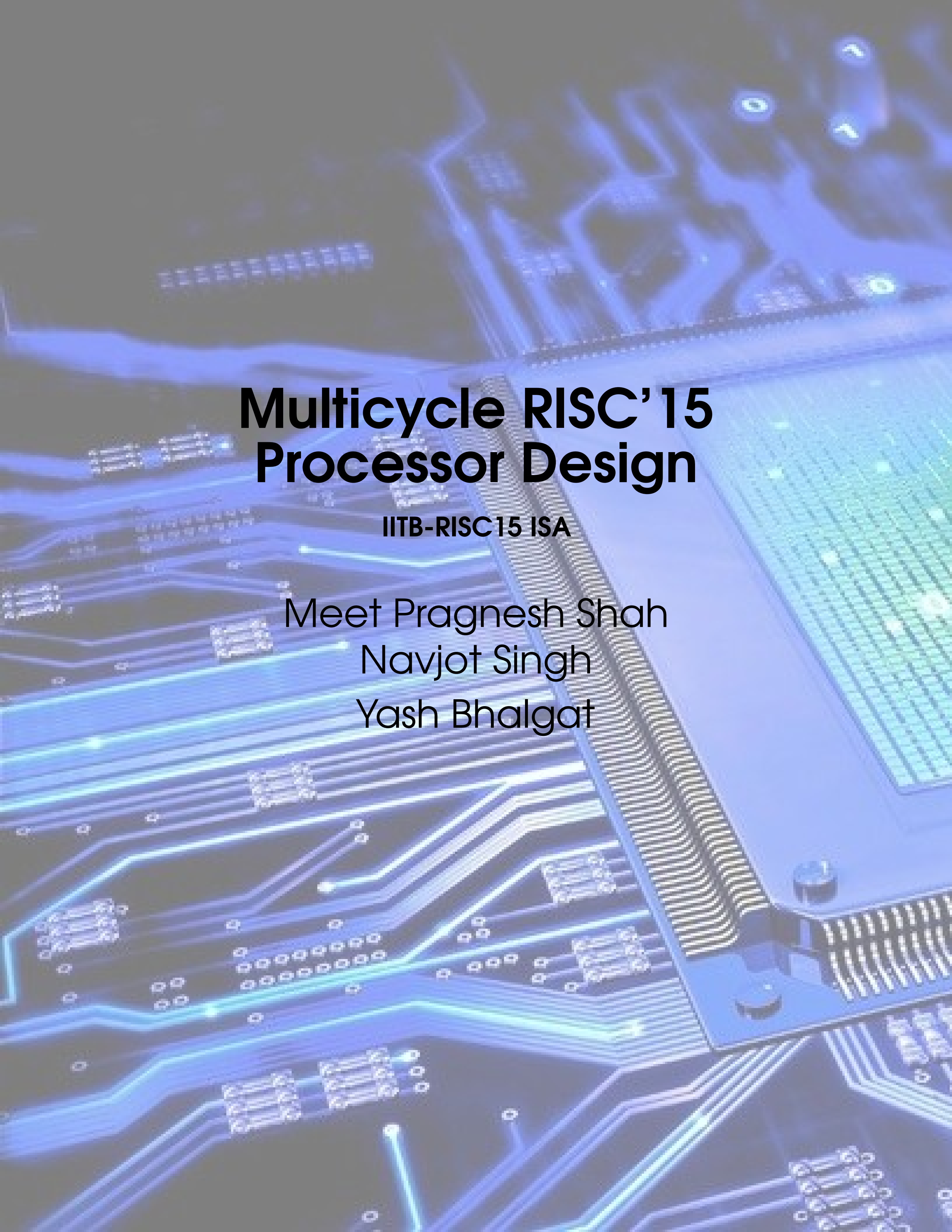


Multicycle RISC'15 Processor Design

IITB-RISC15 ISA

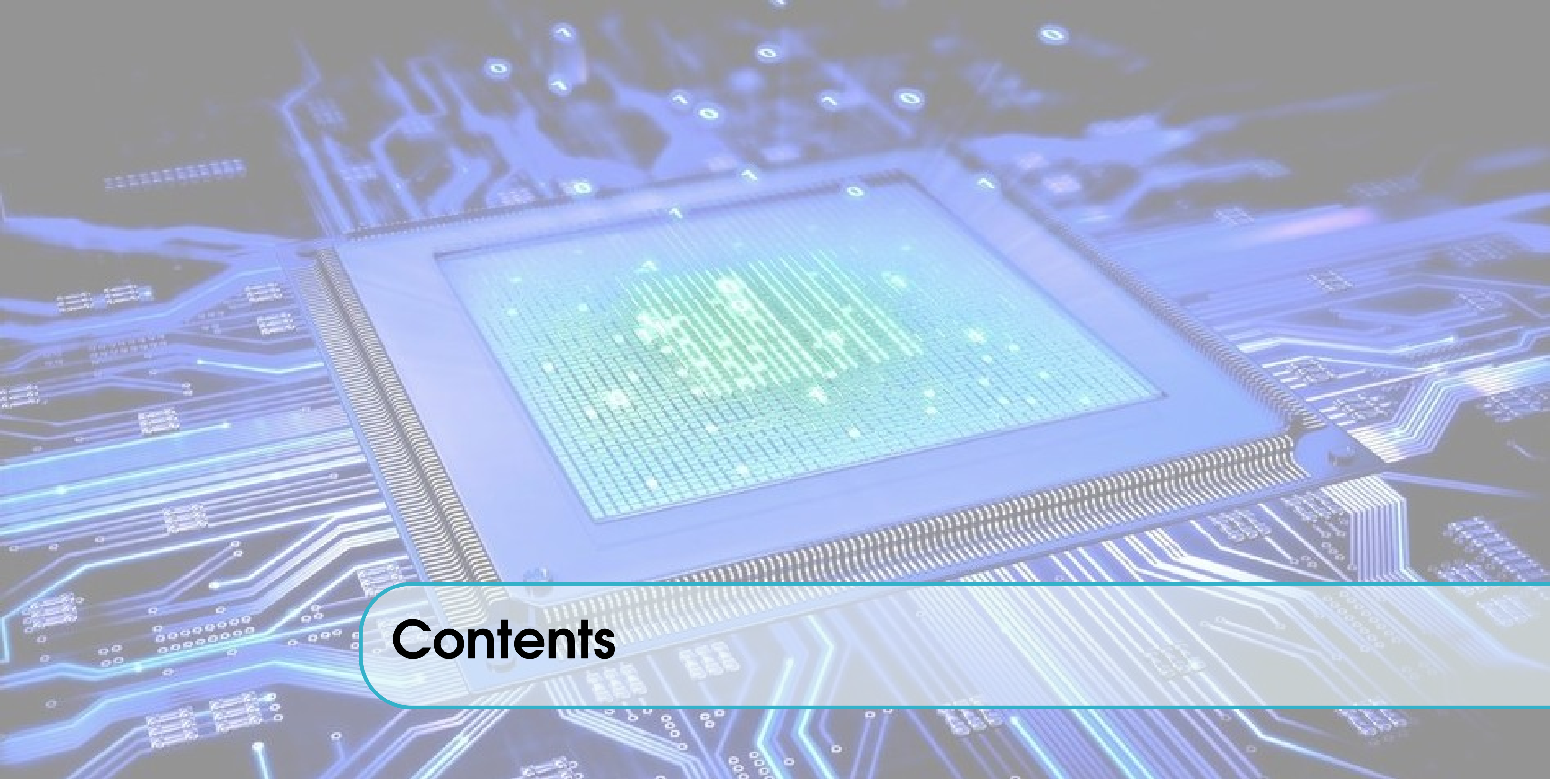
Meet Pragnesh Shah
Navjot Singh
Yash Bhalgat

