CS230 PROJECT

Team:

- Neeshi S Merchant 200050087
- Anushka 200050011
- Anushka Dubey 200050012
- Onkar Borade 200050022

Instruction Encoding

ADD:	00_01	RA	RB	RC	0	00
ADC:	00_01	RA	RB	RC	0	10
ADZ:	00_01	RA	RB	RC	0	01
ADL:	00_01	RA	RB	RC	0	11
ADI:	00_00	RA	RB	6 bit Immediate		
NDU:	00_10	RA	RB	RC	0	00
NDC:	00_10	RA	RB	RC	0	10
NDZ:	00_10	RA	RB	RC	0	01
LHI:	00_00	RA	9 bit Immediate			
LW:	01_01	RA	RB	6 bit Immediate		
SW:	01_11	RA	RB	6 bit Immediate		

LM:	11_01	RA	0 + 8 bits corresponding to Reg R0 to R7 (left to right)	
SM:	11_00	RA	0 + 8 bits corresponding to Reg R0 to R7 (left to right)	
BEQ:	10_00	RA	RB	6 bit Immediate
JAL:	10_01	RA	9 bit Immediate offset	
JLR:	10_10	RA	RB	000_000
JRI	10_11	RA	9 bit Immediate offset	

Instruction Description

Mnemonic	Name & Format	Assembly	Action
ADD	ADD (R)	*add rc, ra, rb*	Add content of regB to regA and store result in regC. *It modifies C and Z flags*
ADC	Add if carry set (R)	*adc rc, ra, rb*	Add content of regB to regA and store result in regC, if carry flaf is set. *It modifies C & Z flags*
ADZ	Add if zero set (R)	*adz rc, ra, rb*	Add content of regB to regA and store result in regC, if zero flag is set. *It modifies C & Z flags*
ADL	Add with one bit left shift of RB (R)	*Adl rc,ra,rb*	Add content of regB (after one bit left shift) to regA and store result in regC *It modifies C & Z flags*
ADI	Add immediate (I)	*adi rb, ra, imm6*	Add content of regA with Imm (sign extended) and store result in regB. *It modifies C and Z flags*
NDU	Nand (R)	*ndu rc, ra, rb*	NAND the content of regB to regA and store result in regC. *It modifies Z flag*

NDC	Nand if carry set (R)	*ndc rc, ra, rb*	NAND the content of regB to regA and store result in regC if carry flag is set. *It modifies Z flag*
NDZ	Nand if zero set (R)	*ndc rc, ra, rb*	NAND the content of regB to regA and store result in regC if zero flag is set. *It modifies Z flag*
LHI	Load higher immediate (J)	*lhi ra, Imm*	Place 9 bits immediate into most significant 9 bits of register A (RA) and lower 7 bits are assigned to zero.
LW	Load (I)	*lw ra, rb, Imm*	Load value from memory into reg A. Memory address is formed by adding immediate 6 bits with content of red B. *It modifies zero flag.*
SW	Store (I)	*sw ra, rb, Imm*	Store value from reg A into memory. Memory address is formed by adding immediate 6 bits with content of red B.
LM	Load multiple (J)	*lw ra, Imm*	Load multiple registers whose address is given in the immediate field (one bit per register, R0 to R7) in order from left to right, i.e., registers from R0 to R7 if corresponding bit is set. Memory address is given in reg A. Registers are loaded from consecutive addresses.

SM	Store multiple (J)	sm, ra, Imm	Store multiple registers whose address is given in the immediate field (one bit per register, R0 to R7) in order from left to right, i.e., registers from R0 to R7 if corresponding bit is set. Memory address is given in reg A. Registers are stored to consecutive addresses.
BEQ	Branch on Equality (I)	beq ra, rb, Imm	If content of reg A and regB are the same, branch to PC+Imm, where PC is the address of beq instruction
JAL	Jump and Link (I)	jalr ra, Imm	Branch to the address PC+ Imm. Store PC+1 into regA, where PC is the address of the jalr instruction
JLR	Jump and Link to Register (I)	jalr ra, rb	Branch to the address in regB. Store PC+1 into regA, where PC is the address of the jalr instruction
JRI	Jump to register (J)	jri ra, Imm	Branch to memory location given by the RA + Imm

VHDL CODES

COMPONENTS

- ALU
- Priority Encoder
- Load/Store Multiple Logic Block
- Registers
- Memory
- Register File
- Sign Extender

MICROPROCESSOR BLOCKS

Data Path: This consists of the entire data path along with all the transfers and the predicates corresponding to an RTL layout of the microprocessor. All the T and S signals are pretty accurately detailed as comments at the beginning of the architecture.

