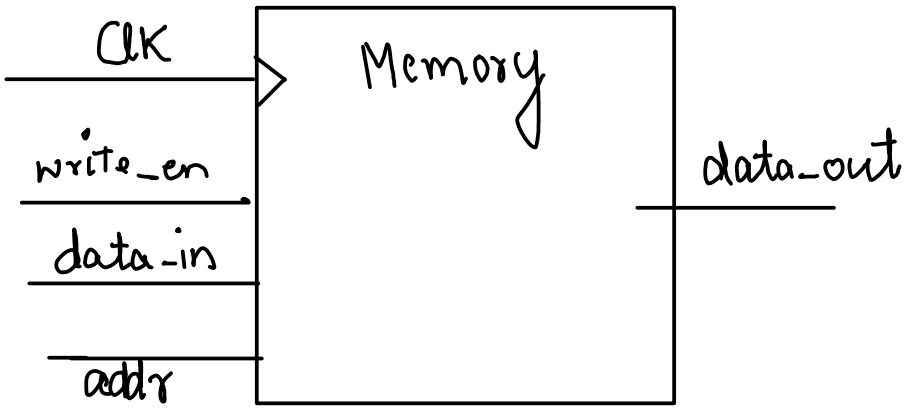
Parshant Arora (200050099) Vidit Goel (200050156)
Shubhkumar (200050134) Aaryan gupta (200050002)

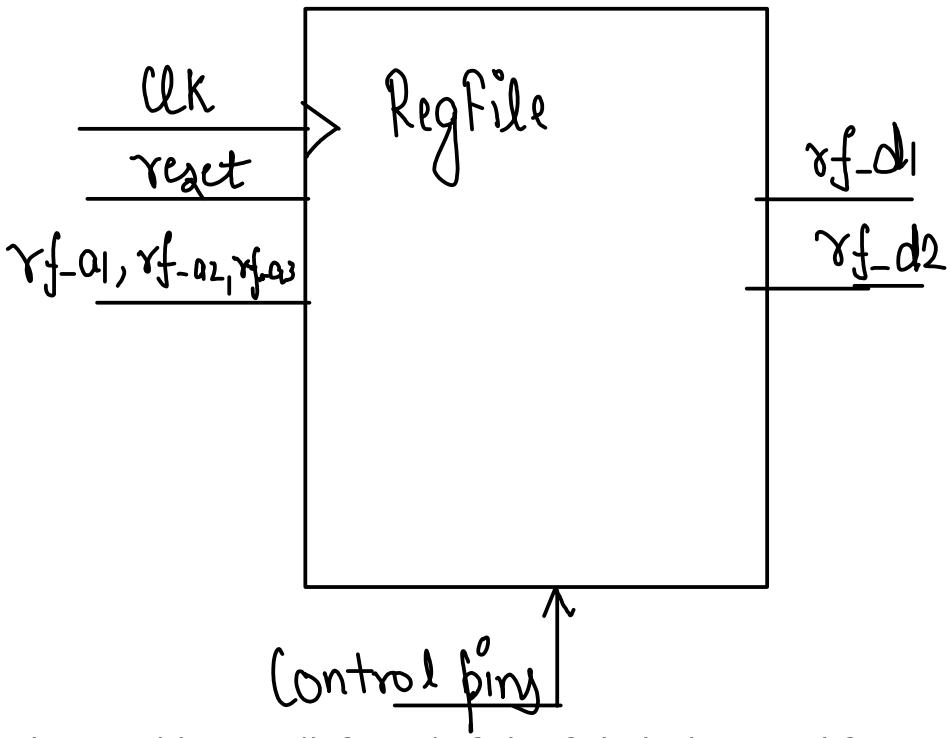
REPORT PROJECT

1)



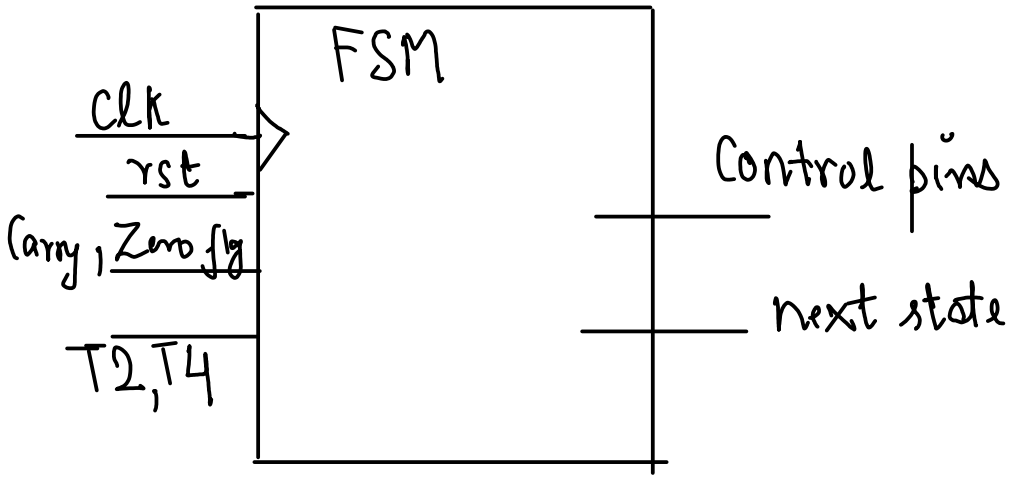
Input is clock , write enable, and data_in and address. If qrite is enabled, it writes the data_in into address and outputs the same data. and if it not enables then just reads the data from address and outputs it.

2)



RegFile takes 2 addresses (infact 3) rf_d1,rf_d2 is data read from addresses rf_a1, rf_a2
There are couple of control pins that decide whether write is enables or not, and for r7, it decides what to write (since it store PC, we have to decide it is a jump or not

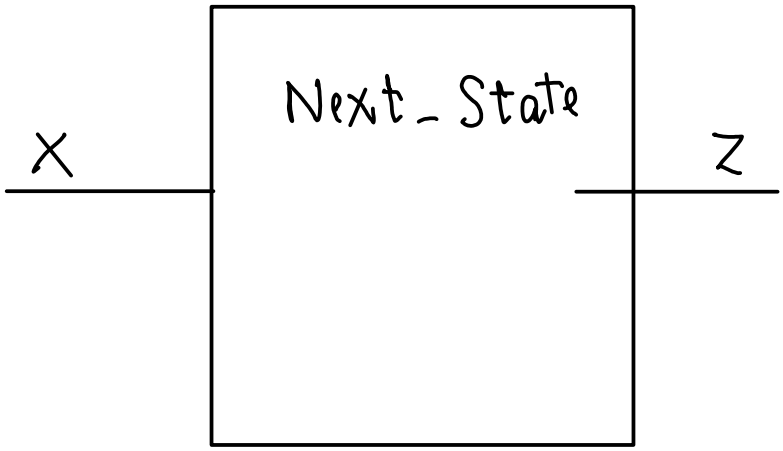
3.



