A simple 8-bit microprocessor

In this assignment, you will implement a simple microprocessor with 8 bit address and data buses. You can work in groups (maximum of 3 members), and submit one combined report per group.

Basic components

The basic components of a microprocessor are:

- ALU (Arithmetic and Logic unit, performs computations)
- Register file (used for temporary storage of data while operating)
- Instruction and Data memory (assumed to be separate for this exercise)
- Program counter (decides which instruction comes next)
- Control unit (generate control signals for all other units)

Instruction Format and Instruction Set

The uP uses 16 bit instructions that are stored in the IMEM, which is a ROM block that has an 8 bit address (since address bus is 8 bits), but gives out 16 bit data. There are 3 types of instructions:

Type	15-12	11-9	8-6	5-3	2-0
R (register)	Opcode	Rd	Ra	Rb	Func
I (immediate)	Opcode	Imm[5:3]	Ra	Rb	Imm[2:0]
J (jump)	Opcode	- don't care -		Addr (use 6:0 inst of 5:0)	

Assume that r0 always stores the value 0 (for easy comparisons) and ra, rb can refer to the same register.

The uP being designed has the following instructions:

Instruction	Opcode/Func	Type	Operation
Add rd, ra, rb	0000 / 000	R-type	Rd ← Ra + Rb
Sub rd, ra, rb	0000 / 010	R-type	Rd ← Ra - Rb
And rd, ra, rb	0000 / 100	R-type	Rd ← Ra AND Rb
Or rd, ra, rb	0000 / 101	R-type	Rd ← Ra OR Rb
Addi rd,ra, imm	0100	I-type	Rd ← Ra + imm
Lw rd, imm(ra)	1011	I-type	Rd ← DMEM[ra+imm]
Sw rd, imm(ra)	1111	I-type	DMEM[ra+imm] ← Rb
Beq rd, ra, imm	1000	I-type	If (ra==rb) pc ← pc + imm*2
J addr	0010	J-type	Pc ← addr*2

Notes:

- The Imm (immediate operand) is split into two parts to make it easier to implement the register file: otherwise Rd will have to be an output for some instructions.
- For the J instruction, the addr is 7 bits (6:0), not 6 bits as in the “imm” code

Implementation

You need 3 clocked units:

- program counter (PC).
- Data memory (DMEM)
- Register file (RF)

Combinational units:

- Instruction memory (IMEM): this need not be a clocked unit
- ALU: takes two inputs + suitable control, and generates output
- Control unit: generates MUX control signals based on the instruction set to make sure the data get processed correctly.

The following signals need to be generated by the control unit (if more are needed, specify clearly in your report):

- MemToReg: decides if data into register comes from memory or from ALU (for load instruction)
- MemWrite: decides if write enable is set for DMEM (store instruction)
- RegWrite: for all R-type operations, since the Rd register value must be updated. Also set for the Lw operation.
- ALUSrc: choose between Register output and Immediate operand for ALU
- Branch: true if the instruction was a Beq
- Jump: true if the instruction was a J (these last two are somewhat trivial, but still helps to think of them as separate control signals)

You need to implement each of the units described above as a separate Verilog module, with test benches to test correct operation for each one. Finally, put everything together and test it by entering a program to generate the Fibonacci series of numbers.

Monitoring and Debugging

- Push button 1 to be used as global system reset: bring PC to 0
- Push button 2 to be used as clock (each time you press, one clock cycle elapses. Obviously both of these buttons must be debounced)

- If SW3 ==0, then LED[7:0] to display contents of register[SW2:SW0] (use the other 3 switches as an index into the register file)
- if SW3 == 1, LEDs show the control signals above.

Report

- Demonstrate working design to TAs. We will also load a separate test code based on the instruction set mentioned here, so you need to make sure your system is able to implement all the instructions.
- Synthesis report: size and speed of your design.
- Analysis: where is the critical path, how can it be speeded up.

Bonus

- Add instructions to interface the DAC code written earlier to this module. How will you adapt the slower speed of the DAC to the processor?
- Hey, the sky is the limit. Use your imagination.

References

1. "Computer Architecture, A Quantitive Approach", J. L. Hennessy and D. A. Patterson, 4th Ed., Morgan Kaufmann, 2006
2. "MiniMIPS: An 8-Bit MIPS in an FPGA for Educational Purposes", C. Ortega-Sanchez, Intl. Conf. On Reconfigurable Computing and FPGAs, 2011.