EE224: Digital Systems

IITB CPU

DESIGN

Members:

23B1247: Aniket Gupta
23B1258: Shreya Nigam
23B1266: Apoorv Goyal
23B1290: Matam Kushaal

IIT-B CPU

INSTRUCTION SET ARCHITECTURE

General properties –

1. 16-bit computer
2. 8 Register means they can be coded in 3 bits.
3. Program Counter (PC)
4. Has 2 flags carry (C) & zero (Z)
5. 3 machine code instruction formats (R, I, J type)

R-type:

15 – 12	11 – 9	8 – 6	5 – 3	2	1 – 0
Operation code	Register A (RA)	Register B (RB)	Register C (RC)	unused	unused
4 bits	3 bits	3 bits	3 bits	(1 bit)	2 bits

I-Type

15 – 12	11 – 9	8 – 6	5 – 0
Operation code	Register (RA)	Register (RB)	~~Register (RC)~~ Immediate
4 bits	3 bits	3 bits	~~3 bits~~ 6 bits signed

J-Type

15 – 12	11 – 9	8 – 0
Operation code	Register (RA)	Immediate
4 bits	3 bit	(9 bits signed)

Following instructions are to be implemented –

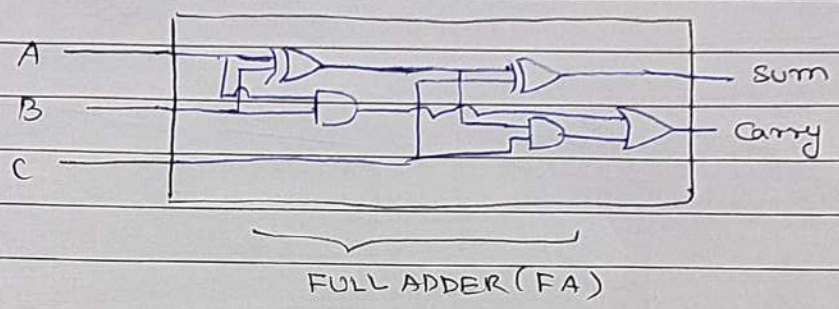
- ADD, SUB, MUL, AND, ORA, IMP } Arithmatic & (R-type) logic
- ADI } Arithmatic (I-type)
- (J) LHI, (J) LLI, (I) LW, (I) SW } Memory Communication
- (I) BEQ, (I) JAL, (I) JLR, (J) J } control flow logic

Components:

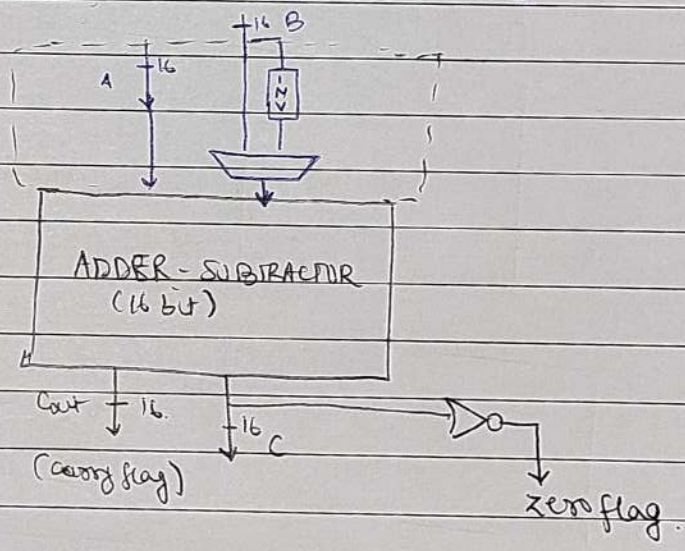
1. Arithmatic & Logic Unit (ALU):
For this we need ADDER-SUB, MULTIPLIER, AND, OR, IMP.

a. ADDER - SUBTRACTOR

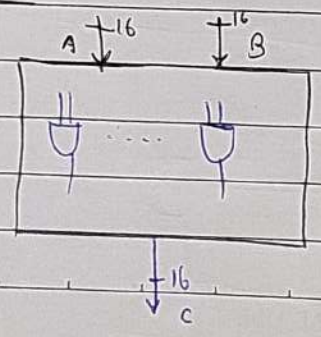
A full adder Unit:



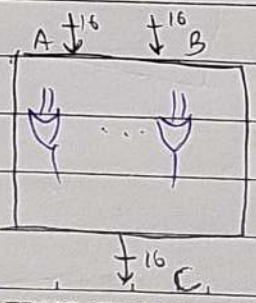
Using 16 such FA we make an adder-subtractor.
$C_{in} = 0$: ADD $C_{in} = 1$ SUB.



