



IITB-RISC

Project 1

Multi-Cycle Implementation

OV Shashank; Avineil Jain; Pratik Brahma; Yogesh Mahajan
GROUP NUMBER : 16

CONTENTS

Microprocessor Design	2
State Elaboration	2
State Flow Diagram	3
Datapath Design.....	5
Instruction Set.....	6
Instructions that affect Flag Settings	6
Instruction Set and Addressing Modes	6
Instruction Encoding	6
Instruction Timing Reference.....	7
Custom Instructions	7
VHDL Codes.....	8
Components.....	8
Microprocessor Blocks	8
Testing.....	8
Assembler	8
Bootloader-Software	8
Quartus Project	9
Bootloader (Edits to support the bootloader)	9
Miscellaneous	10
Team Members	10
Problems Faced	10
Interesting Aspects	10
Testing Procedure	10

MICROPROCESSOR DESIGN

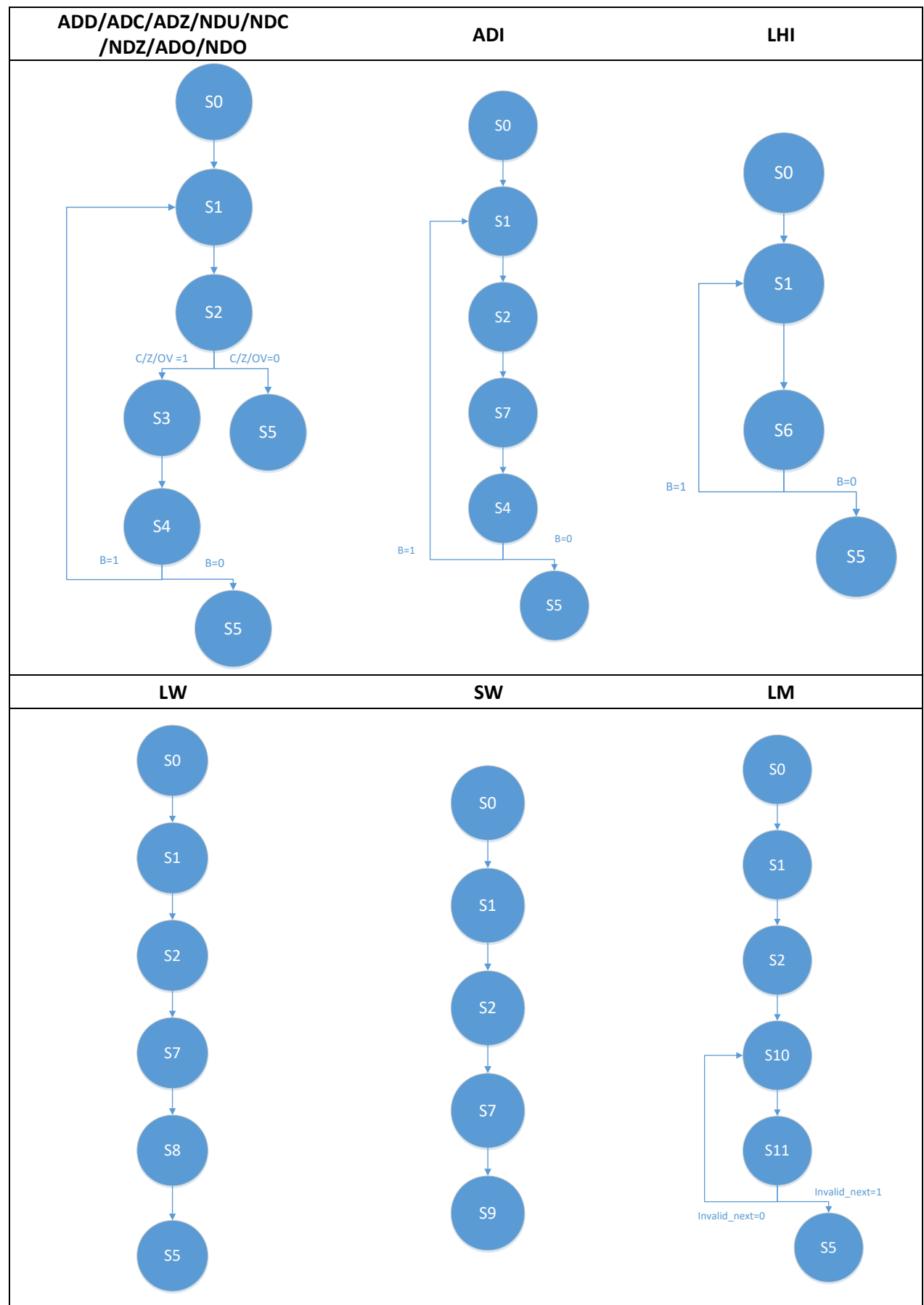
(Does not include the bootload states)