EE309 (MICROPROCESSORS) / EE337 (MICROPROCESSORS LABORATORY)

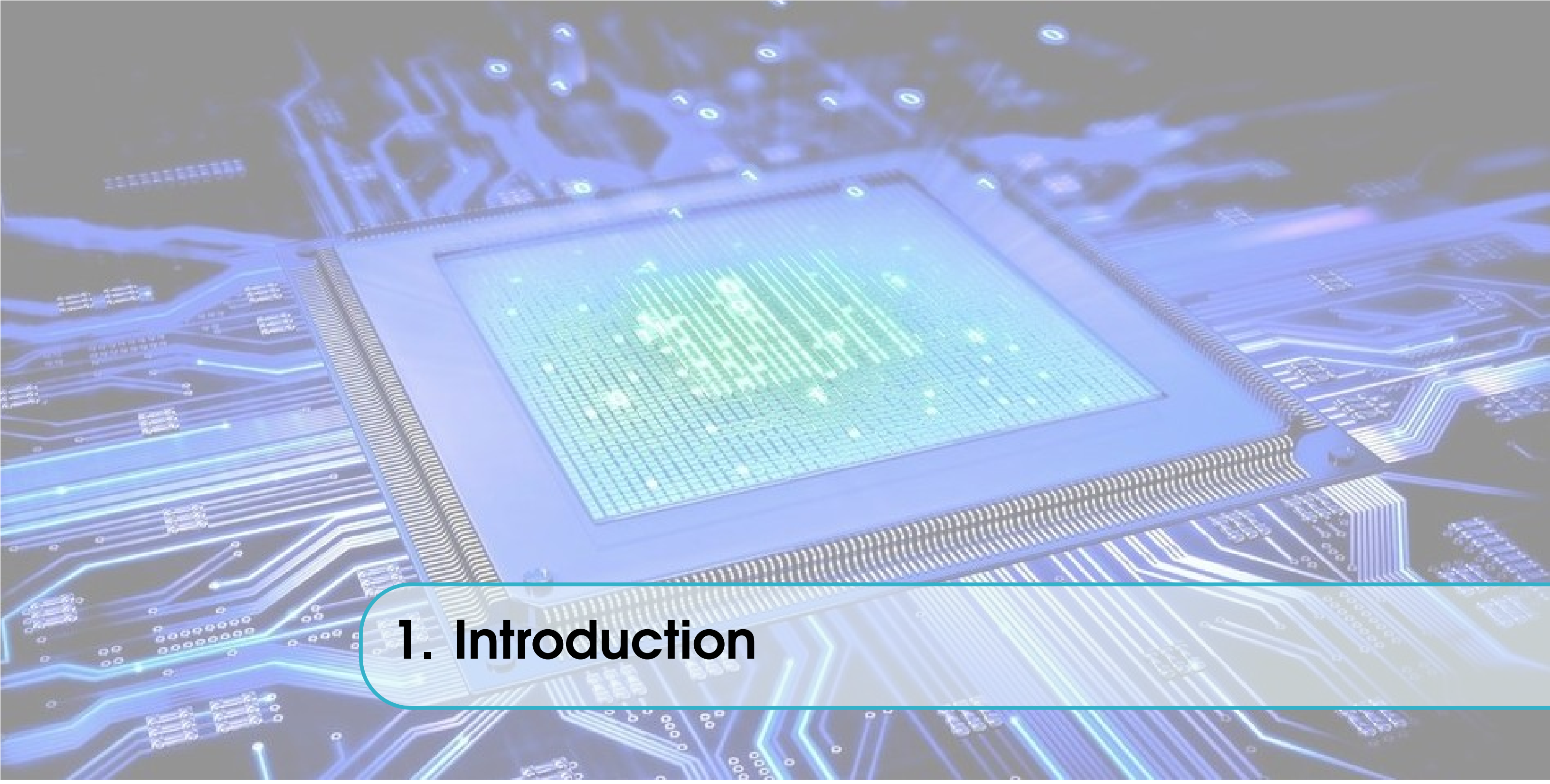
INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY

https://github.com/meetshah1995/Multicycle-RISC15-Design

This project was done under the supervision of Prof. Virendra Singh, instructor for the course along with the lab course instructors Prof. Shankar Balachandran and Prof. Rajbabu Velmurugan and our TA Mrs. Shoba Gopalakrishnan



