EE224 Digital Systems
Project Design

Aaron John Sabu, 170070050
Ayan Sharma, 170020023
Inderjeet Jayakumar Nair, 170020013
Shivanshu Gupta, 170020032

April 2019

1 Introduction

The project intends to simulate the functionality of a multi-cycle processor. As a result, we connect various structural entities in a specific manner whose activity is controlled by a series of control signals.
The idea is to decode the instruction and pave a way for appropriate signalling to execute sequential operations of a particular instruction. We enlist below some of the required entities with their functions. Thereafter we would derive a detailed Finite State Machine which relates the activity of various steps with some clock cycle for a particular operation.

2 Design Elements

2.1 Flip Flop/1-bit Register

This is a component which assigns the input to the output when the clock is high.

2.2 Register File

The Register File consists of eight 16-bit registers alongside logic circuits in order to decode the inputs to the register file for various functions on the register.

2.3 Instruction/Data memory

This block is very similar to the register file in that it stores data and instructions. These can be used when and where required by loading them onto the registers.

2.4 Sign Extender

Depending on the sign of the 6-bit input, 10 identical bits (1 if negative, 0 if positive) are appended to the left of the MSB, hence making the number an equal-value 16-bit number.

2.5 Zero Extender

7 bits, all being 0, are appended to the right of the LSB of a 9-bit number. This is used in the LHI instruction.

3 Design

The following instruction formats are used to implement the processor:

3.1 R-type

Opcode 4-bit	RegisterA (RA) 3-bit	RegisterB (RB) 3-bit	RegisterC (RC) 3-bit	Unused 1-bit	Condition on CZ 2-bit

3.1.1 ADD

This instruction adds the content of RegB to RegA and stores the result in RegC.
If there is a carry from the MSB, the C flag is set. If the output is zero, the Z flag is set.

0000	RA	RB	RC	0	00

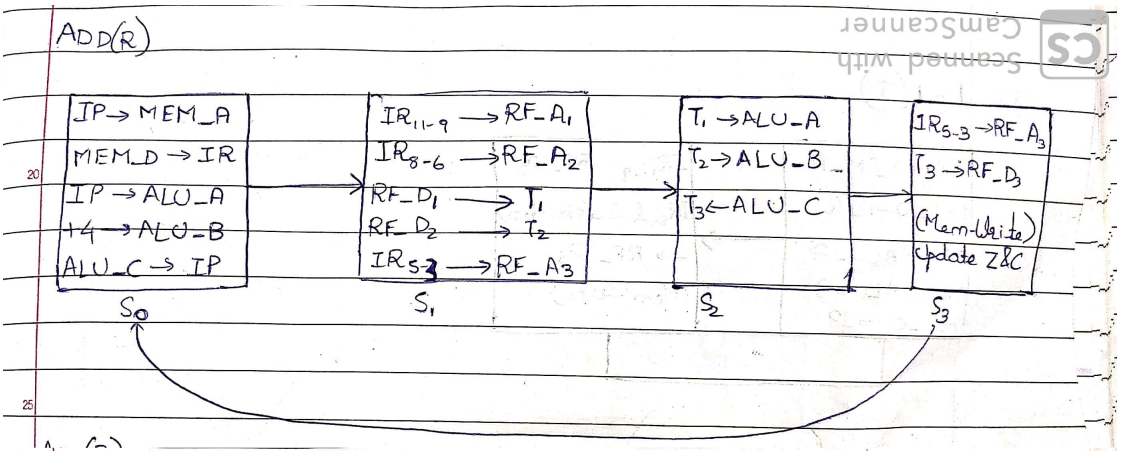

Figure 1: The ADD Design

3.1.2 ADC

This instruction adds the content of RegB to RegA and stores the result in RegC, given that the C flag is set.
If there is a carry from the MSB, the C flag is set. If the output is zero, the Z flag is set.