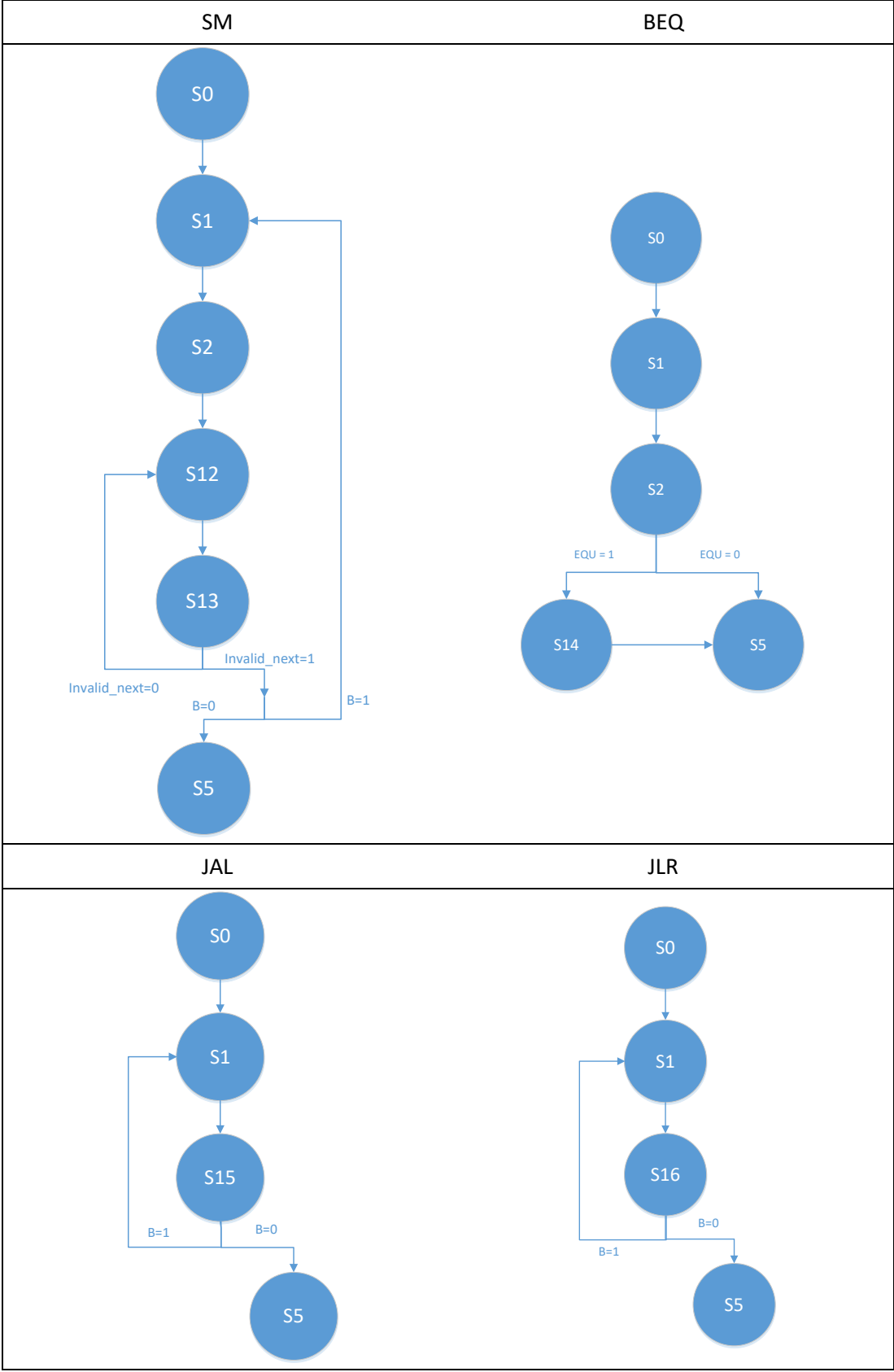
STATE ELABORATION

S1	$R7 \rightarrow \text{MEM (A)}$ $\text{MEM (D)} \rightarrow \text{IR}$ $R7 \rightarrow \text{ALU}$ $+1 \rightarrow \text{ALU}$ $\text{ALU} \rightarrow \text{PC}$
S2	$I_{6-8} \rightarrow A1_{\text{RF}}$ $I_{9-11} \rightarrow A2_{\text{RF}}$ $D1 \rightarrow E1$ $D2 \rightarrow E2, T1$ $I_{0-7} \rightarrow \text{PE}_{\text{INPUT}}$
S3	$E1 \rightarrow \text{ALU}$ $E2 \rightarrow \text{ALU}$ $\text{ALU} \rightarrow T1$
S4	$I_{3-5} / I_{6-8} \rightarrow A3_{\text{RF}}$ $T1 \rightarrow D3_{\text{RF}}$
S5	$\text{PC} \rightarrow D3_{\text{RF}}$ $"111" \rightarrow A3_{\text{RF}}$ $T2 \rightarrow \text{ALU}$ $0 \rightarrow \text{ALU}$
S6	$I_{0-8} \rightarrow \text{SE}_{9-16} \rightarrow \text{LS}_7$ $\text{LS}_7 \rightarrow D3_{\text{RF}}$ $I_{9-11} \rightarrow A3_{\text{RF}}$
S7	$E1 \rightarrow \text{ALU}$ $I_{0-5} \rightarrow \text{SE}_{6-16} \rightarrow \text{ALU}$ $\text{ALU} \rightarrow T1, \text{MEM(A)}$
S8	$\text{MEM (DO)} \rightarrow T2, D3_{\text{RF}}$ $I_{9-11} \rightarrow A3_{\text{RF}}$

