IIT BOMBAY RISC MICROPROCESSORS COURSE PROJECT PROPOSAL

November 8, 2019

Vedant Satav 170070012
Rohan Bansal 170070058
Anwesh Mohanty 170070009
Jaswant Singh 170070032

November 8, 2019

Contents

0.1 Introduction

IITB- RISC is a 16-bit 8-register computer system. It is implemented using VHDL language. The microprocessor involves the use of instructions that are run to perform certain tasks. Instructions are classified into three main categories:

1. R-type
2. I-type
3. J-type

The processor has a total of 6 pipeline stages:

1. Instruction Fetch: In this stage, instruction pointer accesses the memory to fetch the instruction which is stored in the IR register. IP value is also updated depending on various conditions.
2. Instruction Decode: Instruction is broken up into its various components
3. J-type

0.2 IF Stage

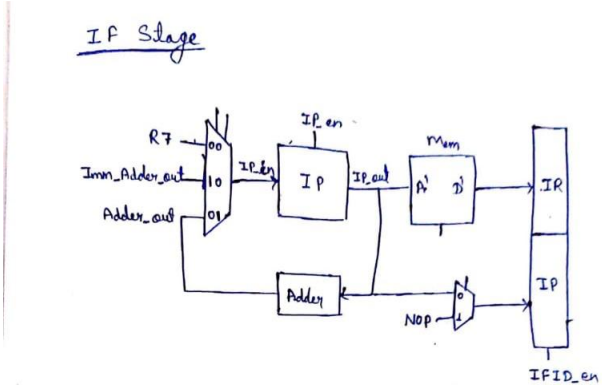

Figure 1: Instruction Fetch Stage

In this state we have following components-

0.2.1 IP

Refers to the 16-bit address in the memory where the current instruction is stored. It accesses the memory via its address1 pins.

0.2.2 Adder

Its a 16 bit adder, used to update the instruction pointer by a value of 1.

0.2.3 Control Store 1

This block manages all the control pins when a given instruction arrives at this stage. This block takes in the first 4-bits (opcode) instruction words of the previous two instructions and as well as the current instruction. Depending on those bits it decides when to send forward the current instruction or a "no operation" signal (stalling). It also controls the various enable pins as well as all the control signals of multiplexers used in this stage. It also contains a reset pin which resets all control pins before starting any operation.

0.3 ID Stage

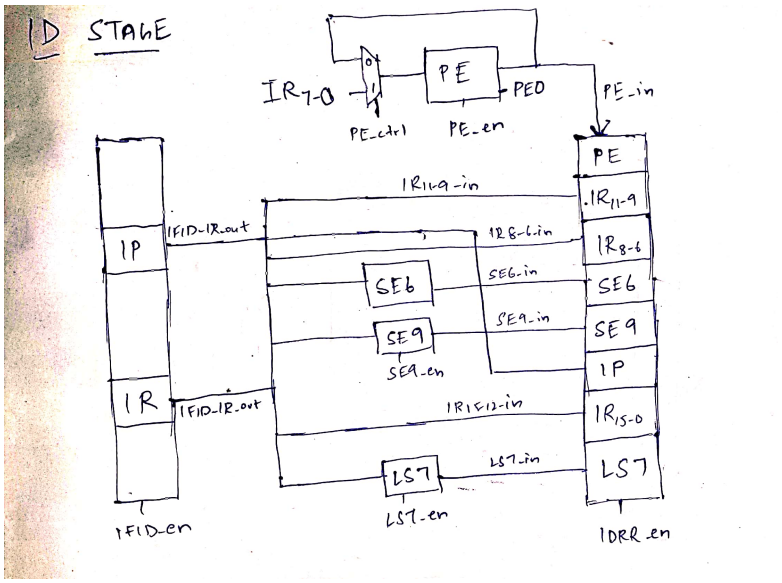

Figure 2: Instruction Decode Stage

0.3.1 SE6 & SE9

SE stands for sign extender. In SE6 we extend 6 bit data into 16 bit and in SE9 we extend 9 bit data into 16 bit. The logic is as follows:
If the MSB of the input data is 1 (or 0), it is preceeded by enough number 1's (or 0's) to make the output of 16-bits.