FSM(finite state machine) takes input of current instruction, carry and zero flags, and data(T2) and decides all the control pins required to execute that instruction

It creates the instances of Next state and control pins

4.)

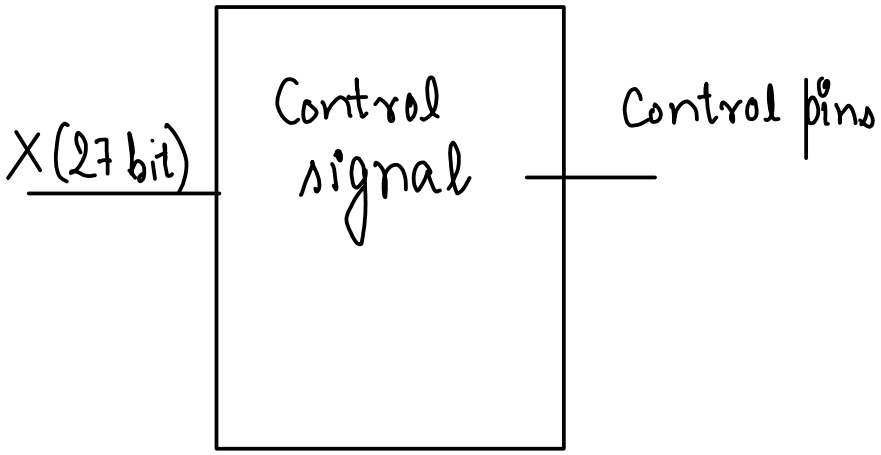


X is 39 downto 0 logic vecot, that contains all the info about the state,

X = IR | Carry | Zero | T2 | temp_z | curr state

It applies decoder on X and retuns the 5 bit next state

5.

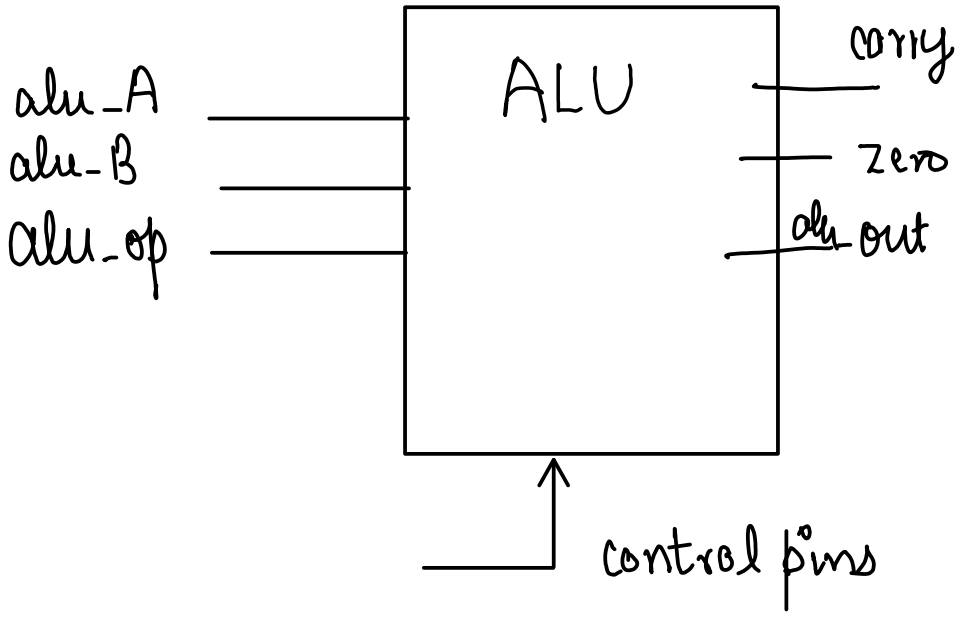


It takes 27 bit std logic vector and outputs the control pins corresponding to that

Format of X : Temp_z | T4 | current state | IR | Carry_reg | Zero_reg

Based upon the current state it assign the control pins their respective value

6.



ALU has 2 input signals, and alu_op that decides the operation that wer have to do.

Various control pins are there that decide whether we are doing normal addition,

, addition if control bit set, or are we adding 1 to PC , or some offset to PC etc

ALU: digital circuit that provides arithmetic and logic operations
Registers: small amount of storage available to the CPU, can be accessed very fast

Datapath:

Memory, registers, ALU, and communication buses. Each step (fetch, decode, execute) requires communication (data transfer) paths between memory, registers and ALU.

Bus: communication system that transfers data between components

like Memory ---(BUS)---> ALU

Types of busses:

1. Address Bus : Carry address
2. Data Bus: carry data
3. Control Bus: carry control signals
4. Power Bus: carry clock/power signals

Inshort a bus carry any information

In component1 ---(Bus)--component2
{component1, bus, component2} = DATA PATH

Now to decide which component to read/write from, we need control signals.
So flow of datapath is decided by control signals.

Control: set up dataflow directions on buses

eg select ALU and memory functions.
Control signals are generated by a control unit consisting of finite-state machines

Instructions and their States

S_1

```
PC → Mem_a/alua
Mem_d → ir
+1 → alu_b
alu1_out → PC
```

Fetch Instruction

S_2

```
ir 9-11 → rf_a1
ir 6-8 → rf_a2
rf_d1 → t1
rf_d2 → t2
```

Read operands from RF

S_3

```
t1 → alu_A
t2 → alu_B
alu_out → t1
```

ALU operations

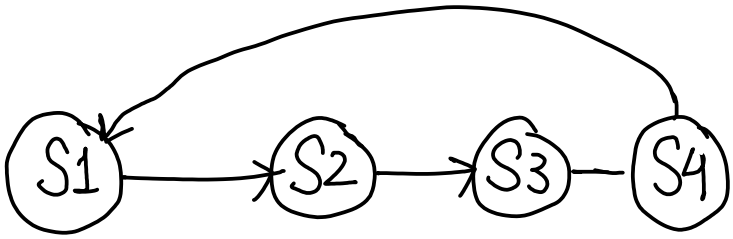
S_4

```
t1 → rf_a3
ir 3-5 → rf_a3
if (rf_a3 = R7) {t1 → PC}
else PC → R7
```

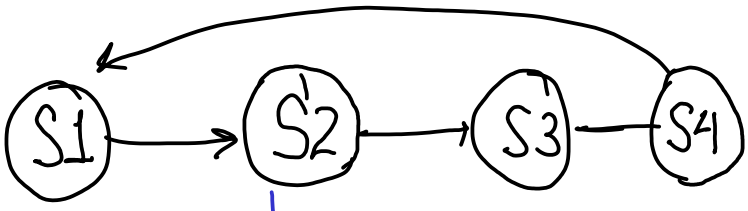
Write back

Instr

ADD(R)