Contents



1. Introduction

1.1 Prologue

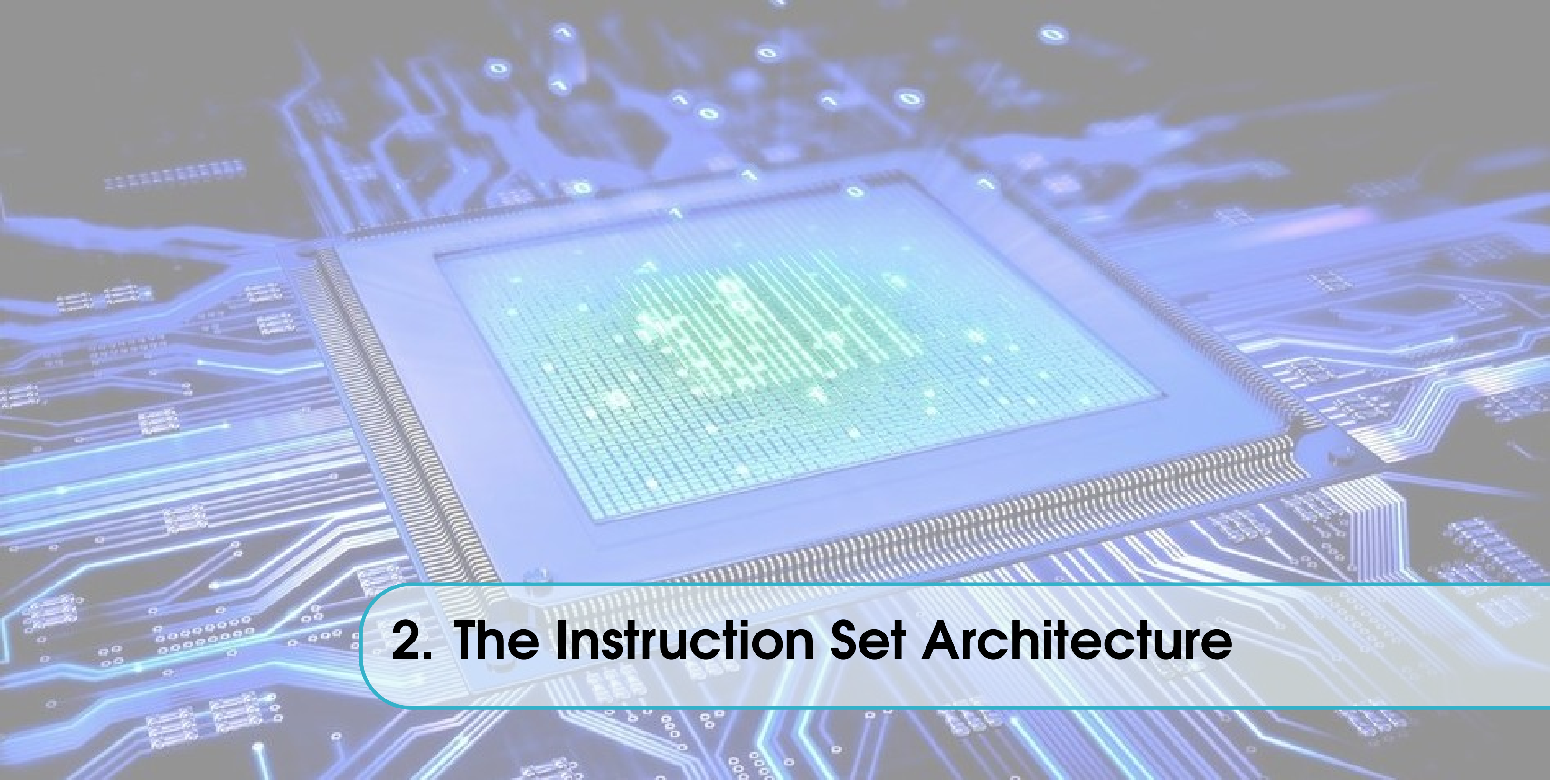
The IITB-RISC'15 processor is based on Little Computer Architecture. It has a 16 bit architecture, having 8 registers (R0 to R7) and uses point to point connections along with a condition code register with flags CY (Carry) and Z (Zero). The IITB-RISC'15 is very simple but, it is general enough to solve complex problems. The architecture allows predicated instruction execution and multiple load and store execution. There are 3 types of instruction formats (namely R, I and J) and a total of 15 instructions. They would be presented in a later chapter.

1.2 Abstract

The Processor can be divided into 3 componenets : (i) Datapath , (ii) Contoller , (iii) Memory .The design is based on multicycle RISC architecture. This is done because the different instructions take different execution times, and with a multicycle implementation, each instruction gets over in lesser time and hence, the performance is optimized. This also reduces the resources which would be required during simulation on FPGA. It is restricted to have the value of PC (program counter) to always be stored in R7.

1.3 Software & Packages Used

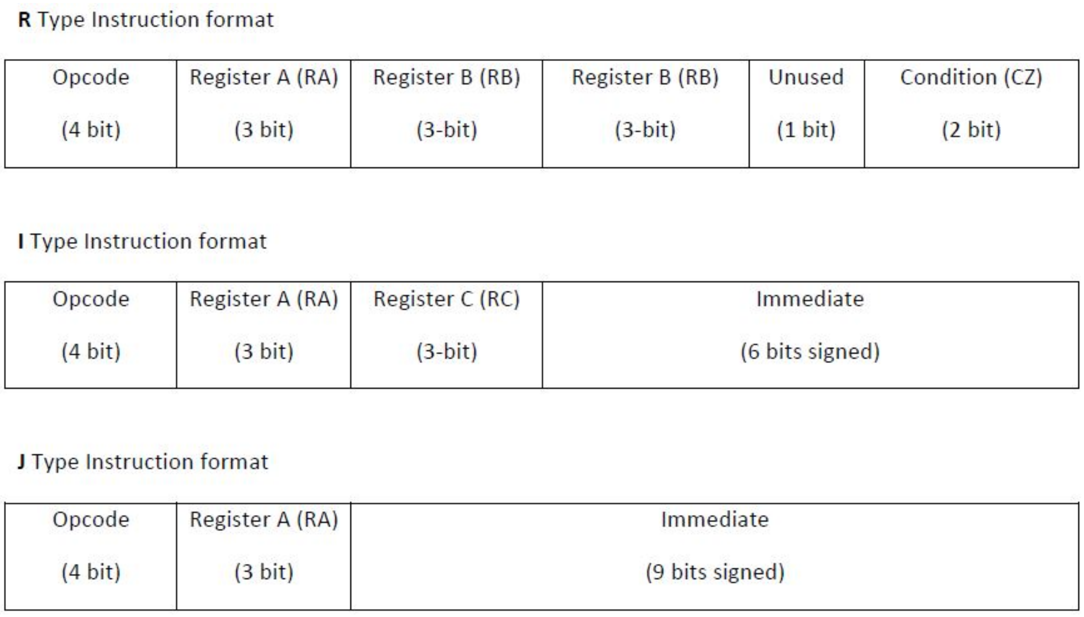
- Altera Quartus v.14.1
- ModelSim
- Sublime Text Editor [Special Thanks]
- Coffee ;)



2. The Instruction Set Architecture

2.1 IITB-RISC'15 ISA

The architecture allows predicated instruction execution and multiple load and store execution. There are 3 types of instruction formats (namely R, I and J) and a total of 15 instructions. The datapath evolves accordingly as these instruction execution flow is designed, using different MUX control signals and read/write enable signals.