0.4 RR Stage

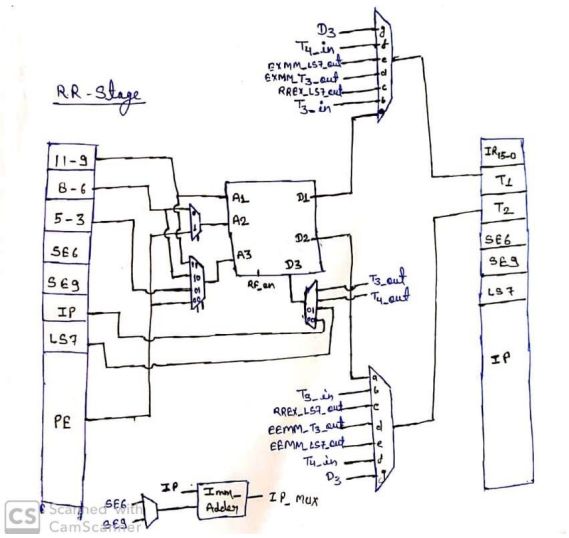

Figure 3: Register Read Stage

In this state we have following components-

0.4.1 Register File

Register file has 5 input pins(RF_{A1}, RF_{A2}, RF_{A3},RF_{D3}, RF_{en}) and 3 output pins(RF_{D1}, RF_{D2}, $R7$). The enable pin is controlled by the control store belonging to the RR stage. Whenever enable pin is high we can write data from pins RF_D3 into Register file in the address given by RF_A3. We can access contents of the Rf when ever we want.

0.4.2 Immediate Adder

This is also a 16 bit adder. We use this adder whenever we need to update IP with immediate value. Its 1^{st} input of is the IP and 2^{nd} input is the output of 2x1 multiplexer which selects between SE6 and SE9 as and when required.

0.4.3 Priority Encoder

The priority encoder block is used for execution of the LM/SM instructions. Input of the PE is chosen between IR(7 downto 0) and the output from the previous instance of the PE by the use of a mux. This is selected by the control store of the RR stage. PE returns the index of the first location in its input where the bit is high.

For ex, if the input were 10010100, the outputs would be the following:
1) The new generated input for the next instance 10010000 which is made by making the first high bit low
2) The index of the first high bit, i.e 010 in this case
3) PE0 = 0
PE0 is a combination circuit which computes a value denoting that all the bits of the input are zero. This marks the end of the RR stage of the LM/SM instruction.

0.4.4 Forwarding Unit

Forwarding have 8 input pins(RF_{A1}, RF_{A2}, IP_{EX}, IP_{MEM}, IP_{WB}, Zen_{EX}, Zen_{MEM}, Zen_{WB}) and 2 output pins ($Forward_{out}1$, $Forward_{out}2$). Here IP_{EX}, IP_{MEM} and IP_{WB} are ip address in EX stage, MEM stage and WB stage. Same for z enable(Zen). $Forward_{out}1$ and $Forward_{out}2$ both are 4 bit outputs. These outputs are connected to corresponding 9x1 muxes(connected to RF_{D1} and RF_{D2}).
FU helps in the task of data forwarding. For example,
Let I_1 be R1 = R2 + R3
Let I_2 be R4 = R1 + R5
Before R1 is written back into the register file, we are trying to access it in the next instruction. To prevent unwanted value, we discard the value of R1 accessed by I_2 and pass the output of the ALU. This is done using multiplexers.

0.4.5 9x1 Mux

9x1 mux is also a part of the forwarding group. It forwards the necessary register to prevent data hazards. The controlling logic of the mux comes from the Forwarding unit.

0.4.6 Control Store 3

This control store takes in the opcode from the current instruction word. Depending on the operation corresponding to the opcode, it determines all the values of the control signals for all the components present in this stage. It sets the pin RF_{en}=1 whenever we write into the register file. We reset the control store to its default values before starting any operation.