Control Path: This consists of the controller Moore FSM. It has been decomposed into three processes; The first one describes the flip flops which control the states, the second one describes the next state logic and finally the third process controls the output logic based on the present state. (The order in the code might not exactly follow this order). All the T signals have been accompanied by the expected result they are to cause in the data path.

IITB_RISC: The top-level entity than combines the data path and the controller FSM. The register 0 has been shown as an output so that the processes of the microprocessor can be displayed outside in hardware.

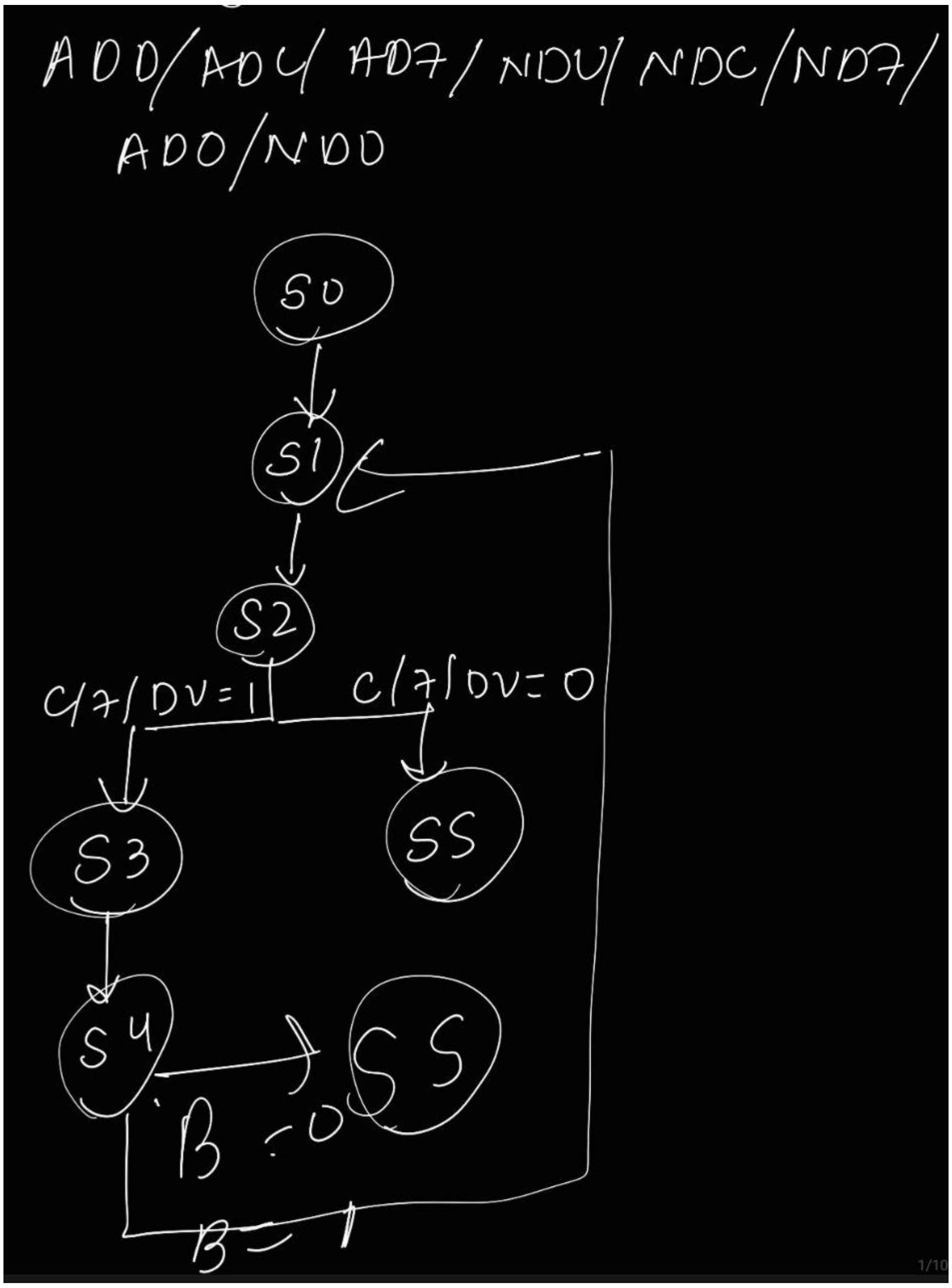
Bootloader: The bootloader block.

