IITB-CPU

Project Guide : Prof. Virendra Singh

Project Report for EE224: Digital Design

Priyansh Jain - 210070063
Sajal Sachin Deolikar - 210070071
Priyanshu Gupta - 210070064
Sannidhya Kaushal - 210070076

Contents

1 Problem Statement

We have to make a 16-bit very simple computer based on the Little Computer Architecture. IITB-CPU is an 8-register, 16-bit computer system. It has 8 general-purpose registers (R0 to R7). Register R7 always stores Program Counter. PC points to the next instruction. All addresses are short word addresses (i.e. address 0 corresponds to the first two bytes of main memory, address 1 corresponds to the second two bytes of main memory, etc.). This architecture uses a condition code register which has two flags Carry flag (C) and Zero flag (Z). IITB-CPU is very simple, but it is general enough to solve complex problems. The architecture allows predicated instruction execution and multiple load and store execution. There are three machine-code instruction formats (R, I, and J type) and 15 instructions. They are illustrated in the figure below.

R Type Instruction format

Opcode (4 bit)	Register A (RA) (3 bit)	Register B (RB) (3-bit)	Register C (RC) (3-bit)	Unused (1 bit)	Condition (CZ) (2 bit)

I Type Instruction format

Opcode (4 bit)	Register A (RA) (3 bit)	Register C (RC) (3-bit)	Immediate (6 bits signed)

J Type Instruction format

Opcode (4 bit)	Register A (RA) (3 bit)	Immediate (9 bits signed)

Instruction Formats

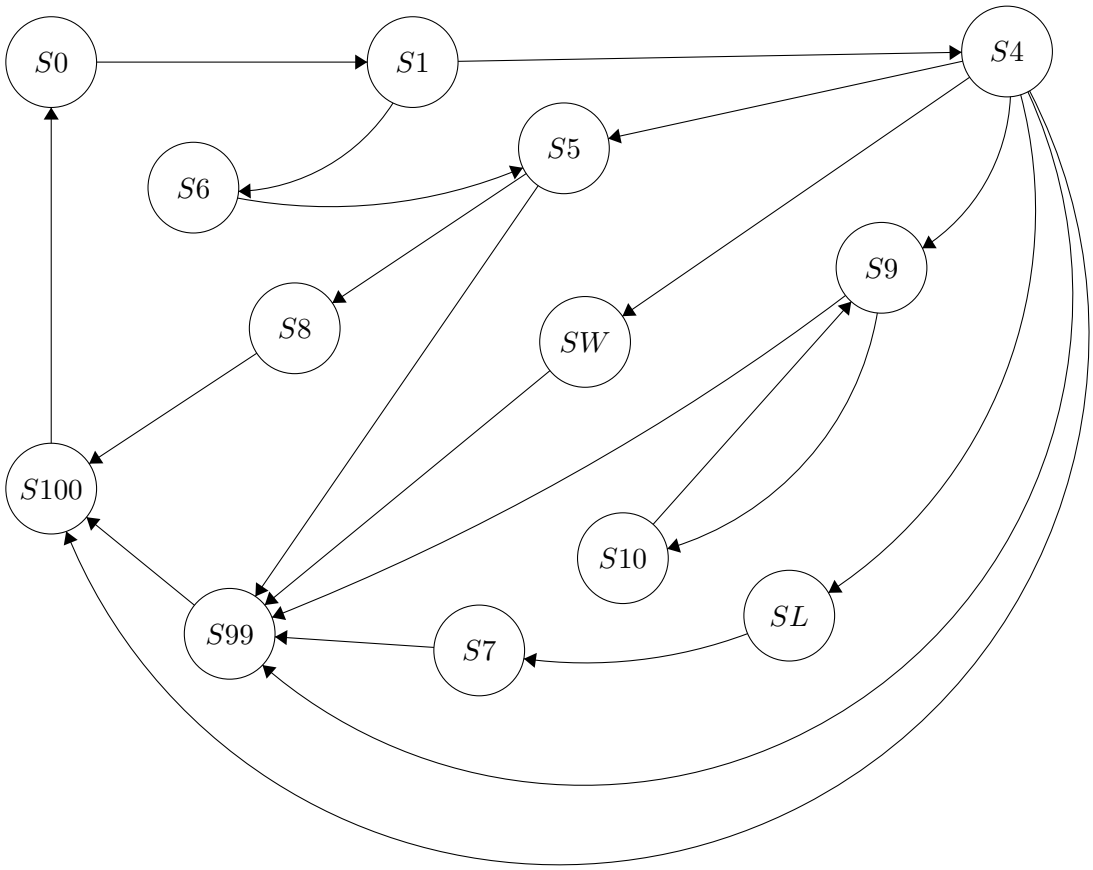
ADD:	00_00	RA	RB	RC	0	00
ADC:	00_00	RA	RB	RC	0	10
ADZ:	00_00	RA	RB	RC	0	01
ADI:	00_01	RA	RB	6 bit Immediate		
NDU:	00_10	RA	RB	RC	0	00
NDC:	00_10	RA	RB	RC	0	10
NDZ:	00_10	RA	RB	RC	0	01
LHI:	00_11	RA	9 bit Immediate			
LW:	01_00	RA	RB	6 bit Immediate		
SW:	01_01	RA	RB	6 bit Immediate		
LM:	01_10	RA	0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