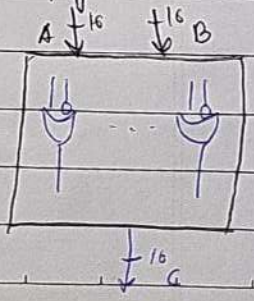
b. AND

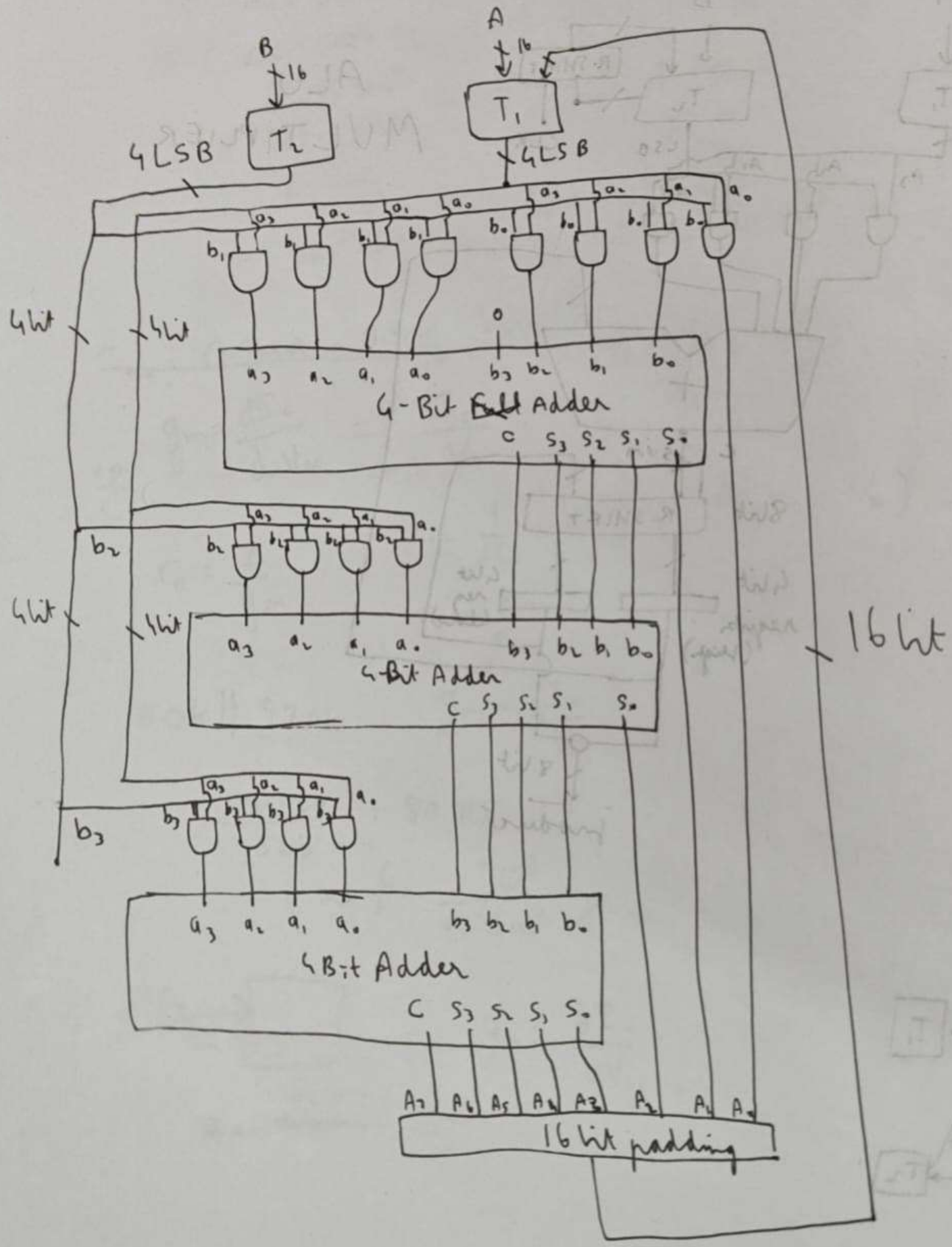


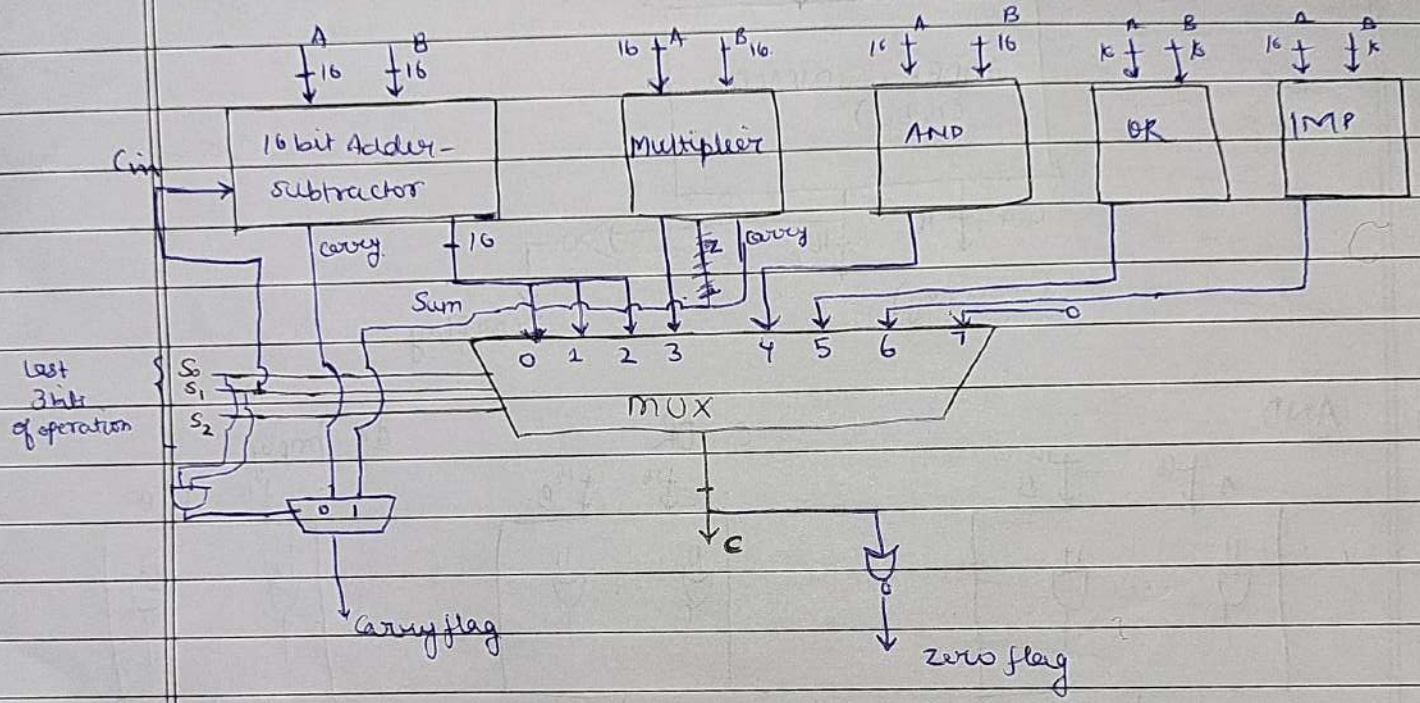
c. OR



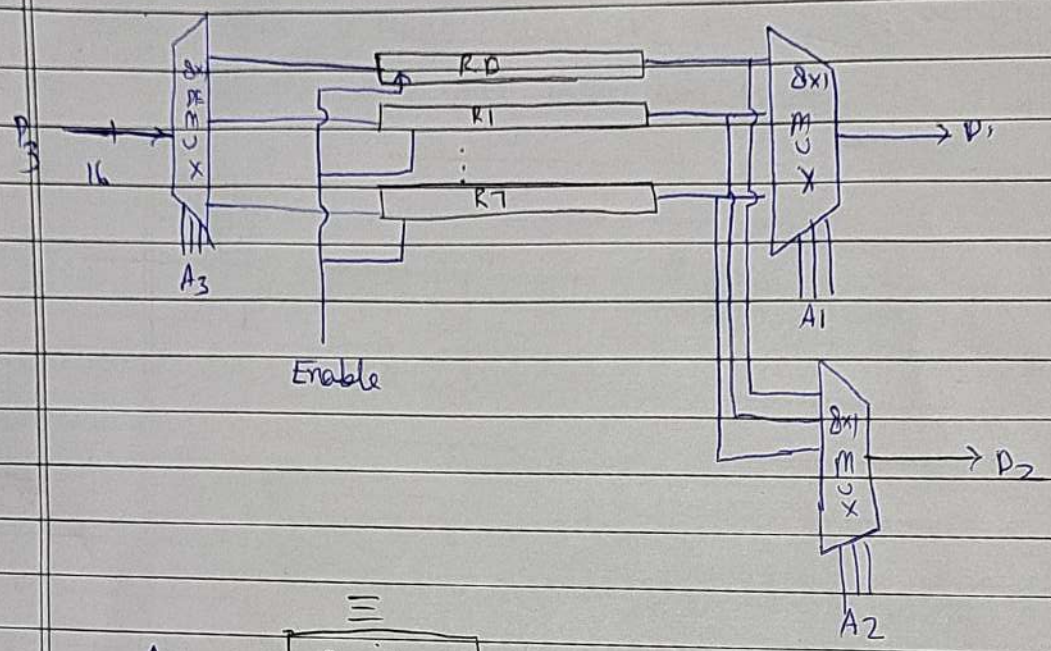
d. Imply



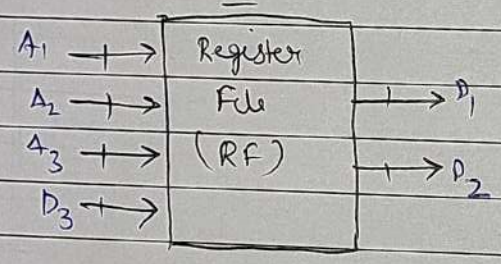

A
16
B
16
T1
T2
4LSB
4LSB
4bit
4bit
4-Bit Adder
4-Bit Adder
4-Bit Adder
16bit
16bit padding



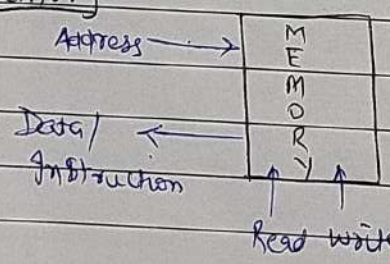
Register file (RF)



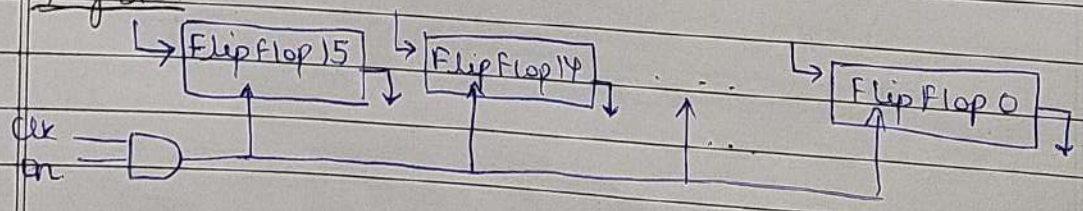
