. Interconnect busses and muxes
2. Controller design details
3. Degree of parallelism
4. Clocking structure of the design