Instructions Encoding

2 Components

- **Multiplexers:** A multiplexer (or mux) selects between several input signals and forwards the selected input to a single output line.
- **Shifter:** A Shifter multiplies or divides by powers of 2. Thus, it shifts a binary number left or right by a specified number of positions.
- **Sign Extenders:** Sign Extender performs an operation in which the number of bits representing a specific value is increased while preserving the original sign and value. A Sign Extender or a Sign Extension Unit is a black box representation of such an operation. In practice, this unit is almost never a unit of its own but rather part of a more complex unit.
- **ALU:** In the computer system, ALU is a main component of the central processing unit, which stands for arithmetic logic unit and performs arithmetic and logic operations. It has the ability to perform all processes related to arithmetic and logic operations such as addition, subtraction, and shifting operations, including Boolean comparisons. Also, binary numbers can accomplish mathematical and bitwise operations. The arithmetic logic unit is split into AU (arithmetic unit) and LU (logic unit). The operands and code used by the ALU tell it which operations have to perform according to input data. When the ALU completes the processing of input, the information is sent to the computer's memory.
- **Clock:** Produces a square wave clock signal with a constant frequency.
- **Memory:** Memory is the electronic holding place for the instructions and data a computer needs to reach quickly. It's where information is stored for immediate use. In the IITB-CPU 16-bit registers are used as memory blocks for the data and instructions storage.
- **Register:** Registers are data storage devices that hold binary information.
- **Temporary Registers:** Used to hold temporary data\intermediate result during an arithmetic and logic operation. These registers are not available to the programmer.
- **Instruction Register:** An instruction register holds a machine instruction currently being executed. A variety of registers serve different functions in a central processing unit (CPU) – the instruction register is to hold currently queued instructions for use.
- **Register File:** A register file is a means of memory storage within a computer's central processing unit (CPU). The computer's register files contain bits of data and mapping locations. These locations specify certain addresses that are input components of a register file. Other inputs include data, the read-write function and the execute function.
- **Program Counter:** The program counter (PC) is a register that manages the memory address of the instruction to be executed next. The address specified by the PC will be + n (+1 for a 1-word instruction and +2 for a 2-word instruction) each time one instruction is executed.
- **FSM (Finite-State Machine):** Used as a mathematical model of computation, it is an abstract machine that can be in exactly one of a finite number of states at any given time. Our Design has 13 states.
- **Datapath:** A datapath is a collection of functional units such as arithmetic logic units or multipliers that perform data processing operations, registers, and buses. Along with the control unit, it composes the central processing unit (CPU).
- **Assembler:** An assembler is a program that takes basic computer instructions and converts them into a pattern of bits that the computer's processor can use to perform its basic operations.
- **Testbench:**A testbench is an HDL module that is used to test another module, called the device under test (DUT). The testbench contains statements to apply inputs to the DUT and, ideally, to check that the correct outputs are produced. The input and desired output patterns are called test vectors.

3 FSM & State Descriptions



3.1 S0

State working	Write Lines	Select Lines
PC − > mem_address	IR - write	
mem_data − > IR		

3.2 S1

State working	Write Lines	Select Lines
IR_{11-9} − > RF_A1		
RF_D1 − > T1	T1 - write	
IR_{8-6} − > RF_A2		
RF_D2 − > T2	T2 - write	

3.3 S4

State working	Write Lines	Select Lines
T1 − > ALU_mux1_00		ALU: ”00” => Sum
T2 − > ALU_mux2_00		ALU: ”01” => Sub
ALU_mux1 − > ALU_a	T3 - write	ALU: ”10” => Bitwise NAND
ALU_mux2 − > ALU_b		ALU mux depends on Opcode
ALU_out − > T3_mux_0		T3_mux <= ’0’
T3_mux − > T3		

3.4 S5

State working	Write Lines	Select Lines
T3 − > RFinput_mux_0		
RFinput_mux − > RF_datain	RF - write	

3.5 S6

State working	Write Lines	Select Lines
IR_{8-0} − > Shifter_input		
Shifter_output − > T3_mux_1		T3_mux <= '1'
T3_mux − > T3	T3 - write	

3.6 S7

State working	Write Lines	Select Lines
T4 − > RF_datain	RF - write	RF_addressmux <= "000", RF_datamux <= '1'

3.7 S8

State working	Write Lines	Select Lines
ALU_out − > T3_mux_0		ALU: "00" => Sum, T3_mux <= '0'
T3_mux − > T3	T3 - write	ALU mux Selection depends on Opcode

3.8 S9

State working	Write Lines	Select Lines
T3 − > mem_address	T1 - write	Mem_address_select <= '1'
mem_data − > T4	T2 - write	meminput_mux <= '1'
T3 − > T1	T4 - write	RFOS <= '1'
A4 − > RFaddress_mux_5		

3.9 S10

State working	Write Lines	Select Lines
(T1 + 1) − > T3	T3 - write	ALU: 00 => Sum
T6 − > T5 (T5 − > Priority_enc)	T5 - write	ALU_mux1<="00", ALU_mux2 <= "10"
if LM, T4 − > RF	RF - write	T5_mux <= "10"
if SM, T2 − > mem_data, T3 − > mem_address		memdata_mux <= '1', memaddres_mux <= '1'

3.10 SL

State working	Write Lines	Select Lines
T3 − > mem_address		
mem_out − > T4	T4 - write	mem_addressmux <= '1'

3.11 SW

State working	Write Lines	Select Lines
T3 − > mem_address		mem_addressmux <= '1'
T1 − > mem_datain	mem - write	

