CS226

Digital Logic Design

Project

Name:	*Roll Number:*
Deepanshu	190050032
Megha Baboria	190050067
Nitin Kumar	190050073
Vikas Panwar	190050129

May 22, 2021

Design Documentation

We have designed an 8 register, 16 bit computer system . It has 8 general-purpose registers (R0 to R7) and can implement 15 basic general purpose instructions. The design of our computing system consists of the following main components:

- controller-FSM
  - IITB_Proc
- datapath (ALU, Registers, Register file etc.)
  - ALU
    - sixteen_bit_nand
    - sixteen_bit_xor
    - Sixteen_bit_adder
  - Register_File
  - Memory

ALU

```
entity ALU is
        port( alu_A,alu_B : in std_logic_vector(15 downto 0);
                op_type : in std_logic_vector(1 downto 0);
                C_out, Z_out: out std_logic;
                alu_C : out std_logic_vector(15 downto 0));
end entity;
```

It has two 16-bit inputs alu_A and alu_B and another two bit control input op_type which determines what operation is to be performed. The different operation performed by ALU are :

- If op_type = “00” :
  Addition of two 16-bit inputs to get a carry and an output

- If op_type = “01”:
  Nand of two sixteen bit inputs.

- If op_type = “10”
  Xor of two sixteen bit inputs.

Note : I have implemented xor instead of subtract to check if the two inputs are equal.

The ALU outputs a 16 bit number along with the following single bit flags:

- C_out : It gets set to 1 if we have carry output resulting by addition.
- Z_out : It gets set to 1 if the output generated by the operation is zero.

Register_File

```
entity Register_File is
	port( A1,A2,A3 : in std_logic_vector(2 downto 0);
		D3: in std_logic_vector(15 downto 0);
		clk,wr,reset: in std_logic ;
		D1, D2: out std_logic_vector(15 downto 0));
end entity;
```

It consists of eight 16-bit registers and allows writing into the register and reading from the registers.
It allows two registers to be read at a time. Which registers are to be read is decided by A1, A2 and their value is read into D1 and D2 asynchronously. A3 determines the register we want to write into and D3 is the value to be written synchronously provided the wr flag is high.

Memory

```
entity memory is
	port (wr_en,rd_en,clk: in std_logic;
```

```
        Addr_in, D_in: in std_logic_vector(15 downto 0);
        D_out: out std_logic_vector(15 downto 0));
end entity;
```

This allows reading from the memory and writing into the memory.
To read : We need to provide the address from which we want to read in Addr_in and make rd_en (read enabler) high.
To write : We provide the address to which we want to write in D_in. It gets written provided the wr_en (write enabler) is high.

IITB Proc

```
entity IITB_PROC is
    port (clk,rst  : in  std_logic);
end entity;
```

This is the heart of our machine. It integrates the various sub-components (Memory, ALU and Register File). It also has temporary registers (T1,T2,T3,T4,IR).
It takes the OpCode (most significant four bits of any instruction) and values of C, Z registers as input in its output logic and for state transition.
We have implemented the finite state machine in this part using behavioural VHDL logic.Our machine is of Mealy type with 22 (S0 to S20, Sres, Spc) different states. The machine is Mealy since the output (i.e control signals) depends on the input (i.e OpCode and C,Z).

The implementation of different instructions by our finite state machine is shown below :

ADD, ADC, ADZ

S_0

PC → Mem_Address
Mem_data_out → IR

S_1

IR[11-9] → RF_A_1
IR[8-6] → RF_A_2
RF_D_1 → T_1
RF_D_2 → T_2.

S_2 — ADD

T_1 → Alu_x
T_2 → Alu_y
Alu_out → T_3
~~carry_out → c~~
~~zero_out → z~~

S_3

T_3 → RF_D_3
IR[5-3] → RF_A_3
carry-out → c
zero-out → z

SPC

PC → Alu_x
+2 → Alu_y
Alu-out → PC

ADI

S_0

PC → Mem_Address
Mem_data_out → IR

S_4

IR[11-9] → RF_A_1
IR[5-0] → SE10_in
RF_D_1 → T_1
SE10_out → T_2.

