Experiment: Design and Synthesize a 32-bit Processor using Verilog

Design a 32-bit RISC-like processor in Verilog with the following specifications:

- Sixteen 32-bit general-purpose registers R0…R15, organized as a register bank with two read ports and one write port.
    - R0 is a special read-only register that is assumed to contain the fixed value of 0.
- Memory is byte addressable with a 32-bit memory address. For simplicity, in this design we shall assume that all operations are on 32-bit data, and all loads and stores occur from memory addresses that are multiples of 8.
- A 32-bit program counter PC
- A 32-bit stack pointer SP
- Addressing modes to be supported:
    a) Register addressing
    b) Immediate addressing
    c) Base addressing for accessing memory (with any of the registers used as base register)
    d) PC relative addressing for branch
    e) Indirect addressing
- The following instruction set has to be implemented:

    a) Arithmetic and logic instructions: ADD, SUB, AND, OR, XOR, NOR, NOT, SL, SRL, SRA, INC, DEC, SLT, SGT, LUI, HAM. There are corresponding immediate addressing versions with a suffixing "I" (like ADDI, SUBI, etc.). Assume that all shift instructions can have either 0 (no shift) or 1 (1-bit shift) as operand. Some example uses are as follows:

```
ADDI  R3,#25      // R3 <= R3 + 25
ADDI  R5,#-1      // R5 <= R5 - 1
ADD   R1,R2,R3    // R1 <= R2 + R3
SLA   R5,R7       // R5 <= R5 << R7[0]
SLAI  R5,#1       // R5 <=  R5 << 1
```

    b) Load and store instructions: LD, ST (all load and stores are 32-bits) and use register indexed addressing (any of the registers R1..R15 can be used). Some example uses are as follows:

```
LD    R2,10(R6)   // R2 <= Mem[R6+10]
ST    R2,-2(R11)  // Mem[R11-2] <= R2
```

    c) Branch instructions: BR, BMI, BPL, BZ. Some example uses are as follows:

```
BR    #10         // PC <= PC + 10
BMI   R5,#-10     // PC <= PC - 10 if (R5 < 0)
BPL   R5,#30      // PC <= PC + 30 if (R5 > 0)
BZ    R8,#-75     // PC <= PC - 75 if (R8 = 0)
```

    d) Register to register transfer: MOVE, CMOV. Some example uses are as follows:

```
MOVE  R10,R5      // R10 = R5
MOVE  R2,R0       // R2 = R0
MOVE  R7,SP       // R7 = SP

CMOV  R1, R2, R3  // if R1 = (R2 < R3) ? R2 : R3
```

e) Program control: HALT, NOP. The **`HALT`** instruction waits for an interrupt on an input pin "INT". **`NOP`** is a dummy instruction that performs no operation.

Steps of implementation:

1. Customize the ALU designed in the previous assignment in Verilog so as to cover all the functions as required to implement the above instruction set architecture. Write a test bench to verify the functionality through simulation.

   (***Deadline for submission: 2nd October 2024***)

2. Design the instruction encoding for the ISA. Make relevant assumptions where necessary, clearly mentioning the same in the documentation with justifications. Provide the overall schematic diagram of the data path, and also the detailed design of the control unit of the processor. You have to design a hardwired control unit.

   (***Deadline for submission: 15th October 2024***)

3. Implement the register bank and integrate the same with the ALU module as designed earlier. Hence write a top-level module for testing the modules by implementing operations like:

   `Rx = Ry op Rz`

   where Rx, Ry, Rz and op can be specified from outside. Also download the design on FPGA and test for the correct operation.

   (***Deadline for submission: 16th October 2024***)

4. Implement memory as a one-dimensional register array. Complete the processor design by going through the following steps:
   a) Implement the data path using structural design methodology.
   b) Prepare a table depicting the (sequence of) RTL micro-operations corresponding to typical instructions like ADD, ADDI, LD, ST, BMI, MOVE, CALL and RET, along with the corresponding control signals required at every step.
   c) Implement the control path using behavioural design of the required FSM.
   d) Download the design on FPGA and test the functionality.

(***Deadline for submission: 30th October 2024, there will be intermediate evaluation to assess the progress***)

5. Write programs corresponding to any three of the following problems and test them on the ISA, show the run on FPGA:

   a) Divide two integers using non-restoring division algorithm.

   b) Multiply two integers using Booth's algorithm.

   c) Sort a set of 10 integers using insertion sort.

   d) Recursive evaluation of factorial.

   (***Deadline for submission: 12th October 2024***)