0.5 EX Stage

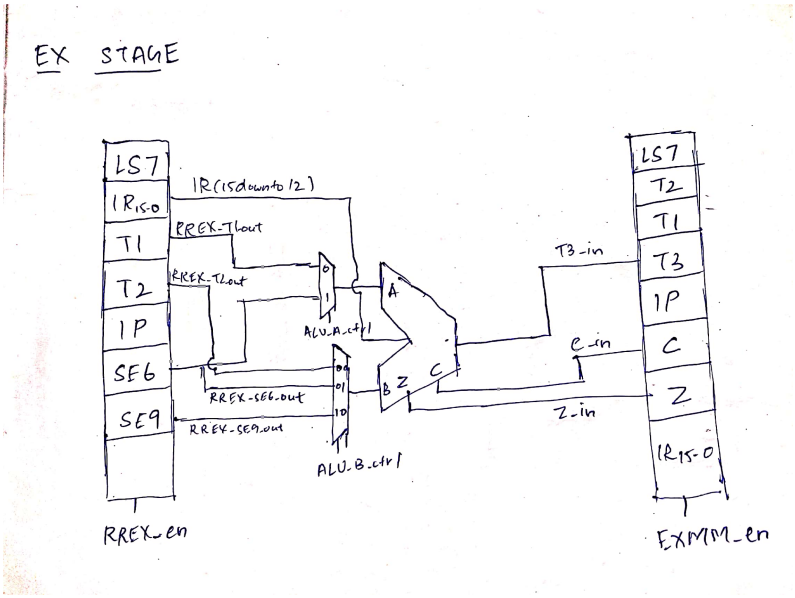

Figure 4: Execution Stage

In this state we have following components-

0.5.1 ALU

ALU have 4 input pins(ALU_A, ALU_B, Opcode, ALU_{en}) and 3 output pins(ALU_C, ALU_{CR}, ALU_Z). Here in ALU we can perform two task ADD and NAND which is decided by opcode. ALU_{CR} is carry-flag and ALU_Z is z-flag. Whenever ALU_{en} is high we modify our z and carry flag.

0.5.2 Control Store 4

This control store selects the inputs to be fed into the ALU by deciding the select signals of the multiplexers connected at the inputs of the ALU depending on the opcode.

0.6 MM Stage

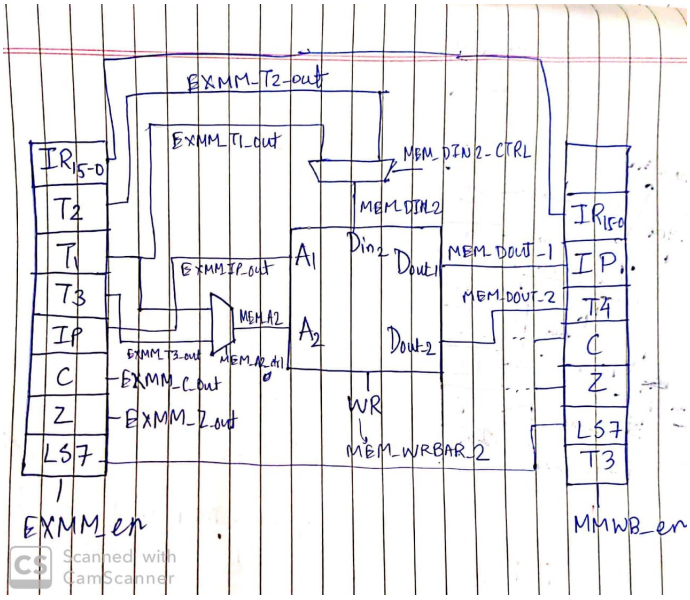

Figure 5: Memory Stage

In this state we have following components-

0.6.1 Memory

Memory has 5 input pins(MEM_{A1}, MEM_{A2}, MEM_{Din}, MEM_{en1}, MEM_{en2}) and 2 output pins(MEM_{D1}, MEM_{D2}).The enable pins are used during the time of memory writing. They are excited by the control stores of their respective stages.
One set of the pins is used for instruction accessing used in the IF stage while the other is used for data accessing during the MEM stage.

0.6.2 Control Store 5

This control store contains the select signals of the multiplexers in this stage as well as the write pin (MEM_WRBAR_2) of the memory as its outputs. Depending on the opcode, we set MEM_WRBAR_2=0 for writing into memory (or 1 if we are not writing into the memory). The control store is reset to its default values before we start any operation by using the reset input.

0.7 WB Stage

Write Back stage carries two a 4x1 multiplexers that serve as the inputs for the RF_A3 and RF_D3 of the register file. The controlling logic comes from the control store of the WB stage.

0.8 Control Store

Depending on the instruction being carried out (derived from the opcode), the control store decides the select pins of the multiplexers to decide what data should be written into which address of the register file. It is reset to default values before the start of any operation.