S_2

T_1 → Alu_x
T_2 → Alu_y
Alu_out → T_3

ADD

S_5

T_3 → RF_D_3
IR[8-6] → RF_A_3
carry_out → c
zero_out → z

Spc

Pc → Alu_x
+2 → Alu_y
Alu_out → Pc

NDU , NDC , NDZ

S_0

PC → Mem_Address
Mem_data_out → IR

↓

S_1

IR [11 − 9] → RF_A_1
IR [8 − 6] → RF_A_2
RF_D_1 → T_1
RF_D_2 → T_2

↓

S_2

T_1 → Alu_x
T_2 → Alu_y
Alu_out → T_3

NAND

↓

S_3

T_3 → RF_D_3
IR [5 − 3] → RF_A_3
zero_out → z

↓

Spc

Pc → Alu_x
+2 → Alu_y
Alu_out → Pc

LHI

S_0

PC → Mem_address
Mem_data_out → IR

↓

S_6

IR[8 − 0] → LHI_in
LHI_out → T_1.

↓

S_7

T_1 → RF_D_3
IR [11 − 9] → RF_A_3

↓

Spc

PC → Alu_x
+2 → Alu_y
Alu_out → pc

LW

S_0

PC → Mem_Address
Mem_data_out → IR

S_8

IR[8-6] → RF_A_1
IR[5-0] → SE10_in
RF_D_1 → T_1