S9	$D2 \rightarrow \text{MEM}_{10}(\text{DI})$ $\text{PC} \rightarrow D3_{\text{RF}}$ $"111" \rightarrow A3_{\text{RF}}$
S10	$\text{do } \{$ $\text{MEM}_{\text{DAT}}(\text{DO}) \rightarrow T2\}$
S11	$T2 \rightarrow D3_{\text{RF}}$ $\text{PE}_{\text{OUTPUT}} \rightarrow A3_{\text{RF}}$ $T1 \rightarrow \text{ALU}$ $+1 \rightarrow \text{ALU}$ $\text{ALU} \rightarrow T1, \text{MEM(DI)}$ $\text{while } (! \text{ invalid_next});$
S12	$\text{PE}_{\text{OUTPUT}} \rightarrow A2_{\text{RF}}$ $T1 \rightarrow \text{MEM(A)}$
S13	$T1 \rightarrow \text{ALU}$ $+1 \rightarrow \text{ALU}$ $\text{ALU} \rightarrow T1$ $\text{while } (! \text{ invalid_next});$
S14	$R7 \rightarrow \text{ALU}$ $I_{0-5} \rightarrow \text{SE}_{6-16} \rightarrow \text{ALU}$ $\text{ALU} \rightarrow \text{PC}$
S15	$\text{PC} \rightarrow D3_{\text{RF}}$ $I_{9-11} \rightarrow A3_{\text{RF}}$ $R7 \rightarrow \text{ALU}$ $I_{0-8} \rightarrow \text{SE}_{9-16} \rightarrow \text{ALU}$ $\text{ALU} \rightarrow \text{PC}$
S16	$D1_{\text{RF}} \rightarrow \text{PC}$ $I_{9-11} \rightarrow A3_{\text{RF}}$ $\text{PC} \rightarrow D3_{\text{RF}}$