0.9 State Diagram

0.9.1 For R-type Instructions

R-type instructions follow the following format of encoding:

0.9.2 ADD

ADD instruction instructs that the values in RegisterA and RegisterB be added and stored in RegisterC. More-over, the carry-set and zero-set needs to be updated.

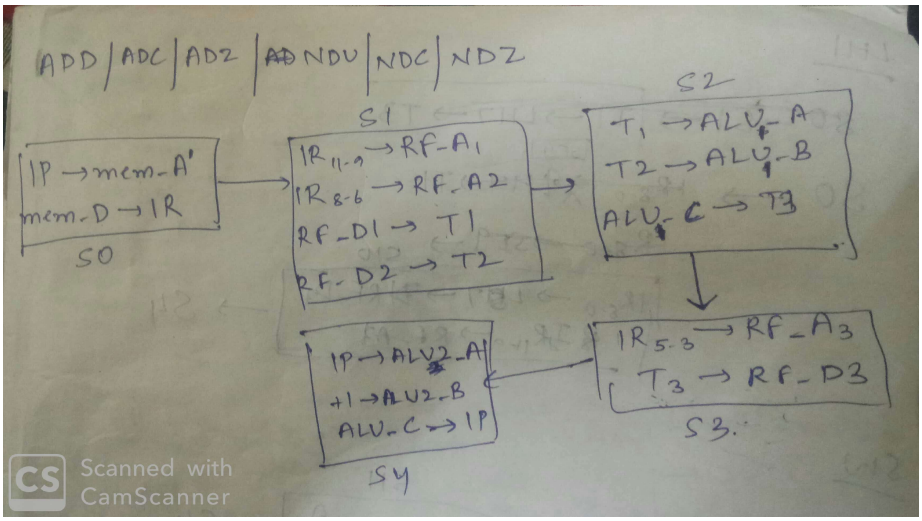

Figure 6: FSM for ADD instruction

0.9.3 ADC

ADC instruction instructs that the values in RegisterA and RegisterB be added and stored in RegisterC, provided the carry-set is high. The carry-set and zero-set needs to be updated.

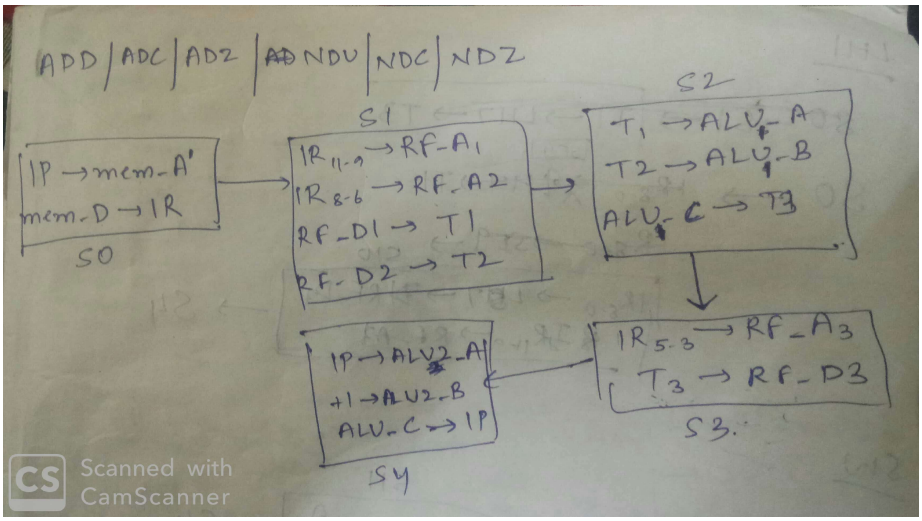

Figure 7: FSM for ADC instruction

0.9.4 ADZ

ADZ instruction instructs that the values in RegisterA and RegisterB be added and stored in RegisterC, provided the zero-set is high. The carry-set and zero-set needs to be updated.