SE10_out → T_2

S_2

T_1 → Alu_x
T_2 → Alu_y
Alu_out → T_3
ADD

S_9

T_3 → Mem_Address
Mem_data_out → RF_D_3
IR[11-9] → RF_A_3

S_{pc}

PC → Alu_x
+2 → Alu_y
Alu_out → PC

SW

S_0

PC → Mem_Address
Mem_data_out → IR

S_8

IR[8-6] → RF_A_1
IR[5-0] → SE10_in
RF_D_1 → T_1
SE_10_out → T_2

S_2

T_1 → Alu_x
T_2 → Alu_y
Alu_out → T_3
ADD

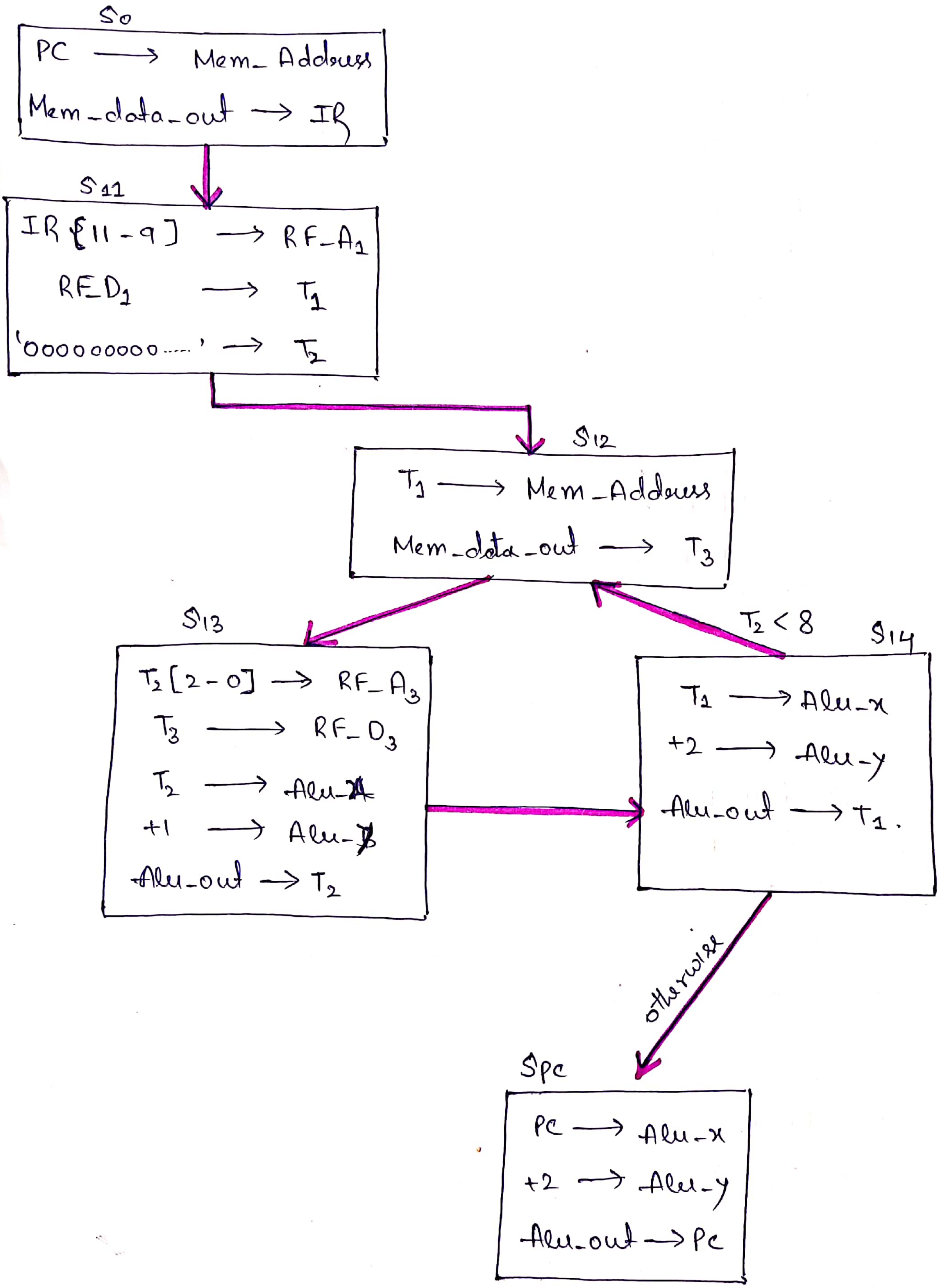
S_{10}

IR[11-9] → RF_A_1.
RF_D_1 → T_2
T_3 → Mem_address
T_2 → Mem_data_in

S_{pc}

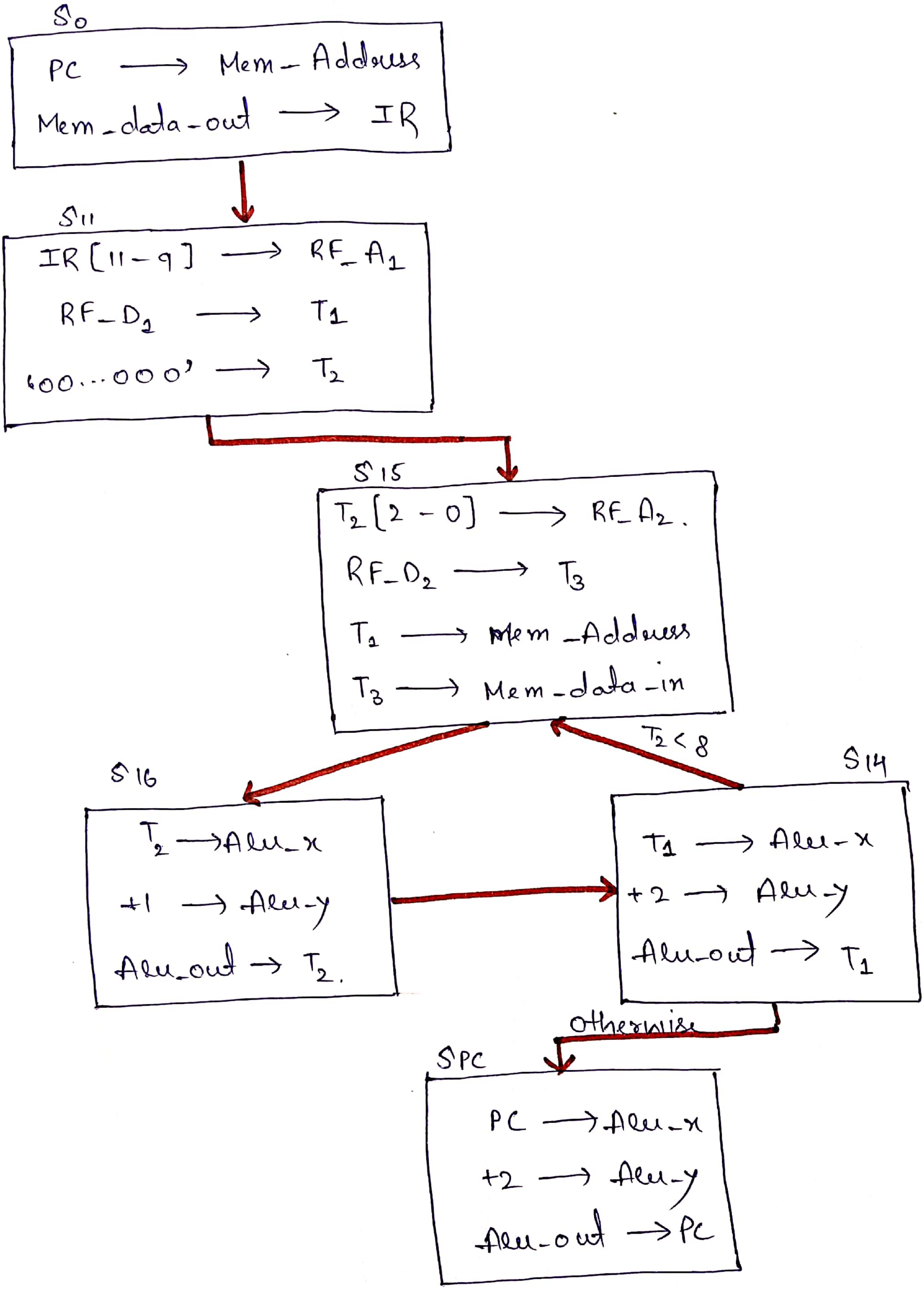
PC → Alu_x
+2 → Alu_y
Alu_out → PC

LA



S0
PC → Mem_Address
Mem_data_out → IR
S11
IR [11-9] → RF_A1
RF_D1 → T1
'000000000.....' → T2
S12
T1 → Mem_Address
Mem_data_out → T3
S13
T2 [2-0] → RF_A3
T3 → RF_D3
T2 → Alu_A
+1 → Alu_B
