PROJECT DESIGN REPORT

Designing a multi-cycle processor,

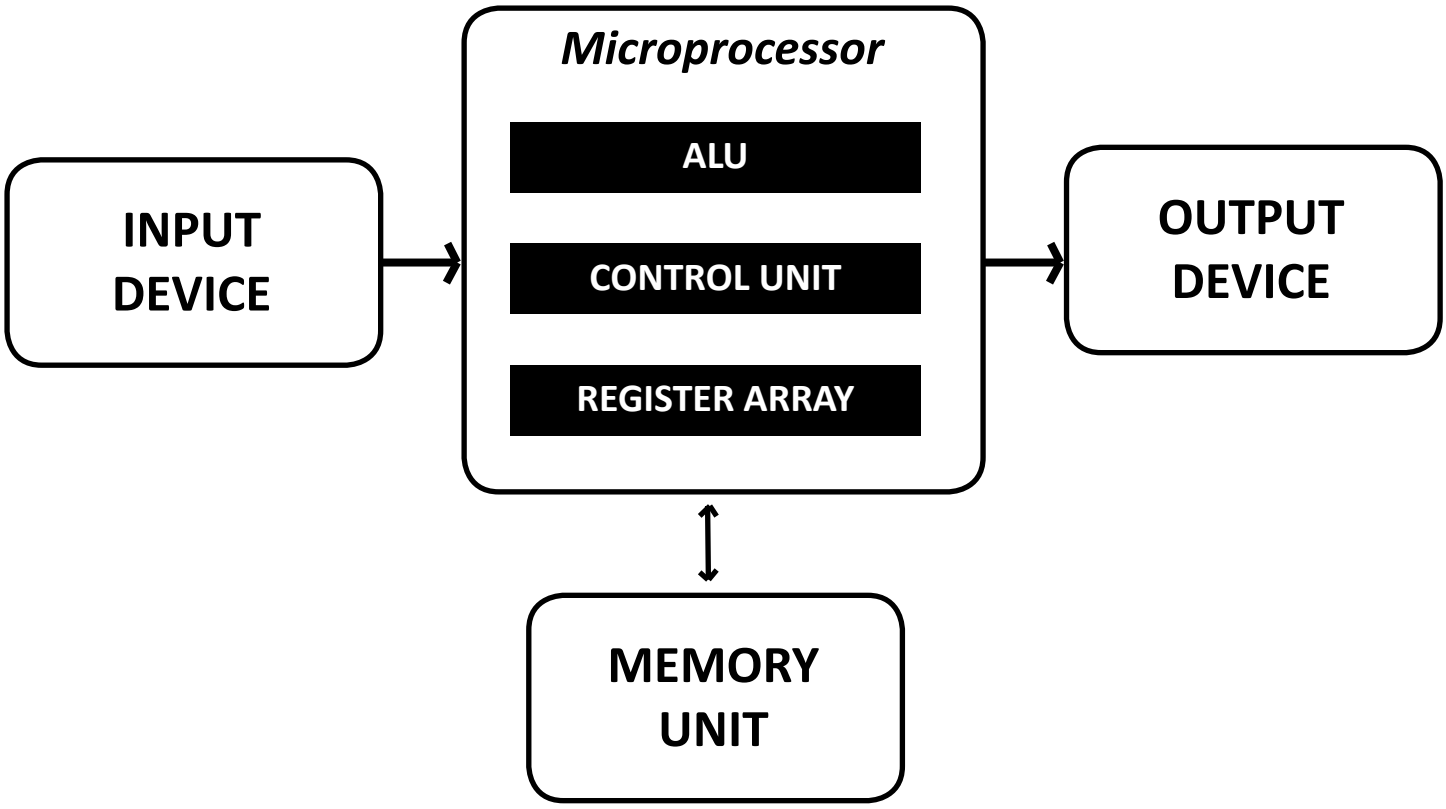
IITB PROC

- **Aryan Lall , 17D070053**
- **Vaidehi Patil , 170110012**
- **Ritij Saini , 17D070027**
- **Sonal Kumar , 17D070045**

AIM

The aim of this project is to use VHDL to implement a multi-cycle processor which is an 8-register, 16-bit computer system which uses point-to-point communication infrastructure (for e.g. A 16-bit very simple computer developed for the teaching purpose) .

The IITB-Proc is general enough to solve complex problems.The architecture allows predicated instruction execution and multiple load and store execution. There are three machine-code instruction formats (R, I, and J type) and a total of 14 instructions.



OVERVIEW

- ***DESIGN***

There are total 14 operations which we have to implement. From these 14 operations, 6 operations include addition and bitwise NAND operations. There are carry and zero registers flags whose value can be changed according to the operations .The system takes clock and reset state as input .Then we fetch instruction register from the memory for the corresponding instruction pointer. Then the operation code is extracted from the instruction register and corrosponding operation is performed which includes ADD , NAND , BEQ etc.

For performing basic operations we have an 16 bit-ALU and various control signals for reading and writing memory and register file . We first figured out the Finite state machine which performs the desired operations in particular order .The instruction pointer is updated at the end of every operation . For updating the registers we have a dedicated register file similarly for updating the memory we have a corresponding memory block .
We have three instruction registers R, J, I as shown below ;-

RType Instruction format

Opcode (4 bit)	Register A (RA) (3 bit)	Register B (RB) (3-bit)	Register C (RC) (3-bit)	Unused (1 bit)	Condition (CZ) (2 bit)

I Type Instruction format

Opcode (4 bit)	Register A (RA) (3 bit)	Register C (RC) (3-bit)	Immediate (6 bits signed)

JType Instruction format

Opcode (4 bit)	Register A (RA) (3 bit)	Immediate (9 bits signed)

• *INSTRUCTION IMPLEMENTATION*

Every instruction set can be implmented in clock cycles which can be described as follows:

• Instruction fetch (IF):
Send the program counter (PC) to memory and fetch the current instruction from memory. Update the PC to the next sequential PC by adding 1 to the PC.

• Instruction decode/register fetch (ID):
Decode the instruction and read the registers corresponding to register source specifiers from the register file. Do the equality test on the registers as they are read, for a possible branch. Sign-extend the offset field of the instruction in case it is needed. Compute the possible branch target address by adding the sign-extended offset to the incremented PC. In an aggressive implementation, which we explore later, the branch can be completed at the end of this stage, by storing the branch-target address into the PC, if the condition test yielded true. Decoding is done in parallel with reading registers, which is possible because the register specifiers are at a fixed location in the architecture.Because the immediate portion of an instruction is also located in an identical place, the sign-extended immediate is also calclated during this cycle in case it is needed.

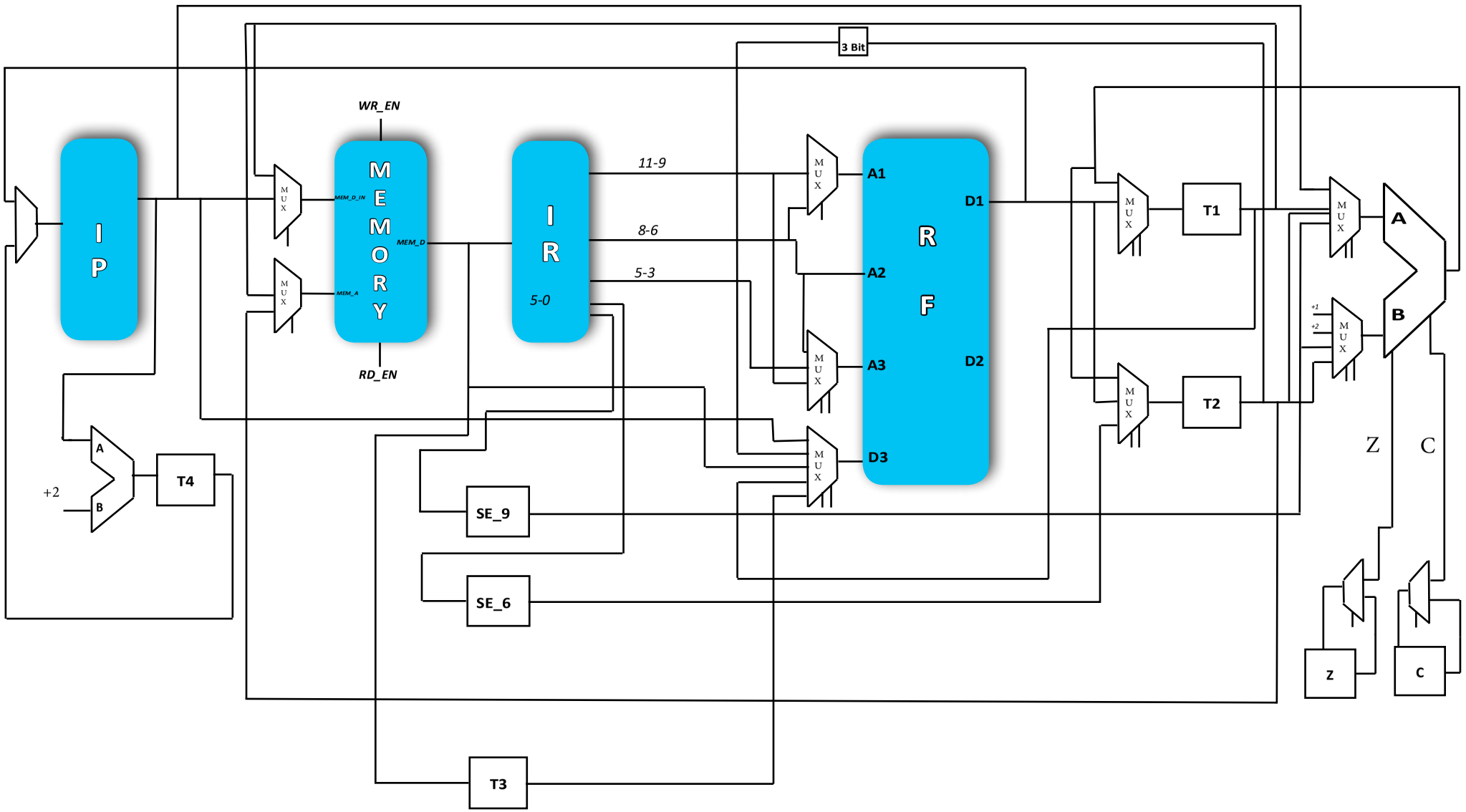
• Execution/effective address cycle (EX):
The ALU operates on the operands prepared in the prior cycle, performing one of three functions depending on the instruction type.
– Memory reference: The ALU adds the base register and the offset to form the effective address.
– Register-Register ALU instruction: The ALU performs the operation specified by the ALU opcode on the values read from the register file.
– Register-Immediate ALU instruction: The ALU performs the operation specified by the ALU opcode on the first value read from the register file and the sign-extended immediate.

• Memory access (MEM):
If the instruction is a load, memory does a read using the effective address computed in the previous cycle. If it is a store, then the memory writes the data from the second register read from the register file using the effective address.