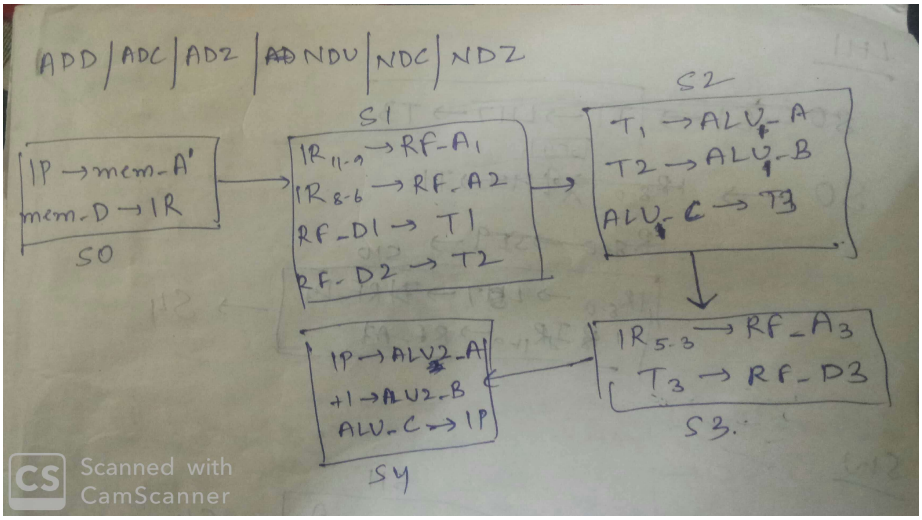

Figure 8: FSM for ADZ instruction

0.9.5 NDU

NDU instruction instructs to store the NAND of the values in RegisterA and RegisterB in RegisterC. The carry-set and zero-set needs to be updated.

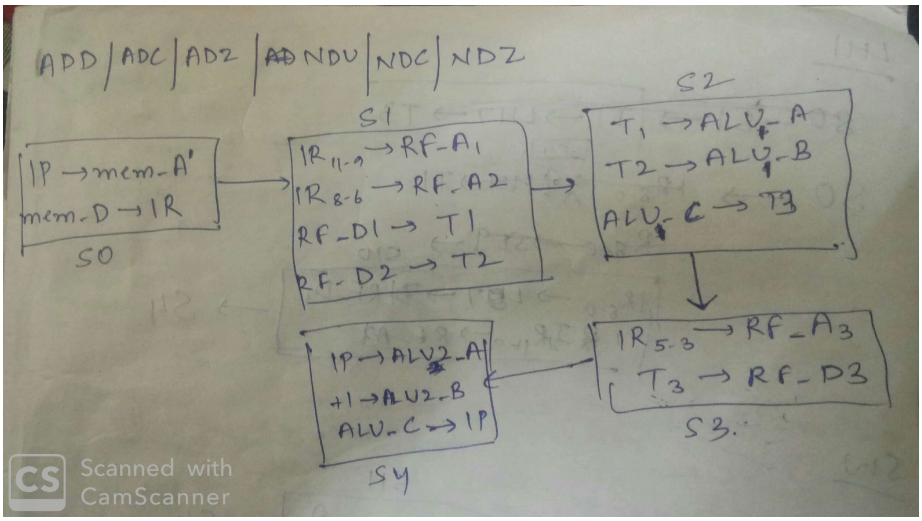

Figure 9: FSM for NDU instruction

0.9.6 NDC

NDC instruction instructs to store the NAND of the values in RegisterA and RegisterB in RegisterC, provided the carry-set is high. The carry-set and zero-set needs to be updated.

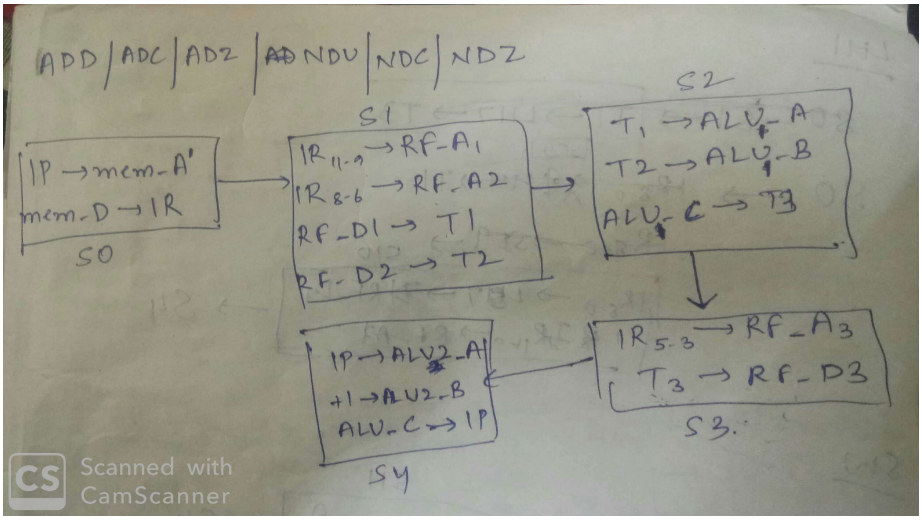

Figure 10: FSM for NDC instruction

0.9.7 NDZ

NDZ instruction instructs to store the NAND of the values in RegisterA and RegisterB in RegisterC, provided the zero-set is high. The carry-set and zero-set

needs to be updated.