Alu_out → T2
T2 < 8
S14
T1 → Alu_x
+2 → Alu_y
Alu_out → T1.
otherwise
Spc
Pc → Alu_x
+2 → Alu_y
Alu_out → Pc

SA


S0
PC → Mem_Address
Mem_data_out → IR
S11
IR[11-9] → RF_A1
RF_D1 → T1
'00...000' → T2
S15
T2[2-0] → RF_A2.
RF_D2 → T3
T1 → Mem_Address
T3 → Mem_data_in
T2 < 8
S16
T2 → Alu_x
+1 → Alu_y
Alu_out → T2.
S14
T1 → Alu_x
+2 → Alu_y
Alu_out → T1
otherwise
SPC
PC → Alu_x
+2 → Alu_y
Alu_out → PC

BEQ

S_0

PC → Mem_Address
Mem_data_out → IR

S_1

IR [11 – 9] → RF_A1.
RF_D1 → T1
IR [8 – 6] → RF_A2
RF_D2 → T2 .

S_2

T1 → Alu_x
T2 → Alu_y
Alu_out → T3
XOR

zero_out = 1 . → S_{17}

S_{17}

PC → Alu_x
IR[5–0] → SE10_in
SE10_out → Alu_y
Alu_out → PC .

else → S_{pc}

S_{pc}

PC → Alu_x
+2 → Alu_y
Alu_out → PC .