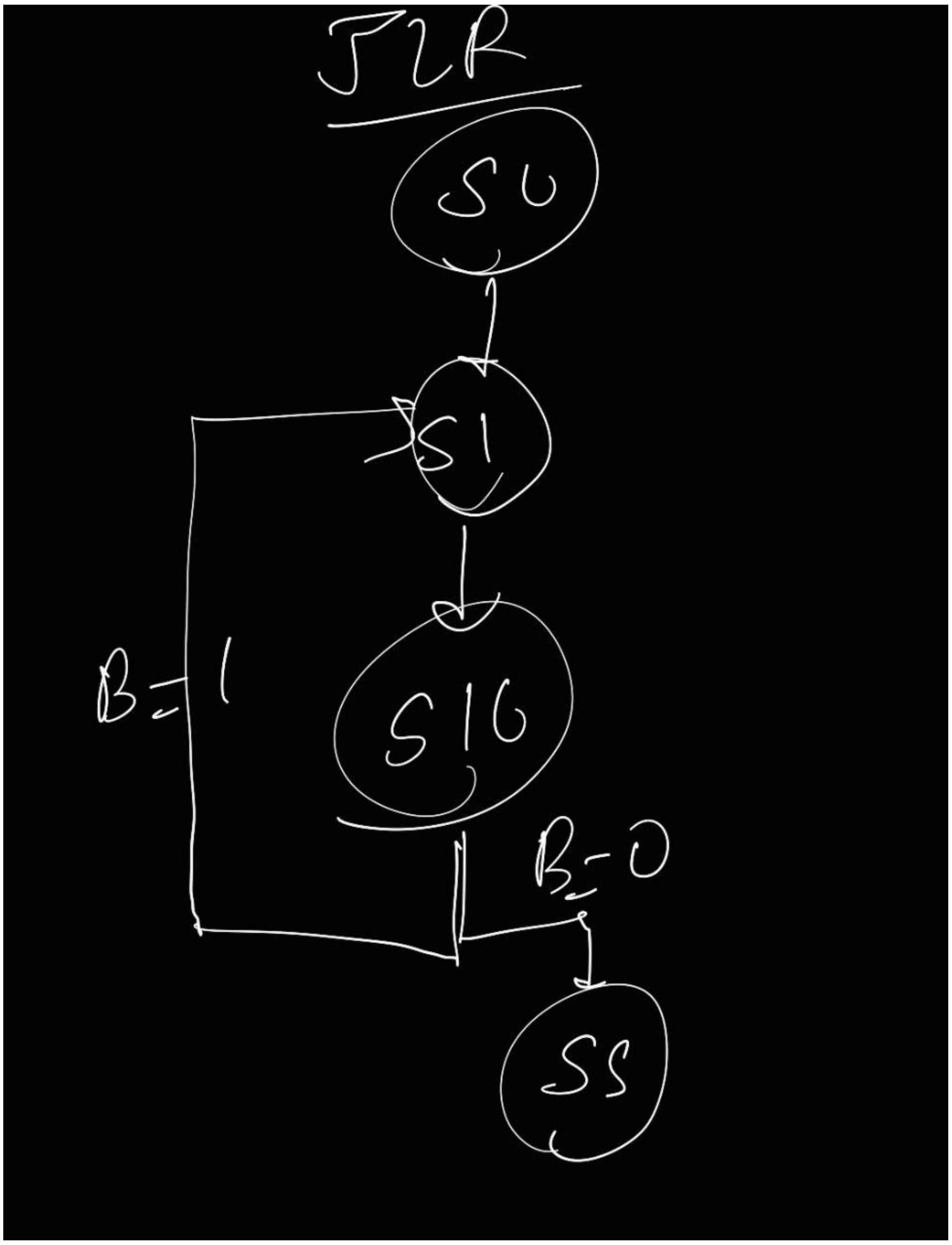
STATES

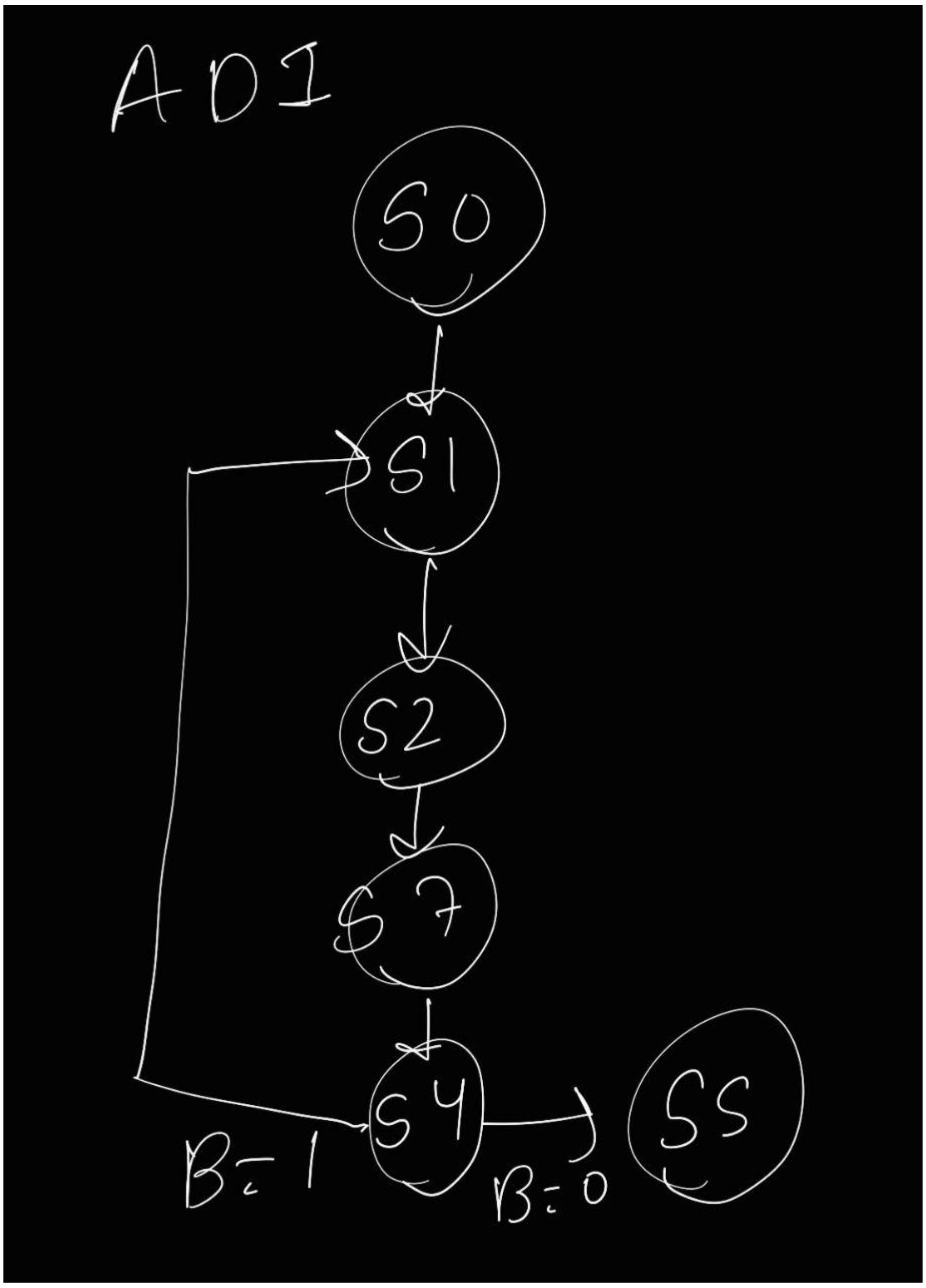
S1	R7 → MEM (A) MEM (D)→ IR R7 → ALU +1 → ALU ALU → PC
S2	I6 – 8 → A1RF I9 – 11 → A2RF D1 → E1 D2 → E2,T1 I0-7 → PEINPUT
S3	E1 → ALU E2 → ALU ALU → T1
S4	I3 – 5 /I6 - 8 → A3RF T1 → D3RF
S5	PC → D3RF “111” → A3RF T2 → ALU 0 → ALU
S6	I0 – 8 → SE9 – 16 → LS7 LS7 → D3RF I9 – 11 → A3RF
S7	E1→ ALU I0 – 5 → SE6 – 16 → ALU ALU → T1 , MEM(A)