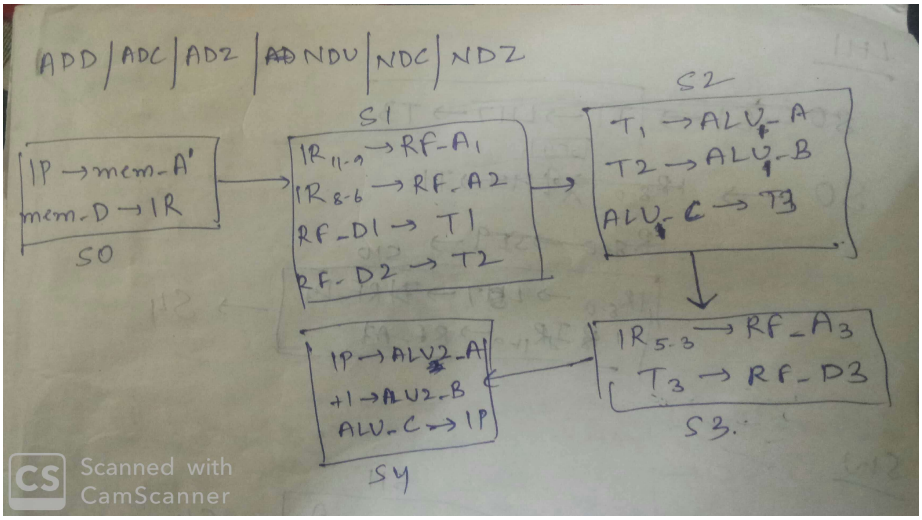

Figure 11: FSM for NDZ instruction

0.9.8 For J-type Instructions

I-type instruction follow the following type of encoding:

0.9.9 LHI

Place 9 bits immediate into most significant 9 bits of register A (RA) and lower 7 bits are assigned to zero.

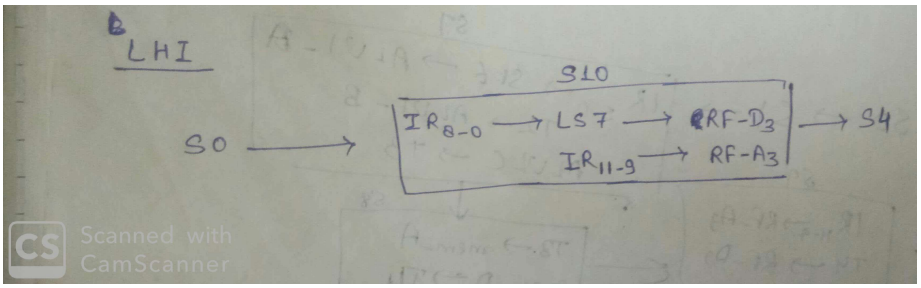

Figure 12: FSM for LHI instruction

0.9.10 LM

The immediate value given in the IR denotes the registers to be loaded from the consecutive memory addresses. If the bit corresponding to R_i is high, the value from memory address 'mem' should be loaded into it. If $R_(i+1)$ is low, skip it. If for $R_(i+2)$ it is high again, the value from memory address 'mem+1' is loaded into it.