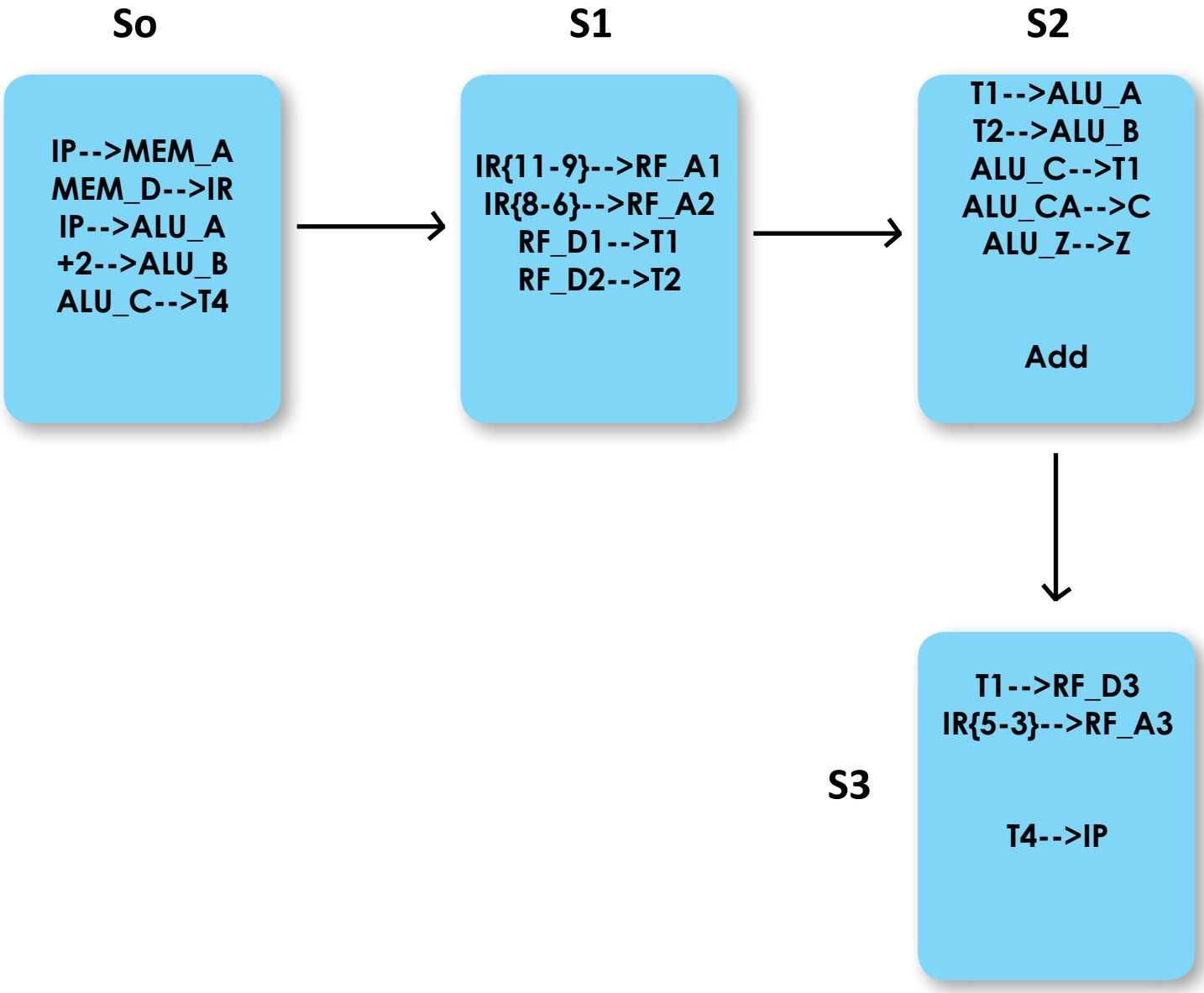
• Write-back cycle (WB):
Write the result into the register file, whether it comes from the memory system (for a load) or from the ALU (for an ALU instruction).

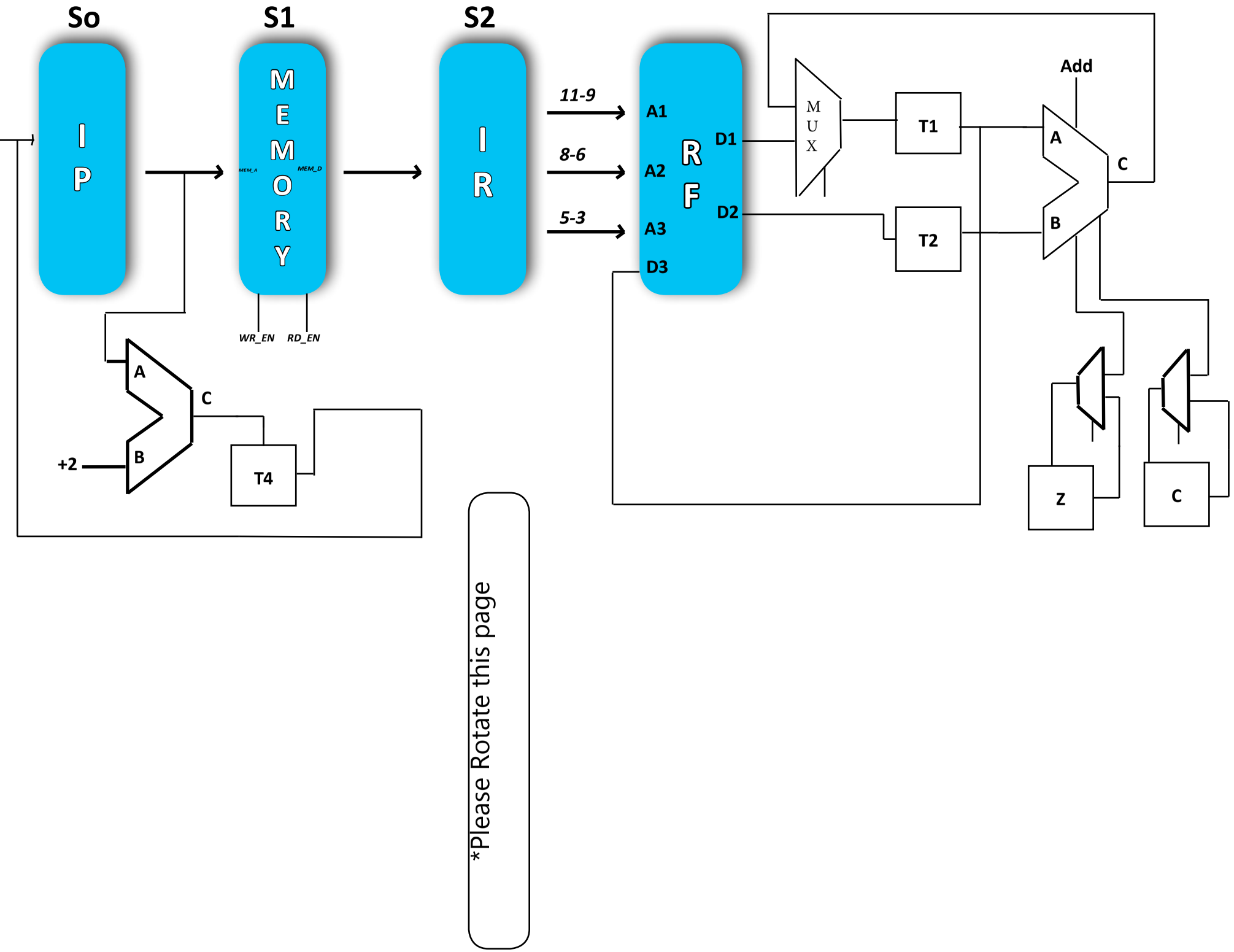
GENERAL DATA PATH is shown on the next page .

GENERAL DATA PATH



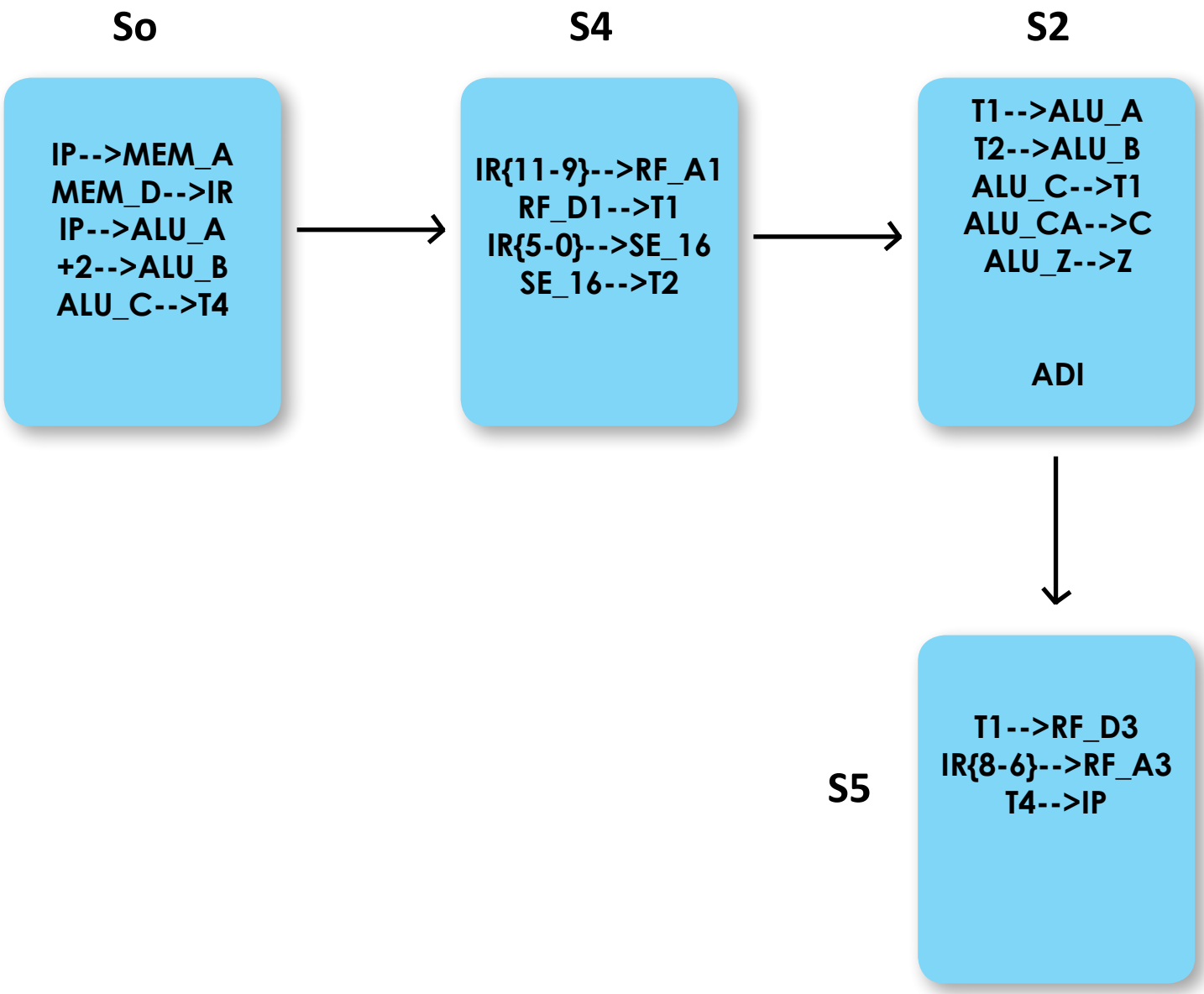
ADD , ADC ,ADZ


So
IP-->MEM_A
MEM_D-->IR
IP-->ALU_A
+2-->ALU_B
ALU_C-->T4
S1
IR{11-9}-->RF_A1
IR{8-6}-->RF_A2
RF_D1-->T1
RF_D2-->T2
S2
T1-->ALU_A
T2-->ALU_B
ALU_C-->T1
ALU_CA-->C
ALU_Z-->Z
Add
S3
T1-->RF_D3
IR{5-3}-->RF_A3
T4-->IP