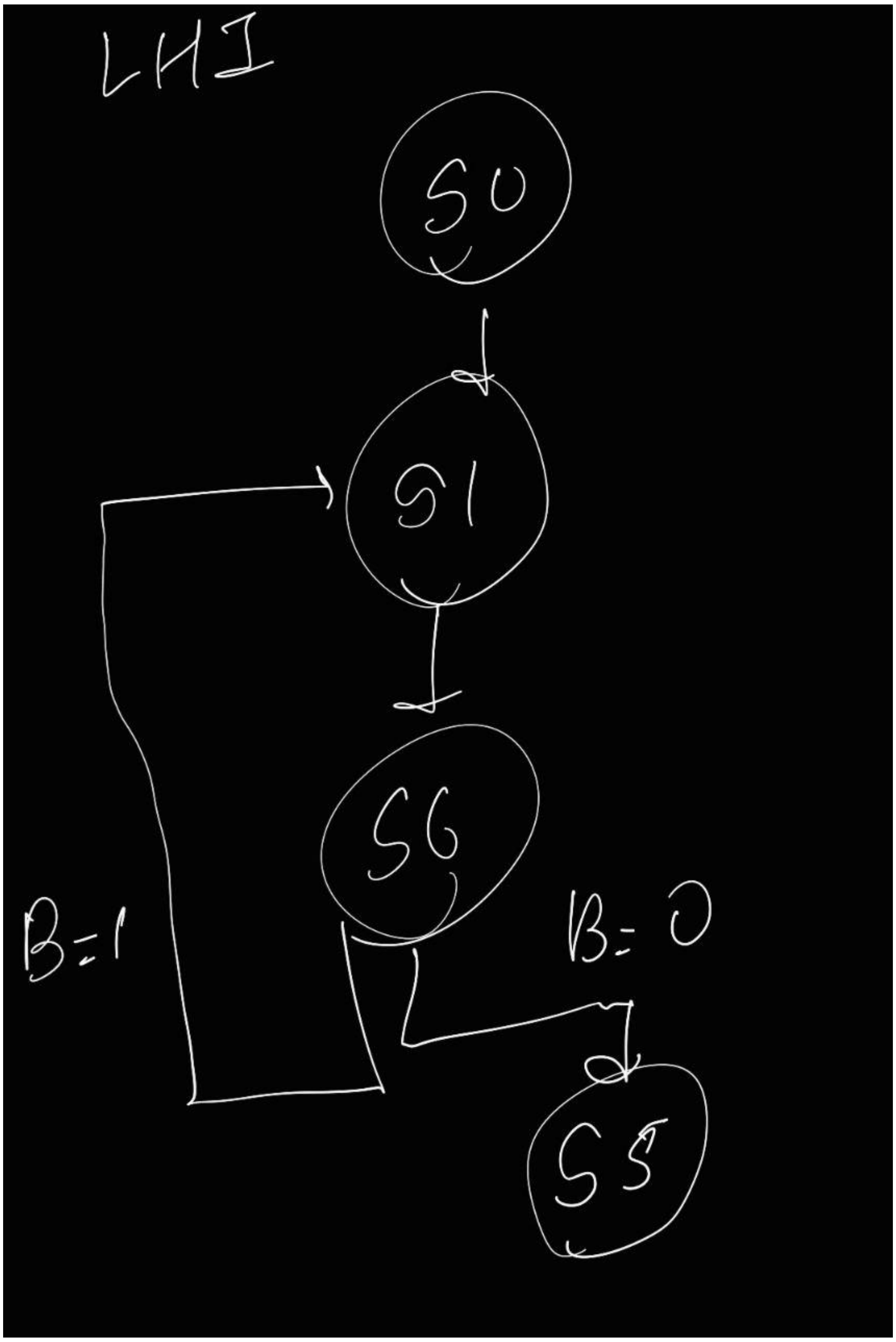
S8	MEM (DO) → T2, D3RF I9 – 11 → A3RF
S9	D2 → MEM10(DI) PC → D3RF “111” → A3RF
S10	do { MEMDAT (DO) → T2}
S11	T2 → D3RF PEOUTPUT→ A3RF T1 → ALU +1 → ALU ALU → T1,MEM(DI) while (! invalid_next);
S12	PEOUTPUT → A2RF T1 → MEM(A)
S13	T1 → ALU +1 → ALU ALU → T1 while (! invalid_next);
S14	R7 → ALU I0 – 5 → SE6 – 16 → ALU ALU → PC
S15	PC → D3RF I9-11 → A3RF R7 → ALU I0 – 8 → SE9 – 16 → ALU ALU → PC
S16	D1RF → PC I9 – 11 → A3RF PC → D3RF