0000	RA	RB	RC	0	10

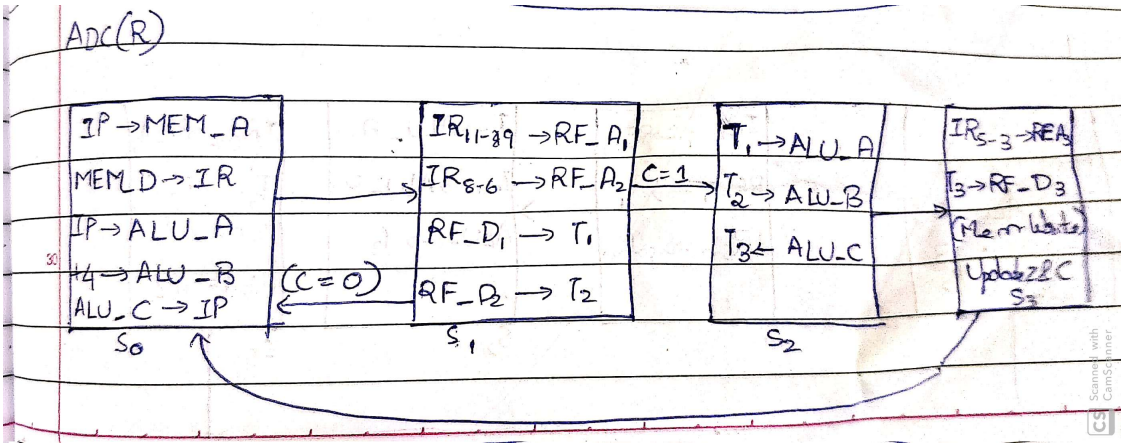

Figure 2: The ADC Design

3.1.3 ADZ

This instruction adds the content of RegB to RegA and stores the result in RegC, given that the Z flag is set.
If there is a carry from the MSB, the C flag is set. If the output is zero, the Z flag is set.

0000	RA	RB	RC	0	01

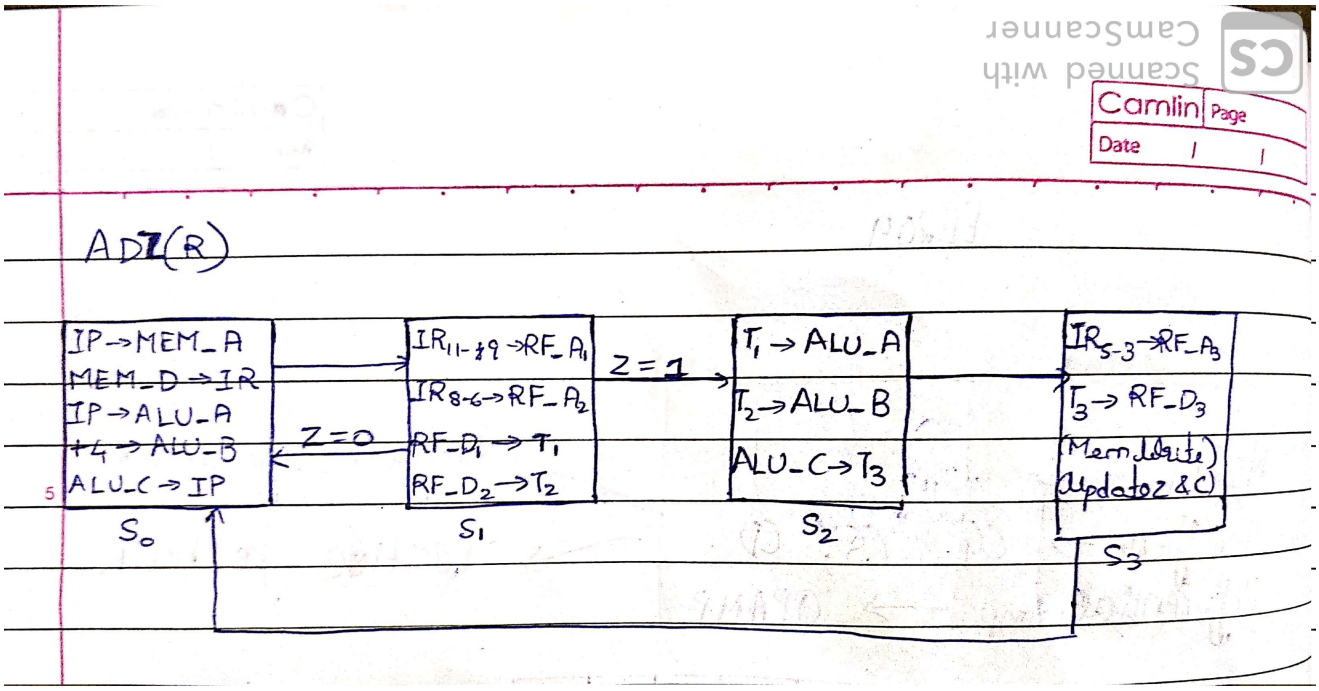

Figure 3: The ADZ Design

3.1.4 NDU

This instruction NANDs the content of RegA and RegB and stores the result in RegC.
If the output is zero, the Z flag is set.

