Instructions :

The basic structure of all the instructions is as follows :

R Type :

opcode (4 bit)	Register A (3 bit)	Reg. B (3 bits)	Reg. C (3 bit)	Unused (1 bits)

condition
(2 bit)

J Type :

opcode (4bits)	Reg. A (3 bit)	Reg C (3 bits)	Immediate (6 bit)

I Type :

opcode (4 bit)	Reg. A (3 bit)	Immediate (9 bit signed)

For each, the implementation follows these steps :

1. Sending the PC to memory and fetching the instruction. Along with this, we can also increment PC.
2. Decode the instructions and read the registers.
   In case of Jump and BEQ, we can also ~~calc~~ calculate the target address by adding immediate to PC,

3. In this step, operations are done on the read values using the ALU.
This includes add / nand for ADD, ADC, ADZ, NDU, NDC etc. Also, immediate is added to base address in this step

4. In case of load / store, memory is read / written to in this step.

5. The result is stored back in RF.

FSM :

1. ADD, ADC, ADZ, NDU, NDC, NDZ

STATES :

S0 : Read Instruction

pc → Mem_~~Add~~

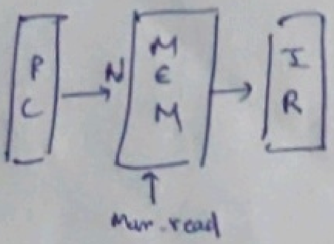
Mem_D~~a~~ta → IR

~~pc → Alu_A~~

~~on~~

mem_Read

IR_write



S0' : Increment PC

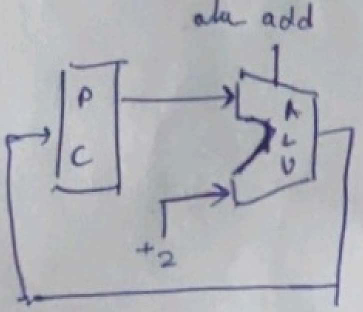
pc → Alu_A

+1 → Alu_B

Alu_C → pc

PC_write

Alu_Add



S1 : Read Registers

$IR_{11-9} \rightarrow rf\_A_1$

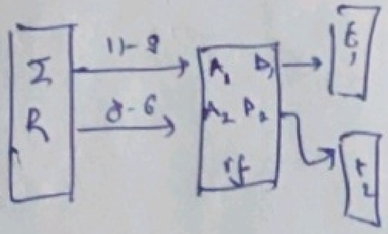
$IR_{8-6} \rightarrow rf\_A_2$

$rf\_D_1 \rightarrow t_1$

$rf\_D_2 \rightarrow t_2$

t_1_write

t_2_write



S2 : ~~A~~ Perform operation

$t_1 \rightarrow$ Alu_A

$t_2 \rightarrow$ Alu_B

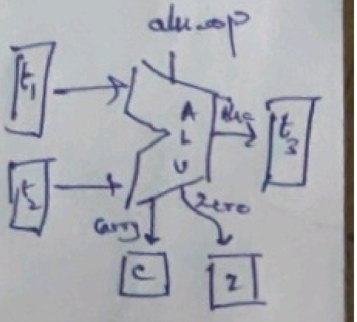
Alu_C → t_3

Alu_Carry → C

Alu_Zero → Z

t_3_write

alu_op (Depending on whether add or nand is to be done)