R Type Instruction format

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

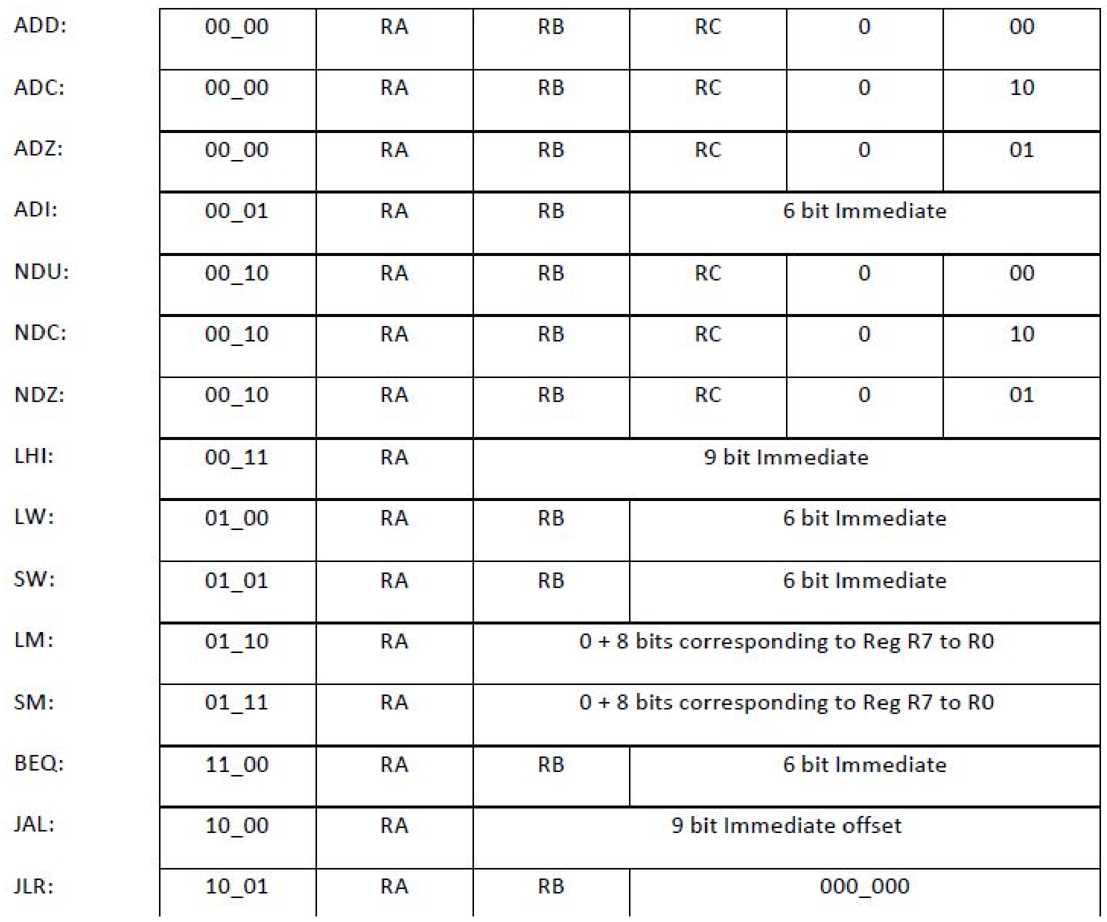
I Type Instruction format

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

J Type Instruction format

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

Figure 2.1: The instruction set



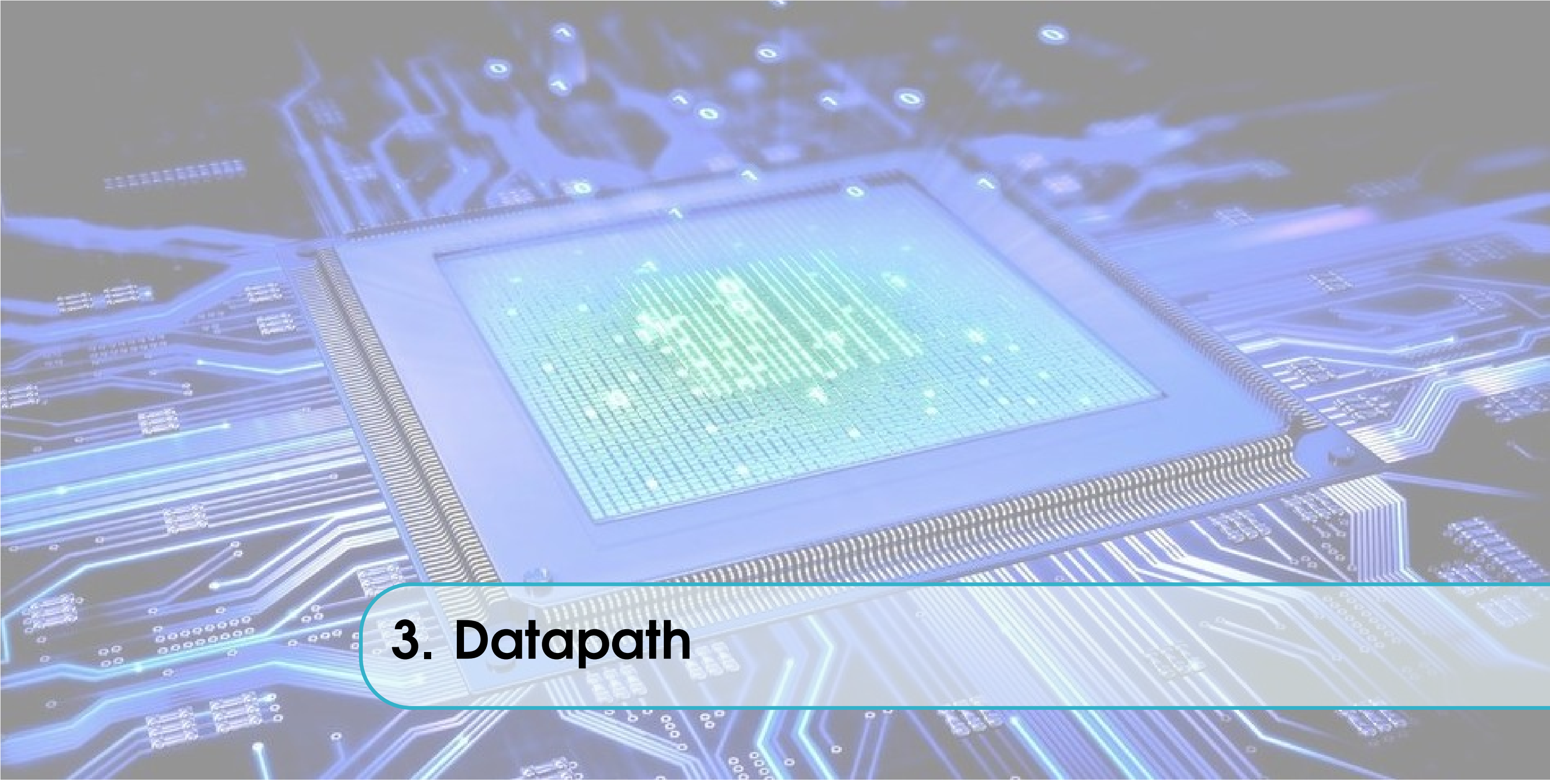
ADD:	00_00	RA	RB	RC	0	00
ADC:	00_00	RA	RB	RC	0	10
ADZ:	00_00	RA	RB	RC	0	01
ADI:	00_01	RA	RB	6 bit Immediate		
NDU:	00_10	RA	RB	RC	0	00
NDC:	00_10	RA	RB	RC	0	10
NDZ:	00_10	RA	RB	RC	0	01
LHI:	00_11	RA	9 bit Immediate			
LW:	01_00	RA	RB	6 bit Immediate		
SW:	01_01	RA	RB	6 bit Immediate		
LM:	01_10	RA	0 + 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		