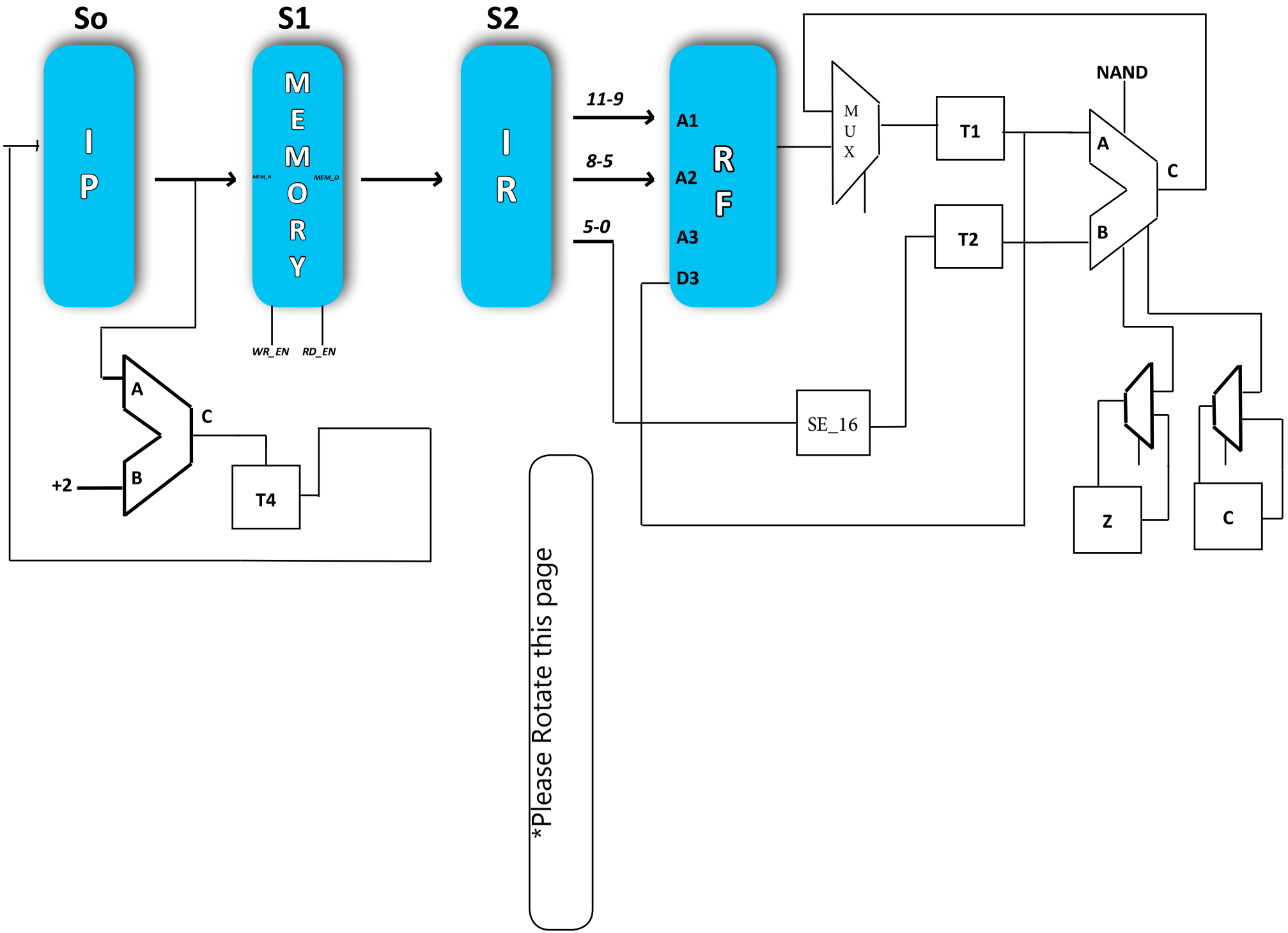


*Please Rotate this page

ADI

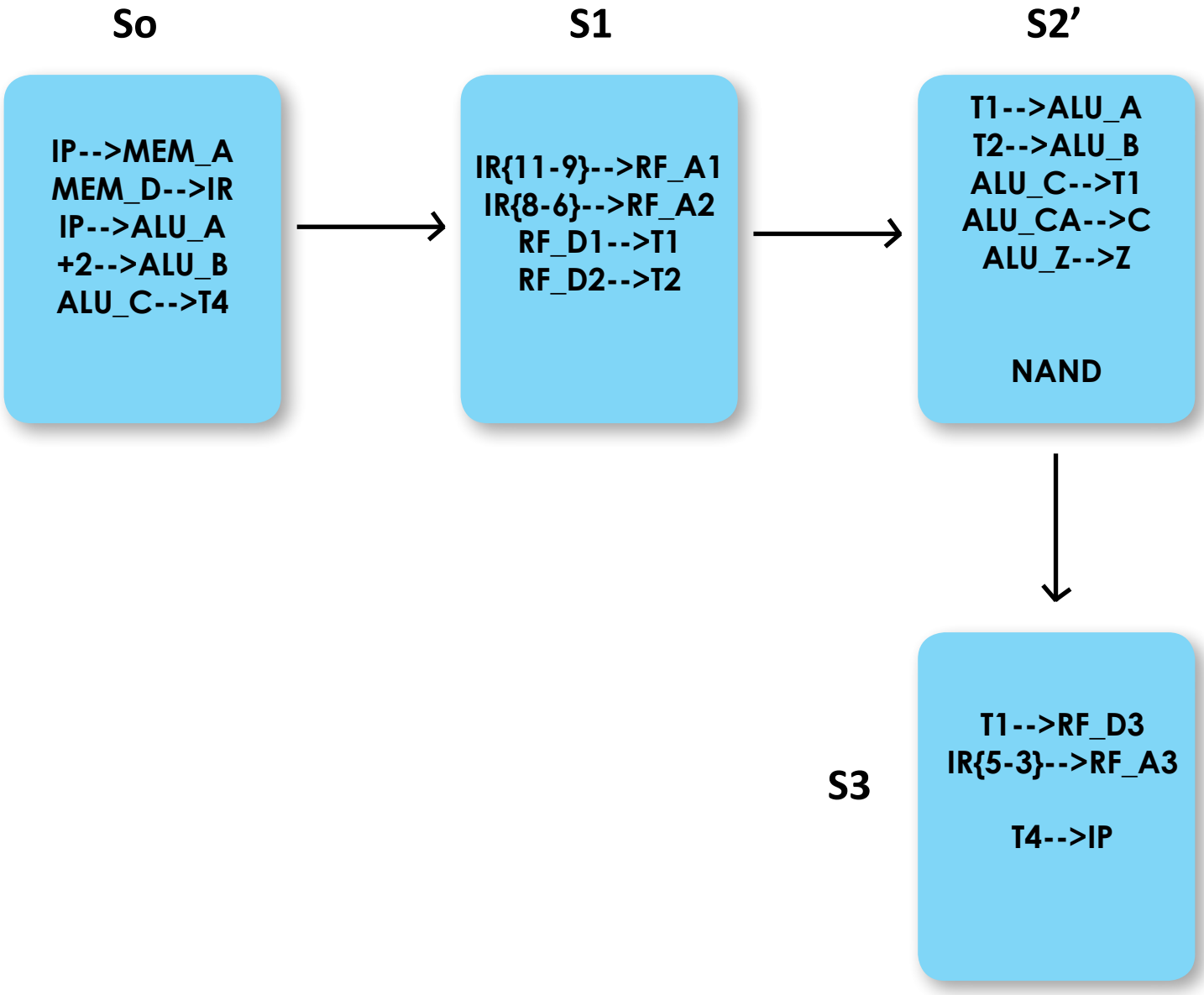

So
IP-->MEM_A
MEM_D-->IR
IP-->ALU_A
+2-->ALU_B
ALU_C-->T4
S4
IR{11-9}-->RF_A1
RF_D1-->T1
IR{5-0}-->SE_16
SE_16-->T2
S2
T1-->ALU_A
T2-->ALU_B
ALU_C-->T1
ALU_CA-->C
ALU_Z-->Z
ADI
S5
T1-->RF_D3
IR{8-6}-->RF_A3
T4-->IP