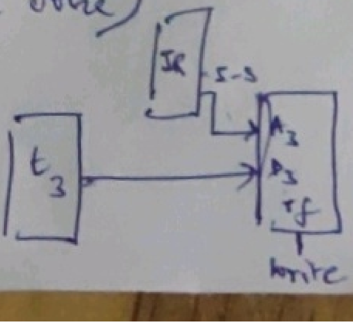


S3 : Write to register file

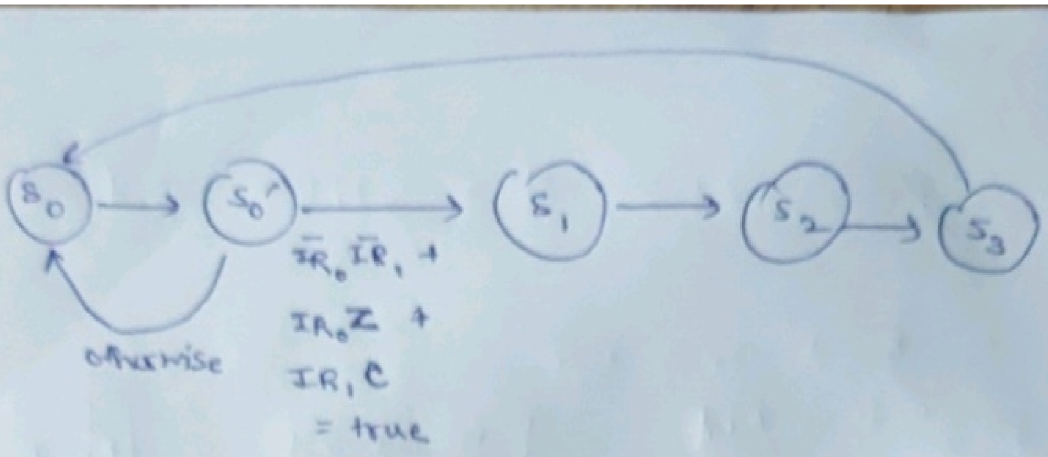
$t_3 \rightarrow rf\_D_3$

$IR_{5-3} \rightarrow rf\_A_3$

rf_write



FSM:

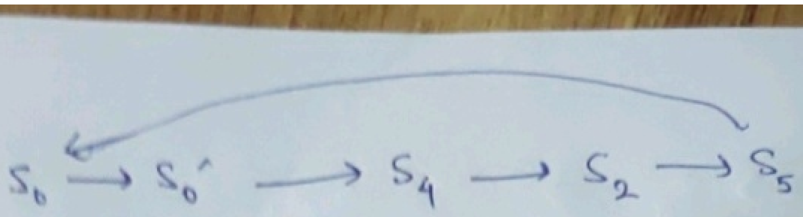


- The condition on $S_0' \to S_1$ is to ensure that the addition is done only if conditions ~~of~~ for instructions are met. For e.g., ADC needs C to be set. So, if the command is ADC (identified by $IR_1 = 1$), we need to make sure $C = 1$. This is done using the given formula.

The condition is written in compact here. We can easily use a decoder to do the transition.

[Use 4 input decoder with inputs being ir_0, ir_1, c, z.]

2. ADI



S4 : Read register and ~~@~~ intermediate

$IR_{11-9} \rightarrow RF\_A1$

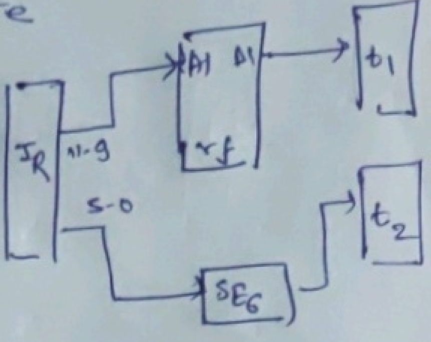
$RF\_D1 \rightarrow t_1$

$IR_{5-0} \rightarrow SE_6$

$SE_6 \rightarrow t_2$

t_1_write

t_2_write

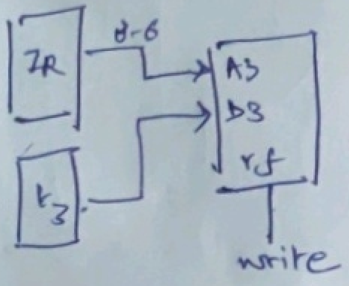


S5 : write to memory

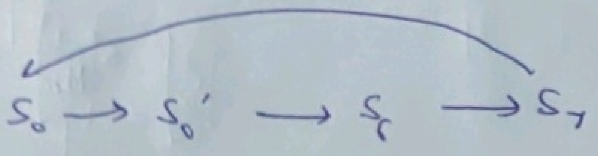
$t_3 \rightarrow rf\_D_3$

$IR_{8-6} \rightarrow rf\_A_3$

rf_write



3. LHI



S6 :

