CS230-RISC Design

Hardik Rajpal 200050048
Harsh Shah 200050049
Harshvardhan Agarwal 200050050
Parth Dwivedi 200050100

Contents

1 Abstract

The goal of this project is to implement a RISC processor, being able to perform all the below given instructions(implementation described in the later sections).
Assumptions: Assumed op code for LHI = 0011. Added extra op code for program termination: OC_TER = 0100

Instructions Encoding:

ADD:	00_01	RA	RB	RC	0	00
ADC:	00_01	RA	RB	RC	0	10
ADZ:	00_01	RA	RB	RC	0	01
ADL:	00_01	RA	RB	RC	0	11
ADI:	00_00	RA	RB	6 bit Immediate		
NDU:	00_10	RA	RB	RC	0	00
NDC:	00_10	RA	RB	RC	0	10
NDZ:	00_10	RA	RB	RC	0	01
LHI:	00_00	RA	9 bit Immediate			
LW:	01_01	RA	RB	6 bit Immediate		
SW:	01_11	RA	RB	6 bit Immediate		
LM:	11_01	RA	0 + 8 bits corresponding to Reg R0 to R7 (left to right)			
SM:	11_00	RA	0 + 8 bits corresponding to Reg R0 to R7 (left to right)			
BEQ:	10_00	RA	RB	6 bit Immediate		
JAL:	10_01	RA	9 bit Immediate offset			
JLR:	10_10	RA	RB	000_000		
JRI	10_11	RA	9 bit Immediate offset			

Figure 1: Instruction Set

2 Overview

The execution of the instructions is carried out by maintaining a set of 14 states. Each state determines what action needs to be preformed, and after the completion of the task, the state is change to either move to the execution of next part of same instruction or to fetch new instruction. The states are as follows:

1. Initialise the memory with pre-defined instructions stored in the memory component
2. Housekeeping: Reading the program counter from the register file, sending it to the ALU component for an increment and to the memory component
3. Instruction fetch: Reading the instruction outputted by the memory and passing it to the instruction register.
4. PC update : Writing the output from the ALU to register 7.
5. Jump and link to register state
6. Load multiple and store multiple
7. Read state of load multiple
8. Store multiple
9. Write back state of load multiple
10. Write state of store multiple
11. General state for write back
12. Execute
13. Memory Access
14. Change program counter

3 Implementation of the memory

The memory component has the following 'in' ports:

1. state: denoting the state of the machine
2. init: to store predefined instructions in the memory
3. mr: bit for memory read
4. mw: bit for memory write
5. dataPointer: denoting address in the memory
6. di: data in

It has only one 'out' port:

1. do: data out

4 Implementation of the instructions

4.1 ADD/ADC/ADZ/ADL/ADI

The execution all the above instructions was carried out using the ALU component. The data stored in the operand registers are given as inputs to the ALU and addition operation is selected using the selector of the ALU. The conditions(for zero bit and carry bit) is also checked in the same state as the one for giving inputs to the ALU. For addition of immediate value, one of the operands is not a register address. The state is then changed to "WBTR(write back to register)".

4.2 NDU/NDC/NDZ

Similar to the addition instructions, NAND operations are also carried out using ALU operations. However in this case, the operation selector of the ALU is changed to execute NAND operation. Again the next state is changed to "WBTR(write back to register)".

4.3 LHI

Since this instruction involves changing the value of register, the immediate value, once fetched from the instruction, is passed to the component of register file, with the write bit set to '1'. The state is changed to the housekeeping state thereafter.

4.4 LW/SW

For both the above instructions, the execution state involves use of ALU component to find the memory address after adding immediate value to address stored at register B. Then the state is changed to "memory access" state. In the memory access state, the data for register file component and write bit is set appropriately.

4.5 LM/SM

The execution state of the above instructions involves setting the intial register(where the data may be inserted based on the first bit of immediate part of instruction). The flow is then passed to ST_LMSM, which is only used to branch the flow based on load or store instructions. Irrespective of the branch taken, the program would continue executing the read state for 8 times(the number of registers), after which the flow is passed to the housekeeping state.

4.6 BEQ

The execution state of this instruction again involves setting the inputs (current pc value and the immediate 6-bit value from the instruction) to ALU. The selector of ALU is addition for this case. The flow is passed to CPC(Change Program Counter) state. There, the program counter is changed (that is, branching takes place) only if the outputs from the register file (fetched appropriately) are equal. Flow is then passed to housekeeping state.

4.7 JAL/JLR/JRI

For JAL, the inputs are set for ALU, whereas for JLR the inputs to register file are changed. In the next state of above to instructions, the required inputs for writing to register file are set. For JRI, both register file and ALU are required to be accessed, and hence the inputs are set accordingly. Finally, the flow of all three instructions move to CPC, to change the program counter.

5 Testing of the architecture

Testing of the above described implementations of instructions using the “report” functionality in VDHL. We also probed the data stored in “Memory” and “registerfile” components to verify the outputs of the instructions.