P.T.O

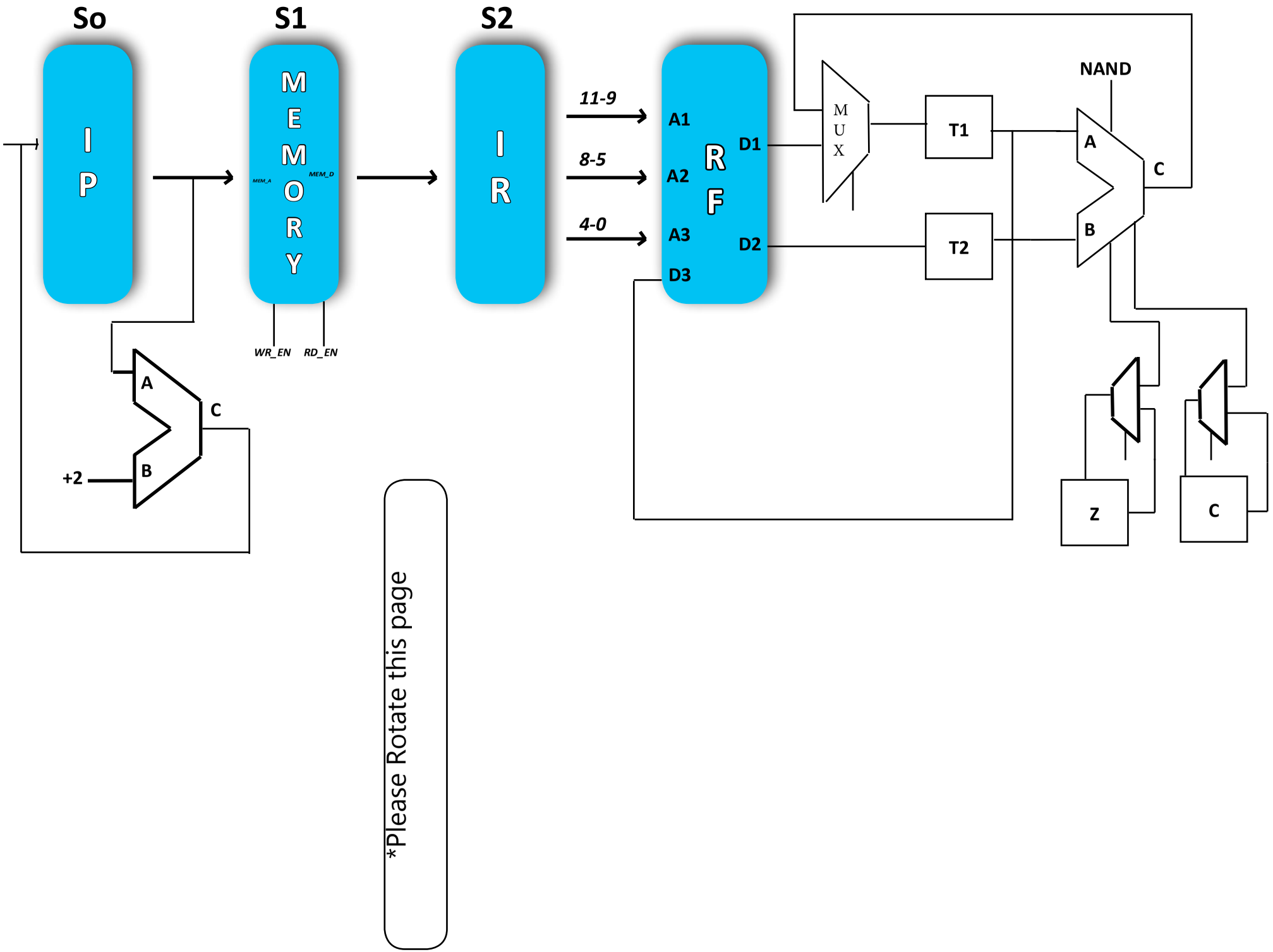


*Please Rotate this page

NDU , NDC , NDZ

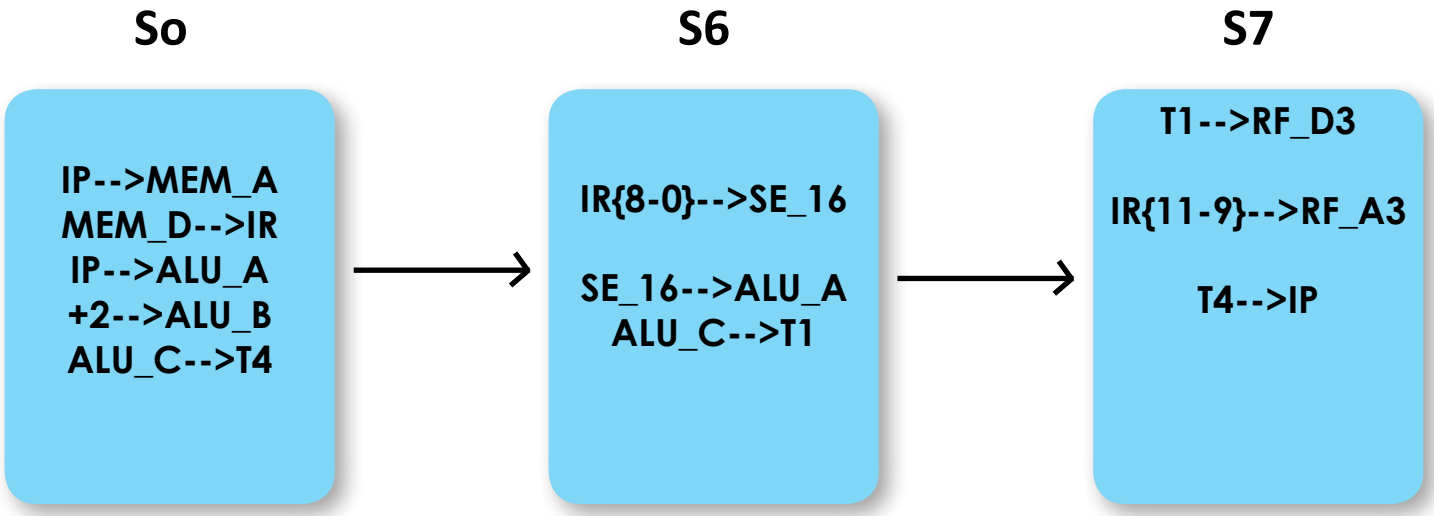


P.T.O

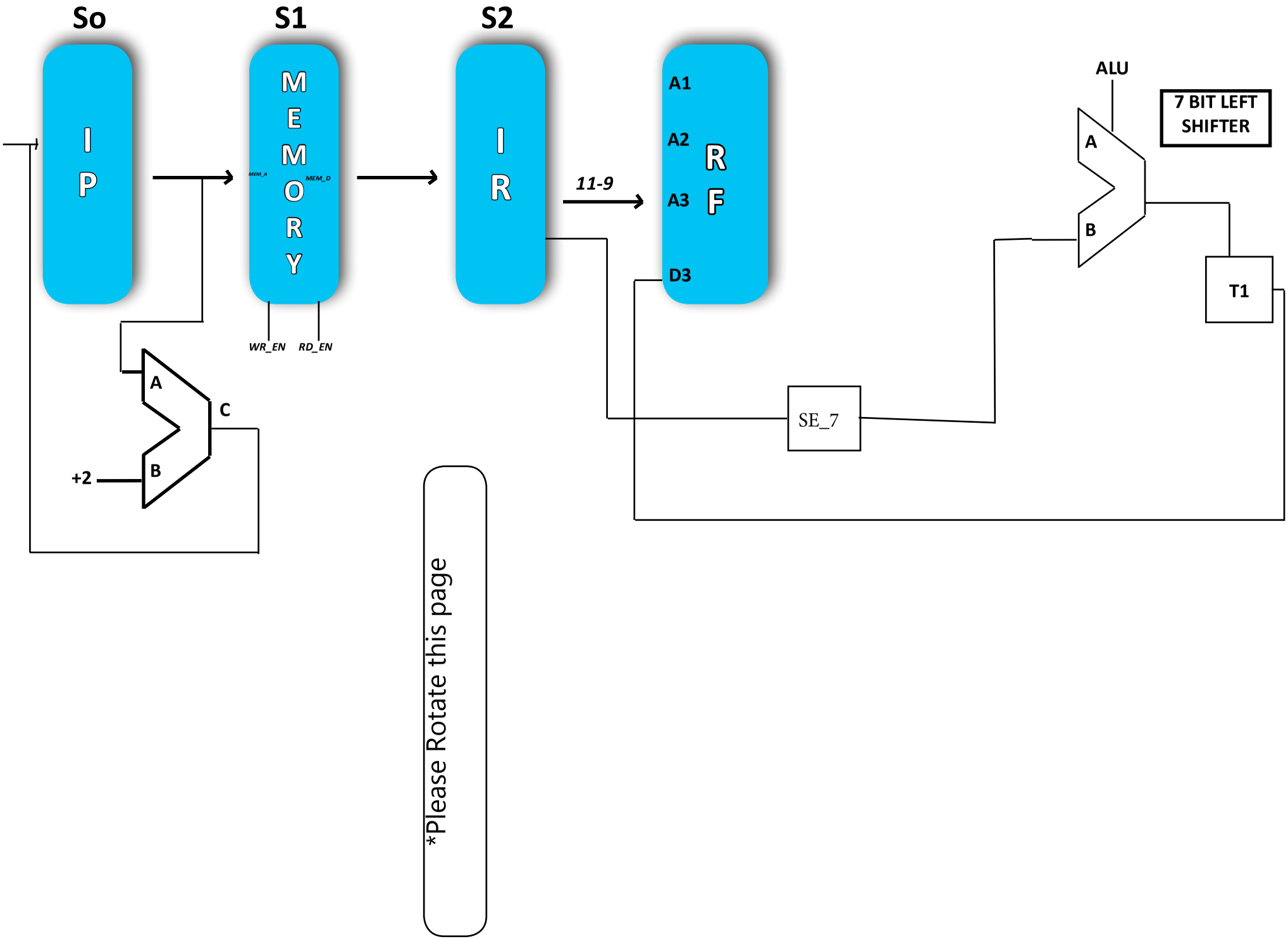


*Please Rotate this page

LHI

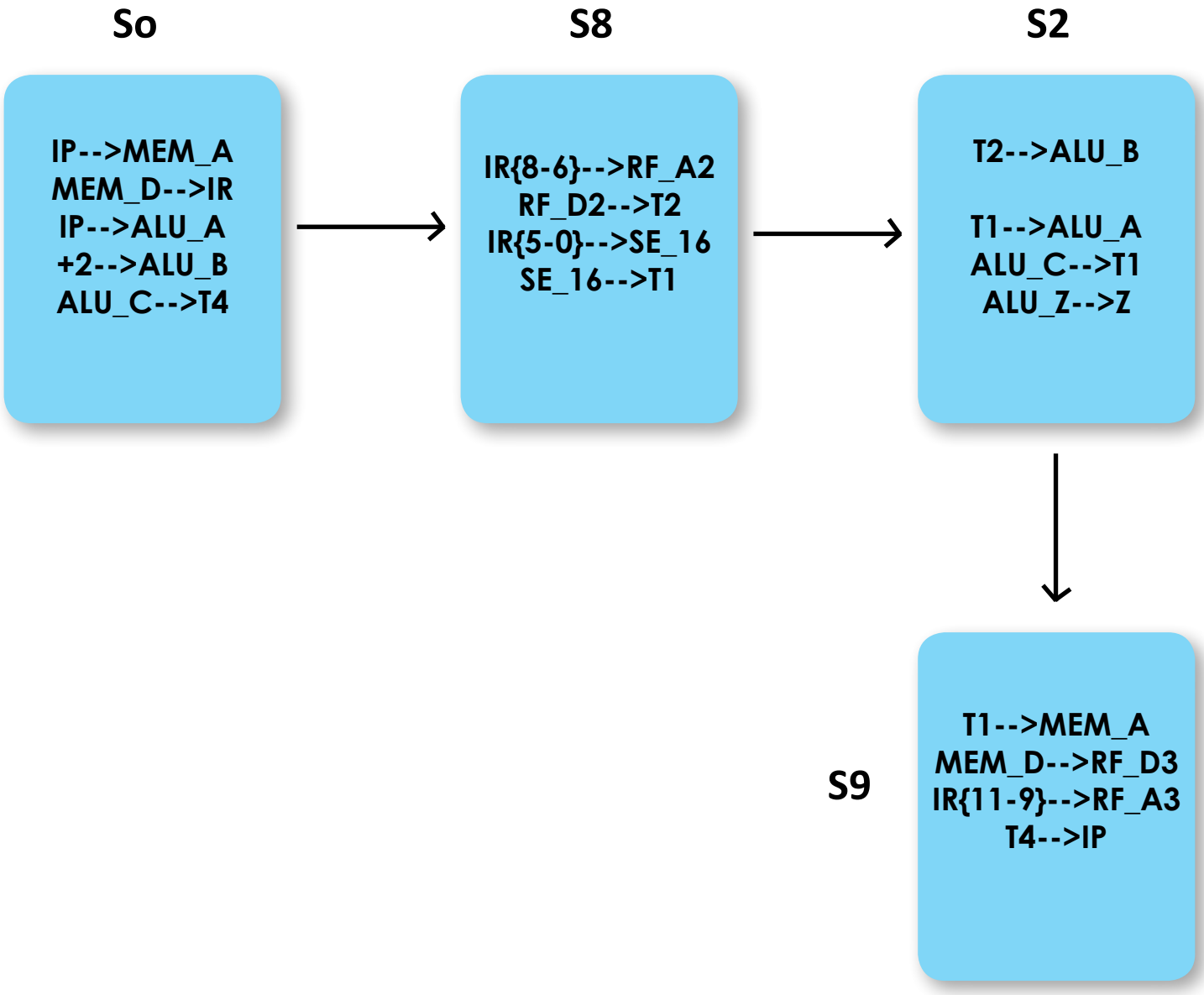

So
IP-->MEM_A
MEM_D-->IR
IP-->ALU_A
+2-->ALU_B
ALU_C-->T4
S6
IR{8-0}-->SE_16
SE_16-->ALU_A
ALU_C-->T1
S7
T1-->RF_D3
IR{11-9}-->RF_A3
T4-->IP