$IR_{8-0} \rightarrow SE_9$

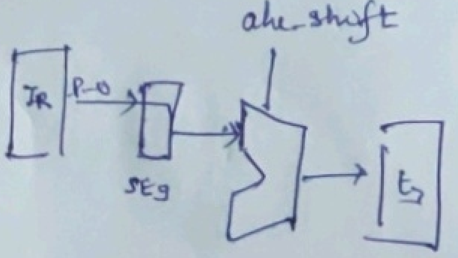
$SE_9 \rightarrow Alu_A$

$Alu_C \rightarrow t_3$

~~sign~~

7- bit Shifter

t_3 - write

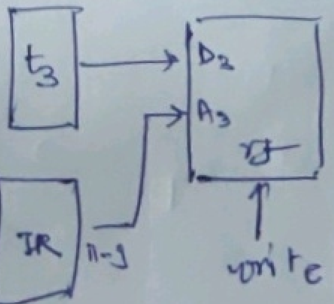


S7 :

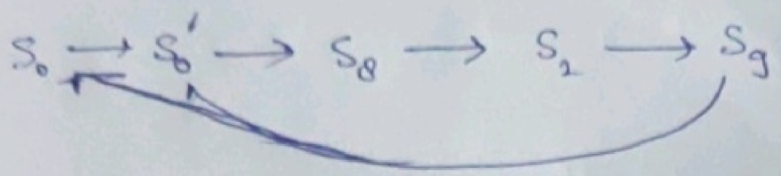
$t_3 \rightarrow$ ~~R~~ rf_D_2

$IR_{11-9} \rightarrow rf\_A_3$

rf_write



4. LW



S_8: Calculating Address

$IR_{8-6} \longrightarrow rf\_A2$

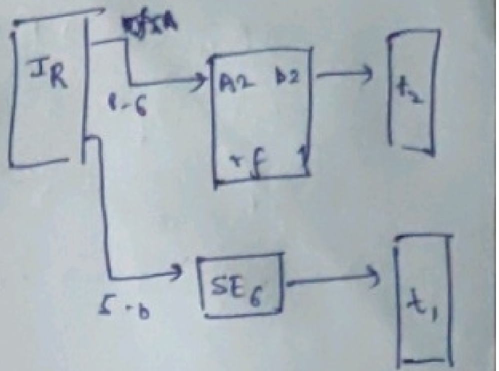
$rf\_D_2 \longrightarrow t_2$

$IR_{5-0} \longrightarrow SE_6$

$SE_6 \longrightarrow t_1$

t_1-write

t_2-write



S_9: Load

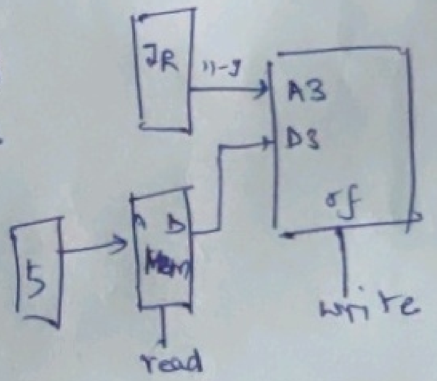
$t_3 \longrightarrow Mem\_A$

$Mem\_D \longrightarrow$ ~~B~~ rf_d_3

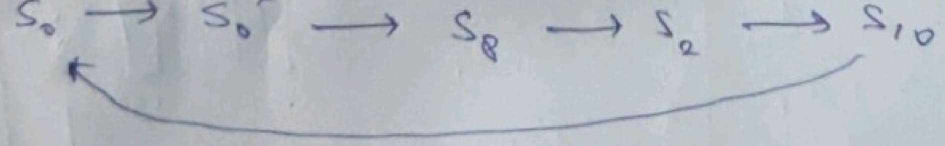