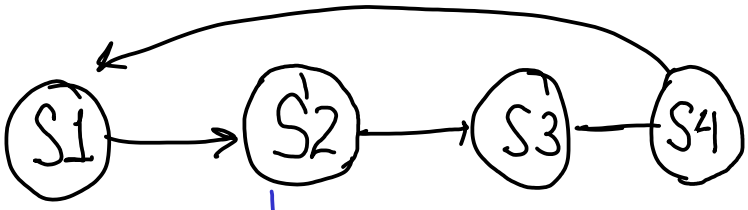


ADC(R)



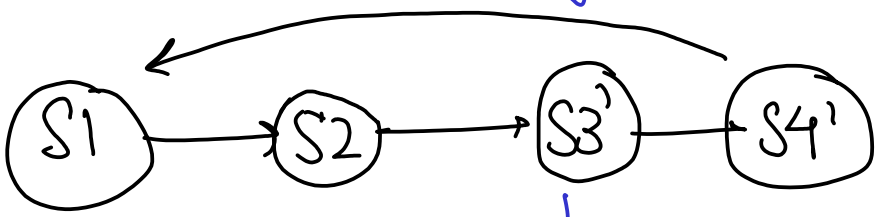
↳ if carry flag set then PC→R7

ADZ(R)



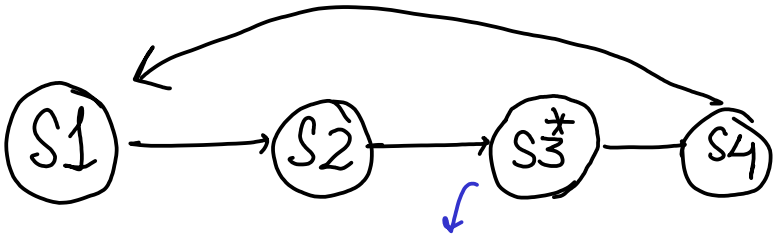
↳ if zero flag set then PC→R7

ADI(I)



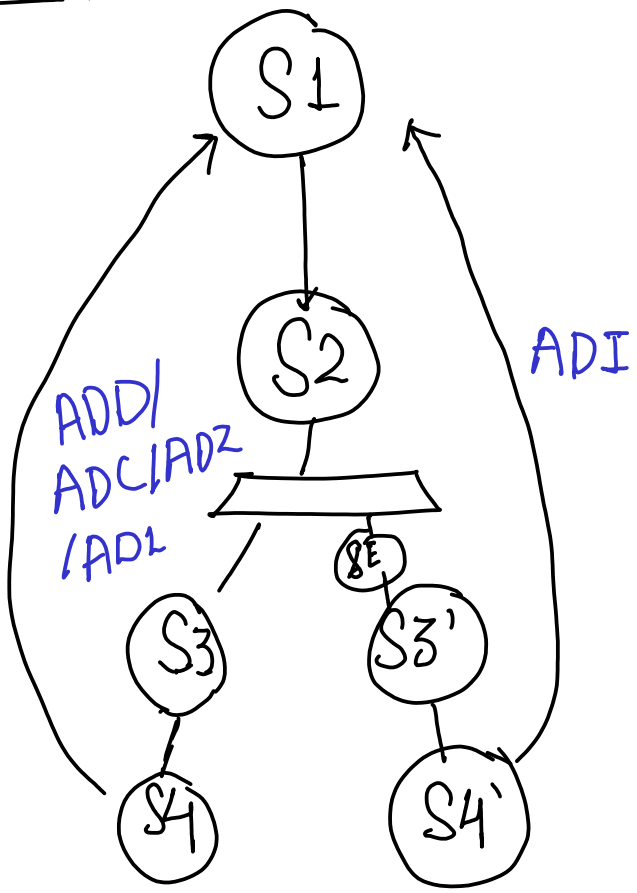
↓ use SE to make offset 16 bit

ADL(R)