0010	RA	RB	RC	0	00

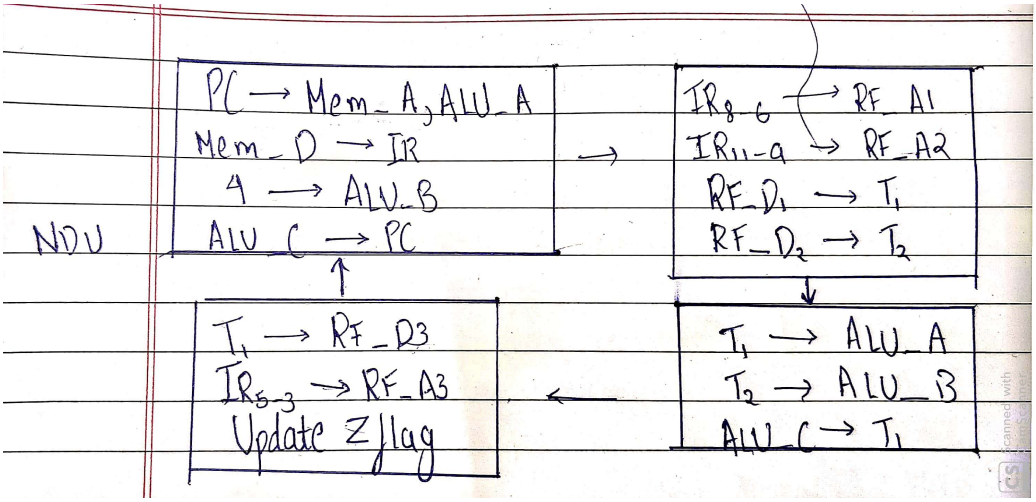

Figure 4: The NDU Design

3.1.5 NDC

This instruction NANDs the content of RegA and RegB and stores the result in RegC, given that the C flag is set.
If the output is zero, the Z flag is set.

0010	RA	RB	RC	0	10

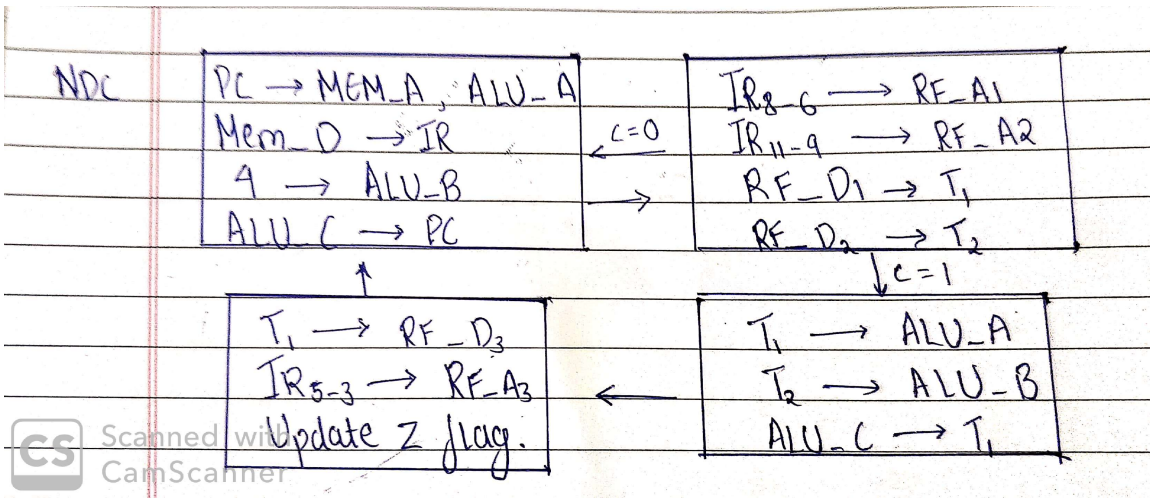

Figure 5: The NDC Design

3.1.6 NDZ

This instruction NANDs the content of RegA and RegB and stores the result in RegC, given that the Z flag is set.
If the output is zero, the Z flag is set.