STATE FLOW DIAGRAMS


ADD/ADC/ADZ/NDU/NDC/NDZ/
ADO/NDO


JZR
S0
S1
S16
B=1
B=0
S5


ADI
S0
S1
S2
S7
S4
S5
B=1
B=0

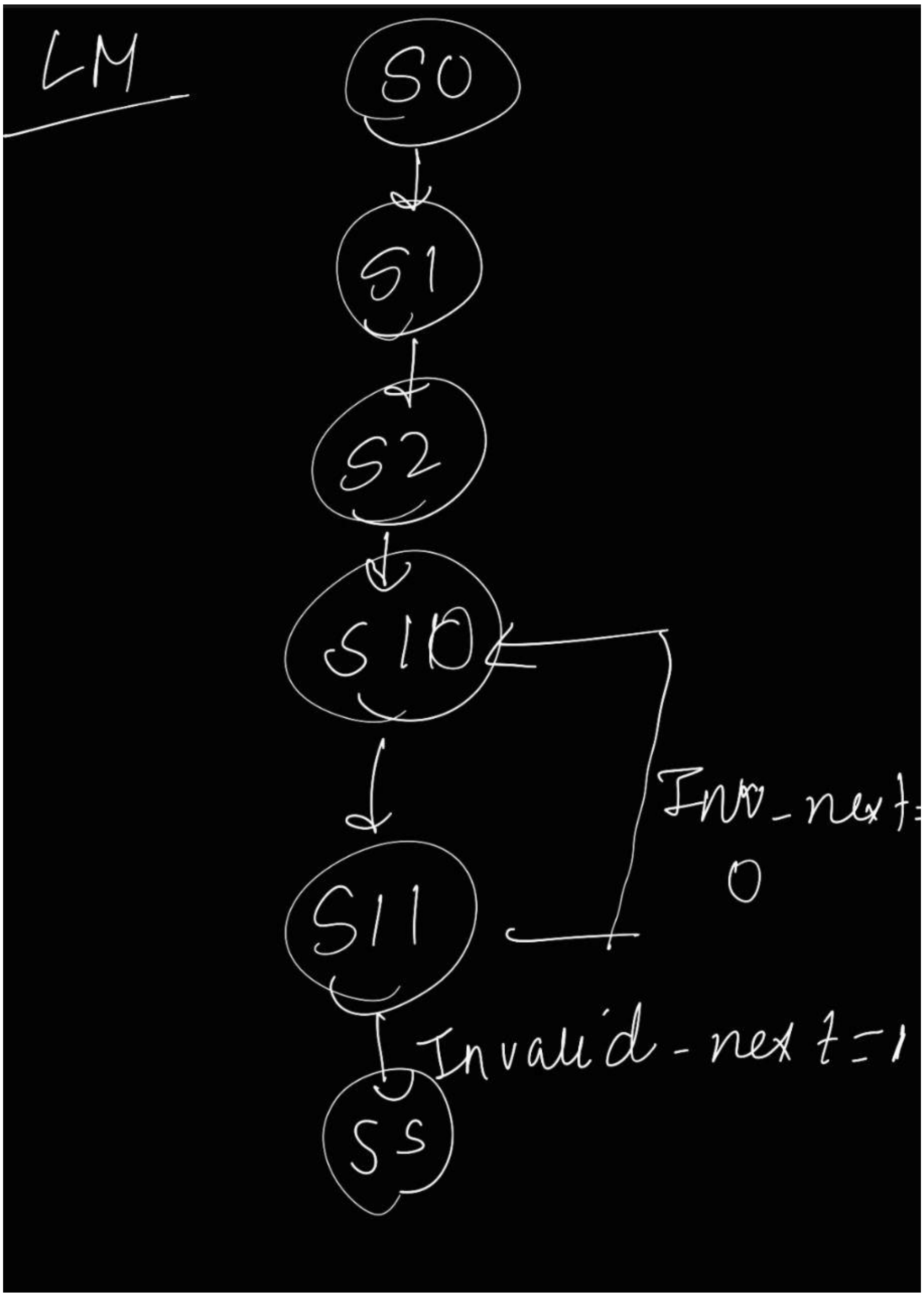

LHI
S0
S1
S6