↓ Shift RB and add in alu

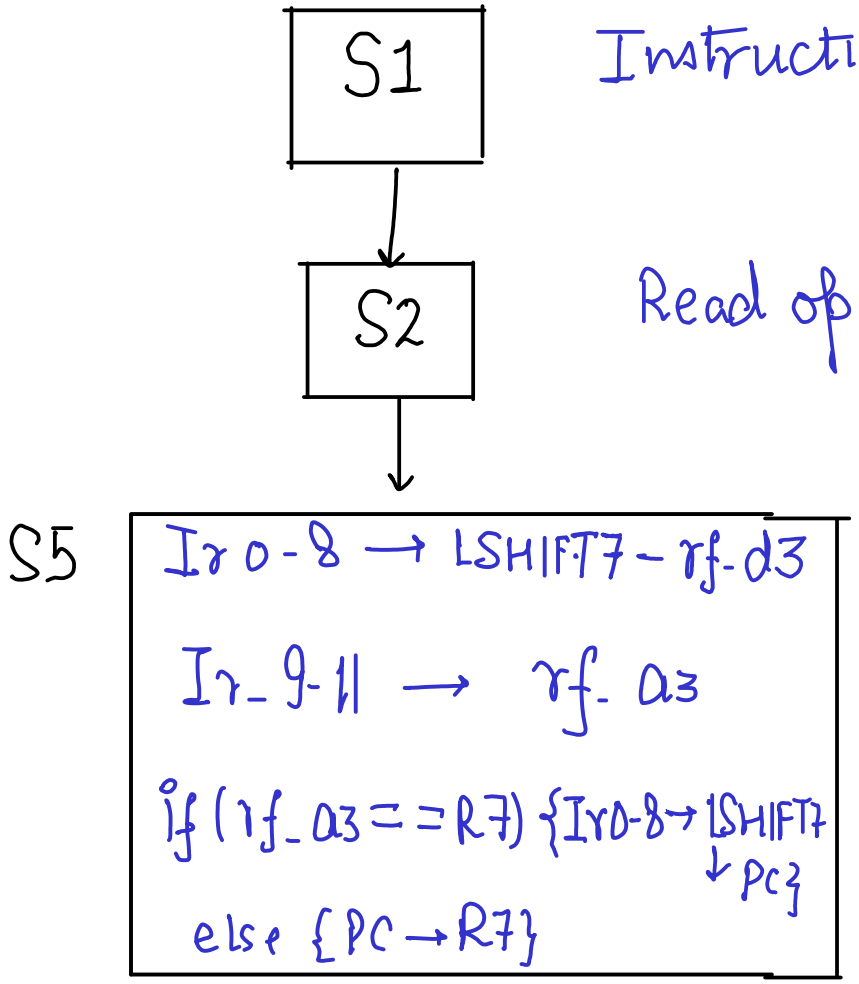
ADD FSM



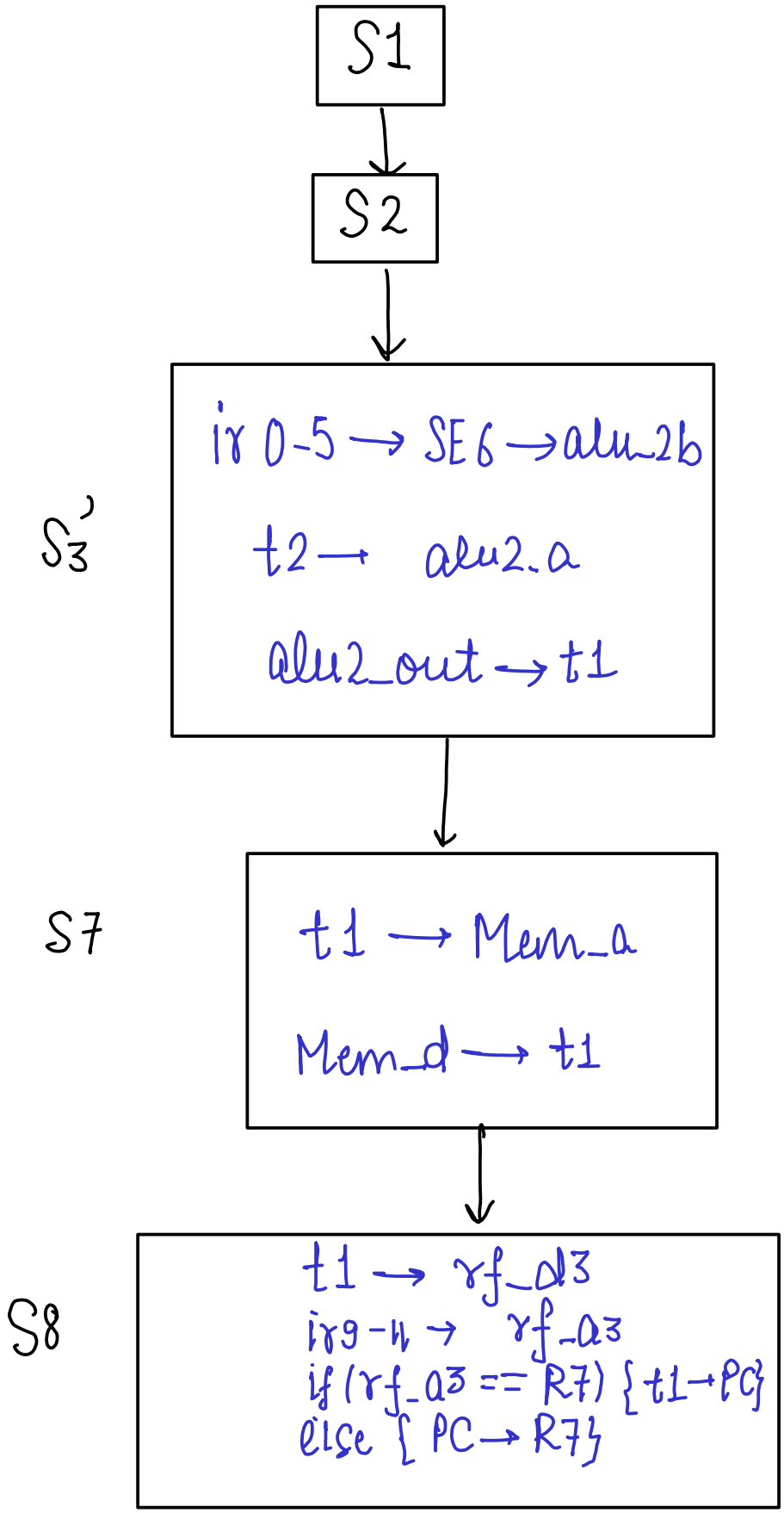
Similar is extended to NAND instructions just the alu opr^n control pins are different.