P.T.O

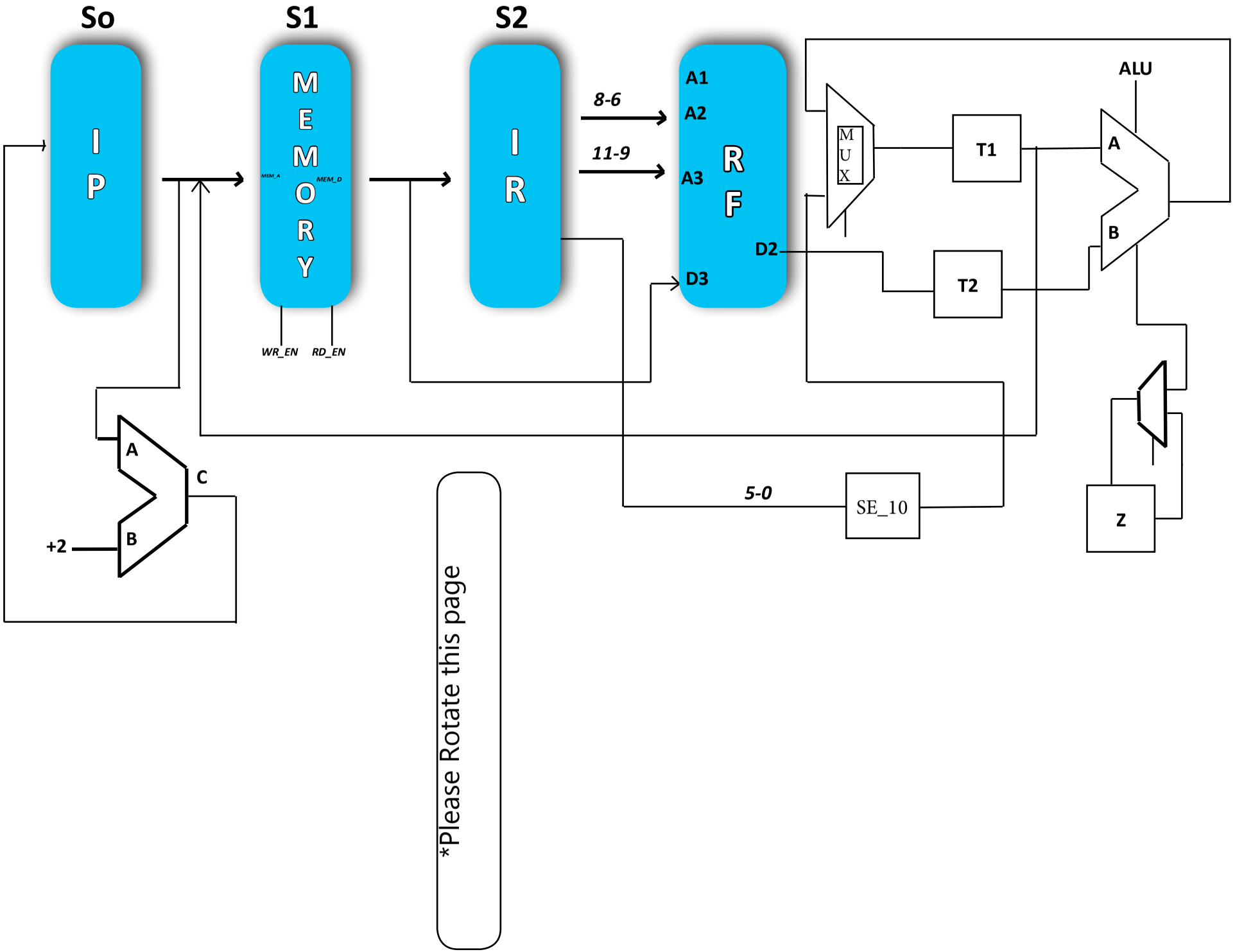


*Please Rotate this page

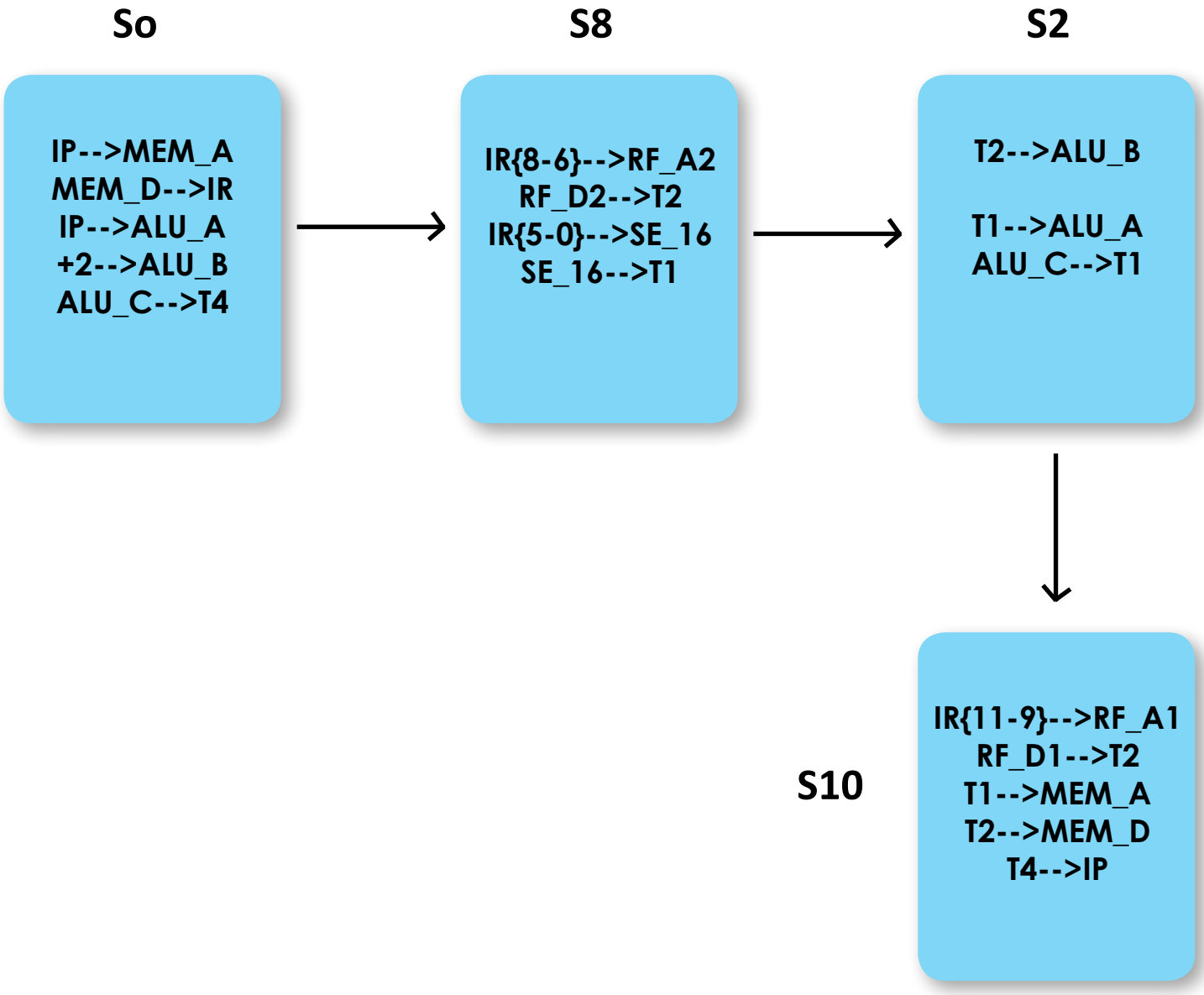
LW


So
IP-->MEM_A
MEM_D-->IR
IP-->ALU_A
+2-->ALU_B
ALU_C-->T4
S8
IR{8-6}-->RF_A2
RF_D2-->T2
IR{5-0}-->SE_16
SE_16-->T1
S2
T2-->ALU_B
T1-->ALU_A
ALU_C-->T1
ALU_Z-->Z
S9
T1-->MEM_A
MEM_D-->RF_D3
IR{11-9}-->RF_A3
T4-->IP

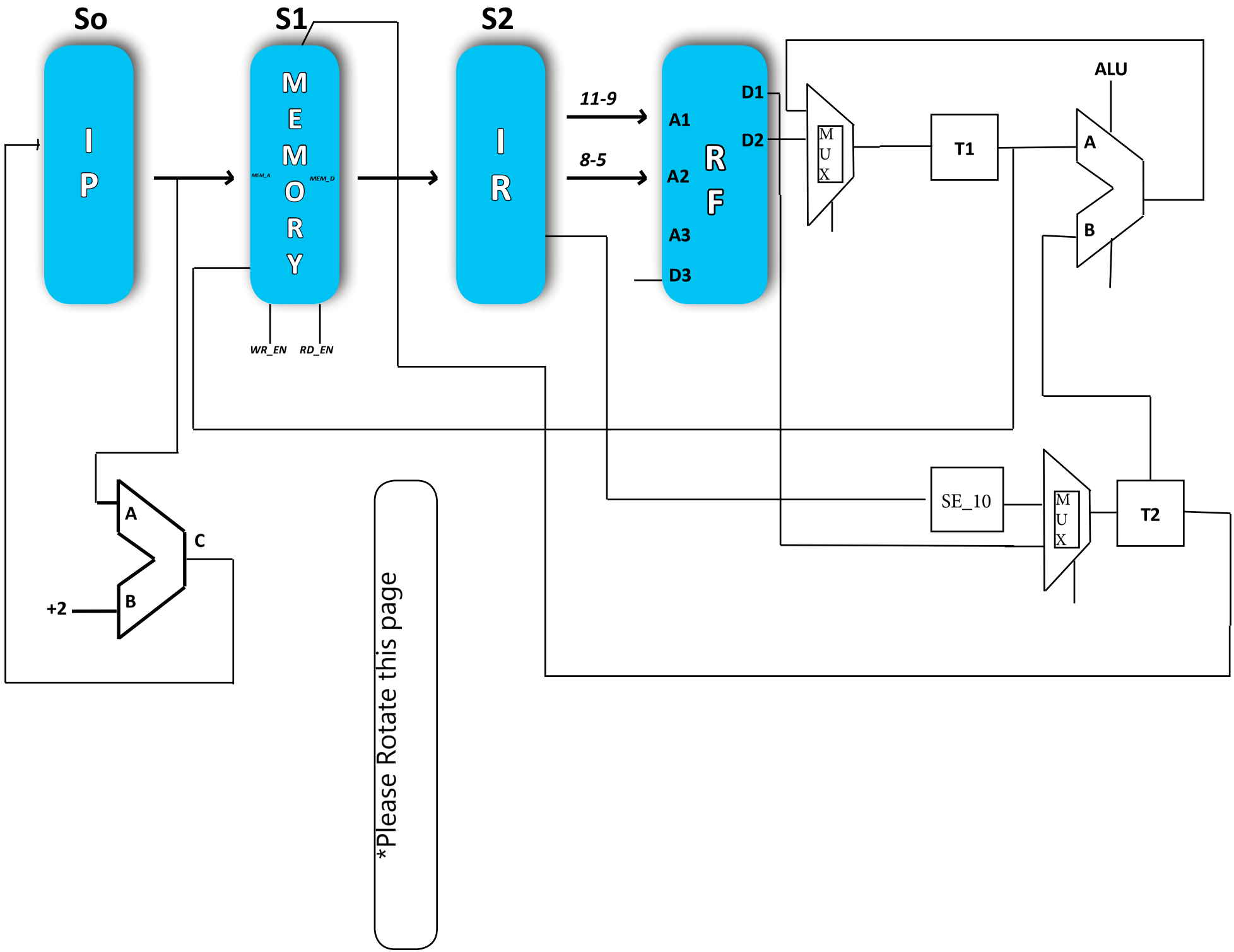
P.T.O


So
S1
S2
I
P
MEMORY
MEM_A
MEM_D
WR_EN
RD_EN
I
R
8-6
11-9
A1
A2
A3
RF
D3
D2
MUX
T1
T2
ALU
A
B
A
B
C
+2
5-0
SE_10
Z
*Please Rotate this page