Figure 2.2: 16-bit instruction encoding

RA: Register A

RB: Register B

RC: Register C



3. Datapath

3.1 Elements

The datapath consists of the described units:

(1) Register File : The module consists of eight 16-bit clocked registers (R0 to R7). This module has 4 inputs : 3 input addresses (add1, add2, add3) 8bit each and 1 data input (16bit). There are 2 output data sequences, corresponding to the first two input addresses, add1 and add2. Control signal Rf_wen is involved to enable the writing to the registers.

(2) ALU : The logic unit supports two operations : ADD and NAND, to implements different instructions and for incrementing desired variables. The module is combinational in nature, meaning the desired data output would be obtained in the same cycle when the two operands are fed. Two flags CY (carry) and Z (zero) are associated with the ALU, which would be written to the CZ module defined in the datapath. A clocked register, T1, stores the computed ALU result.

(3) Multiplexers : For proper instructor execution, different modules need to be enabled or disabled, along with proper routing of data to these modules. This is achieved using MUXes, having variable no. of control signals bits. These would be described later.

(4) CZ module : This module is used to generate a conditional write_enable signal to the Register File module based on previous CY and Z bits. Thus, this conditional signal would be required to implement ADZ, ADC, NDZ and NDC instruction.

(5) Registers : Temporary registers A and B are used to store output results of the register file. Register T1 is similarly used to store the ALU result.

(6) Memory* : This module is connected to different modules in the datapath. It has a size of 64bytes, controlled by write_enable signal, with multiplexed input data.

(7) Priority Encoder : This module serves an essential purpose in reducing the cycles in LM and SM instructions by rendering the RF addresses corresponding to the set bits in instructions last 8 bits.

(8) Sign Extenders : This module appends zeroes at the beginning of the immediate bits,
Usage: extending Imm6 and Imm9 in immediate addressing

(9) Data Shifters : This module is used to convert a 9 bit value to 16 bit by padding zeroes at the end.

3.2 Design Specifications

Design Specifications to incorporate all corner cases (well to the best of our knowledge :)) are as follows :

- R7 is our PC and we have PC_enable to to update it when needed.
- R7 = Ra + Rb is a branch instruction and in our case in any instruction the PC updation part occurs in the last states of the operation so it wouldn't affect the corner case.
- There is a slave register CZ_reg which takes care of this and is written to only in the ADD family of operations as mentioned in the project specifications. As we mentioned in the lab, we have a temp_A register to store value of A before address increment which takes care of this case.
- We are using a priority encoder to optimize the states and cycles taken by the LM, SM instructions. Now the cycles are equal to the number of bits set in the imm8 given in the instruction.
- We have extended the sign of the bits of the immediate field to ensure compatibility.

3.3 Control Signals Interpretation

The different control signals to MUX's are mentioned here:
These multiplexers have different combination of no. of inputs and control signals, hence they are defined differently in the datapath.

Mux1_ALU_B (3bit)	
Mux Signal	Input
000	16'd0
001	16'd01
010	B
011	imm6
100	addIncrement

Mux2_ALU_A (3bit)	
Mux Signal	Input
000	16'd0
001	16'd01
010	shift7
011	imm6
100	imm9
101	A
110	tempA

Mux3_RF_wen (2bit)	
Mux Signal	Input
00	1'b0
01	1'b1
10	CZ
11	addrIncrement_out

Mux_4_RF_wadd (3bit)	
Mux Signal	Input
000	regA
001	regC
010	lm_sm_addr
011	3'b111
100	regB

Mux5_RF_read2 (2bit)	
Mux Signal	Input
00	regB
01	lm_sm_addr
10	3'b111

Mux6_RF_dataIn (1bit)	
Mux Signal	Input
0	memout
1	T1

Mux8_memwrite (2bit)	
Mux Signal	Input
00	1'b0
01	1'b1
010	mux7
11	imm6

Mux9_memDataIn (1bit)	
Mux Signal	Input
0	A
1	B

3.4 Datapath

High Resolution SVG version of the datapath diagram can be found at :
`https://www.ee.iitb.ac.in/student/~meetshah1995/files/datapath.svg`