* assign rf-a3 as 111 for loading PC

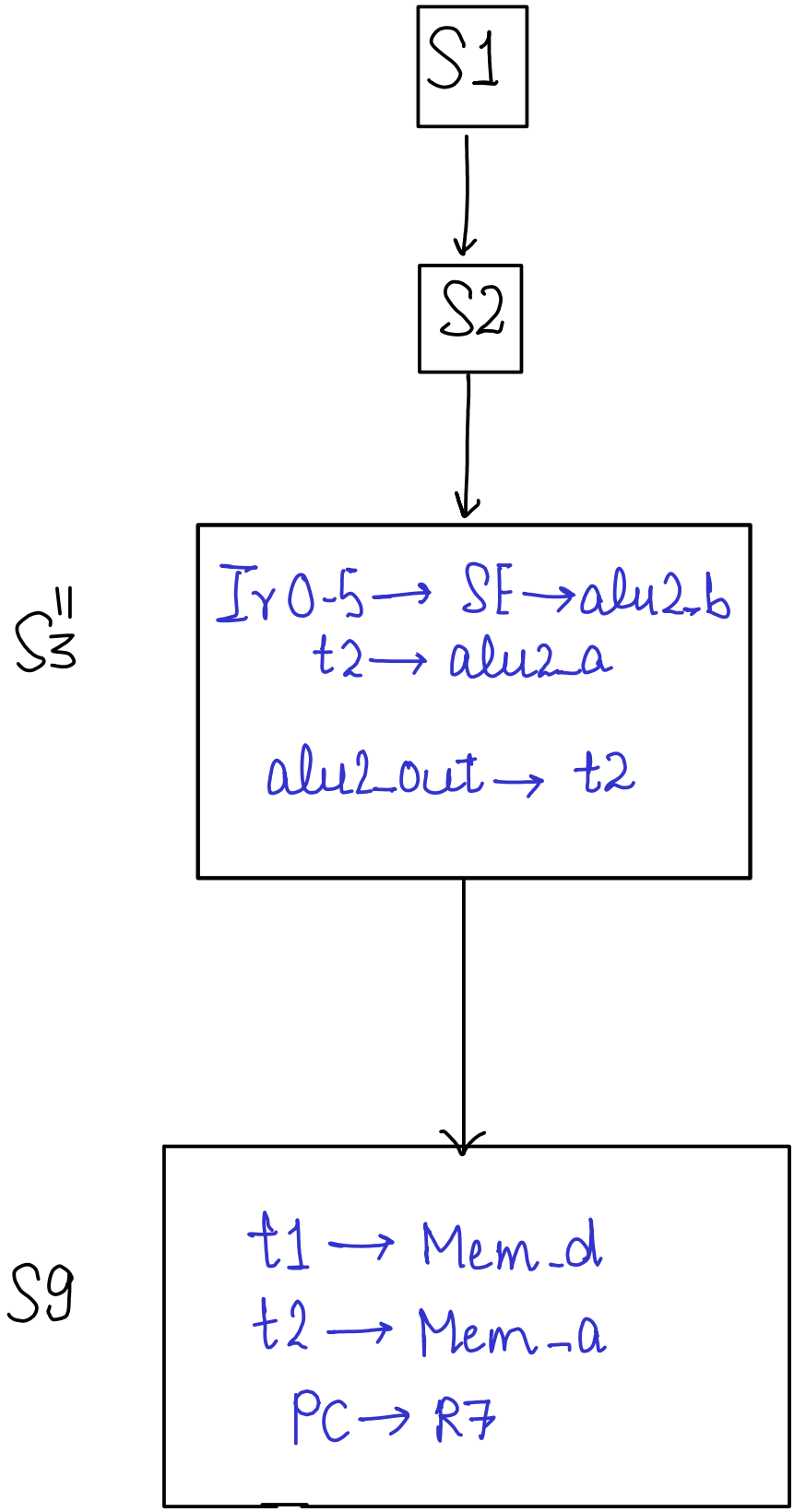
LHI


S1
Instruction fetch
S2
Read operands
S5
Ir 0-8 → LSHIFT7 - rf_d3
Ir_9-11 → rf_a3
if (rf_a3 == R7) {Ir0-8 → LSHIFT7 ↓ PC}
else {PC → R7}