$IR_{11-9} \longrightarrow rf\_a_3$

Mem_read

rf_write



5: SW



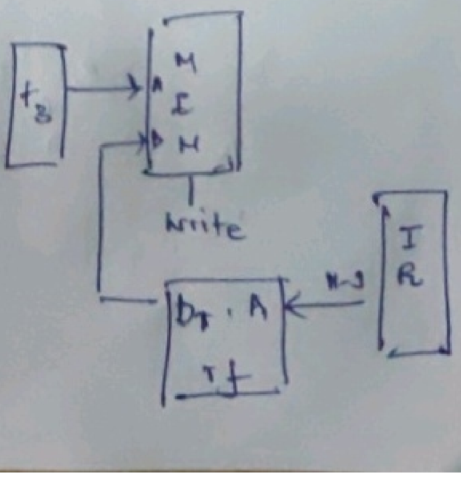
S_{10}: Writing to Memory

$t_3 \longrightarrow Mem\_A$

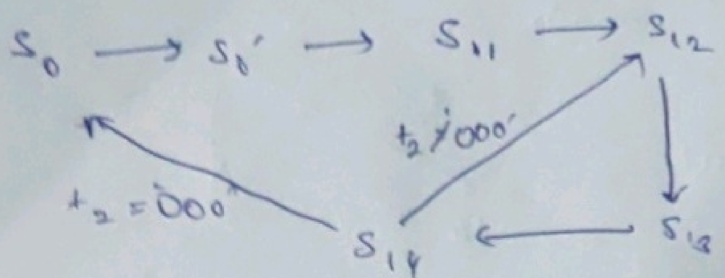
$IR_{11-9} \longrightarrow rf\_a_1$

$rf\_d_1 \longrightarrow mem\_D$

mem_write

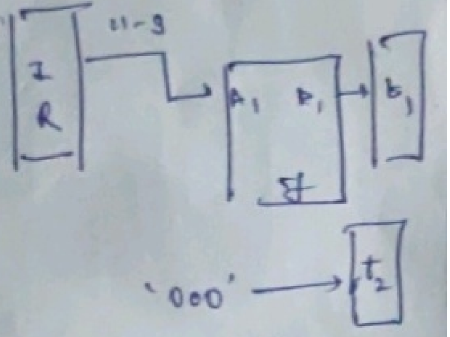


6. LA



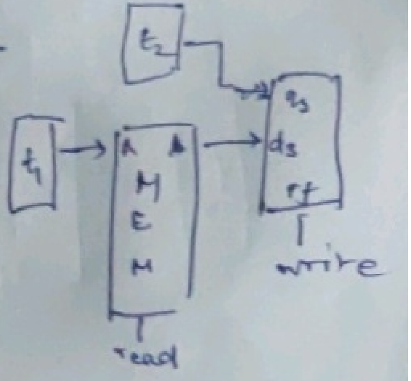
S_{11} : read memory address and initialise counter

$IR_{11-9} \rightarrow rf\_A_1$	t_1_write
$rf\_D_1 \rightarrow t_1$	t_2_write
'000' $\rightarrow t_2$	



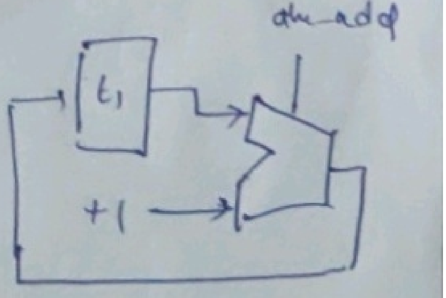
S_{12}: read memory and write to rf

$t_1 \rightarrow mem\_A$	mem_read
$mem\_D \rightarrow rf\_d_3$	rf_write
$t_2 \rightarrow rf\_a_3$	