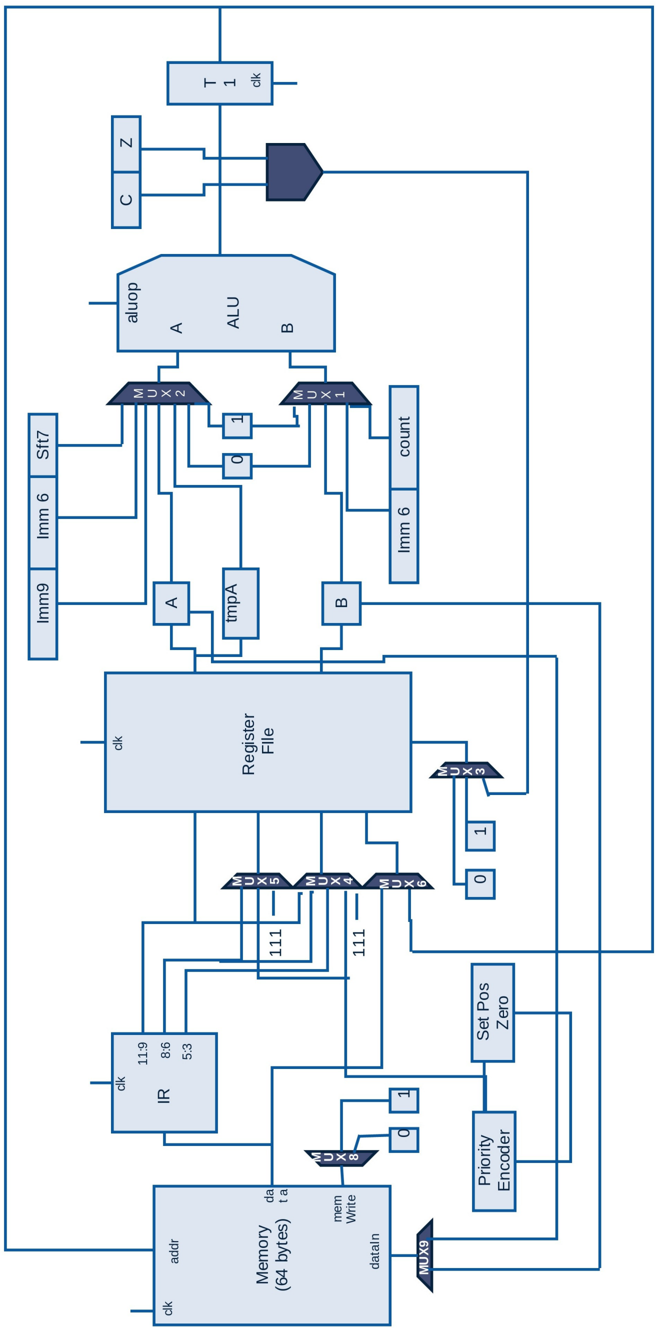
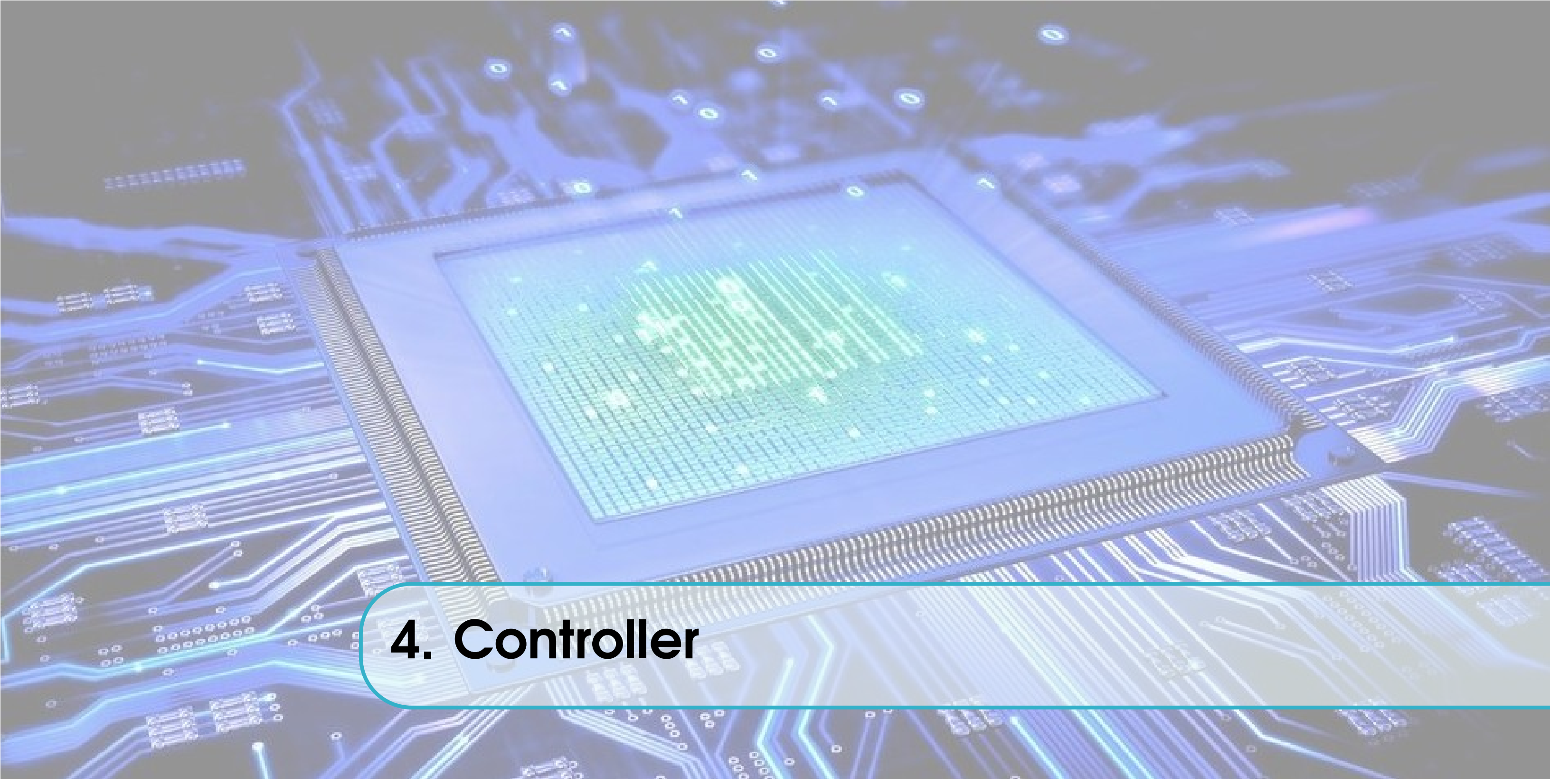


Figure 3.1: Proposed Datapath



4. Controller

4.1 State Description

Writeup for different states involved for each instruction(These are shown sequentially going from left to right):

1. ADD

0	1	2	3	4
Instruction gets stored in IR	Values of reg read into A and B for Add	Added answer written into Rc	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

2. ADC

0	1	5	7	3	4
Instruction gets stored in IR	Values of reg read into A and B for Add	Logical combination of the carry and zero bits is stored in CZ_reg for Add	Depending upon logical combination of the carry and zero bits added value is written in Rc	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