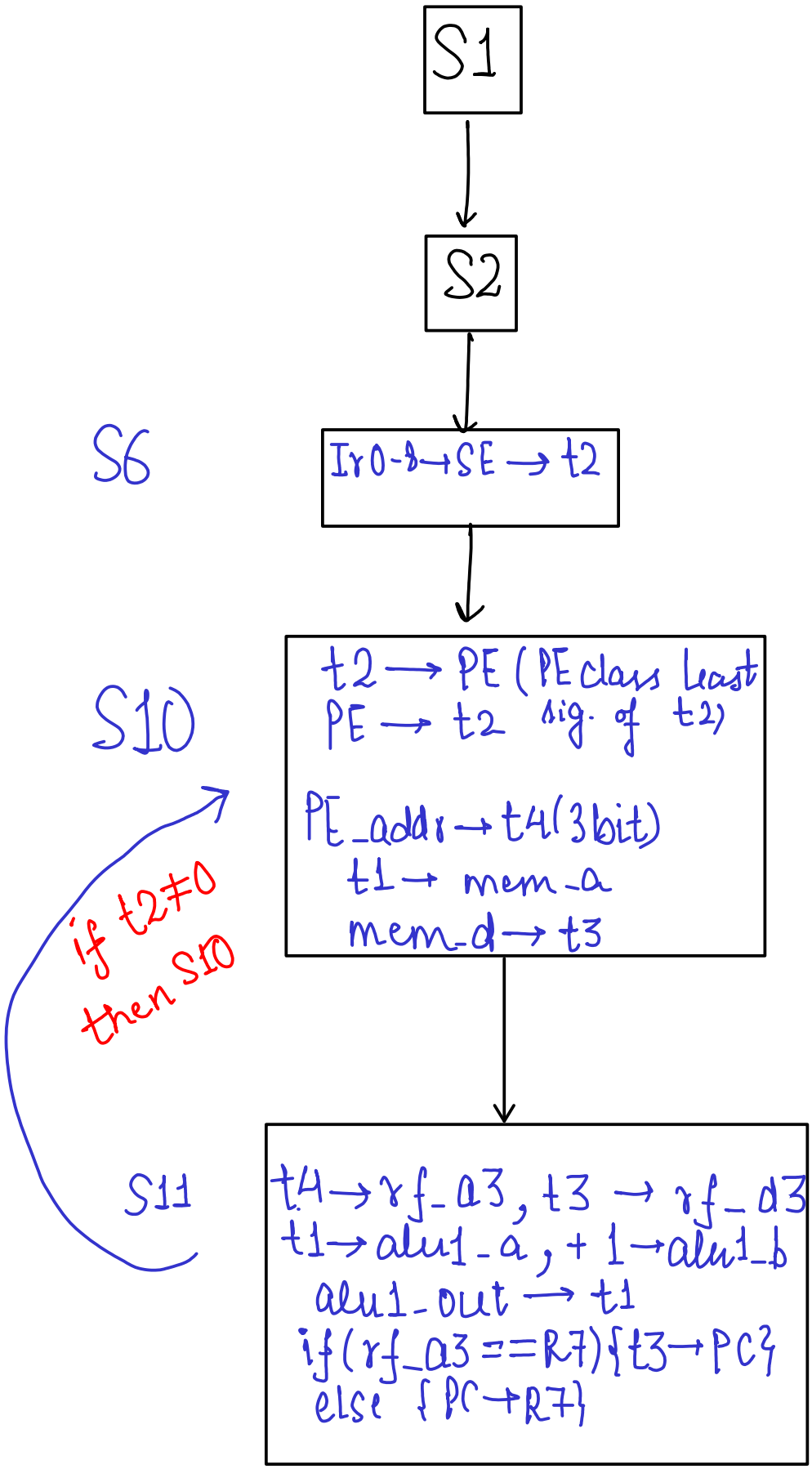
LW

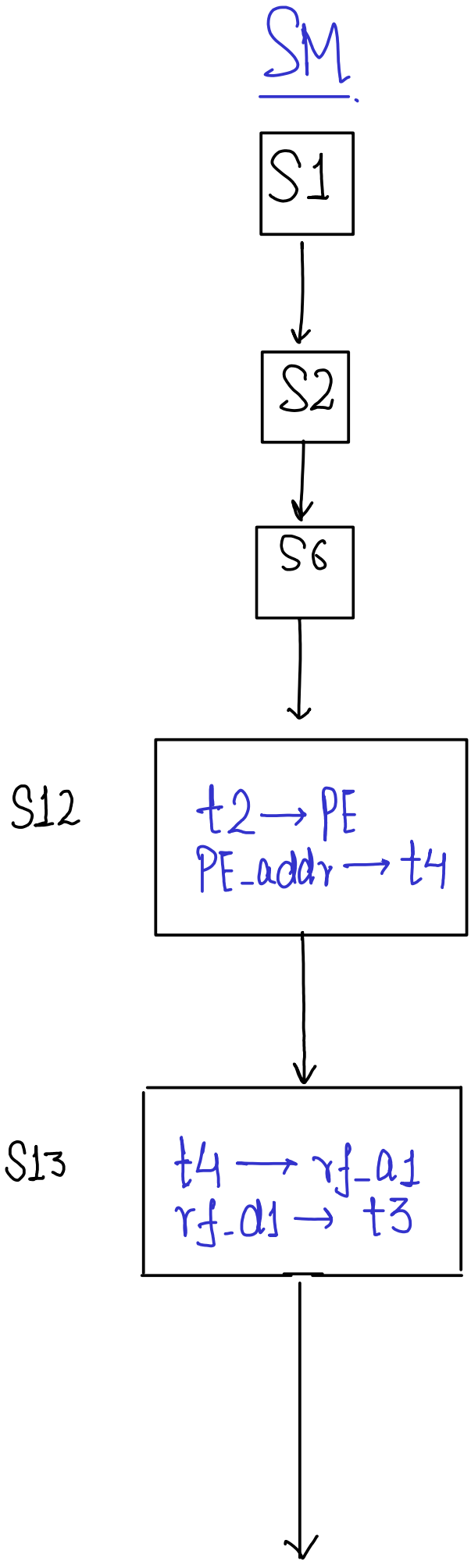

S1
S2
ir 0-5 → SE6 → alu_2b
t2 → alu2_a
alu2_out → t1
S3'
t1 → Mem_a
Mem_d → t1
S7
t1 → rf_a3
ir 9-11 → rf_a3
if (rf_a3 == R7) {t1 → PC}
else { PC → R7}
S8

SW

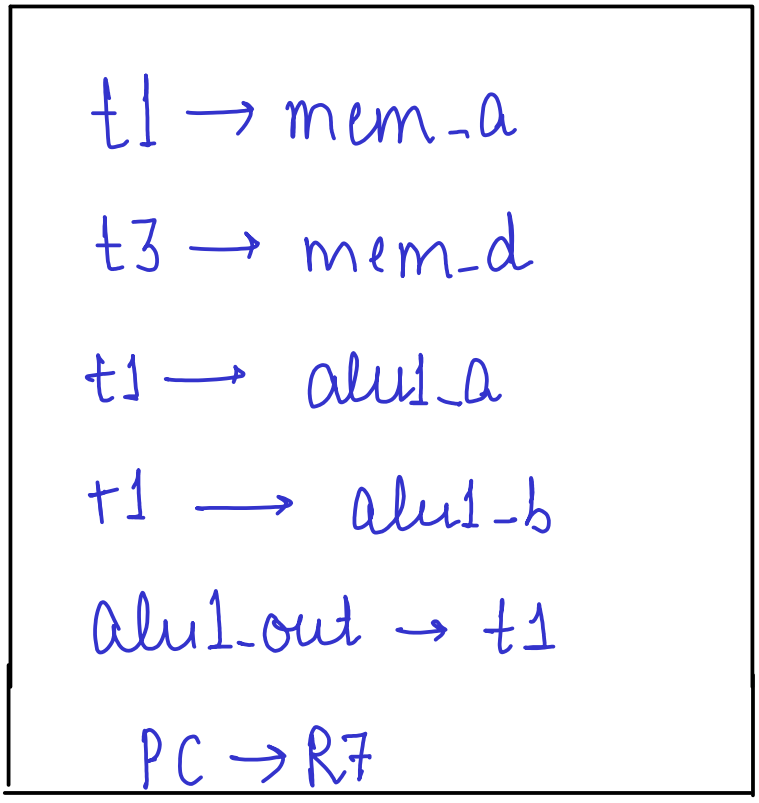

S1
S2
S3
Ir 0-5 → SE → alu2_b
t2 → alu2_a
alu2_out → t2
S9
t1 → Mem_d
t2 → Mem_a
PC → R7

LM


S1
S2
S6
Ir 0-8 → SE → t2
S10
t2 → PE (PE class least sig. of t2)
PE → t2
PE_addr → t4 (3 bit)
t1 → mem_a
mem_d → t3
if t2≠0 then S10
S11
t4 → rf_a3, t3 → rf_d3
t1 → alu1_a, + 1 → alu1_b
alu1_out → t1
if (rf_a3 == R7) {t3 → PC}
else {PC → R7}