B=1
B=0
S5

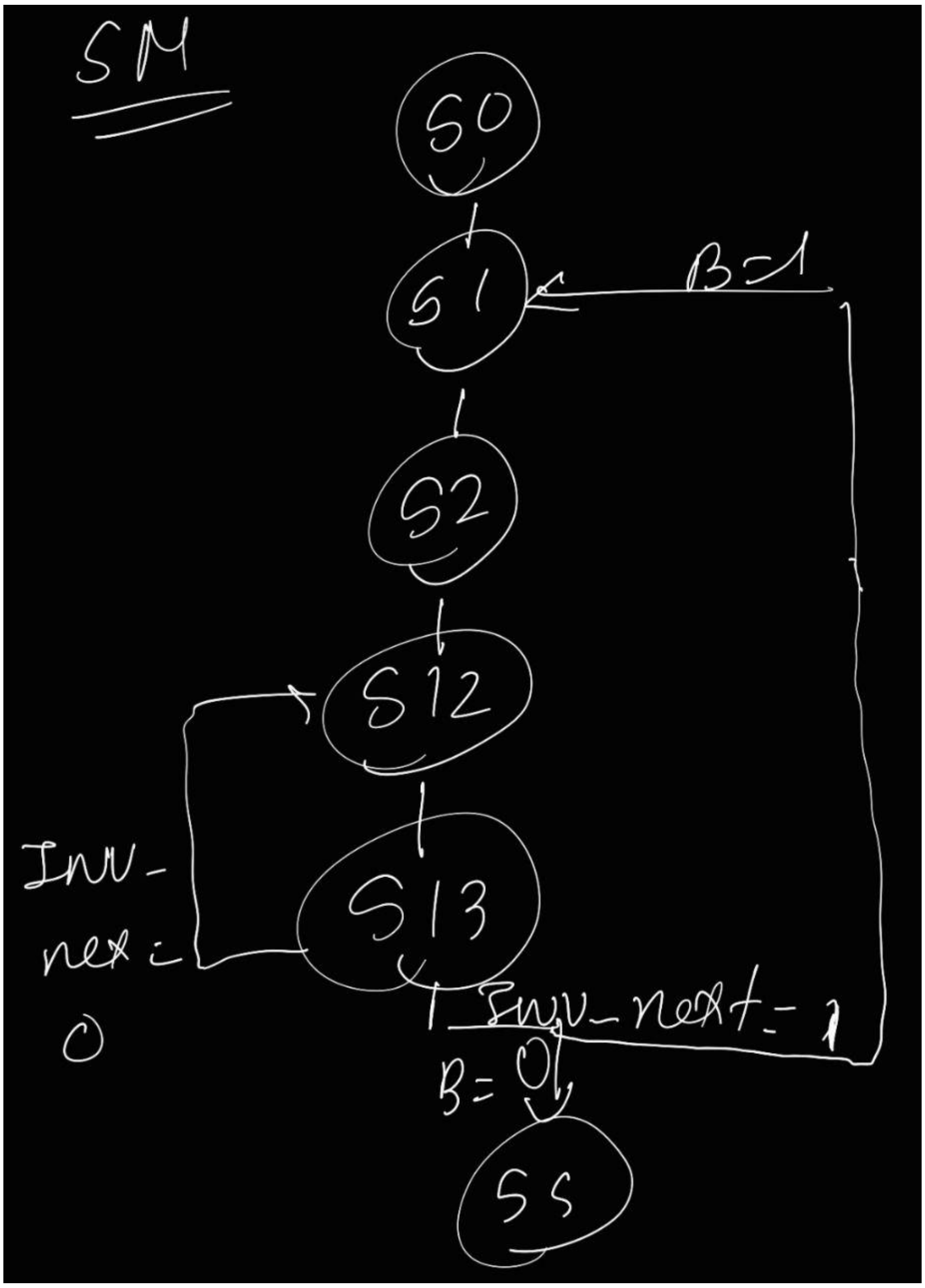
LW

$$\begin{tikzcd}
S0 \arrow[d] \\
S1 \arrow[d] \\
S2 \arrow[d] \\
S7 \arrow[d] \\
S8 \arrow[d] \\
S5
\end{tikzcd}$$