≡



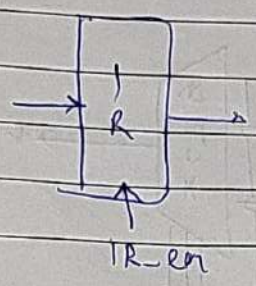
Memory



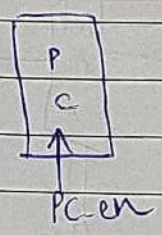
Register



Instruction Register



Program Counter



State Machine Description For instructions

For ADD, SUB, MUL, AND, ORA IMP.

① Tasks.

15	12 11	9 8	6 5	3 2	1 0
0---	RA	RB	RC	0	00

Instruction

State S_0: Fetch Instruction

Update P.C.

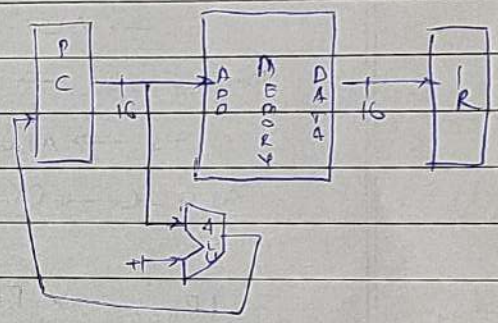
State S_1: Read operands

state S_2: Do operation