SM
S1
S2
S6
S12
t2 → PE
PE_addr → t4
S13
t4 → rf_a1
rf_d1 → t3

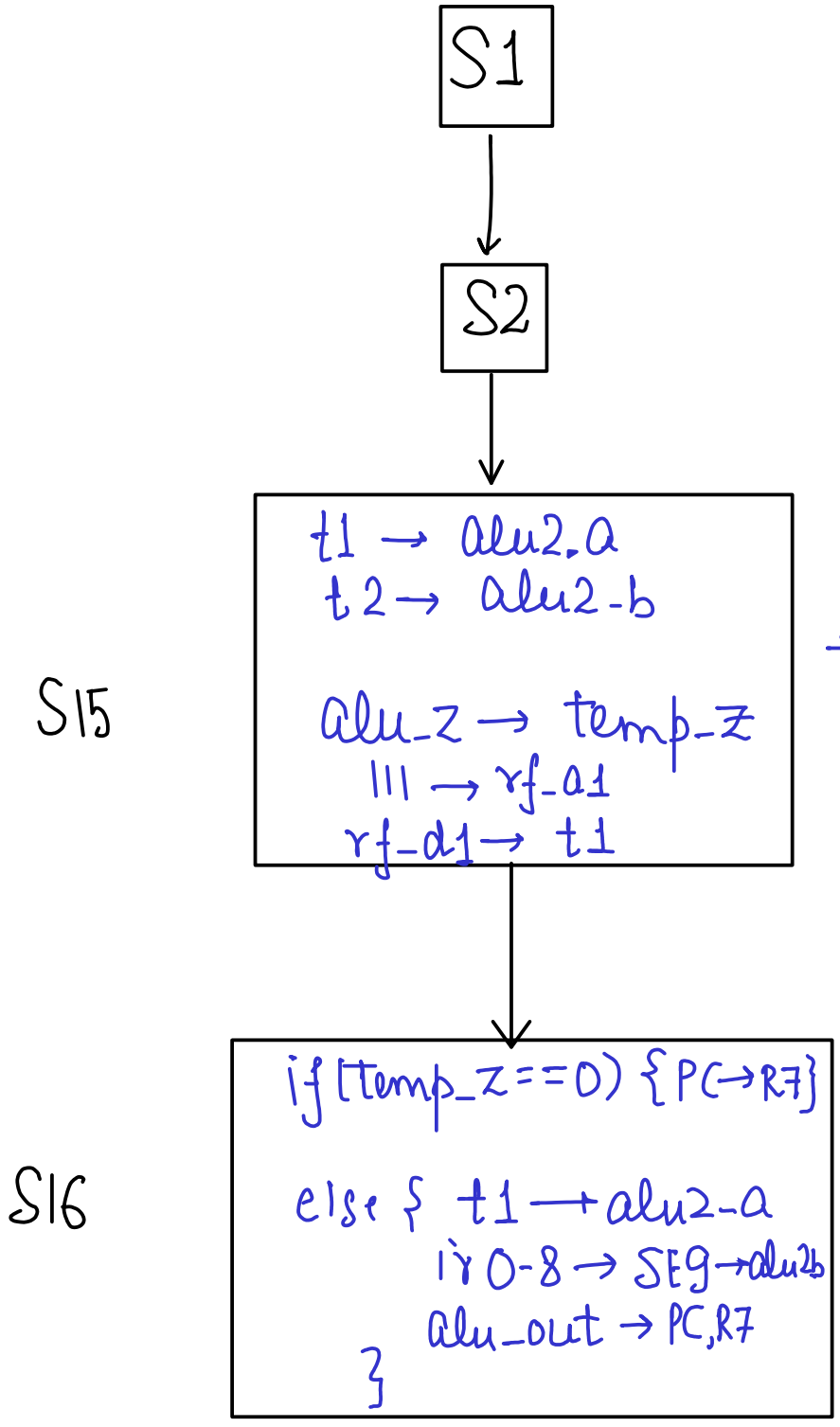
S14



→ loopback (S12)

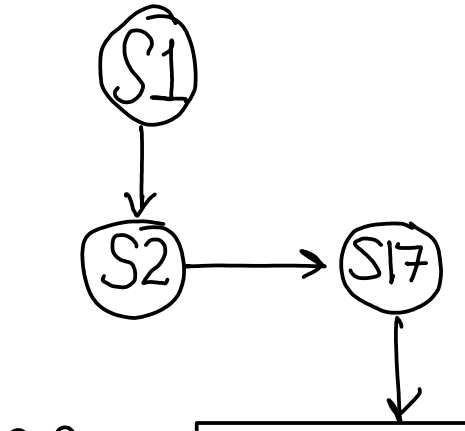
if t2=0 then S1

BEQ



Alu Subtract to compare and check if result is zero (ie if temp_z = 1)

JAL



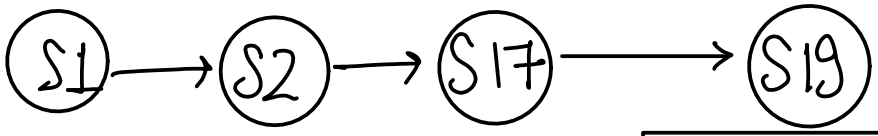
111 → rf_a1

rf_d1 → t3

S18

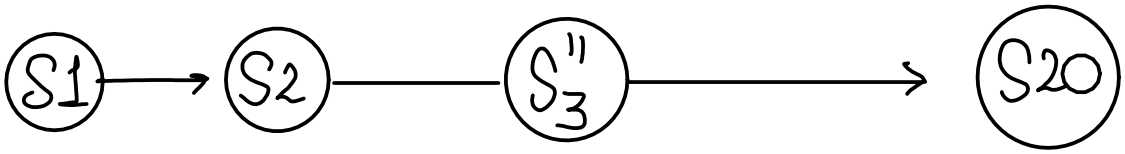
IR0-8 → SEG → alu2a
111 → rf_a1, rf_d1 → t3
t3 → alu2_b, rf_d3
alu2_out → PC, R7
ir9-11 → rf_a3