0010	RA	RB	RC	0	01

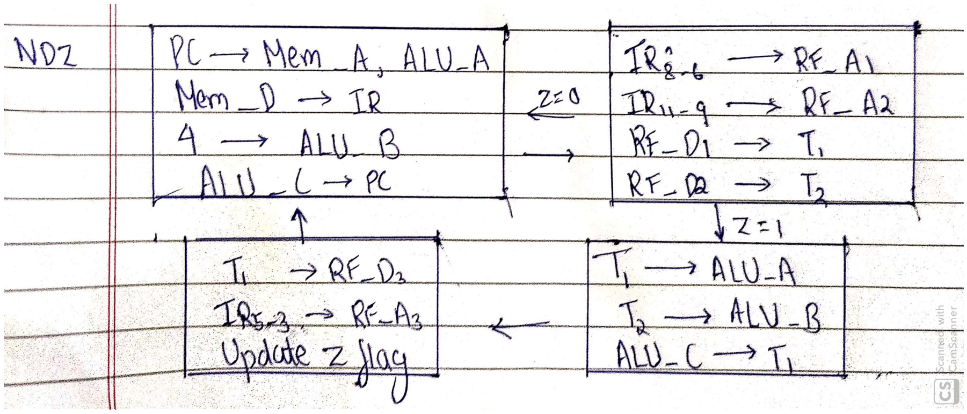

Figure 6: The NDZ Design

3.2 I-type

Opcode 4-bit	RegisterA (RA) 3-bit	RegisterC (RC) 3-bit	Immediate 6-bit (signed)

3.2.1 ADI

This instruction adds the value of *Immediate* to RegA and stores the result in RegB.
If there is a carry from the MSB, the C flag is set. If the output is zero, the Z flag is set.

0001	RA	RB	6-bit Immediate

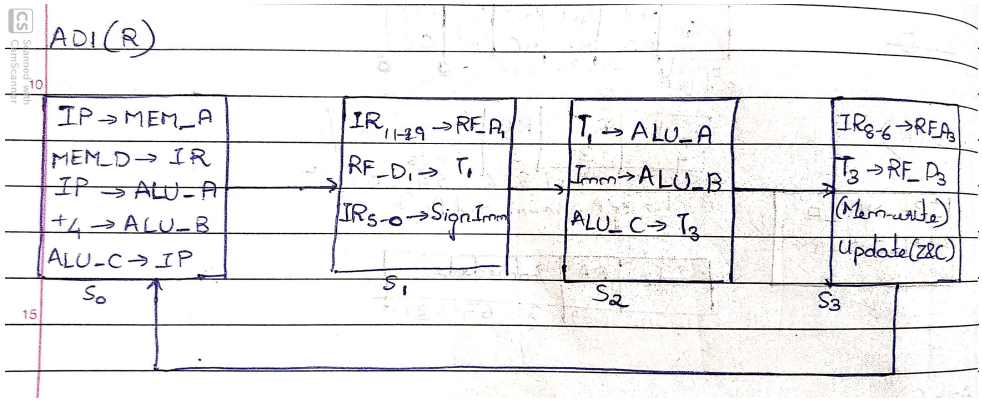

Figure 7: The ADI Design

3.2.2 LW

The address stored in RegB is added to the *6-bit Immediate* and the content stored in the corresponding memory location is loaded onto RegA.
If the output is zero, the Z flag is set.