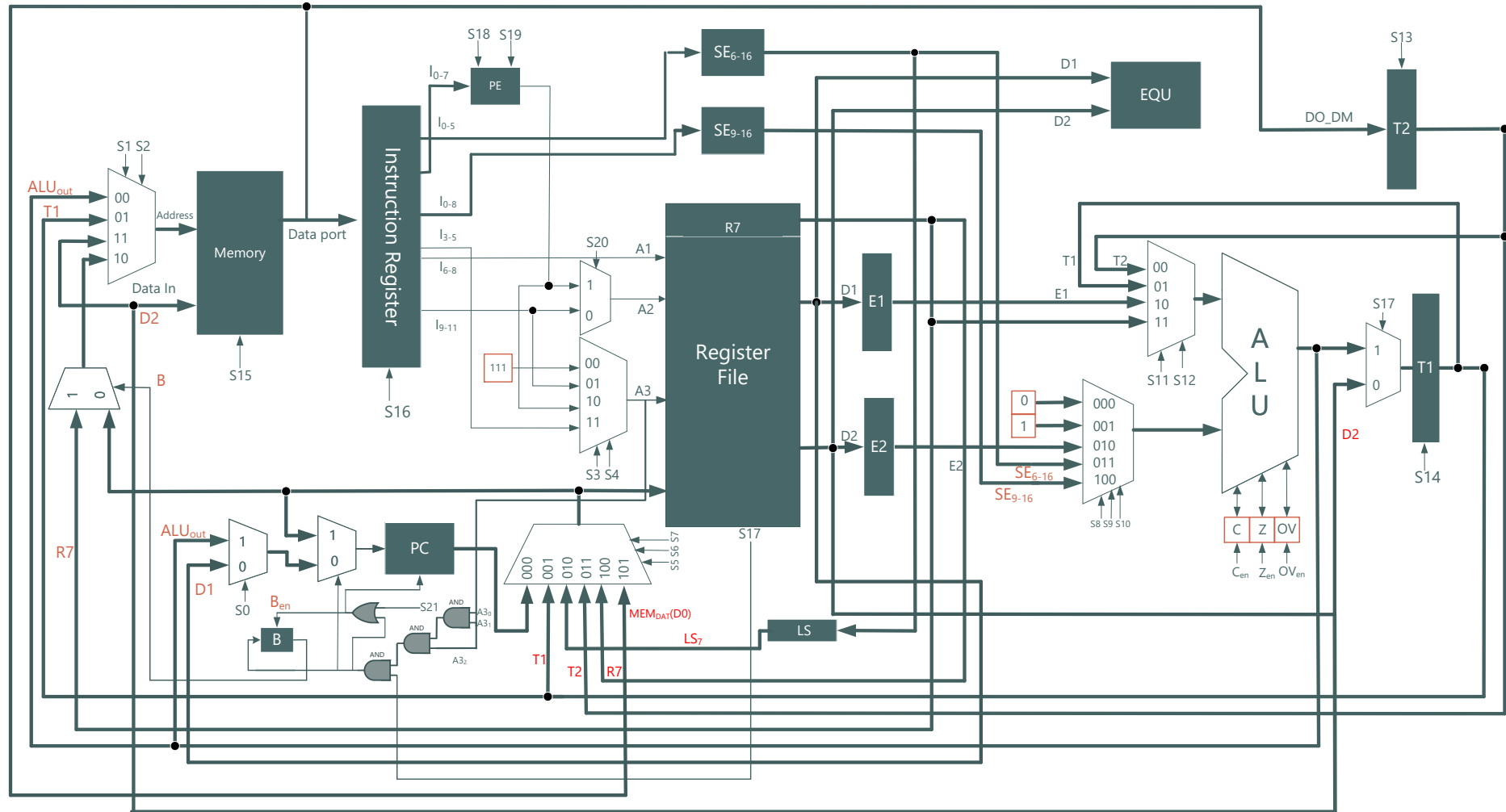
STATE FLOW DIAGRAM





DATAPATH DESIGN

(Does not reflect the presence of the bootloader block)