SW

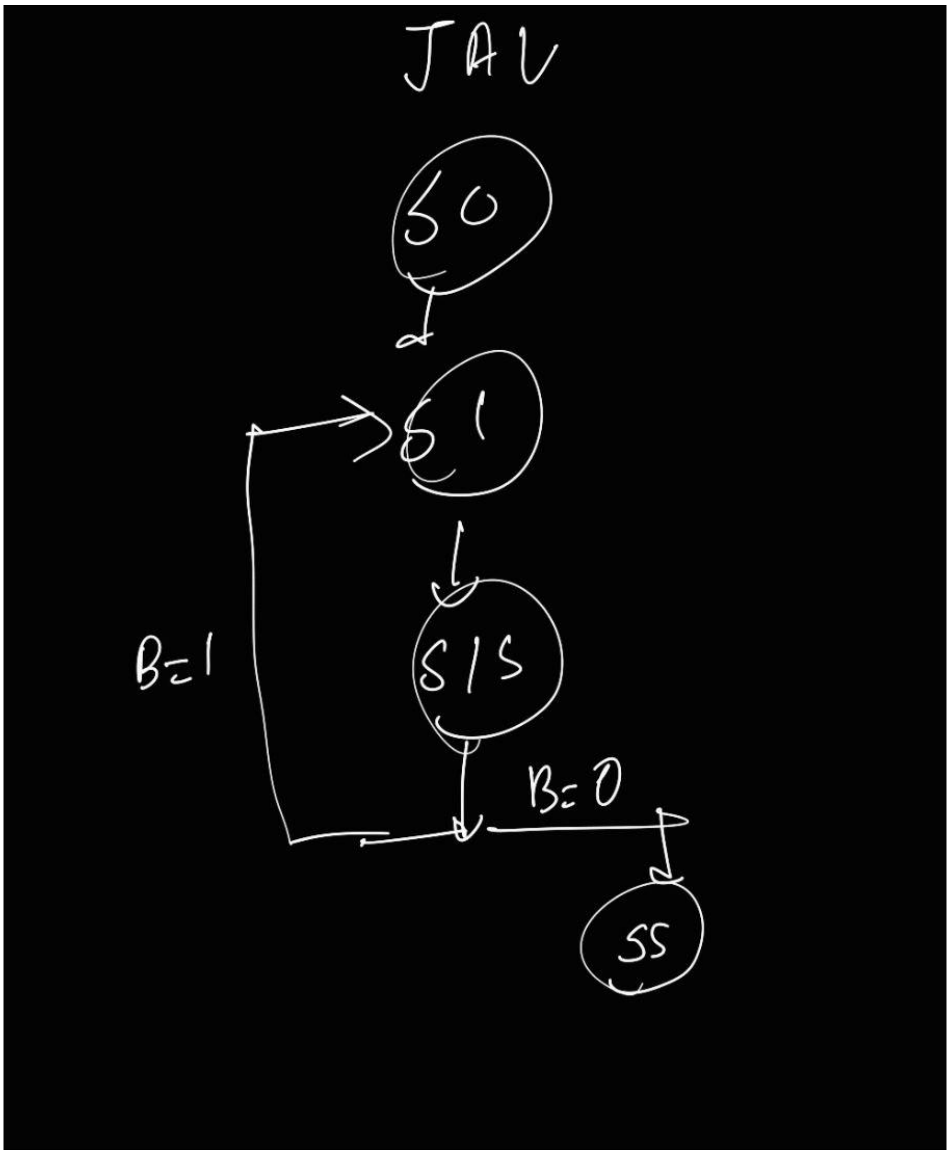
$$\begin{tikzcd}
S0 \arrow[d] \\
S1 \arrow[d] \\
S2 \arrow[d] \\
S7 \arrow[d, leftrightarrow] \\
S9
\end{tikzcd}$$


LM
S0
S1
S2
S10
S11
Inv_next=
0
Invalid-next=1
SS


SM
S0
S1
B=1
S2
S12
S13
Inv_next=0
Inv_next=1
B=0
SS

BEQ

$$\begin{tikzcd}
 & S0 \arrow[d] & \\
 & S1 \arrow[d] & \\
 & S2 \arrow[dl, "EQU=1"'] \arrow[dr, "EQU=0"] & \\
S14 & & S5 \arrow[ll]
\end{tikzcd}$$


JAL
S0
S1
S1S
B=1
B=0
SS