3. ADZ

0	1	5	7	3	4
Instruction gets stored in IR	Values of reg read into A and B for Add	Logical combination of the carry and zero bits is stored in CZ_reg for Add	Depending upon logical combination of the carry and zero bits added value is written in Rc	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

4. NDU

0	8	9	3	4
Instruction gets stored in IR	Values of reg read into A and B for NAND	NANDed answer written into Rc	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

5. NDC

0	8	10	7	3	4
Instruction gets stored in IR	Values of reg read into A and B for NAND	Logical combination of the carry and zero bits is stored in CZ_reg for NAND	Depending upon logical combination of the carry and zero bits added value is written in Rc	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

6. NDZ

0	8	10	7	3	4
Instruction gets stored in IR	Values of reg read into A and B for NAND	Logical combination of the carry and zero bits is stored in CZ_reg for NAND	Depending upon logical combination of the carry and zero bits added value is written in Rc	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

7. ADI

0	11	12	3	4
Instruction gets stored in IR	Pass Imm6 to input B of ALU to be added in the next state	Added answer written into Rb	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

8. LHI

0	20	21	3	4
Instruction gets stored in IR	Pass Shifted 7 Value and 0 to ALU for loading	Loaded Value Returned to Ra	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

9. SW

0	15	16	17	3	4
Instruction gets stored in IR	Imm 6 and reg B is passed to ALU	Resultant memory address, is passed to memory block	Desired data is written in the memory	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

10. LW

0	15	18	3	4
Instruction gets stored in IR	Imm 6 and reg B is passed to ALU	Data output from memory is stored in the Register File	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

11. BEQ

0	22	23	24 (if compare high)	3 (else)	4
Instruction gets stored in IR	reg A and reg B values are loaded	Same as previous state to ensure proper data storage in T1 register	Imm 6 and reg B is passed to ALU	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

12. JAL

0	3	25	26	27	4
Instruction gets stored in IR	Read PC into B and +1 passed to ALU	Store PC+1 in reg A	add imm6 with PC, caring not to set the carry	Store the obtained result in PC	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

13. JLR

0	3	25	28	29	4
Instruction gets stored in IR	Read PC into B and +1 passed to ALU	Store PC+1 in reg A	Pass reg B and 0 to ALU	Store obtained result in PC	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

14. LM

0	30	31	32	33 (Decide Repeat)	3	4
Instruction gets stored in IR	Set the Mux4 to take address from the priority encoder and enable the Priority encoder	Enable data writing for RF	Disable write for Atmp	Values written into RF, disable wRF, T1write and increment counter	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

15. SM

0	34	35	36	37 (Decide Repeat)	3	4
Instruction gets stored in IR	Set the Mux5 to take input from the priority encoder address and enable the Priority encoder	Enable the memory_write signal in the controller	Wait for the memOut to Load into the corresponding memory address	Disable the memory_write, disable the priority encoder and increment the counter	Read PC into B and +1 passed to ALU	PC+1 stored in T1; ready to write that in PC (will be updated automatically in next cycle)