INSTRUCTION SET

IITB - RISC

INSTRUCTION SET

INSTRUCTIONS THAT AFFECT FLAG SETTINGS

Instruction	Flags			Instruction	Flags		
	C	Z	OV		C	Z	OV
ADD	X	X	X	NDU		X	
ADC	X	X	X	NDC		X	
ADZ	X	X	X	NDZ		X	
ADO	X	X	X	NDO		X	
ADI	X	X	X	LW		X	

INSTRUCTION SET AND ADDRESSING MODES

R _n	Registers R6-R0 of the Register Bank
data 6	Signed 6-bit constant included in instruction
data 9	Signed 9-bit constant included in instruction
rel 6	Signed (two's complement) 6-bit offset byte. Used by BEQ. Range is -32 to +31 short words relative to the present instruction
rel 9	Signed (two's complement) 9-bit offset byte. Used by JAL. Range is -256 to +255 short words relative to the present instruction
reg	8-bits representing registers R7-R0. Used by LM and SM
R ₇	Program Counter
R	Registers R7-R0 of the Register Bank

INSTRUCTION ENCODING

ADD:	00_00	RA	RB	RC	0	00
ADC:	00_00	RA	RB	RC	0	10
ADZ:	00_00	RA	RB	RC	0	01
ADO:	00_00	RA	RB	RC	0	11
ADI:	00_01	RA	RB	data 6		
NDU:	00_10	RA	RB	RC	0	00
NDC:	00_10	RA	RB	RC	0	10
NDZ:	00_10	RA	RB	RC	0	01
NDO:	00_10	RA	RB	RC	0	11
LHI:	00_11	RA	data 9			
LW:	01_00	RA	RB	rel 6		
SW:	01_01	RA	RB	rel 6		
LM:	01_10	RA	reg			
SM:	01_11	RA	reg			
LLI:	10_11	RA	data 9			
BEQ:	11_00	RA	RB	rel 6		
JAL:	10_00	RA	rel 9			
JLR:	10_01	RA	RB	000_000		

INSTRUCTION SET

IITB - RISC

INSTRUCTION TIMING REFERENCE

Mnemonic		Flags			Oscillator Period
		C	Z	OV	
ADD	R _n , R, R	X	X	X	5
ADD	R ₇ , R, R	X	X	X	4
ADC	R _n , R, R	0	X	X	5
ADC	R ₇ , R, R	0	X	X	4
ADC	R _n , R, R	1	X	X	3
ADZ	R _n , R, R	X	0	X	5
ADZ	R ₇ , R, R	X	0	X	4
ADZ	R _n , R, R	X	1	X	3
ADO	R _n , R, R	X	X	0	5
ADO	R ₇ , R, R	X	X	0	4
ADO	R _n , R, R	X	X	1	3
NDU	R _n , R, R	X	X	X	5
NDU	R _n , R, R	X	X	X	4
NDC	R _n , R, R	0	X	X	5
NDC	R ₇ , R, R	0	X	X	4
NDC	R _n , R, R	1	X	X	3
NDZ	R _n , R, R	X	0	X	5
NDZ	R ₇ , R, R	X	0	X	4
NDZ	R _n , R, R	X	1	X	3
NDO	R _n , R, R	X	X	0	5
NDO	R ₇ , R, R	X	X	0	4
NDO	R _n , R, R	X	X	1	3