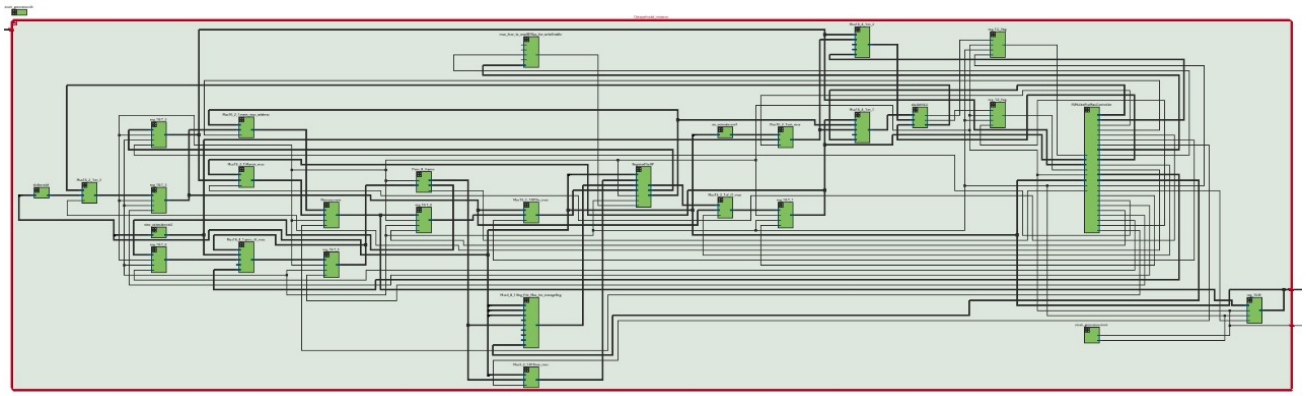
3.12 S99

State working	Write Lines	Select Lines
(ALUa_2 + ALUb_3) − > T3_mux_0		ALU: 00 => Sum
T3_mux − > T3	T3 - write	ALU_mux1<="01", ALU_mux2 <= "10"

3.13 S100

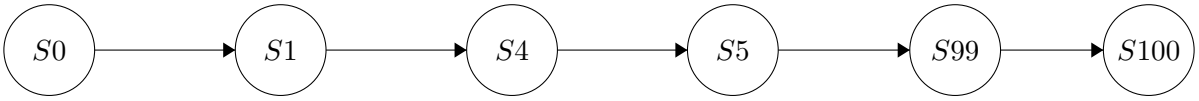
State working	Write Lines	Select Lines
T3 − > RF_input	RF - write	RF_addressmux <= ”011”