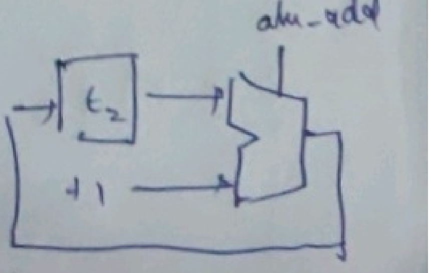
S_{13}: increase memory address

$t_1 \rightarrow alu\_A$	t_1_write
$+1 \rightarrow alu\_B$	alu_add
$alu\_C \rightarrow t_1$	

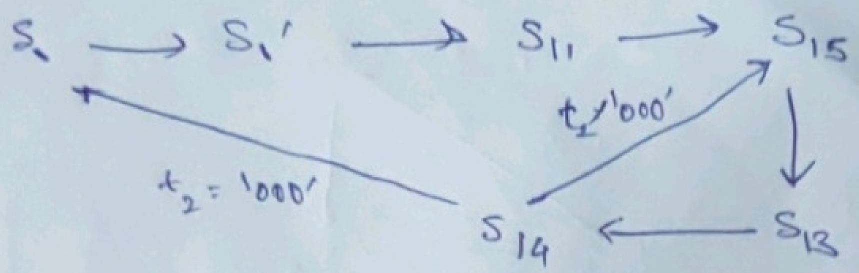


S_{14}: increase counter

$t_2 \rightarrow alu\_A$	t_2_write
$+1 \rightarrow alu\_B$	alu_add
$alu\_C \rightarrow t_2$	



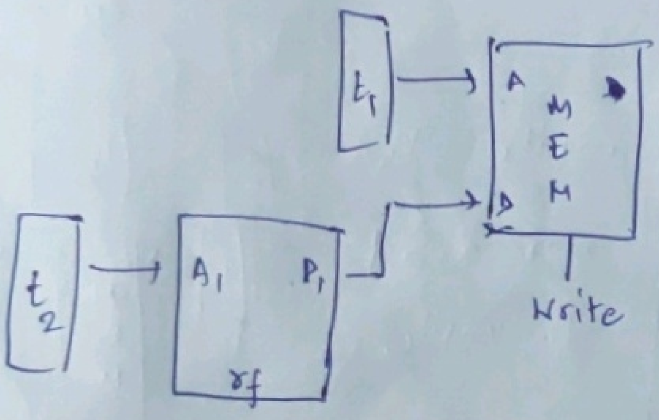
7: SA



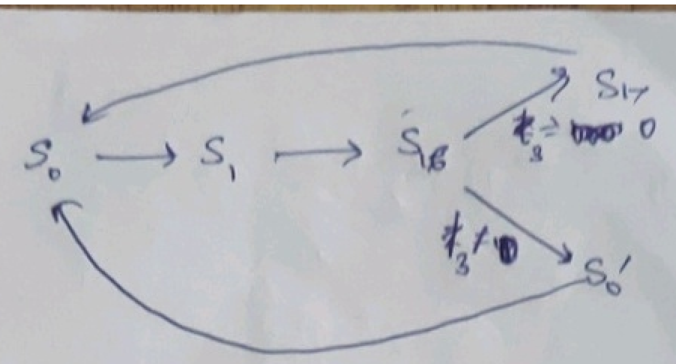
S_{15}: write to memory

$t_2 \rightarrow$ rf_A$_1$
rf_D$_1 \rightarrow$ mem_D
$t_1 \rightarrow$ mem_A

mem_write



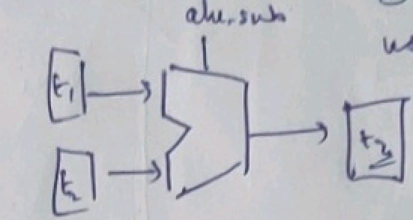
8. BEQ



S_{16} : Check if regA, regB have same content

$t_1 \rightarrow$ alu_A
$t_2 \rightarrow$ alu_B
~~alu_zero → t_3~~
~~alu → C~~
alu_C $\rightarrow t_3$