Mnemonic		Flags/ Condition	Oscillator Period
ADI	R _n , R, data6	X	5
ADI	R ₇ , R, data6	X	4
LHI	R _n , data9	X	3
LHI	R ₇ , data9	X	2
LLI	R _n , data9	X	3
LLI	R ₇ , data9	X	2
LW	R, R, rel6	X	5
SW	R, R, rel6	X	4
LM	R, reg	X	3+2*reg_count
SM	R, reg	X	3+2*reg_count
JAL	R _n , rel9	X	3
JAL	R ₇ , rel9	X	2
JLR	R _n , R	X	3
JLR	R ₇ , R	X	2
BEQ		Not Equal	3
BEQ		Equal	4

CUSTOM INSTRUCTIONS

- ADO: ADD if the overflow flag is set. The rest of the instruction format is the same as the other conditional execution instructions
- NDO: NAND if the overflow flag is set. The rest of the instruction format is the same as the other conditional execution instructions
- LLI: Load lower immediate. Load the immediate 9 bits into the register as mentioned in the instruction without sign extension by padding zeros in the upper 7 bits.

NOTE: The microprocessor is supposed to be a signed microprocessor and hence the overflow bit was introduced. It may be considered as the signed equivalent of the carry bit. The carry bit here still represents the unsigned version and hence the two must be used carefully.

CODES

IITB - RISC

VHDL CODES

(All the codes have been hyperlinked here for easy viewing; Notepad++ or gedit is recommended)

COMPONENTS

- [ALU](#)
- [Priority Encoder](#)
- [Load/Store Multiple Logic Block](#)
- [Registers](#)
- [Memory](#) (Generated using Altera's MegaWizard – Takes no HEX Input File – Use Bootloader)
- [Register File](#)
- [Sign Extender](#)

MICROPROCESSOR BLOCKS

- [Data Path](#): This consists of the entire data path along with all the transfers and the predicates corresponding to an RTL layout of the microprocessor. All the T and S signals are pretty accurately detailed as comments at the beginning of the architecture.
- [Control Path](#): This consists of the controller Moore FSM. It has been decomposed into three processes; The first one describes the flip flops which control the states, the second one describes the next state logic and finally the third process controls the output logic based on the present state. (The order in the code might not exactly follow this order). All the T signals have been accompanied by the expected result they are to cause in the data path.
- [IITB_RISC](#): The top-level entity that combines the data path and the controller FSM. The register 0 has been shown as an output so that the processes of the microprocessor can be displayed outside in hardware.
- [Bootloader](#): The bootloader block. More information can be found [here](#).

TESTING

- [Test_final](#): This is a test bench which can be used to simulate the entire microprocessor in Altera's modelsim. It basically functions to provide the clock and reset signals to the microprocessor.

Testing results have been stored as [images](#) and are also available along with this document.

ASSEMBLER

The code [assembler.py](#) was developed as a combined effort of Kalpesh and Shashank. It basically works by taking as its input a text file containing code written in an assembly like language, and creates an INTEL HEX file which can be directly provided to Quartus which uses it to preload memory. A [sample hex](#) file produced by the code is provided along with this document. Also, a [sample text](#) input file has also been included. More information on the syntax can be found [here](#).

BOOTLOADER-SOFTWARE

The code [serial_boot.py](#) can be used in conjunction with the bootloader hardware as described in the [README](#) to program the microprocessor memory to contain any set of instructions. More information on its utilisation can be found [here](#).