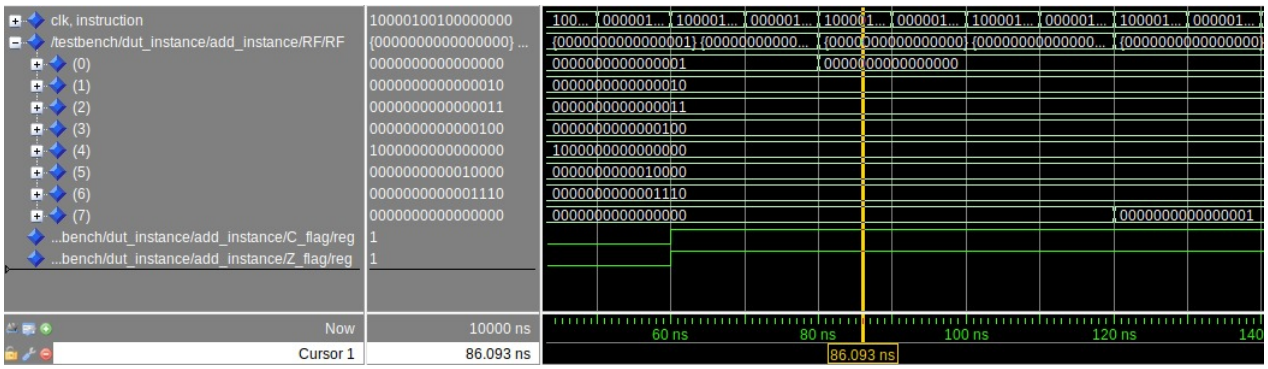
4 Datapath Netlist



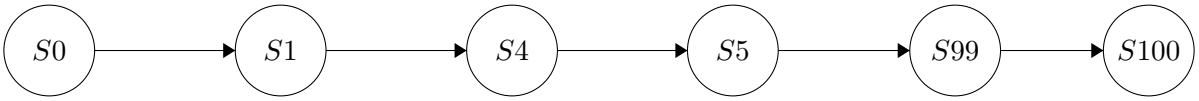
5 Flowcharts & Simulations

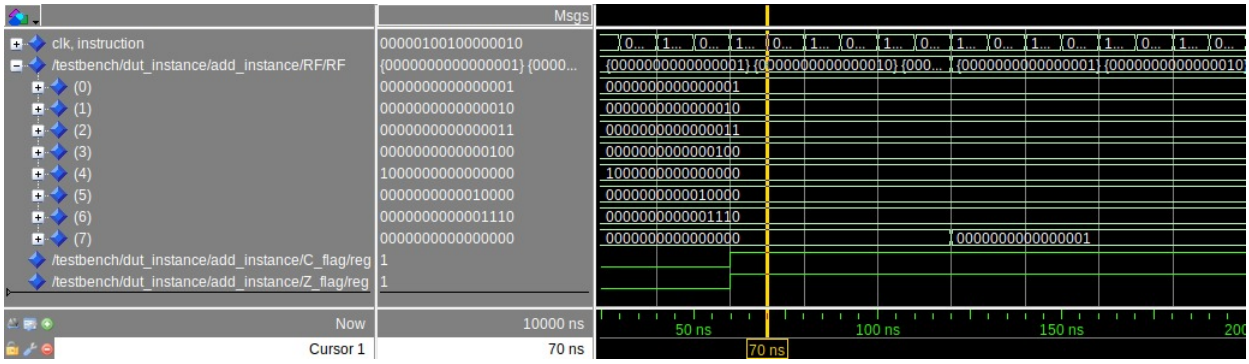
5.1 ADD



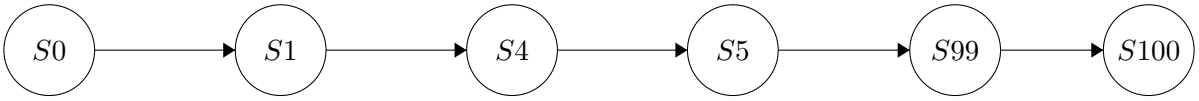


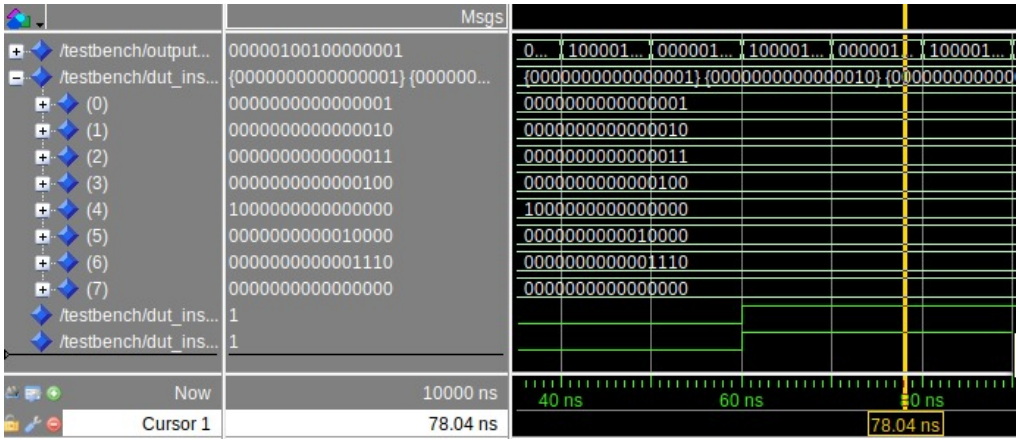
5.2 ADC



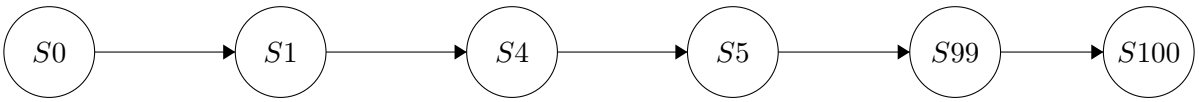


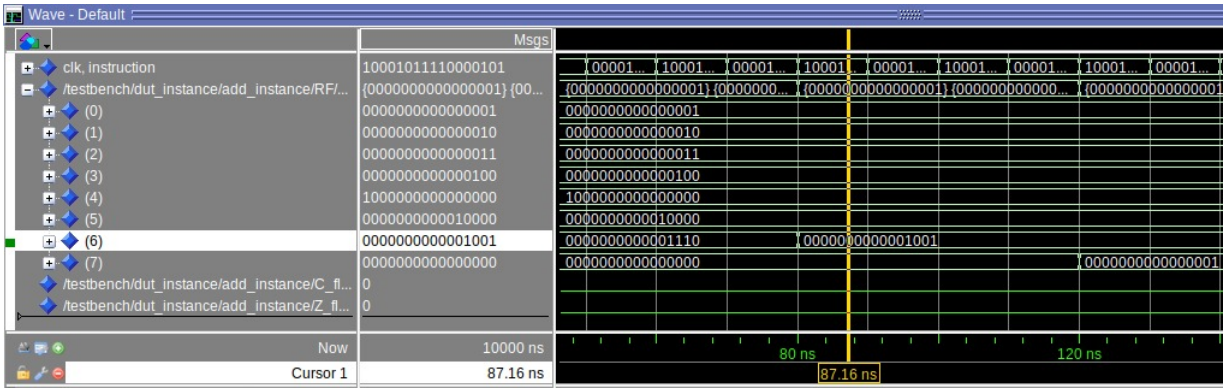