4.2 Finite State Machine

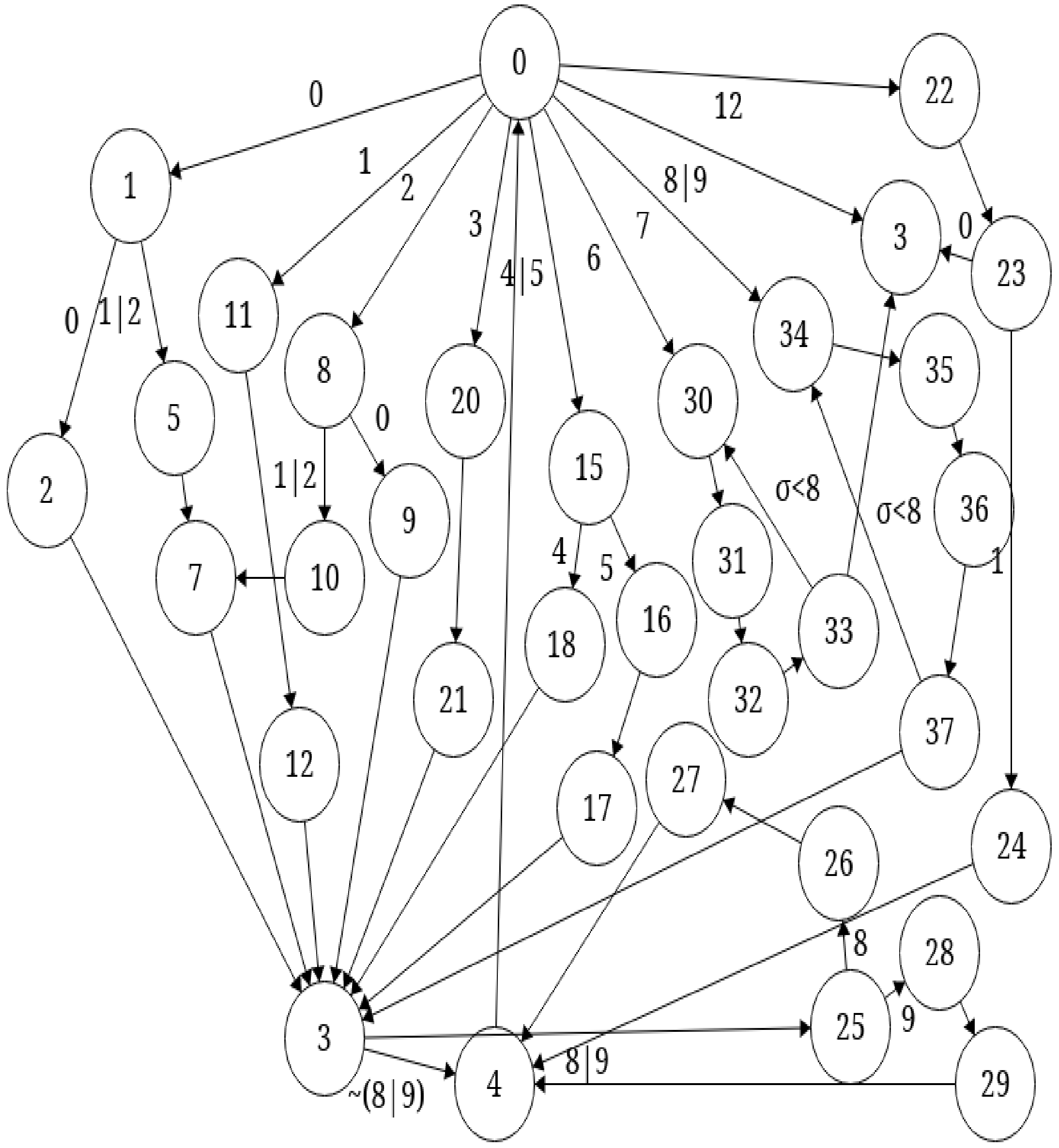
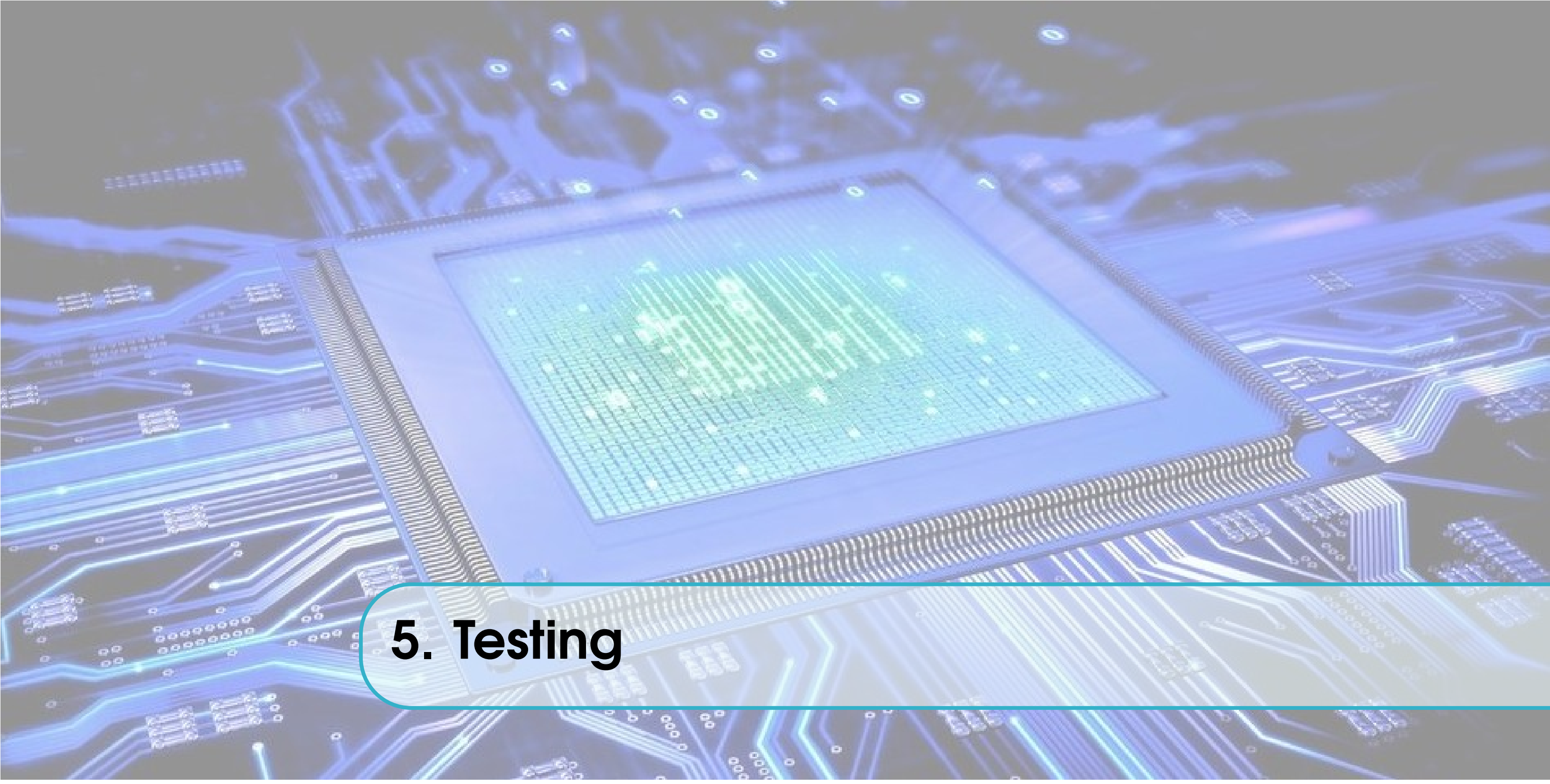


Figure 4.1: The State Transition flow



5. Testing

5.1 Test Cases

Default register values :
reg0 16'b0000000000000011
reg1 16'b1111111111111111
reg3 16'b0000000000000001

TEST 1:

```
mem[0] <= 16'b0000001011110000; // ADD R1 R3 R6
mem[1] <= 16'b0010001011101000; // NDU R1 R3 R5
mem[2] <= 16'b0000001011100010; // ADC R1 R3 R4
mem[3] <= 16'b0010001011010001; // NDZ R1 R3 R2
mem[4] <= 16'b0001001011110000; // ADI R3 R1 6'b110000
```

TEST 2:

```
mem[0] <= 16'b1100001011000010; // BEQ R1 R3 6'b000010
mem[1] <= 16'b0011001101101001; // LHI R1 9'b101101001
mem[2] <= 16'b0101001011000011; // SW R1 R3 6'b000011
mem[3] <= 16'b0100010011000011; // LW R2 R3 6'b000011
```

TEST 3:

```
mem[0] <= 16'b1000000000000011; // JAL R0 9'b000000011
```

mem[1] <= 16'b0000001011100010; // ADC R1 R3 R4
mem[2] <= 16'b0010001011101000; // NDU R1 R3 R5
mem[3] <= 16'b0010001011010001; // NDZ R1 R3 R2
mem[4] <= 16'b0001001011110000; // ADI R3 R1 6'b110000

TEST 4:
mem[0] <= 16'b1001000010000000; // JLR R0 R2
mem[1] <= 16'b0000001011100010; // ADC R1 R3 R4
mem[2] <= 16'b0010001011101000; // NDU R1 R3 R5

TEST 5:
mem[0] <= 16'b0110000001100100; // LM R0 9'b001100100
mem[1] <= 16'b0000001011100000; // ADD R1 R3 R4
mem[2] <= 16'd1;
mem[3] <= 16'd2;
mem[4] <= 16'd3;
mem[5] <= 16'd4;
mem[6] <= 16'd5;
mem[7] <= 16'd6;
mem[8] <= 16'd7;
mem[9] <= 16'd8;

TEST 6:
mem[0] <= 16'b0111000001100100; // SM R0 9'b001100100
mem[1] <= 16'b0000001011100000; // ADD R1 R3 R4