CODES

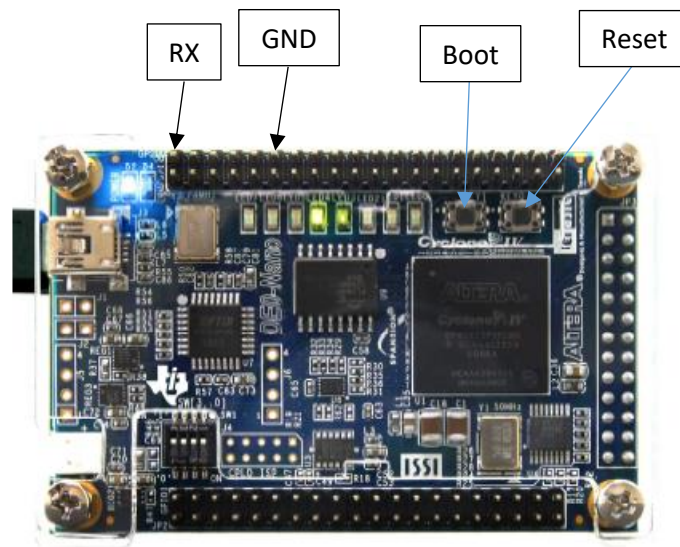
IITB - RISC

QUARTUS PROJECT

The [quartus project folder](#) used for testing is also included along with this submission. This can be used for quick compilation and viewing in Quartus. Necessary settings have been implemented to allow for timing analysis and RTL and gate level simulation. The bootloader has been integrated into the design.

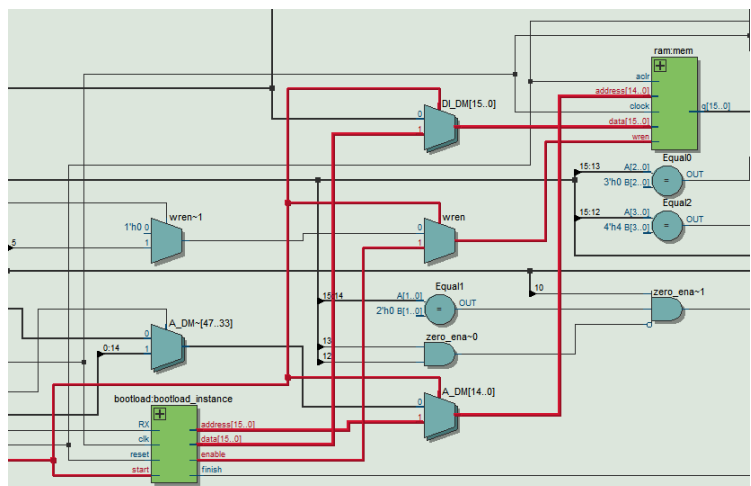
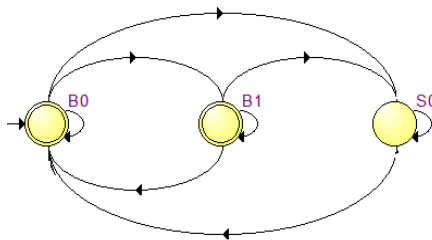
Pin Mapping:

- Last 8 Bits of the Register R0 have been mapped to the LEDs. It can be used to see visible results.
- KEY [0] is the RESET Button. It is active LOW.
- KEY [1] is the BOOTLOAD Button. It is too active LOW.
- GPIO_00 is the UART RX Pin.



To send into bootloader mode, while keeping the Boot Key pressed, press and release the Reset Button and then release the Boot Key. The microprocessor enters the boot mode.

BOOTLOADER (EDITS TO SUPPORT THE BOOTLOADER)



CODES

IITB - RISC

MISCELLANEOUS

TEAM MEMBERS

1. Avineil Jain
2. Pratik Brahma
3. Yogesh Mahajan
4. OV Shashank

PROBLEMS FACED

- Minor Bugs in code leading to improper execution of instructions
- Dealing with delays due to input registers present in the Altera MegaWizard Generated RAM
- Behavioural to Synthesisable: Only change that was needed was to replace the memory. Everything else had been taken care of from the beginning itself.
- As such did not find anything challenging

INTERESTING ASPECTS

- Having the microprocessor run on the FPGA
- Running the Bootloader successfully
- Creating this document...

TESTING PROCEDURE

- GHDL Simulation
- Gate Level Simulation of all instructions
- On FPGA: Ran a code that started a counter on R0 with variable delay times. The delay times could be measured and were pretty accurate.