5.3 ADZ



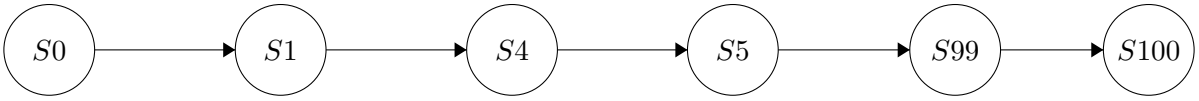


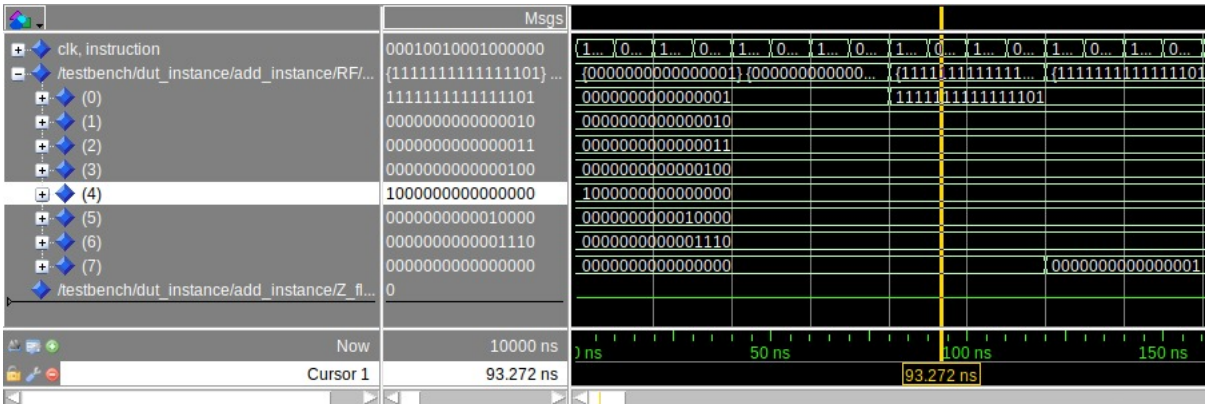
5.4 ADI



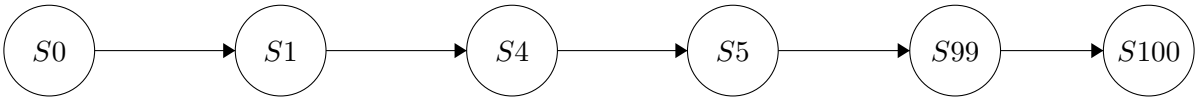


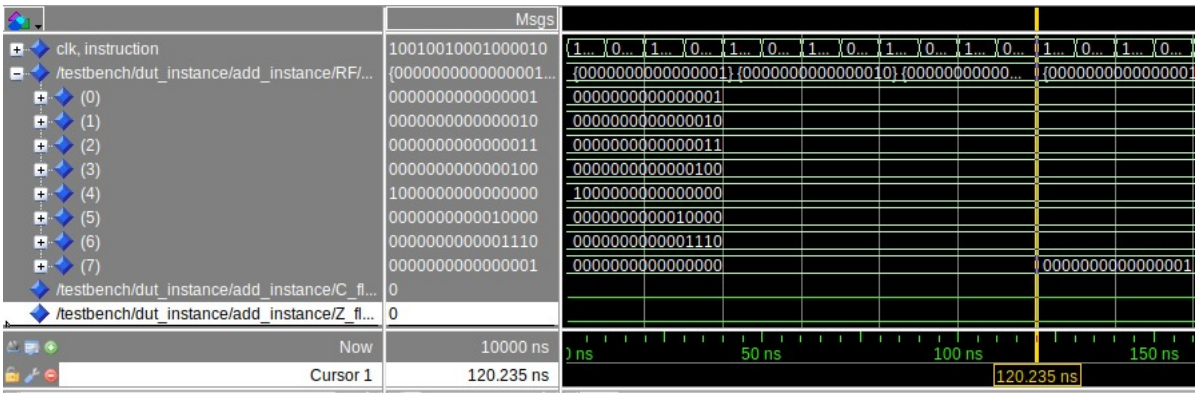
5.5 NDU



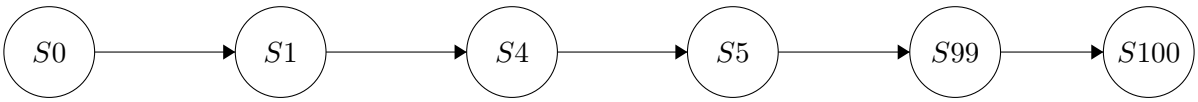


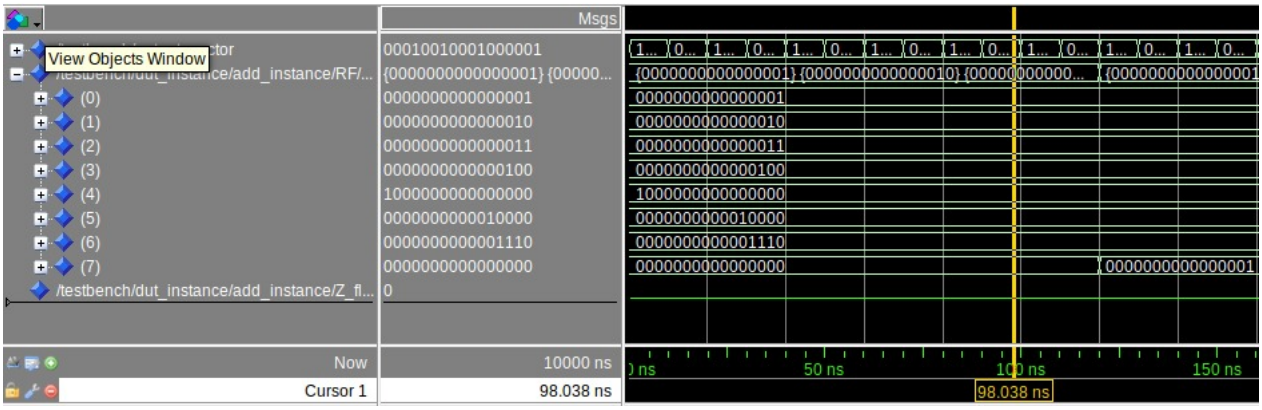
5.6 NDC



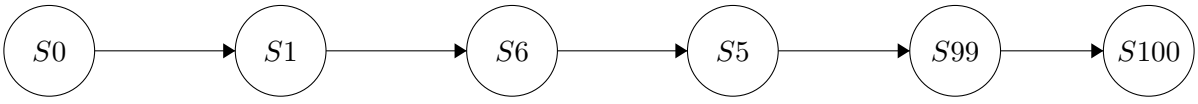