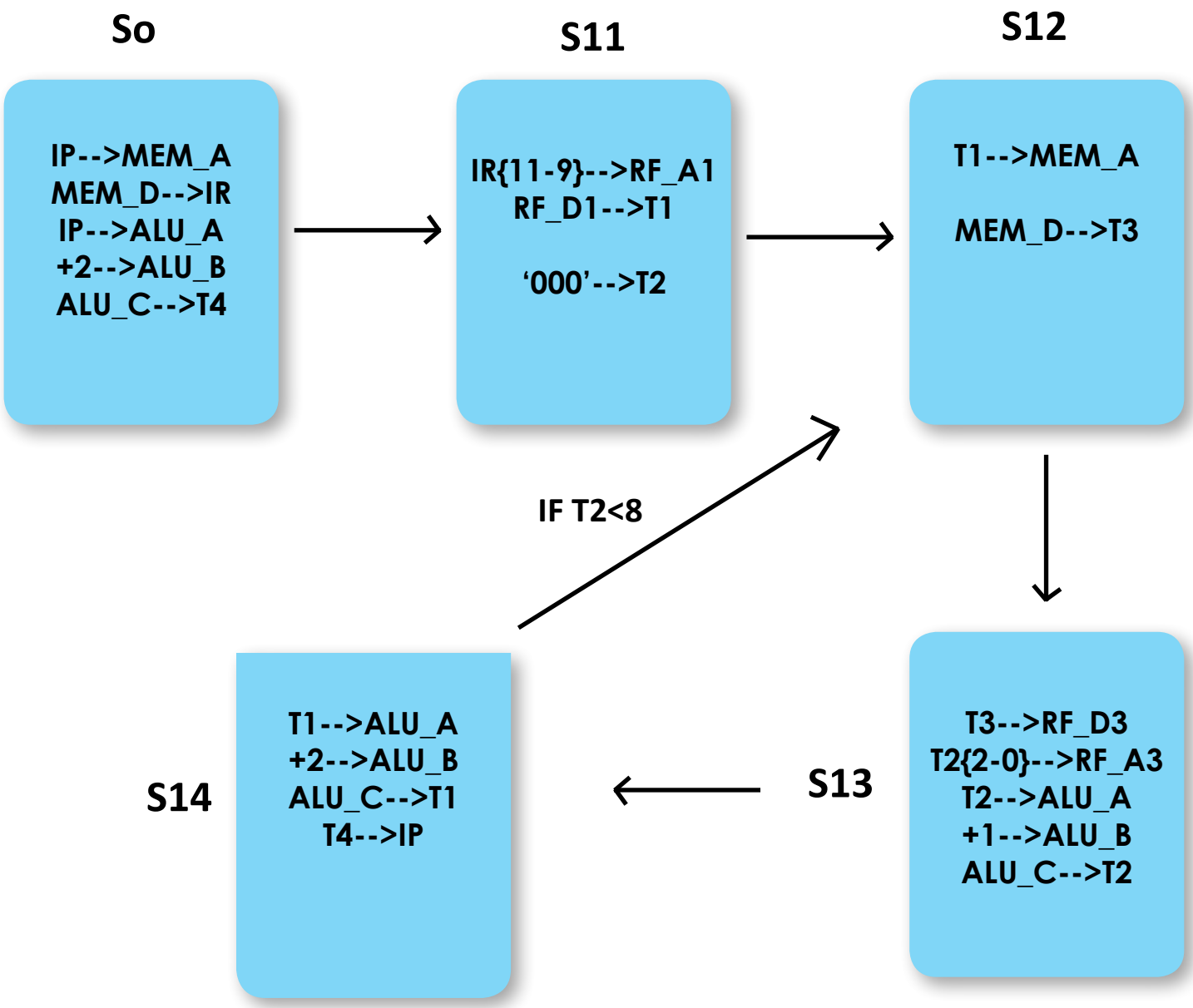
SW


So
IP-->MEM_A
MEM_D-->IR
IP-->ALU_A
+2-->ALU_B
ALU_C-->T4
S8
IR{8-6}-->RF_A2
RF_D2-->T2
IR{5-0}-->SE_16
SE_16-->T1
S2
T2-->ALU_B
T1-->ALU_A
ALU_C-->T1
S10
IR{11-9}-->RF_A1
RF_D1-->T2
T1-->MEM_A
T2-->MEM_D
T4-->IP

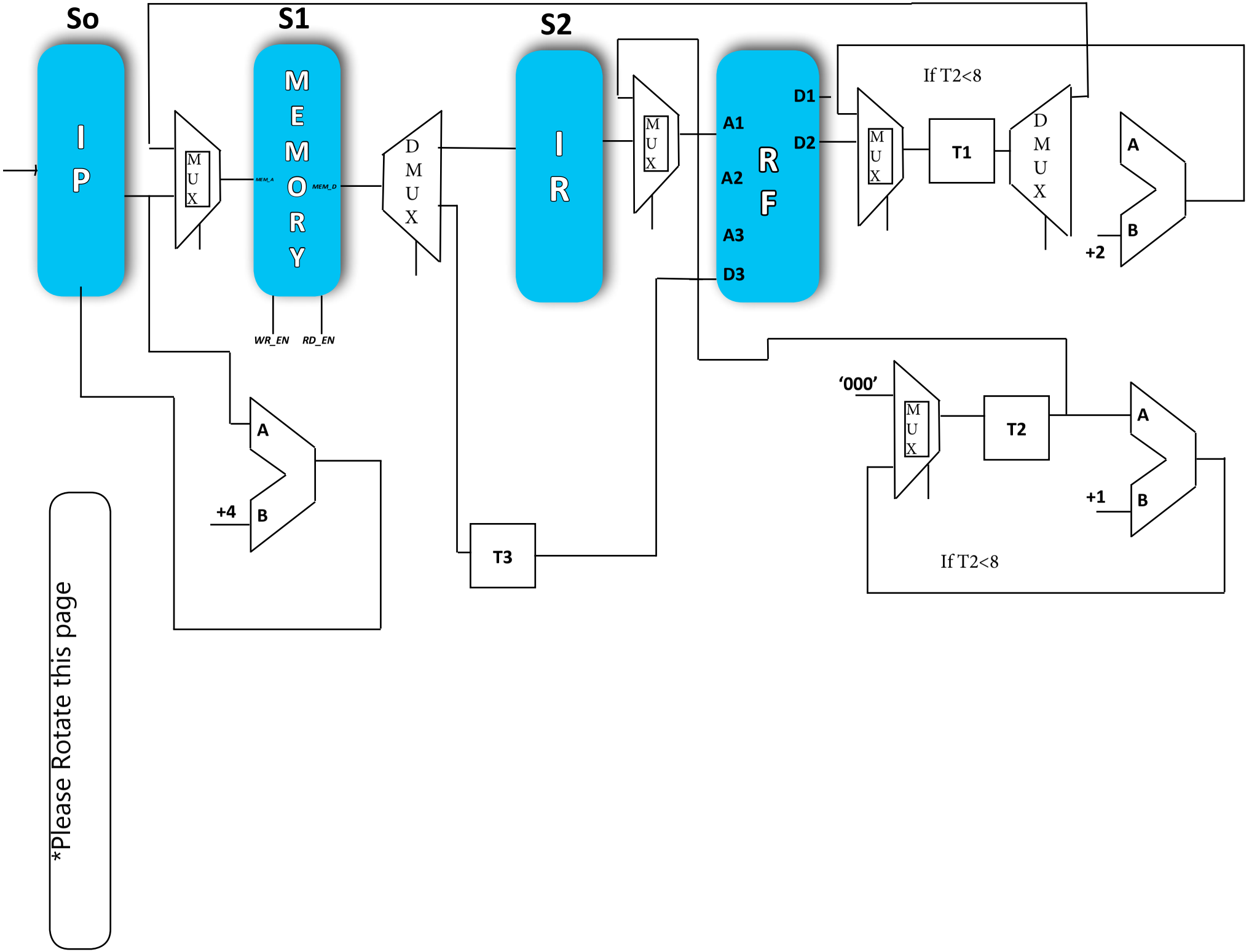
P.T.O


So
S1
S2
IP
MEMORY
MEM_A
MEM_D
WR_EN
RD_EN
IR
11-9
8-5
RF
A1
A2
A3
D1
D2
D3
MUX
T1
ALU
A
B
SE_10
MUX
T2
A
B
C
+2
*Please Rotate this page

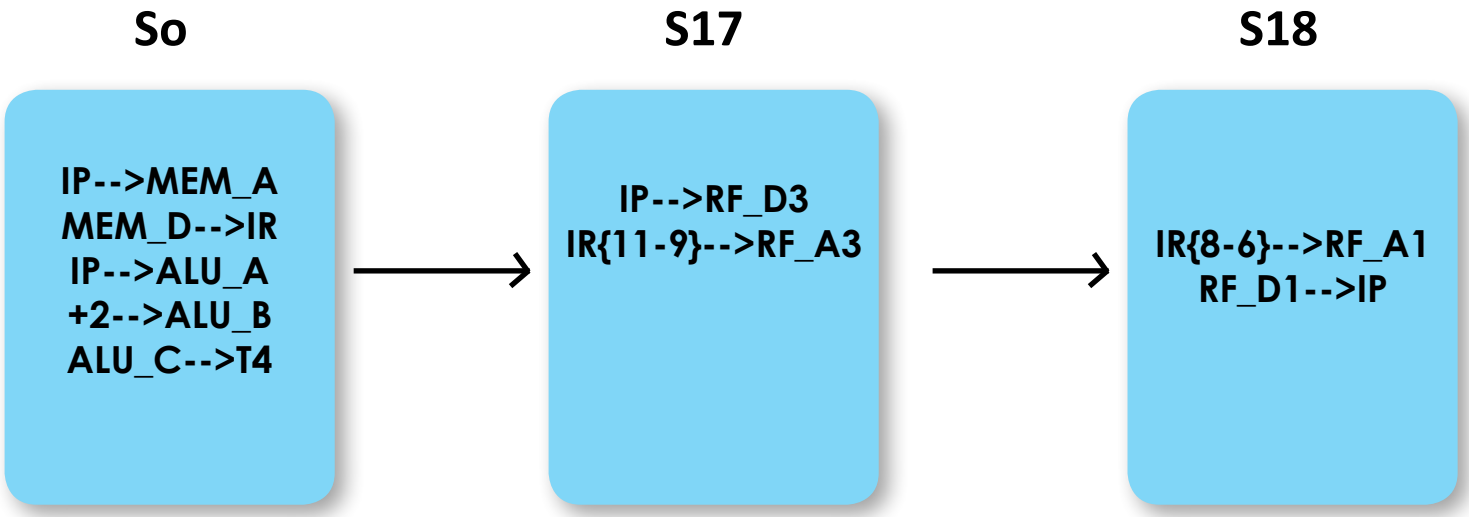
LA


So
IP-->MEM_A
MEM_D-->IR
IP-->ALU_A
+2-->ALU_B
ALU_C-->T4
S11
IR{11-9}-->RF_A1
RF_D1-->T1
'000'-->T2
S12
T1-->MEM_A
MEM_D-->T3
IF T2<8
S13
T3-->RF_D3
T2{2-0}-->RF_A3
T2-->ALU_A
+1-->ALU_B
ALU_C-->T2
S14
T1-->ALU_A
+2-->ALU_B
ALU_C-->T1
T4-->IP

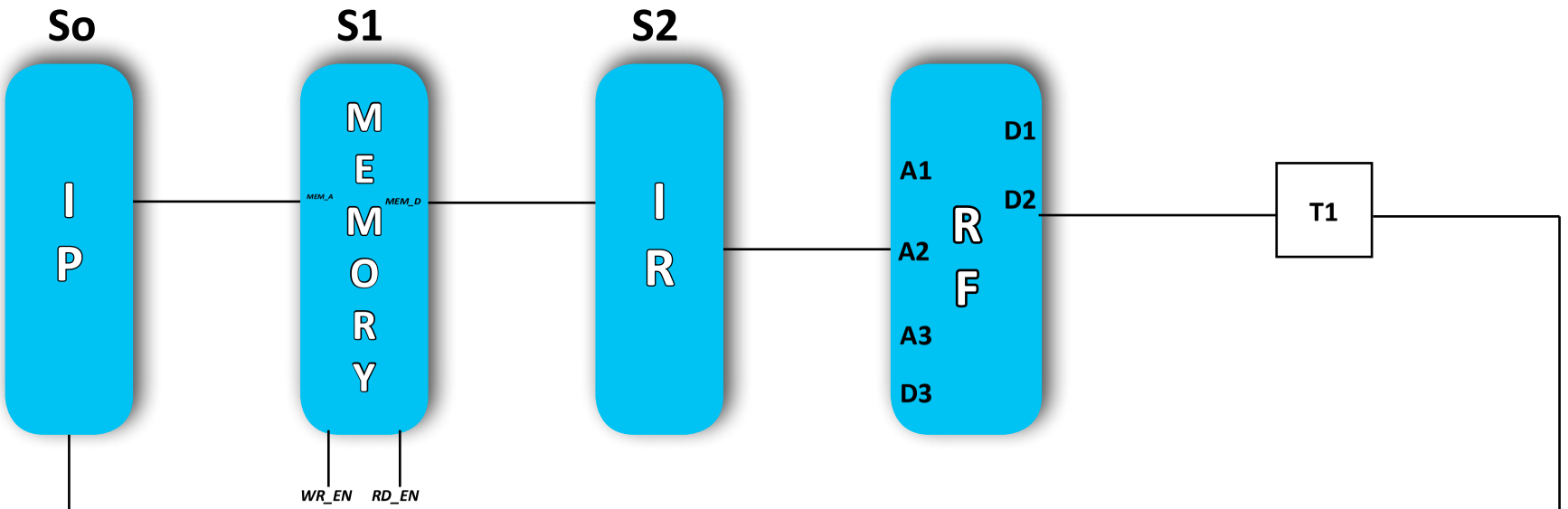
P.T.O


So
S1
S2
IP
MUX
MEMORY
MEM_A
MEM_D
WR_EN
RD_EN
DMUX
IR
MUX
A1
A2
A3
D3
D1
D2
RF
If T2<8
MUX
T1
DMUX
A
B
+2
A
+4
B
T3
'000'
MUX
T2
A
+1
B
If T2<8
*Please Rotate this page