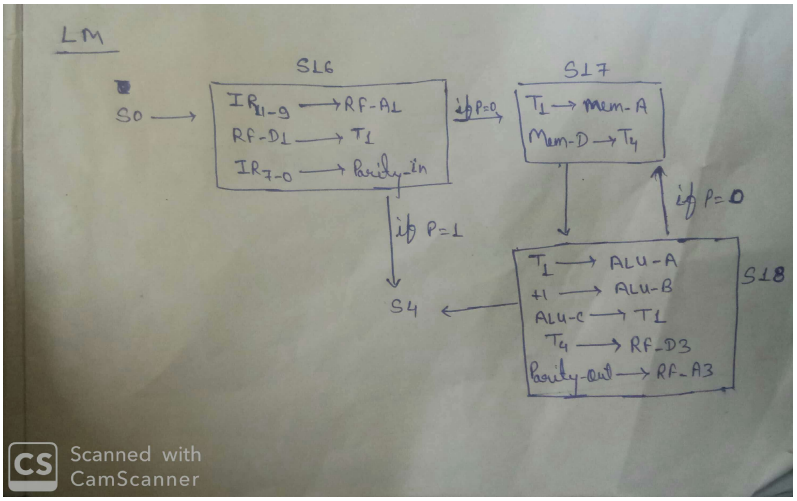

Figure 13: FSM for LM instruction

0.9.11 SM

This instruction is similar to LM.
The immediate value given in the IR denotes the registers to be stored with data from the consecutive memory addresses. If the bit corresponding to R_i is high, its value should be stored in memory location given by 'mem'. If $R_(i+1)$ is low, skip it. If for $R_(i+2)$ it is high again, its value is stored in memory location 'mem+1'.

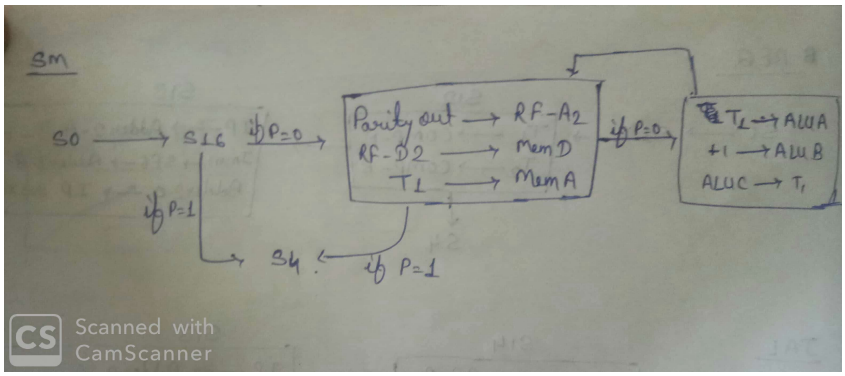

Figure 14: FSM for SM instruction

0.9.12 For I-type Instructions

I-type instructions follow the following type of encoding:

0.9.13 ADI

ADI instructions instructs to add the value in RegisterA and the sign extended value in Imm and to store the result in RegisterB. Both the carry- and zero-set needs to be updated.

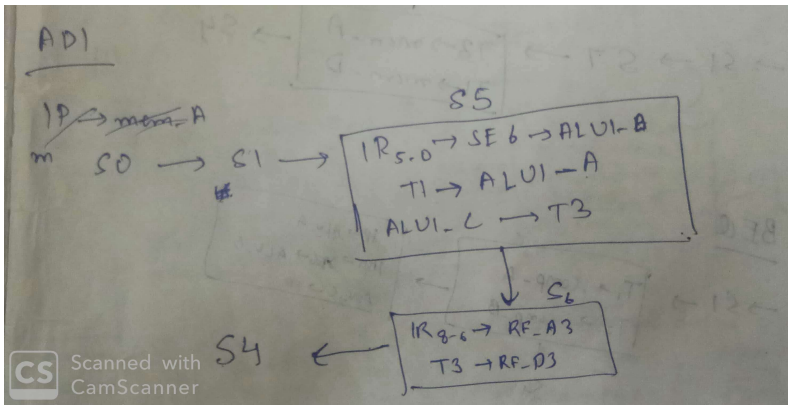

Figure 15: FSMO of ADI instruction

0.9.14 LW

LW loads the value from a memory address and stores it into RegisterA. The memory address is calculated by adding immediate 6-bits and content of RegisterB. Flag z is modified by checking whether the loaded value is zero or not.

