CS 220 : PROJECT

PROJECT DESCRIPTION :

We designed a general purpose computer with the following funtionalities:

1) Load-store architecture
2) 8 bit instructions
3) Memory: 64 bytes for program instructions
4) Program: 64 bytes for program data
5) A combinatorial module for addition and substraction having two CLA adders combined together to perform 8-bit operations.
6) A booth multiplier optimised to multiply even negative numbers and complete the multiplication in 8 cycles.
7) A display module which will display the 8-bit binary pattern in an integer format on the three seven segment displays.
8) Both the memory modules are optimised to used least possible circuitary (one 16x1 MUX and one 4x1 MUX). A read operation takes 2 cycles and a write operation takes 3 cycles.

DESIGN:
Memory Design : We have created two memory modules as follows -

1) 64 bytes to store 8-bit program instructions
2) Second 64 bytes are further divided as follows:
    a) First 32 bytes to store and save data by user
    b) Another 32 bytes for maintaining stack in case of function calls.

USER INTERFACE:
FPGA configuration:

1) Toggle Buttons:
    a) btn0 : Program reset – It will clear all data and stack memory and put program counter to 0 to re- enter the program.

    b) btn1 : Execute – The program counter is updated to 0 in order to begin program execution.

    c) btn2 : Program input – This toggle button is pressed each time a 8 -bit instructions is fed into program memory using switches and program counter is increased by 1 for next instruction.

    d) btn3 : This button helps move program counter backwards in case

any previous instruction has to be edited.

2) Switches : Switches are used to provide 8-bit instructions and user input if some value has to be read from user.

Our system has following user interface :

1) Program counter is displayed in integer format while feeding instruction set in order to keep the track of line number.
2) The system waits for user to give input until execute button(btn1) is pressed again. Till then “Inp” is displayed. It, therefore, helps taking multiple input from user.

CPU REGISTERS:

1) We have used four CPU registers (r0, r1, r2, r3) accesible to user to carry out all the computations.
2) In addition three registers, which are not accesssible to user (pc, link, sp) are used for following purposes -
   a) pc(program counter) – To keep track of current program line which is executing. Jump and branch instructions require updating pc to specified line.

   b) link – This register is maintained for jump instruction. It maintains stack address for storing the pc before updating it for jump in order to return back to same address, allowing multiple function calls. This stack grows upwards from memory address '63'.

   c)sp(stack pointer) – This register keep track of current stack address for storing or retrieving memory.This stack grows downwards from memory address '32'.

EXTRA FUNCTIONALITIES:

1) In addition to basic opcodes, we have implemented following additional opcodes for following instructions:
   a) Taking input from user during program execution.
   b) Branch jump after comparing values stored in registers.
   c) Swapping values stored in 2 registers.
   d) Display any register value.
   e) Sleep for number of seconds specified by user.
   f) Return back to caller after jump instruction.
   g) Stack implementation

2) Our system can implement multiple function calls by maintaing stack pointer and link registers as follows:
   a) Every time a jump instruction is executed, registers specified by user is stored in stack memory which can be restored by user before the function ends.
   b) Link register to keep track of latest function call by storing its return address in stack memory.
3) Our system provides enhanced user interface.

OPCODE LIST:

SNo.	OPCODES FORMAT	DESCRIPTION	NUMBER OF CYCLES
1	00000000	No opcode	1
2	00000001	User input: waits for user to give input from switches and press btn1 during program execution	1
3	00000010	Return back to the address pointed by the link in stack memory and incrementing link pointer by 1.	3
4	00000011	Halt	1
5	000001<ra>	Right shift value stored in register <ra> by 1.	1
6	000011<val>	Sleep for <val> number of seconds	NA
7	000010<ra>	Load immediate 8bit following this instruction to register <ra>	3
8	0001<ra><rb>	r0=<ra>+<rb> Add values stored in registers <ra> and <rb> and store the result in r0	2
9	0010<ra><val>	r0=<ra>+<val> Add value stored in <ra> and immediate <val> and store the result in r0	2
10	0011<ra><rb>	r0=<ra>-<rb> Subtract values stored in registers <ra> and <rb> and store the result in r0	2
11	0100<ra><val>	r0=<ra>-<val>Subtract value stored in <ra> and immediate <val> and store the result in r0	2
12	0101<ra><rb>	{r0,r1} = <ra> *<rb> Multiply values stored in <ra> and <rb> and store the higher bits in r1 and lower bits in r0	12
13	011 <address>	Store instruction : It stores the value of r0 in memory according to the <address>(5 bits)	4

		specified by the user	
14	100 {0,1}{0,1} <target>	Branch instruction : It jumps to the address (<target> * 8) by comparing values in registers. Registers are selected as: a) If fourth bit from left is 0, r0 will be selected else r1, say ra. b) similarly, if fifth bit from left is 0, r2 will be selected else r3, say rb. Now, jump will take place, if ra >= rb	1
15	1010<ra><rb>	Move value stored in <rb> to <ra>, i.e <ra> = <rb>	1
16	1011<ra><rb>	Swap values stored in <ra> and <rb>	2
17	110 <address>	Load instruction: Load the value stored at memory address <address> specified by the user and store it in register r0	4
18	1111<ra><val>	Load immediate: Load the immediate value <val> to the register <ra>	1
19	111000<ra>	Jump instruction : Jump to the instruction by updating pc to the address stored in register <ra> Link pointer store the current pc before jumping in stack memory and decrementing link pointer by 1.	5
20	111001<ra>	Display the value stored in the registers <ra> in integer format	1
21	111010<ra>	Load the value stored in the register <ra> in stack memory and incrementing sp by 1	4
22	111011<ra>	Restore the value pointed by sp in stack memory in register <ra> and decrementing sp by 1.	3

LIST OF PROGRAMS THAT CAN BE IMPLEMENTED:

Our system is capable of running following programs:
Factorial, Fibonacci, Countdown and Countup timer, Multiplication table, Calculator and many more.....

Program with multiple function calls and even recursive functions can be run on the system.