0100	RA	RB	6-bit Immediate

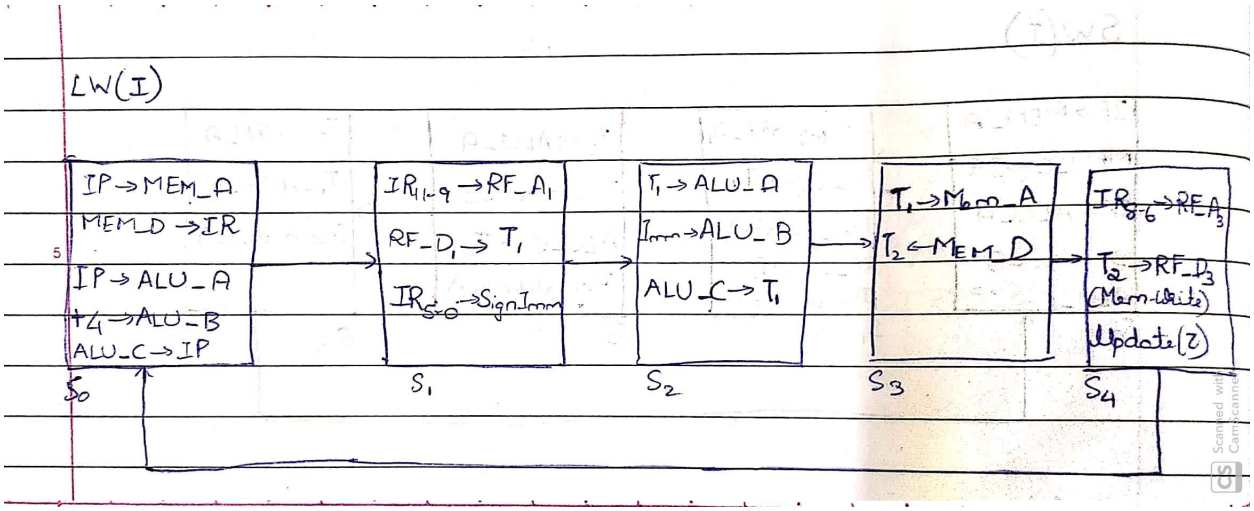

Figure 8: The LW Design

3.2.3 SW

The address stored in RegB is added to the *6-bit Immediate* and the content stored in RegA is loaded onto the memory location obtained from the sum described above.

0101	RA	RB	6-bit Immediate

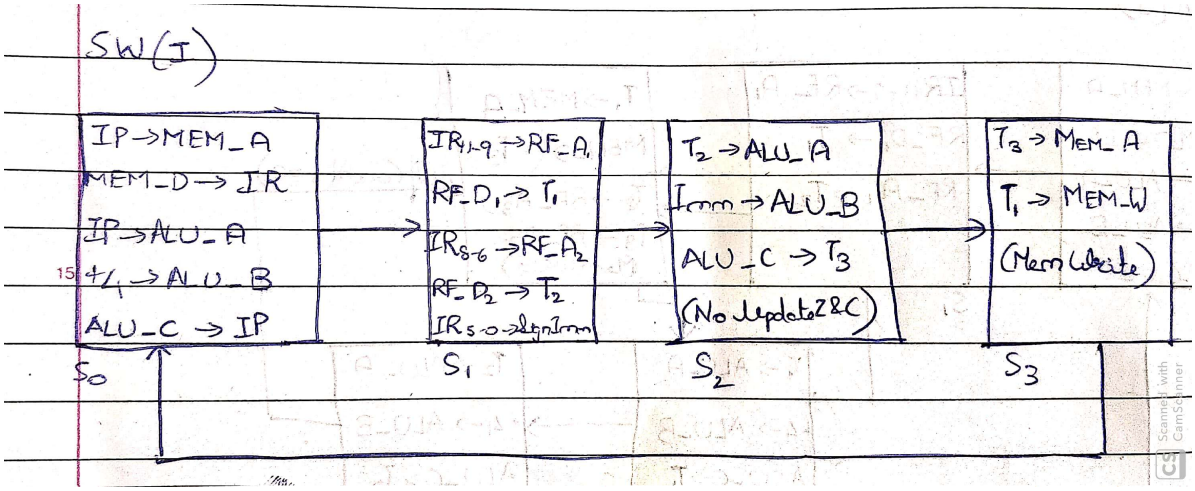

Figure 9: The SW Design

3.2.4 BEQ

Given that the contents of RegA and RegB are the same, the instruction pointer is incremented by *Immediate* bits. Else the instruction pointer follows the usual sequence.