JLR

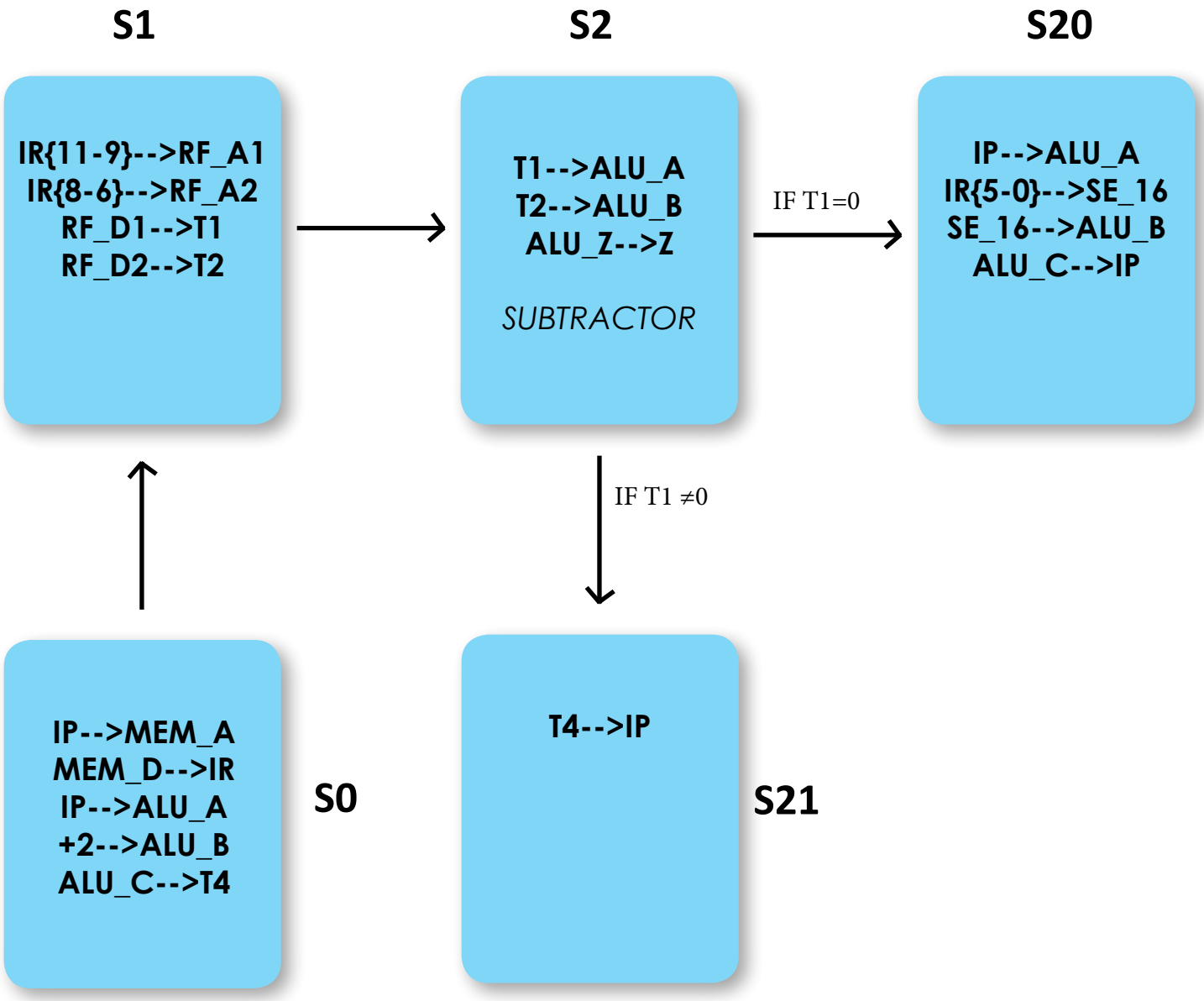

So
S17
S18
IP-->MEM_A
MEM_D-->IR
IP-->ALU_A
+2-->ALU_B
ALU_C-->T4
IP-->RF_D3
IR{11-9}-->RF_A3
IR{8-6}-->RF_A1
RF_D1-->IP

P.T.O

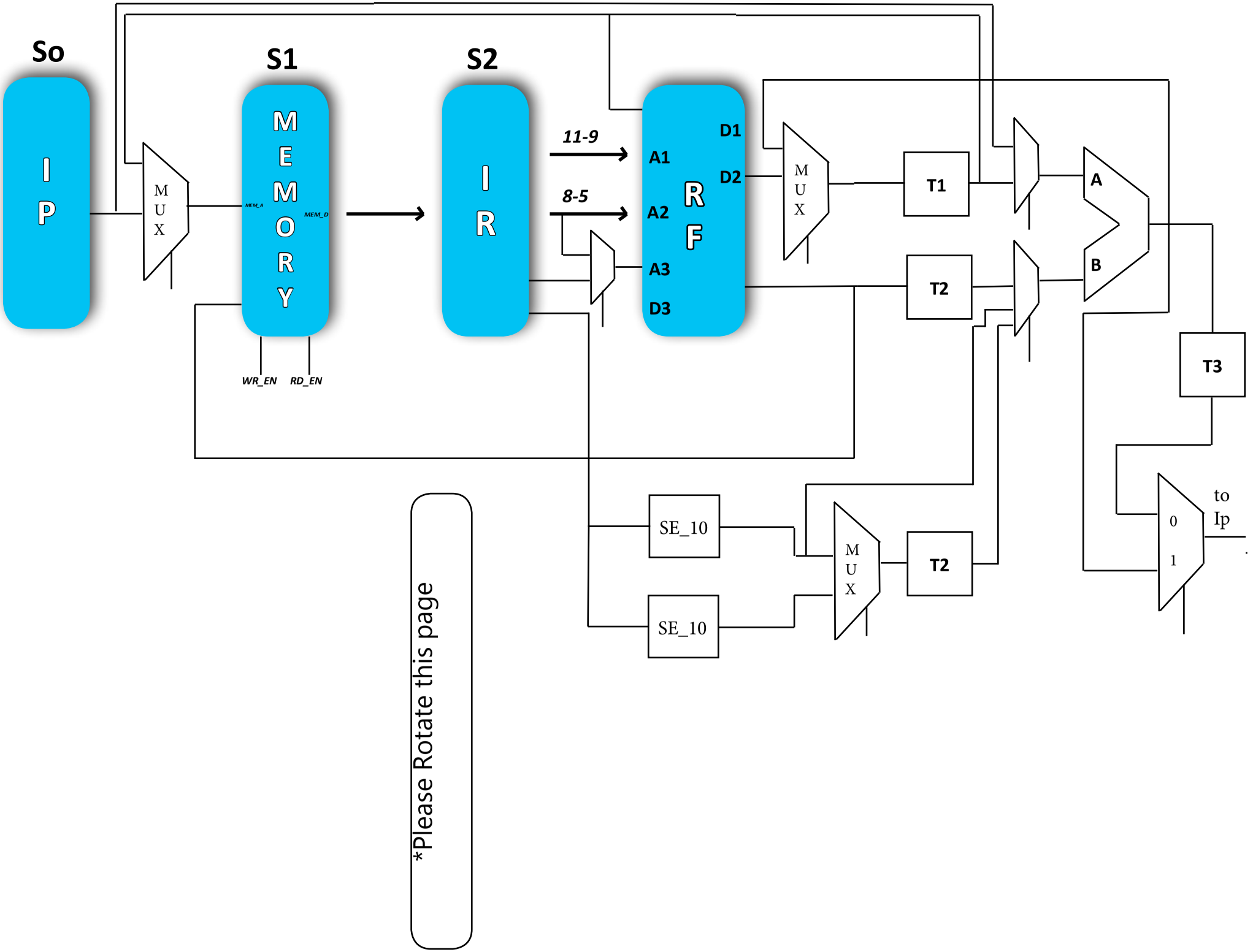


*Please Rotate this page

BEQ

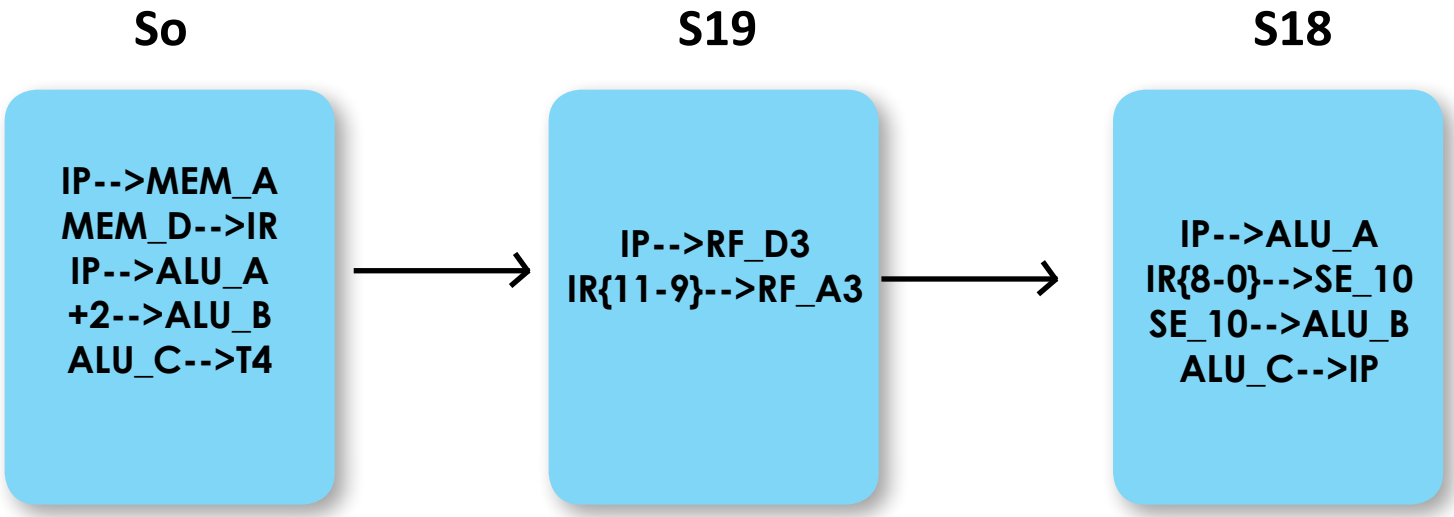

S1
S2
S20
IR{11-9}-->RF_A1
IR{8-6}-->RF_A2
RF_D1-->T1
RF_D2-->T2
T1-->ALU_A
T2-->ALU_B
ALU_Z-->Z
SUBTRACTOR
IF T1=0
IP-->ALU_A
IR{5-0}-->SE_16
SE_16-->ALU_B
ALU_C-->IP
IF T1 ≠0
IP-->MEM_A
MEM_D-->IR
IP-->ALU_A
+2-->ALU_B
ALU_C-->T4
S0
T4-->IP
S21