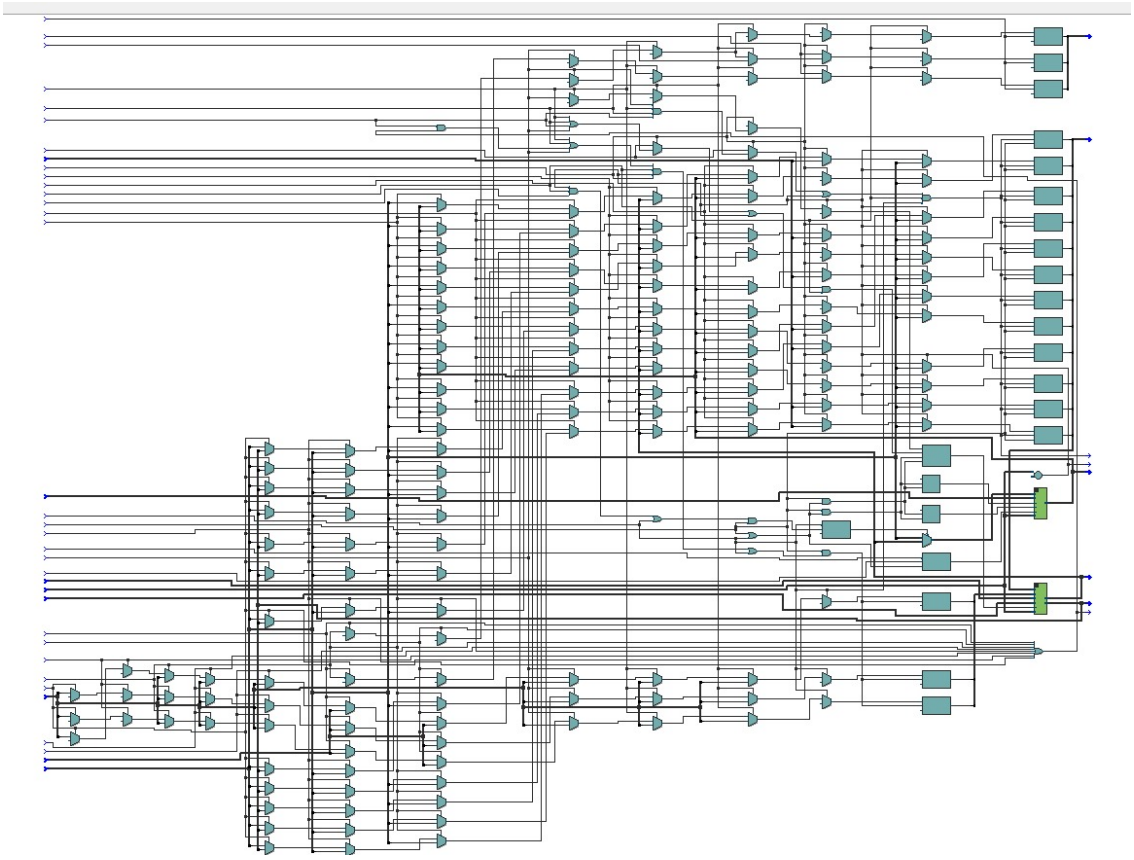

Figure 2: Final architecture(Part 1)

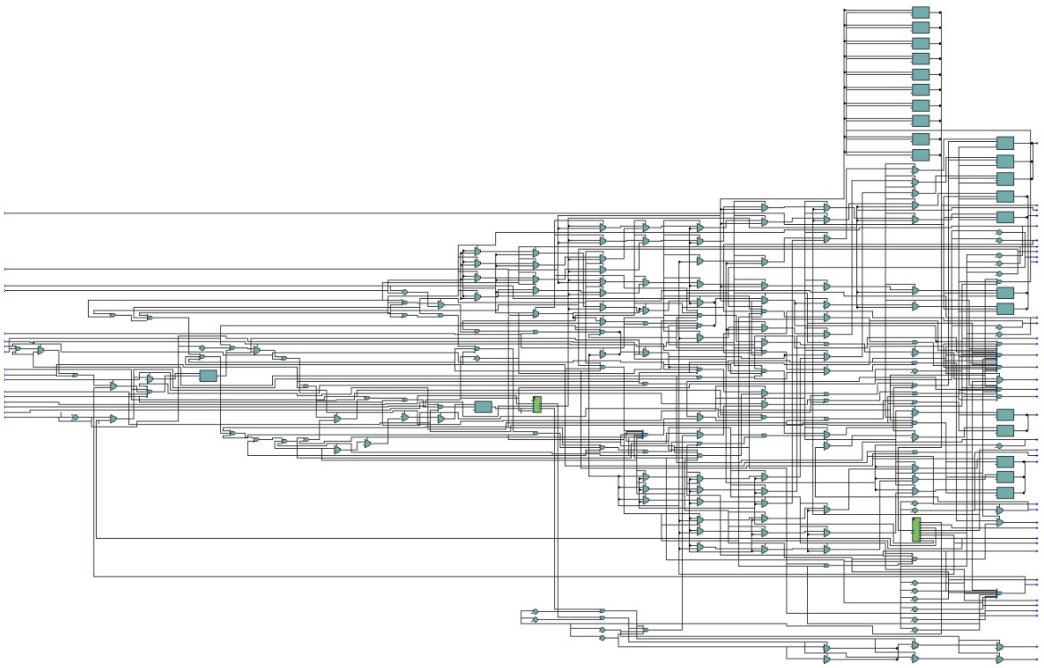

Figure 3: Final architecture(Part 2)