CS_213: System Specification March 2006
M Seisenberger

Assignment 2: Microprocessor Specification

The Microprocessor - Overview The microprocessor you must specify is a 'generic' RISC processor that borrows from a number of existing architectures. The machine is a general purpose register machine - that is, any register can be used for any purpose (with one exception - the Zero register).

The State The state of the machine consists of the following:

- Memory. There is one memory, shared by programs and data. It consists of bytes, and memory addresses are 32-bits long.
- Registers. There is one set of 64 general purpose registers. Each register contains a 32-bit word. Register Zero always contains 0. That is, instructions can write results to register Zero, but they will be discarded, and when the contents of register Zero is next read, it will contain 0.
- Program counter. A 32-bit word.

Instruction Format All instructions are 32-bits long and have the same format. The format will consist of three register addresses and one opcode. Note that the number of registers dictates the size of a register address. The precise instruction format, and the assignment of opcodes to instructions, is left up to you. The same holds for the choice: Big-Endian or Little-Endian architecture for handling memory storage. The instructions available are listed below. In the following, `Ra, Rb` and `Rc` refer to any register. In each instruction (except after a jump) the program counter should be incremented by 4.

Instruction Set: Arithmetical/Logical

- **ADD32 Ra, Rb, Rc:** Perform the operation `Ra + Rb` and store the result in `Rc`.
- **ADD8 Ra, Rb, Rc:** As above except that only the least significant 8 bits of `Ra` and `Rb` should be added and the result stored in `Rc` should be sign extended to 32 bits.
- **MULT Ra, Rb, Rc:** Perform the operation `Ra * Rb` and store the result in `Rc`.

- **AND Ra, Rb, Rc:** Perform the operation `Ra & Rb` and store the result in `Rc`.
- **OR Ra, Rb, Rc:** Perform the operation `Ra | Rb` and store the result in `Rc`.
- **NOT Ra, Rc:** Perform the operation `not Ra` and store the result in `Rc`. `Rb` is not used.
- **SLL Ra, Rb, Rc:** Perform the operation `Ra << Rb` and store the result in `Rc`. (`Ra << Rb` shifts the contents of register `Ra`, (contents of) `Rb` positions to the left and fill the new positions with '0's.)

Instruction Set: Load/Store

- **LD32 Ra, Rb, Rc:** Load the four bytes starting at memory address `Ra + Rb` into register `Rc`.
- **ST32 Ra, Rb, Rc:** Store the contents of register `Rc` at memory address starting at `Ra + Rb`.

Instruction Set: Conditional/Jump

- **EQ Ra, Rb, Rc:** if `Ra == Rb` then store **0** in `Rc`; otherwise write **-1** to `Rc`.
- **GT Ra, Rb, Rc:** if `Ra > Rb` when `Ra` and `Rb` are considered to be unsigned numbers, then store **0** in `Rc`; otherwise write **-1** to `Rc`.
- **JMP Ra, Rb, Rc:** if `Ra == 0` then store the current program counter, incremented by 4, at memory address starting at `Rb` and set the program counter to `Rc`; otherwise increment the program counter by 4. (Note that this instruction could be used for both, a (conditional) jump and a (conditional) jump to subroutine.)

Expected is a full specification of the Microprocessor, i.e., Modules defining words, the instruction format, the registers, the memory, the state, etc,
A first class solution also contains a test module that runs the microprocessor. The example run should include tests from each instruction set and contain a jump to a subroutine and a return from it.

Submission: Please hand in your solution (yoursurname.maude) via Blackboard.