1100	RA	RB	6-bit Immediate

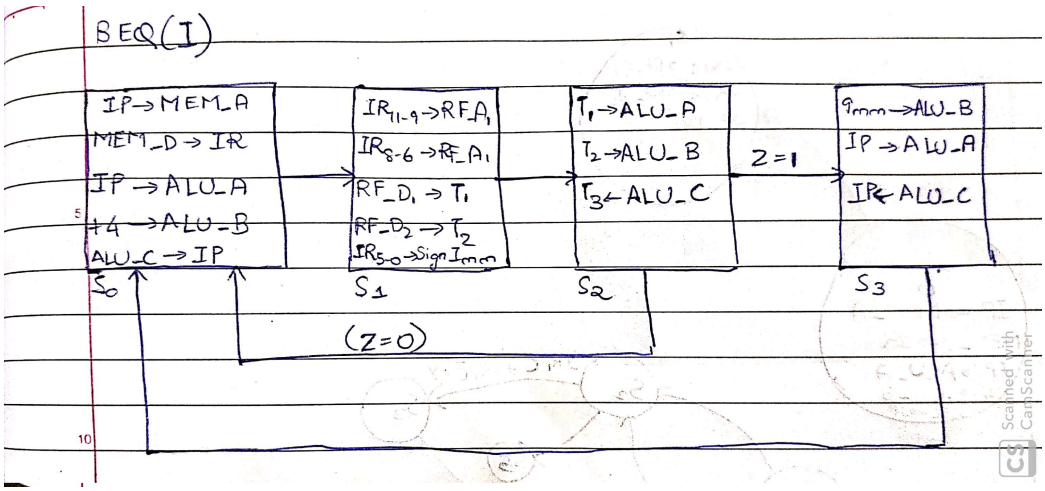

Figure 10: The BEQ Design

3.2.5 JLR

The instruction pointer is stored in RegA, following which the instruction pointer jumps to the address in RegB.

1001	RA	RB	000000

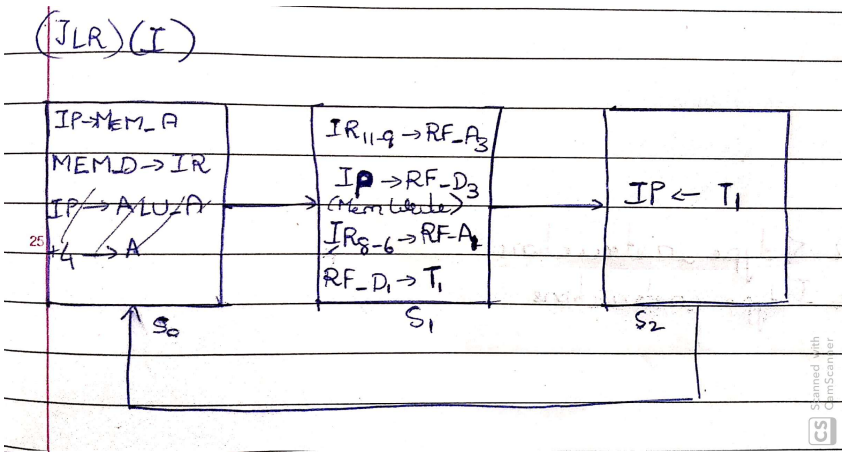

Figure 11: The JLR Design

3.3 J-type

Opcode 4-bit	RegisterA (RA) 3-bit	Immediate 9-bit (signed)

3.3.1 LHI

The *9-bit Immediate* bits are assigned to the most significant 9 bits of RA. The remaining 7 bits of RA as reset (assigned to 0).

0011	RA	9-bit Immediate

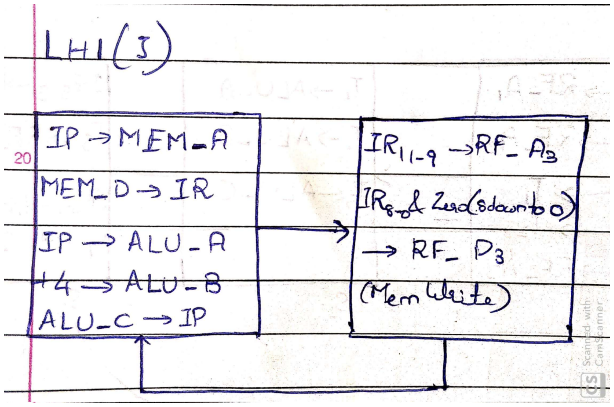

Figure 12: The LHI Design

3.3.2 LA

Starting from the memory location stored in RegA, the eight 16-bit values are loaded onto the eight 16-bit registers.