P.T.O

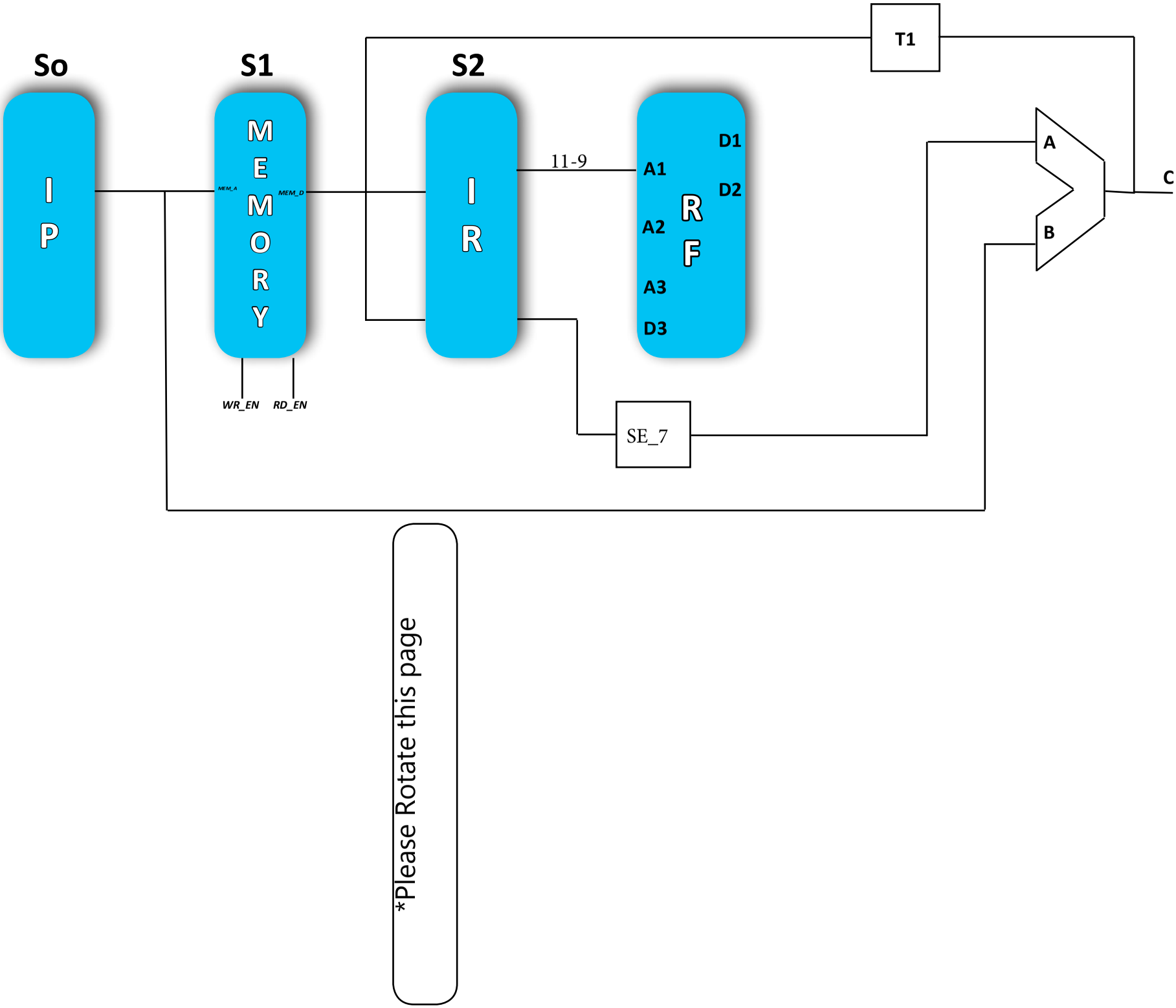


*Please Rotate this page

JAL

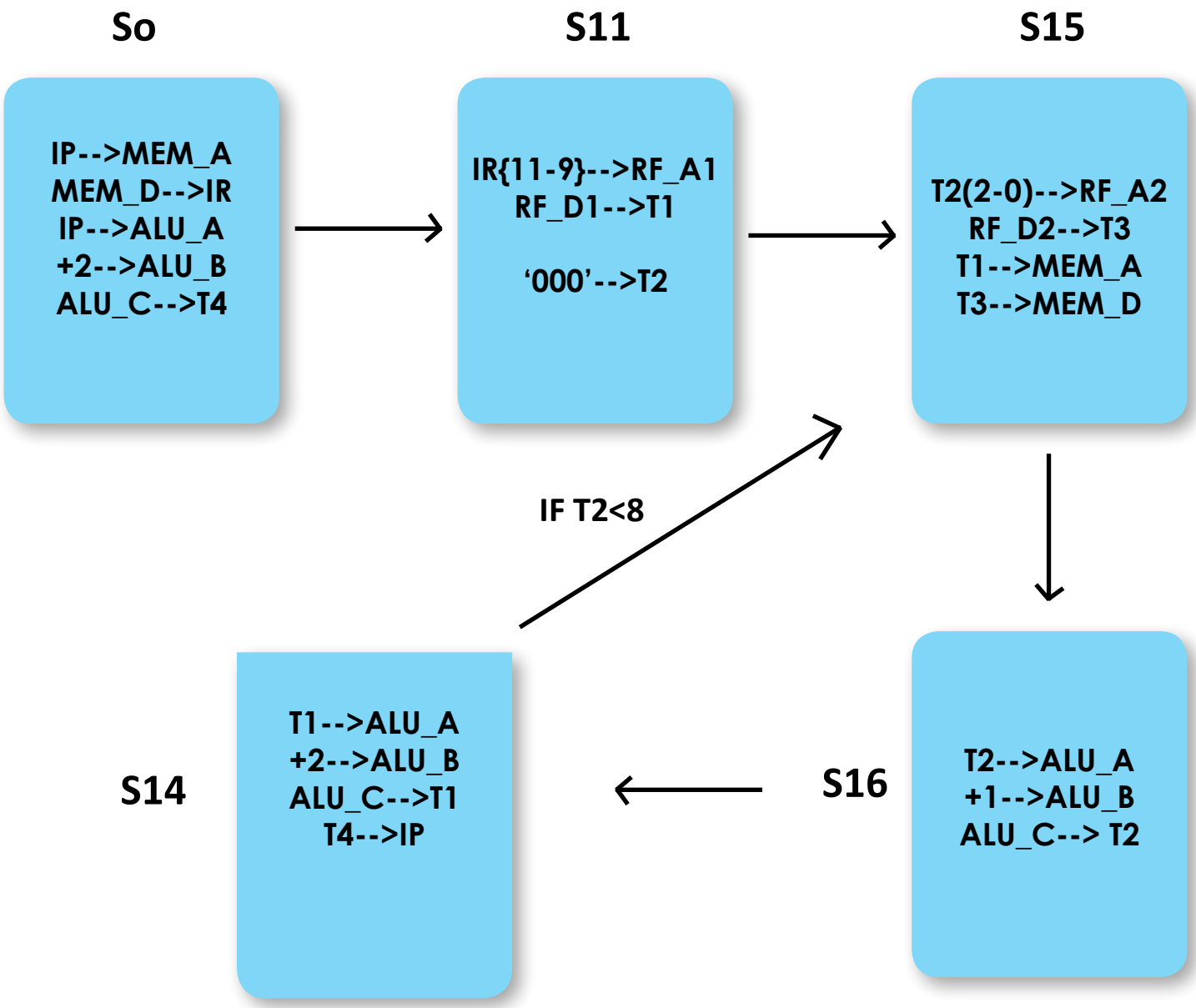

So
IP-->MEM_A
MEM_D-->IR
IP-->ALU_A
+2-->ALU_B
ALU_C-->T4
S19
IP-->RF_D3
IR{11-9}-->RF_A3
S18
IP-->ALU_A
IR{8-0}-->SE_10
SE_10-->ALU_B
ALU_C-->IP

P.T.O



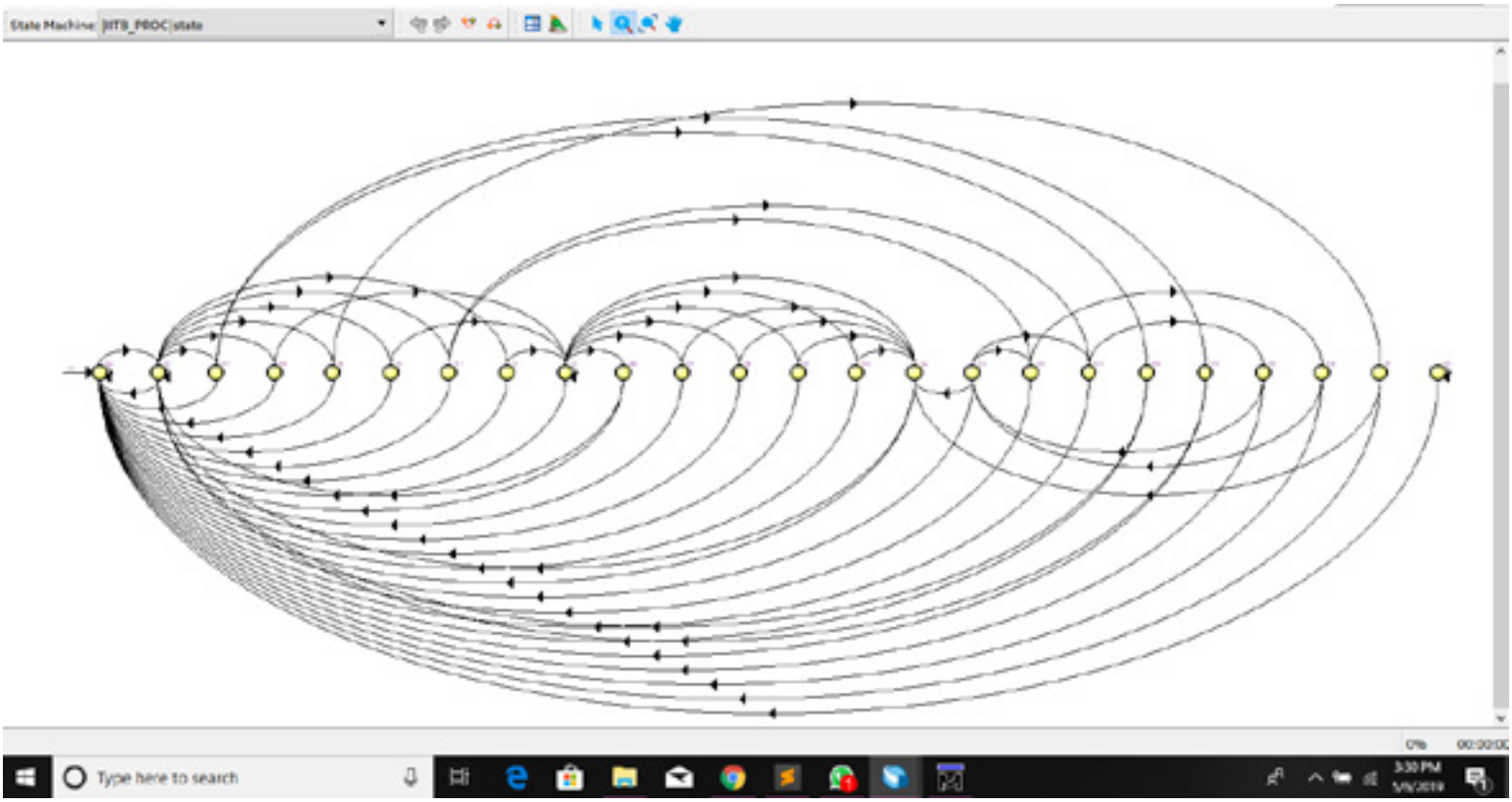
*Please Rotate this page

SA


So
IP-->MEM_A
MEM_D-->IR
IP-->ALU_A
+2-->ALU_B
ALU_C-->T4
S11
IR{11-9}-->RF_A1
RF_D1-->T1
'000'-->T2
S15
T2(2-0)-->RF_A2
RF_D2-->T3
T1-->MEM_A
T3-->MEM_D
IF T2<8
S14
T1-->ALU_A
+2-->ALU_B
ALU_C-->T1
T4-->IP
S16
T2-->ALU_A
+1-->ALU_B
ALU_C--> T2

P.T.O

NETLIST VIEWER



STATE DIAGRAM