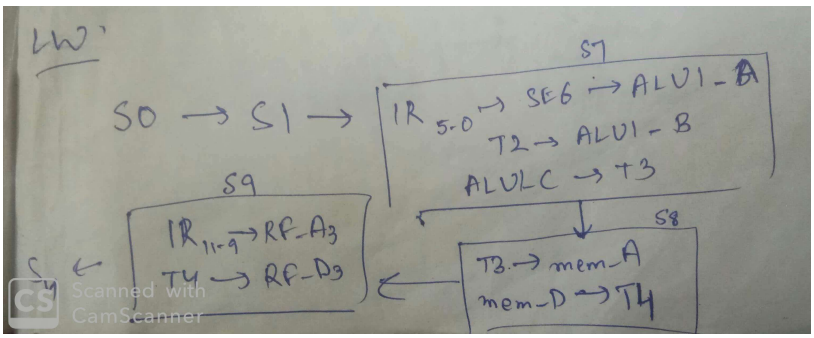

Figure 16: FSMO of LW instruction

0.9.15 SW

SW stores the value in RegisterA into a memory location which is calculated by adding the immediate 6-bits and content in RegisterB.

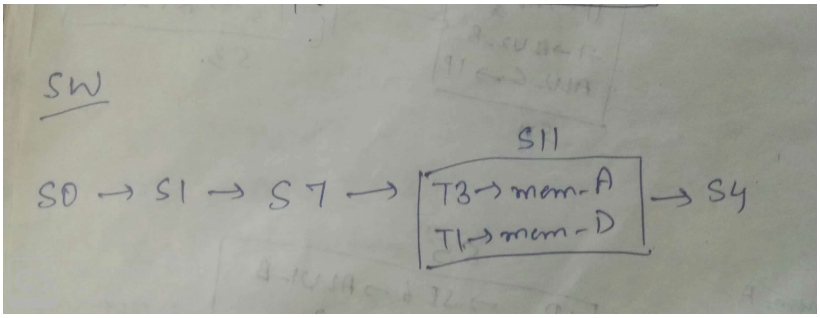

Figure 17: FSMO of SW instruction

0.9.16 BEQ

If content of RegisterA and Register are the same, branch to PC+Imm, where PC is the address of BEQ instruction.

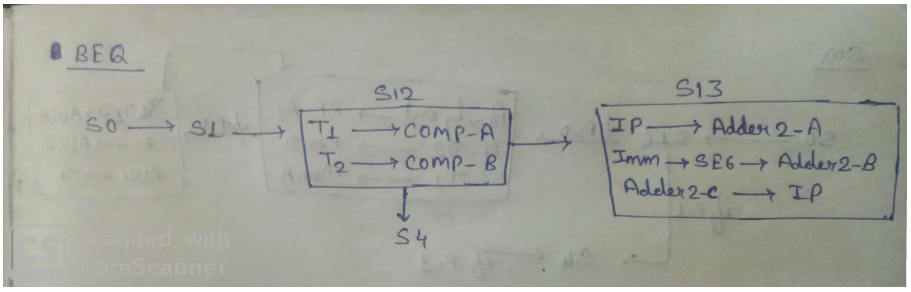

Figure 18: FSMO of BEQ instruction

0.9.17 JAL

JAL (Jump And Link) branches to the address PC+ Imm. PC is stored into RegisterA, where PC is the address of the JAL instruction

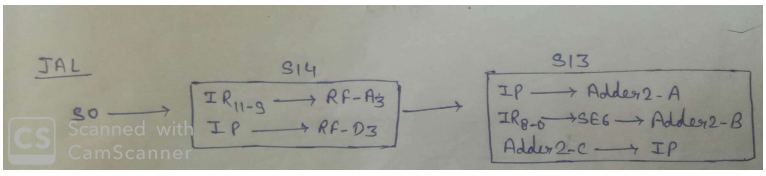

Figure 19: FSMO of JAL instruction

0.9.18 JLR

Branch to the address in RegisterB. Store PC into RegisterA, where PC is the address of the JLR instruction

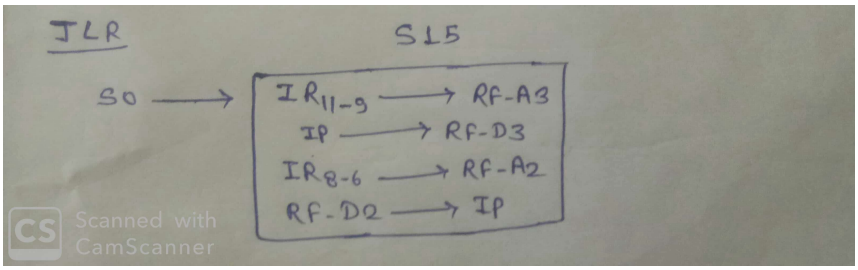

Figure 20: FSMO of JLR instruction