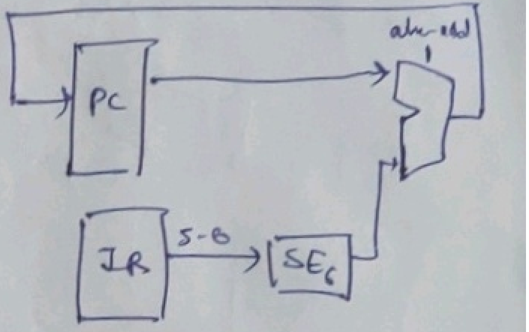
alu_sub
t_3_write (This can be done by giving cin = 1 and using alu_add)



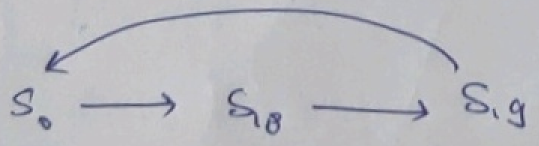
S_{17} : Update pc

pc $\rightarrow$ Alu_A
$IR_{5-0} \rightarrow SE_6$
$SE_6 \rightarrow$ Alu_B
Alu_C $\rightarrow$ pc

pc_write
alu_add



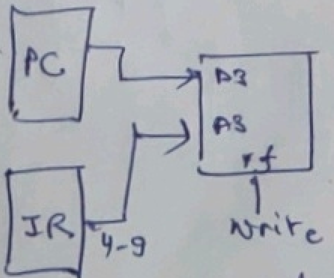
9. JAL



S_{18} : store pc

$IR_{4-3} \rightarrow rf\_A_3$
pc $\rightarrow rf\_D_3$

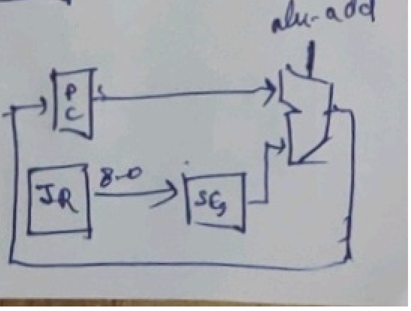
rf_write



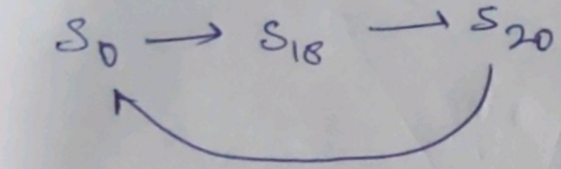
S_{19} : update pc

pc $\rightarrow$ Alu_A
$IR_{8-0} \rightarrow SE_9 \rightarrow$ Alu_B
Alu_C $\rightarrow$ PC

PC_write
alu_add



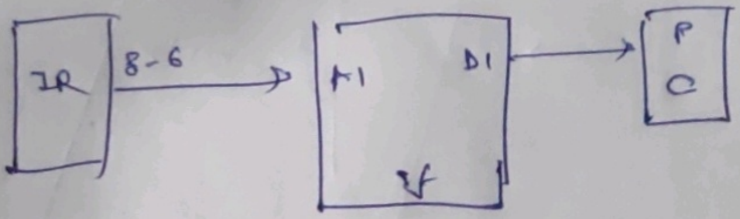
10: JLR:



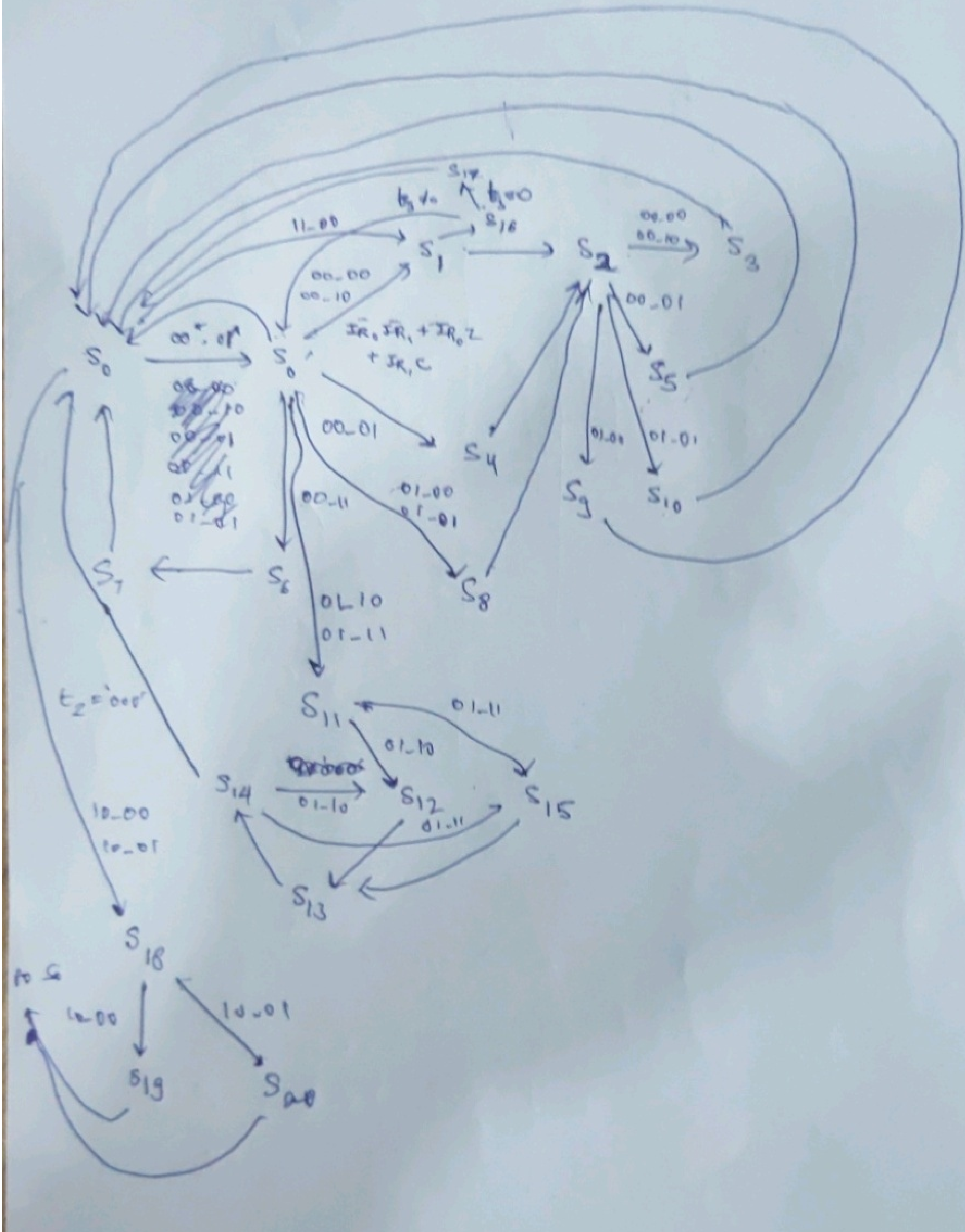
S_{20}:

$IR_{8-6} \longrightarrow$ rf-A_1 | pc-write

rf-$D_1 \longrightarrow$ pc



Complete State Diagram:



S17
t3≠0
t3=0
S16
11_00
S1
S2
00.00
00.10
S3
00_00
00_10
00_01
S0
00*. 01*
S0
IR0 IR1 + IR0 Z
+ IR1 C
S5
00_01
S4
01.00
01_01
S9
S10
01_00
01_01
00_11
S6
S7
S8
01_10
01_11
t2='000'
S11
01_11
01_10
S14
01_10
S12
01_11
S15
S13
10_00
10_01
S18
to S0
10_00
10_01
S19
S20