Project 1 Part 2 of Assignment 1

The objective of this project is to develop a pipelined processor as follows:

1) Alternatives:
    a. Design, simulate, and implement using:
        i. VHDL or Verilog or schematic, or a combination thereof
        ii. Target physical implementation in an Altera (Flex10K) or Xilinx FPGA (Virtex II-Pro)
    b. Develop in a high level programming language of your choice (C, C++, Phyton, etc.), an architecture simulator, with a simple, text in terminal based interface. Free and good, i.e. sufficient, Windows based compilers can be found at: http://www.bloodshed.net. Alternatively, you can use the gcc under Linux or cygwin (Linux on Windows).
2) Register File:
    a. Four 8-bits registers: R0, R1, R2, and R3
3) Memory Organization:
    a. Harvard architecture
    b. 256 bytes of program memory (PM)
    c. 256 bytes of data memory (DM)
4) Four interrupt sources, with interrupt vectors located in the top four program memory locations. The priority is fixed, with the highest for the one located at the last location.
5) During a CALL, the state of the system is automatically stored on the stack. This is a software stack, with a base address at the last location of the data memory. The address of the Top of Stack (TOS) is stored in a special register: the stack pointer (SP). This is not user (programmer) accessible. The stack will grow from high to low addresses, so during a PUSH the SP is automatically decremented, and during a POP it is automatically incremented. The initial (after power-up or reset) value of the SP points to the top address in the data memory.
6) Interrupts are serviced starting with the end of an instruction cycle. The state of the system is saved as described in 5).
7) Upon a return from a subroutine (RET) or an interrupt service routine (ISR), the state of the system is restored from the stack, and execution continues with the next instruction.
8) Input / Output Peripherals are mapped in a separate address space than the memory address spaces
9) Addressing Modes:
    a. Immediate (IAM)
    b. Displacement (DAM)
10) Instruction Set:
    a. CPY Rx, Ry ; copy Rx ← Ry
    b. SWAP Rx, Ry ; swap Rx ← Ry
    c. LD Rx, #n ; load Rx ← immediate value n
    d. LD Rx, Ry, #d, M[Aeff] ; load Rx ← M[Aeff = Ry + d]; #d = displacement value
    e. ST Rx, Ry, #d, M[Aeff] ; store Rx → M[Aeff = Ry + d]; #d = displacement value

f. IN Rx, P[I/O-PAddrs] ; input Rx ← Input Peripheral pointed by I/O-PAddrs
g. OUT Rx, P[I/O-PAddrs] ; output Rx → Output Peripheral pointed by I/O-PAddrs
h. ADD Rx, Ry ; add Rx ← Rx + Ry
i. SUB Rx, Ry ; subtract ← Rx – Ry
j. AND Rx, Ry ; Rx ← Rx (bitwise AND) Ry
k. CEQ Rx, Ry ; compare if Rx and Ry are equal; if yes Z = 1
l. CLT Rx, Ry ; compare if Rx is less than Ry; if yes N = 1
m. NOT Rx ; bitwise complement Rx
n. MUL Rx, Ry ; multiply Rx ← Rx * Ry; to be implemented later
o. DIV Rx, Ry ; divide Rx ← Quotient(Rx/Ry); Ry ← Remainder(Rx/Ry); to be implemented later
p. SHLA Rx ; shift left arithmetic
q. SHLL Rx ; shift left logical
r. SHRA Rx ; shift right arithmetic
s. SHRL Rx ; shift right logical
t. JMP JTA ; jump unconditionally to the jump target address; the C, N, Z, V fields are all 0
u. BR <__>, BTA ; branch conditionally if one of the following conditions is true:

C	N	Z	V	<__>	Condition
1	0	0	0	C	C=1
0	1	0	0	N	N=1
0	0	1	0	Z	Z=1
0	0	0	1	V	V=1
1	1	0	0	CN	C&N=1
1	0	1	0	CZ	C&Z=1

v. CALL CTA ; call to call target address; the C, N, Z, V fields are all 1
w. RET <Rx> ; return from subroutine; optionally a return value in register Rx
x. RETI <Rx> ; return from an interrupt service routine (ISR); optionally a return value in register Rx
y. SIMD ; to be implemented later

Instruction Type	B7	B6	B5	B4	B3	B2	B1	B0	Instruction Mnemonic	# of OpCodes used	IW width (Bytes)
RR	OpCode				SRC1/ DEST1		SRC2/ DEST2		CPY, SWAP	2	1
LS	OpCode				SRC/ DEST		IAM/ DAM	LD/ ST	LD, ST	1	2
	Immediate Value / SRC || Displacement Value										
IO	OpCode				SRC/ DEST		X	I/O	IN, OUT	1	2
	I/O-Ps Address										
AL	OpCode				SRC1/ DEST		SRC2				1
SR	OpCode				SRC/ DEST		L/R	Arith/ Logic	SHLA, SHLL, SHRA, SHRL	1	1
JBC	OpCode				C	N	Z	V	JMP, BR	1	2
	Jump/Branch/Call Target Address										
RET	OpCode				Intr/ Subr	Ret. Val.	DEST RET Val		RET, RETI	1	1
SIMD	OpCode				TBD				TBD	1	2

OpCode = Operation Code

SRC = Source Register Address (Name)

DEST = Destination Register Address (Name)

IAM/DAM = 0/1

L/R = Shift Direction (Left = 0; Right = 1)

I/O = Input/Output = 0/1

Arith/Logic = 0/1

Intr/Subr = Interrupt/Subroutine = 0/1

Ret.Val = Return Value Exists (No = 0; Yes = 1)

The project is due on Tuesday 04/03/12 at 10pm in the corresponding dropbox on mycourses.

To test your processor, in this iteration of the project you are free to use any instruction sequence (program), as long as it contains at least one instance of every instruction of the instruction set, except for MUL, DIV, and SIMD, which will be implemented later. You will have to provide an annotated assembly code listing, which you will have to convert manually to machine code.

You have to make provisions to collect the following metrics during the execution of a benchmark:

- Total instruction count (IC)
- The use frequency of each instruction
- The use frequency of each addressing mode
- Total number of pipeline stalls

For alternative 1a, you can use structural or behavioral (functional) modeling, or a combination thereof. You can further use any number of necessary busses, muxes, tristates, and manipulation units. The control unit (CU) can be hardwired or microprogrammed, or a combination thereof.

For alternative 1b, the architecture simulator will be an executable program, which will read in the user program, and eventual input data, and outputs the above metrics and the last state of the system (for debugging purposes) to the terminal or a file.

In both cases, use as a template the MIPS pipelines discussed in class.

Keep in mind that in the next iterations of the project you will be asked to enhance the processor with additional features. Thus, creating a modular, well-annotated design or program is highly recommended.

By Tuesday 03/27/12 I want a one page, informal description of:

- The alternative you have decided to implement
- How you are gone do it
    - Development environment and hardware used
    - High level block diagram – DP and CU

Grading:

- Working RR, AL, and SR instruction types = 20
- Working LS and IO instruction types = 15
- Working JBC and RET instructions = 15
- Informal report describing what you have done (1 page)