JLR



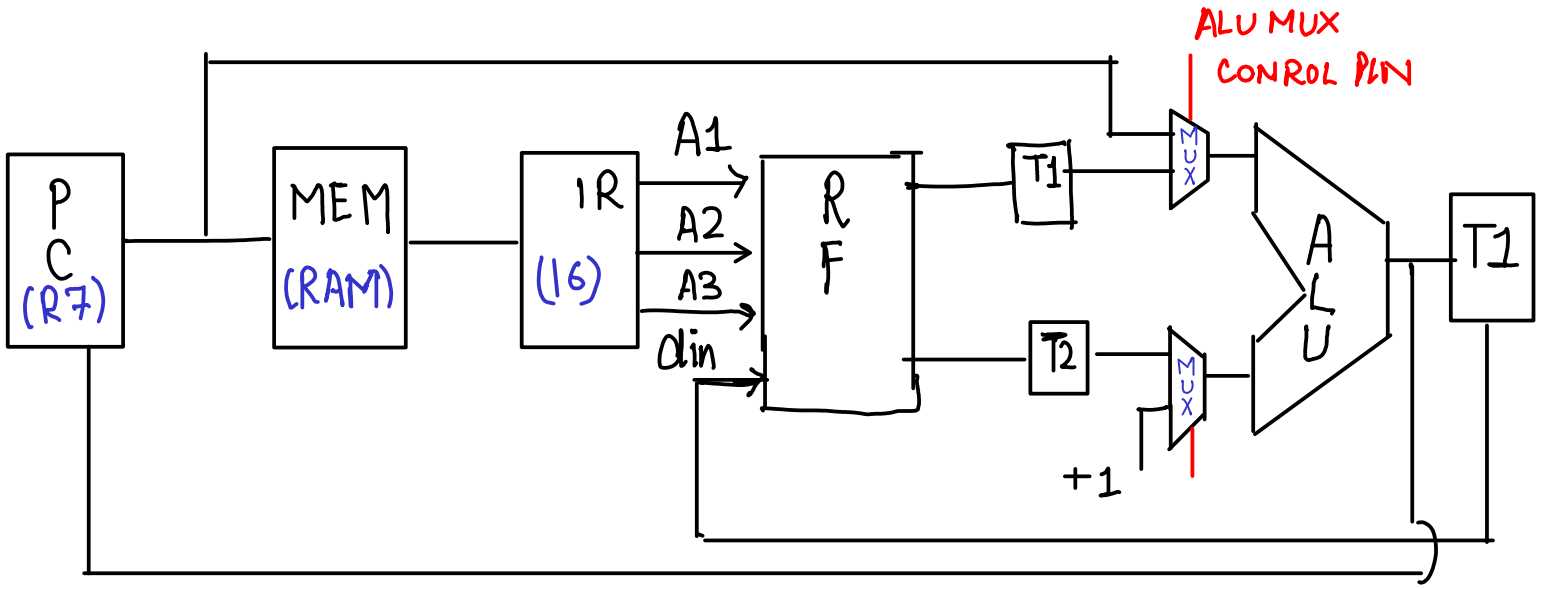
t3 → rf_d3
ir9-11 → rf_a3
t2 → PC, R7

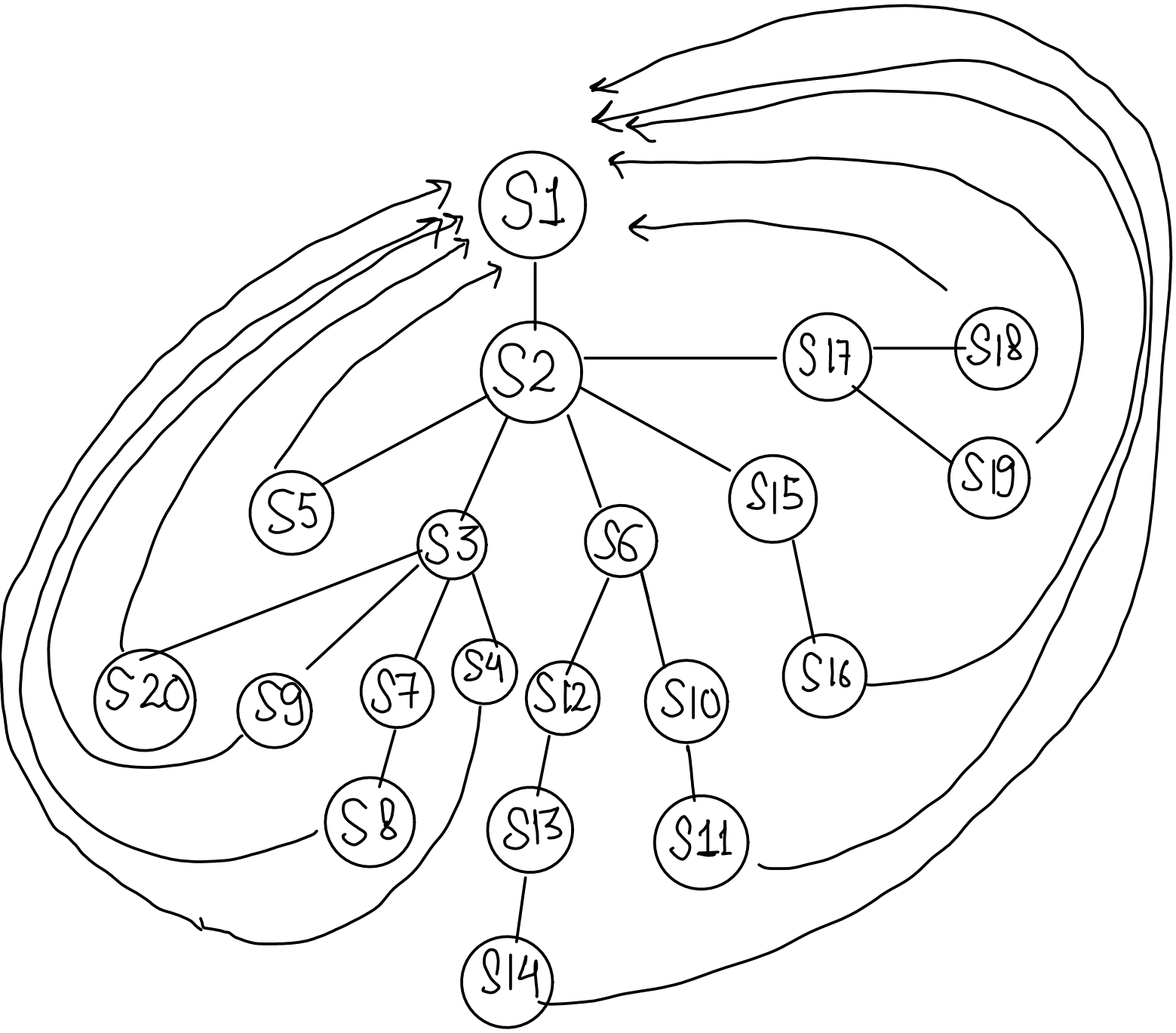
JRI



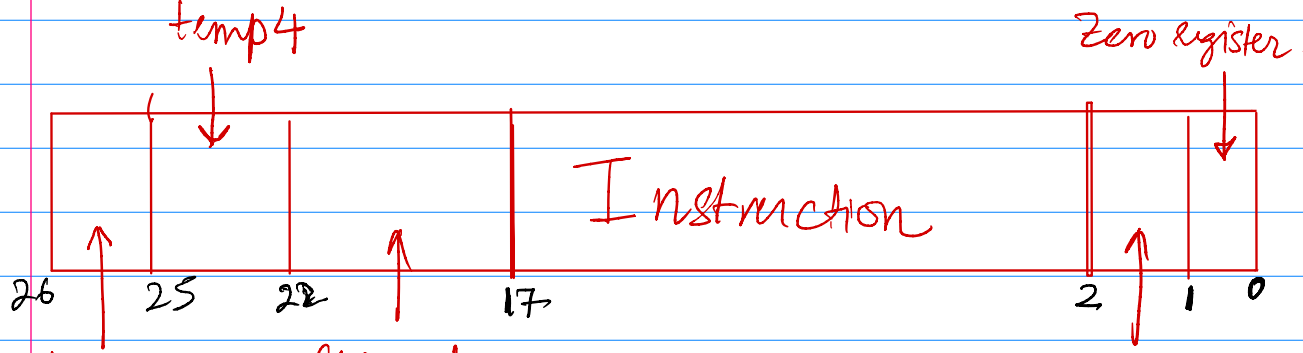
Ir0-8 → SEG → alu1_b
t1 → alu1_a
alu1_out → t1

t1 → PC
PC → R7


ALU MUX
CONROL PIN
P
C
(R7)
MEM
(RAM)
IR
(16)
A1
A2
A3
Din
R
F
T1
T2
MUX
MUX
A
L
U
T1
+1


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

Control signal input



(temporary Zero value)

(carry register)

Control signal Output

- ALU_op : ALU operation code.
- alua_mux : decides from where ALU_a takes input
- alub_mux : decides from where ALU_b takes input
- register_en : register file (active / not)
- reg write mux : decides who writes in R7
- reg addr1 mux : rf_a1 mux
- reg addr 3 mux : rf_d3 mux
- reg dest 3 mux : rf_d3 mux
- mem wr bar : enables memory write
- memory addr mux : mem_a mux (decides input in mem_a)
- memory dest mux : mem_d mux
- Encode t1 : Active high (t1 register)
- Enwde t2 : Active high (t2 register)

- encode t3: active high (T3 register)
- encode t4: active high (T4 register)
- pcencode: enables PC write
- instregencode: enables instruction register.
- temp Zencode: enables writing in temp Z
- flagC encode: enables writing in Carry register.
- flag Z encode: enables writing in Zero register.
- tem1 multi: decides input for t1
- tem 2 multi: decides input for t2
- tem3 multi: decides input for t3
- progcount multi: decides who writes in PC

Given a State and few control pins; control signal sets various multiplexers based on requirements of the particular stage.