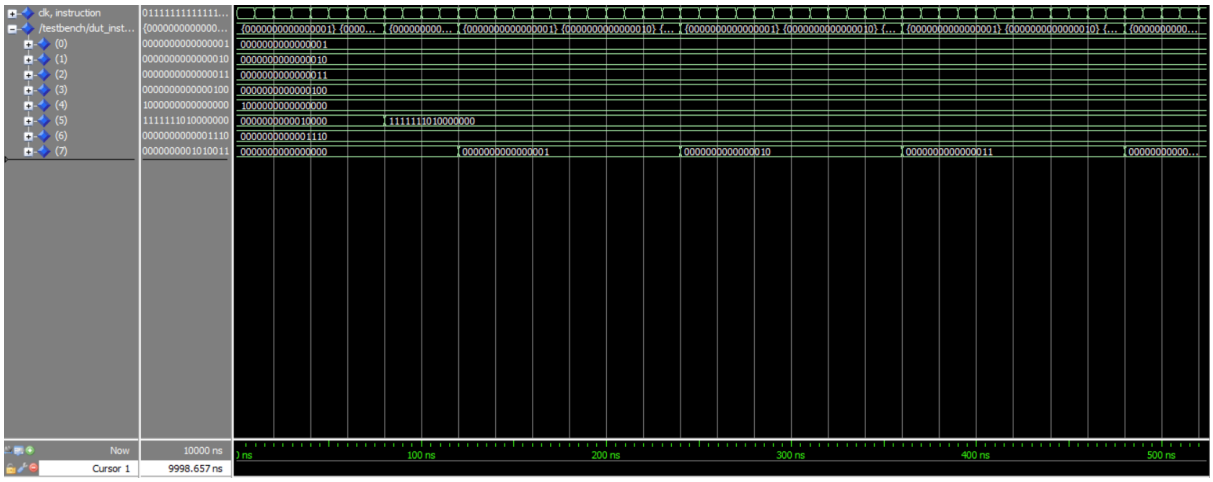
5.7 NDZ



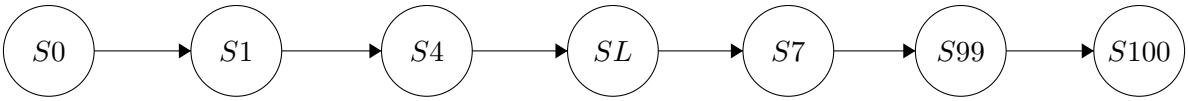


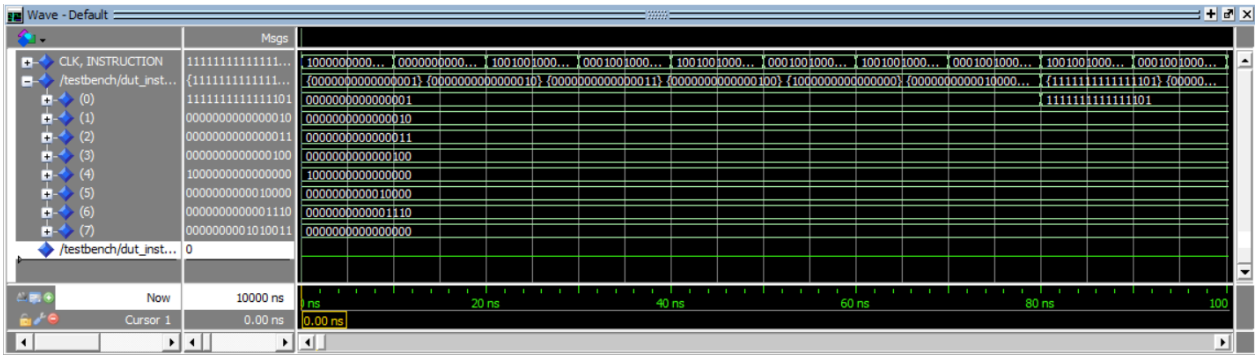
5.8 LHI



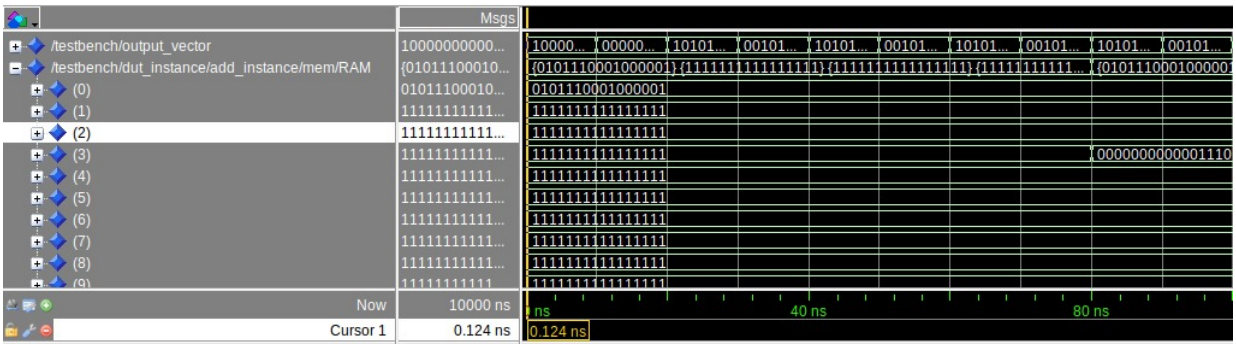


5.9 LW

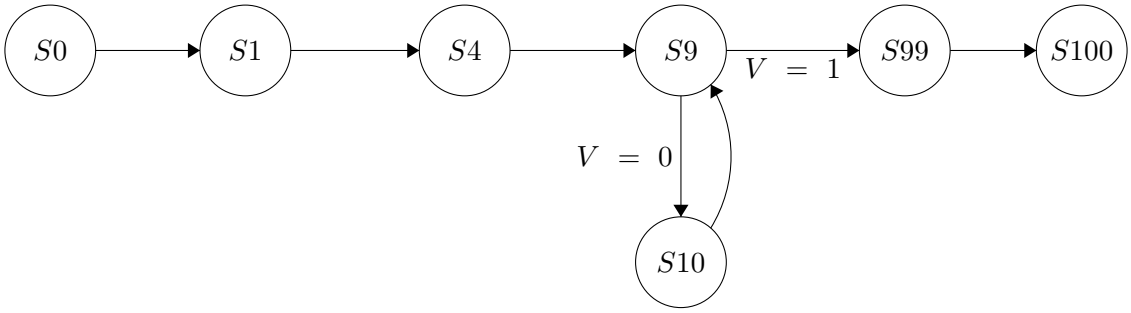


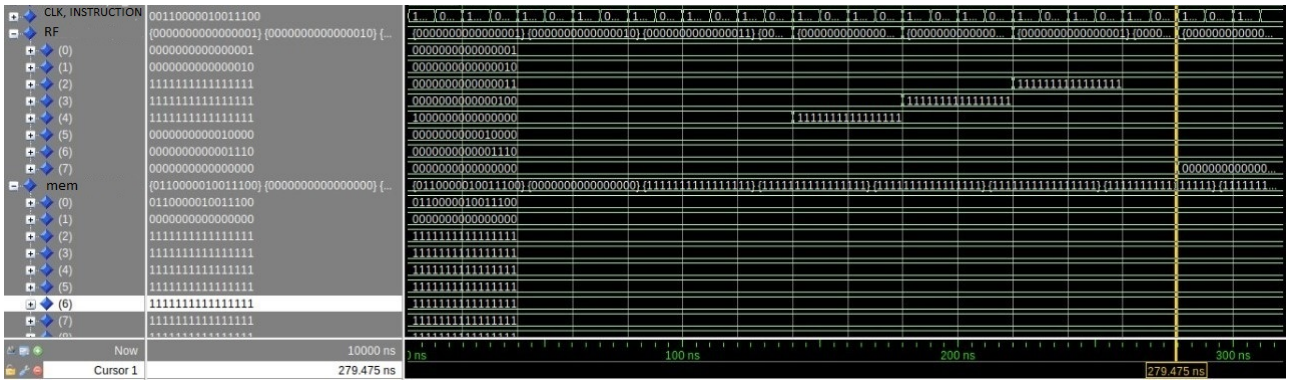


5.10 SW

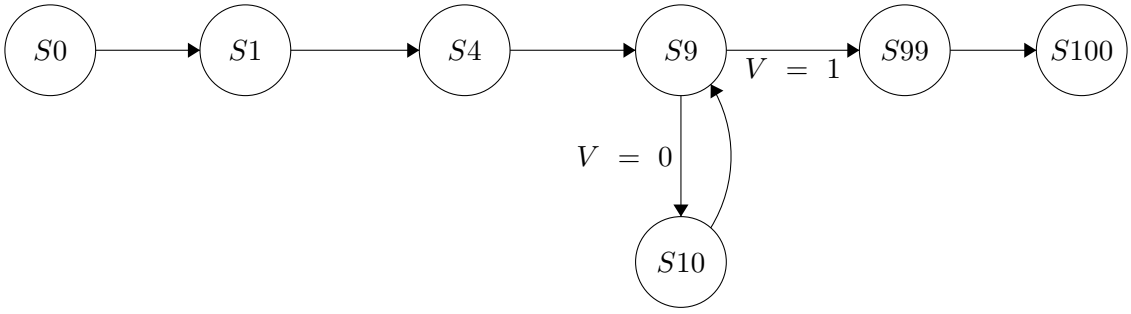