0110	RA	

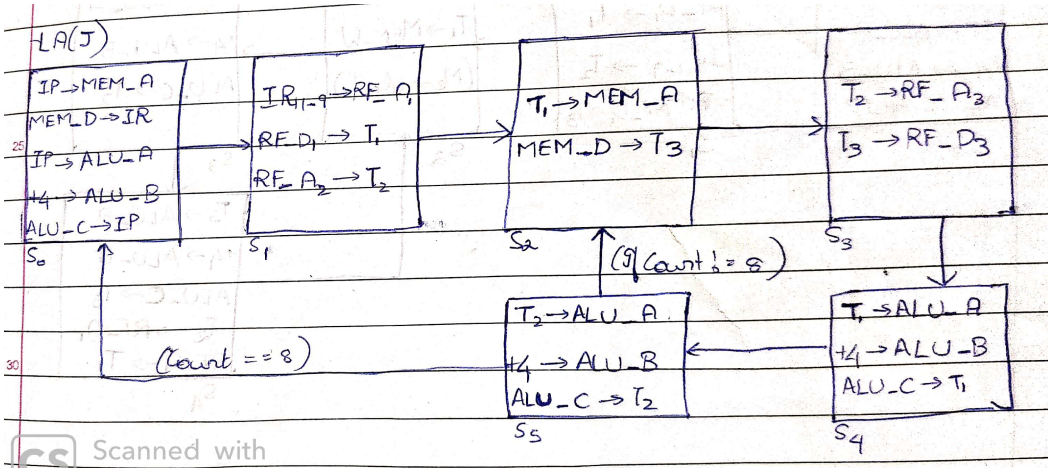

Figure 13: The LA Design

3.3.3 SA

Starting from the memory location stored in RegA, the contents in the eight 16-bit registers are stored onto eight 16-bit locations in the memory.

0111	RA	

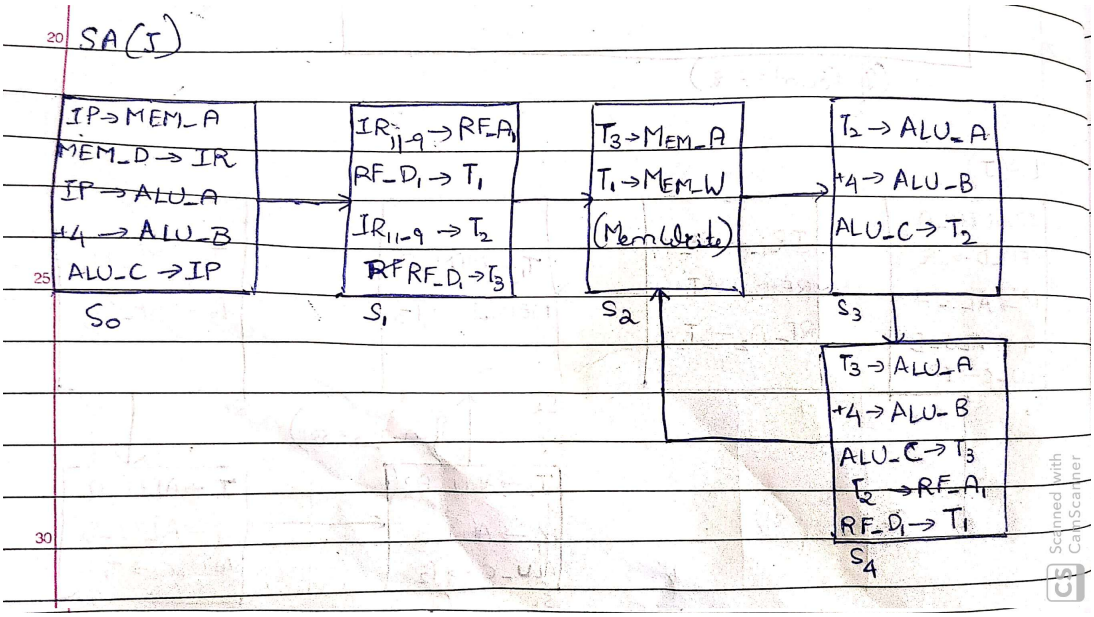

Figure 14: The SA Design

3.3.4 JAL

The instruction pointer is incremented by *Immediate* bits.

1000	RA	9-bit Immediate Offset

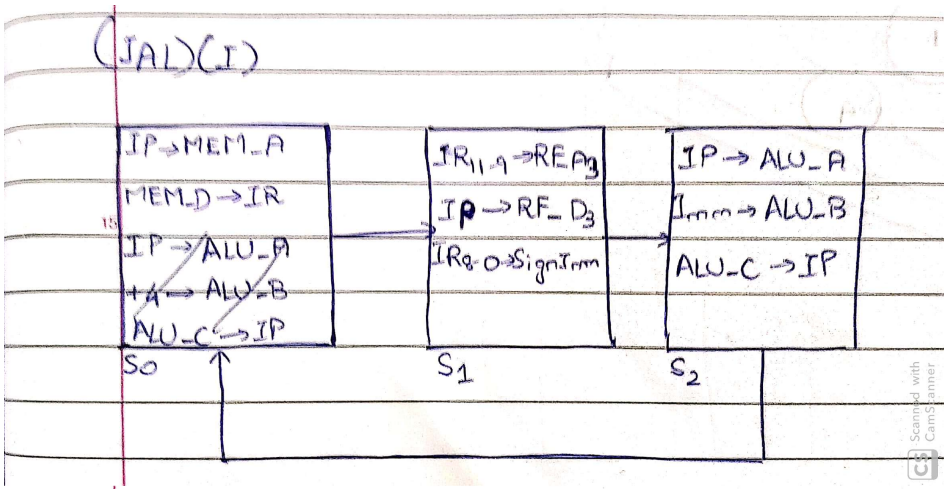

Figure 15: The JAL Design

4 FSM reduced representation

The above Finite State Machines are minimised, hence leading us to the following overall Finite State Machine for the entire processor:

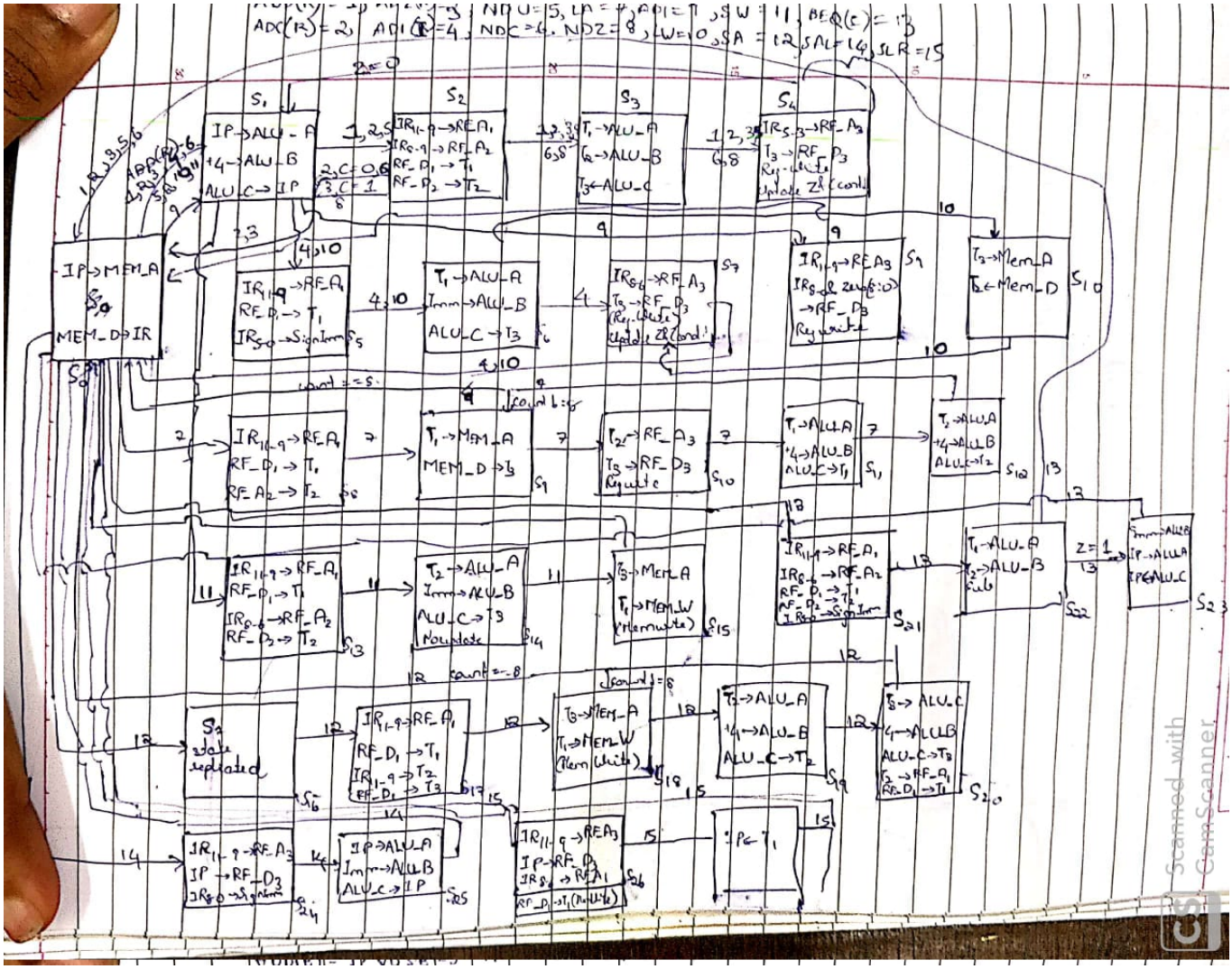

Figure 16: Minimised FSM