JAL

S_0

$PC \rightarrow Mem\_Address$

$Mem\_data\_out \rightarrow IR$

S_{18}

$IR[11-9] \rightarrow RF\_A_3$

$PC \rightarrow RF\_D_3$

S_{19}

$PC \rightarrow Alu\_x$

$IR[8-0] \rightarrow SE7\_in$

$SE7\_out \rightarrow Alu\_y$

$Alu\_out \rightarrow PC$

JLR

S_0

$PC \rightarrow Mem\_Address$

$Mem\_data\_out \rightarrow IR$

S_{18}

$IR[11-9] \rightarrow RF\_A_3$

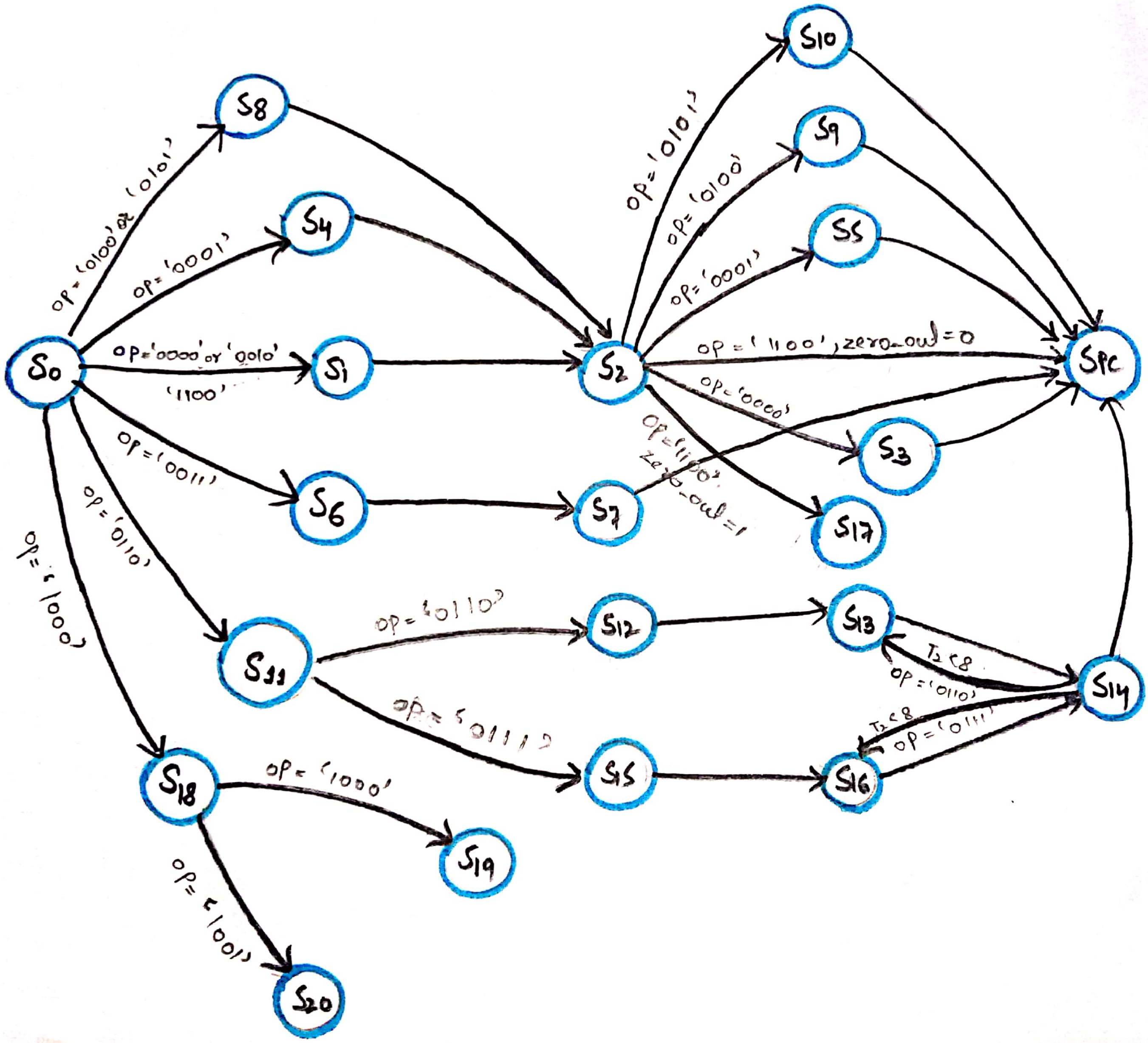
$PC \rightarrow RF\_D_3.$

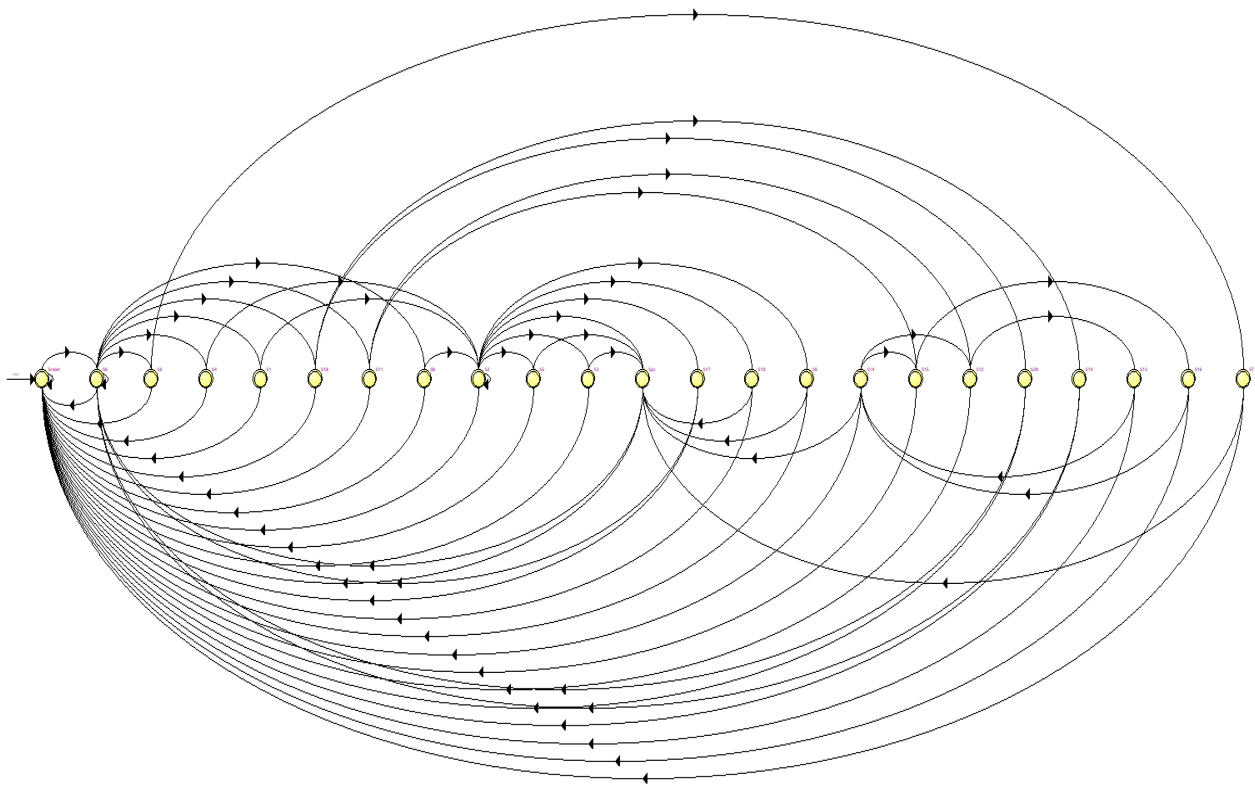
S_{20}

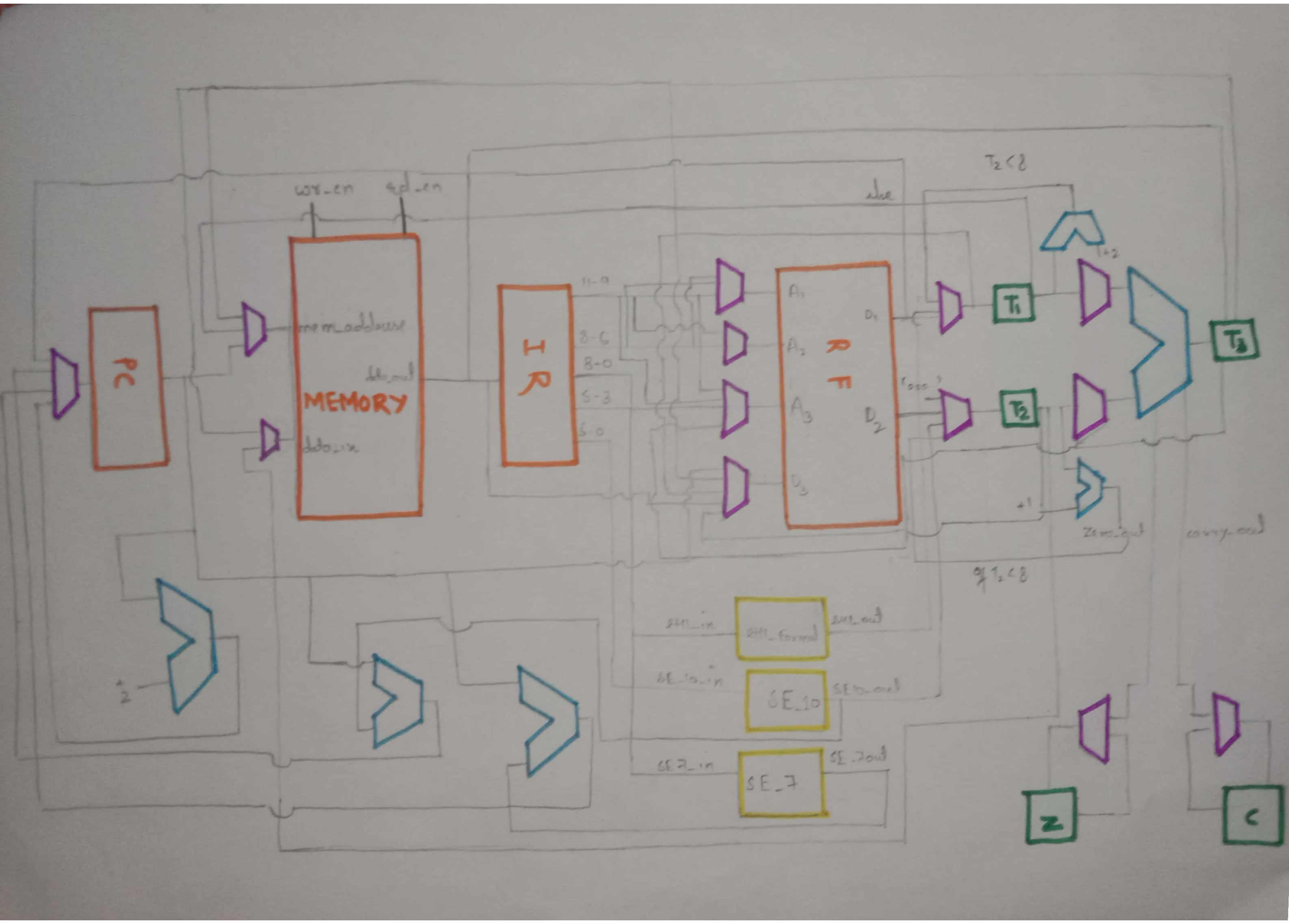
$IR[8-6] \rightarrow RF\_A_1$

$RF\_D_1 \rightarrow PC$

COMPLETE STATE DIAGRAM




Sreset
S0
S6
S4
S1
S18
S11
S8
S2
S5
S3
Spc
S17
S10
S9
S14
S15
S12
S20
S19
S13
S16
S7


wr_en
rd_en
PC
MEMORY
I R
R F
T1
T2
T3
Z
C
SE_10
SE_7

For testing purposes, we initialized state to restart state and pointer pc to “0000000000000000” and then observed the various internal signals (RF_A1,RF_A2,RF_A3,RF_D1,..etc) in each cycle.

Following waveform shows variation in following :
RF_A1,RF_A2,RF_A3,RF_D1,RF_D2,RF_D3,IR,PC,Mem_address,Mem_data_in,
Mem_data_out