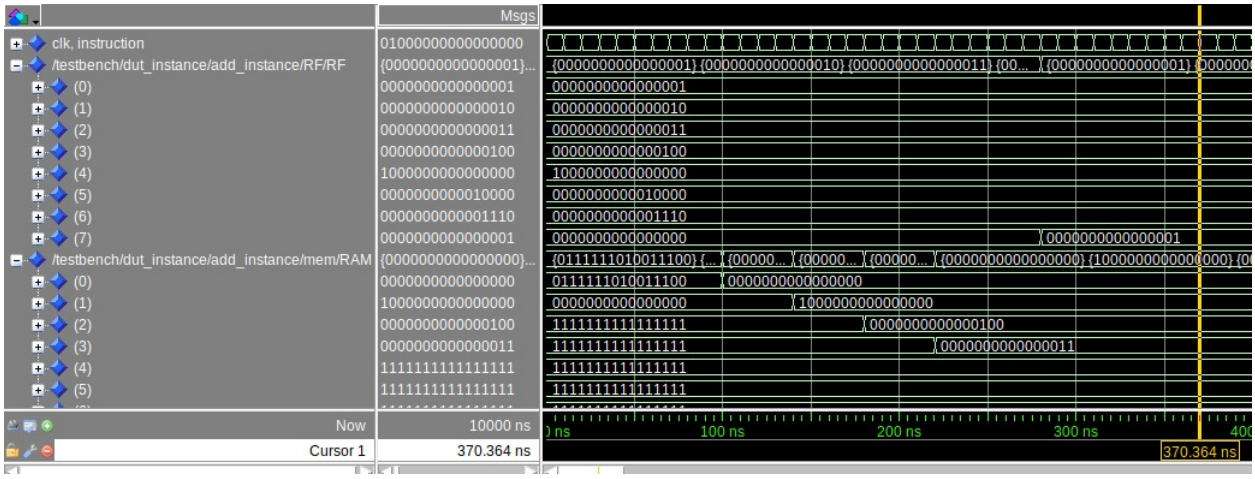
5.11 LM



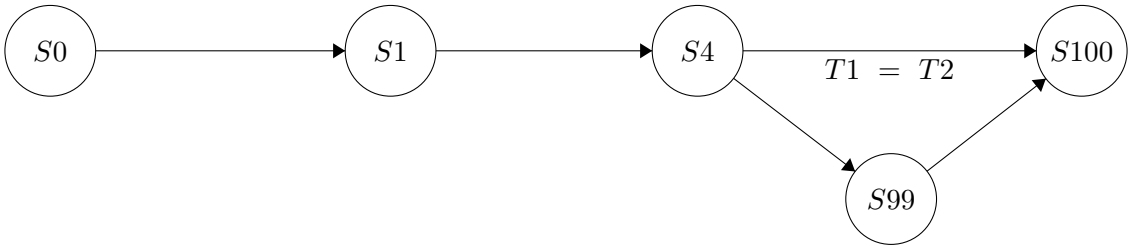


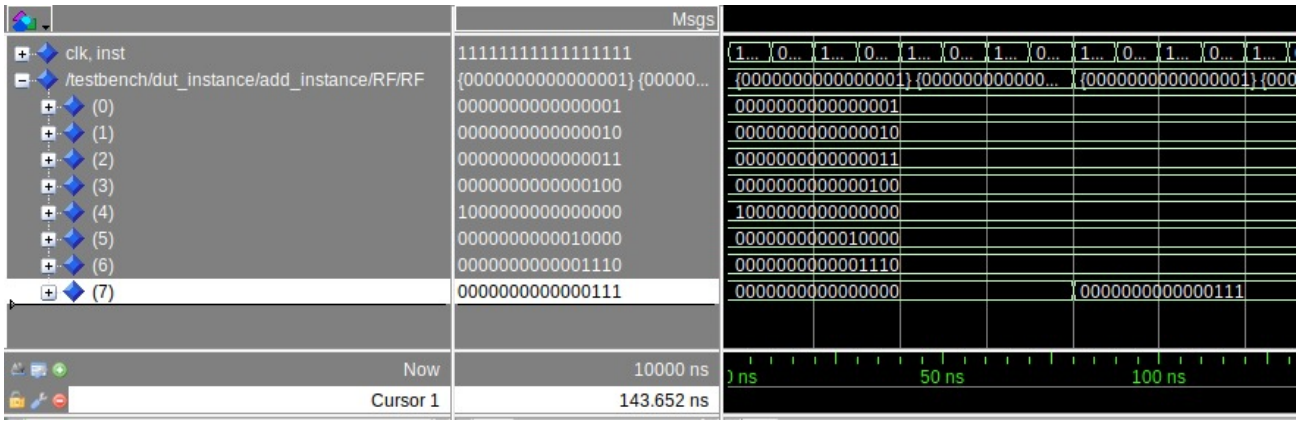
5.12 SM



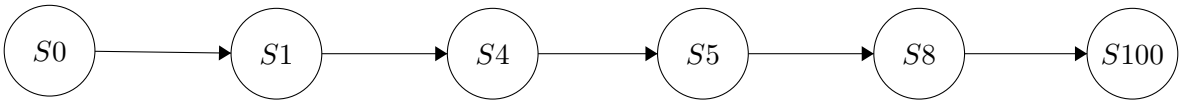


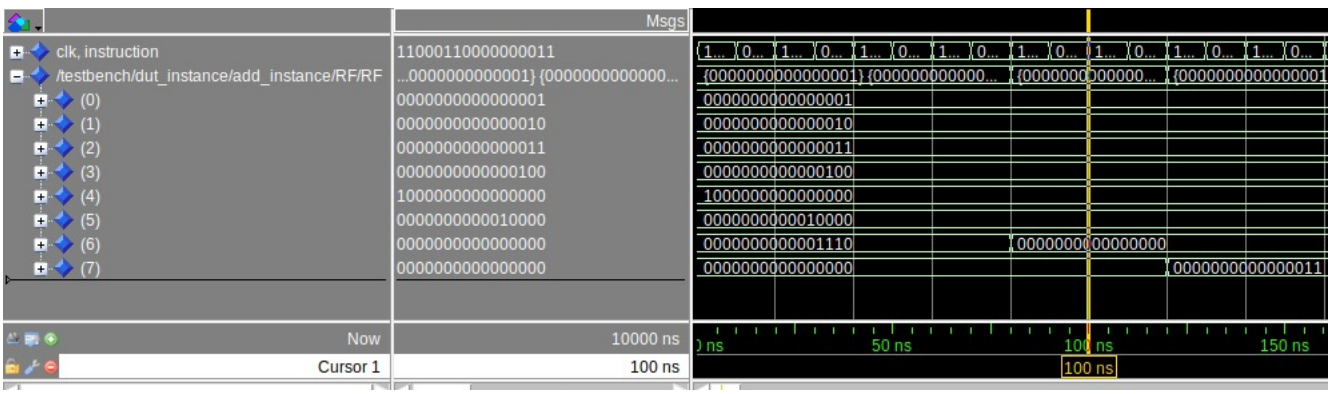
5.13 BEQ





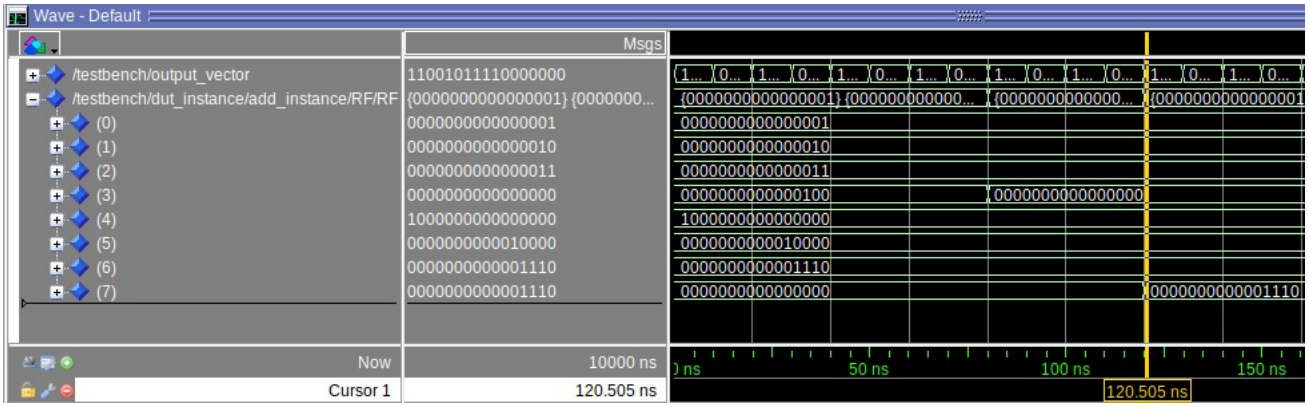
5.14 JAL





5.15 JLR


S0
S1
S4
S5
S8
S100


Wave - Default
Msgs
/testbench/output_vector
/testbench/dut_instance/add_instance/RF/RF
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
11001011110000000
{0000000000000001} {0000000...
0000000000000001
0000000000000010
0000000000000011
0000000000000000
1000000000000000
0000000000010000
0000000000001110
0000000000001110
Now
10000 ns
Cursor 1
120.505 ns
50 ns
100 ns
150 ns
120.505 ns