state S_3: Update Result.

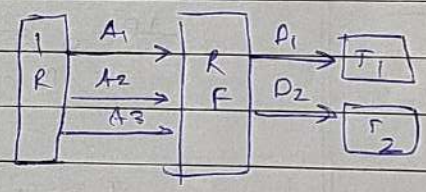
S_0:

operation / connection	signals
PC → Memory_Address	Memory_Read
→ ALU_A	ADD
Memory_Data → IR	PC_en
+1 → ALU_B	IR_en
ALU_C → PC	



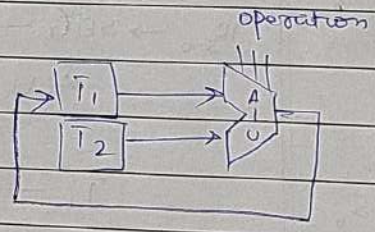
S_1

IR_{11-9} → RF_A1	RF_en
IR_{8-6} → RF_A2	T_1 - en
RF_D1 → T1	T_2 - en
RF_D2 → T2	



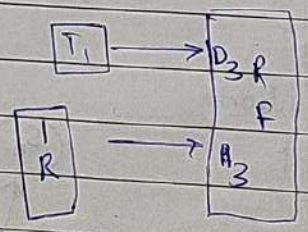
S_2

T_1 → ALU_A	operation
T_2 → ALU_B	IR_{10-8} → ALU_select
ALU_C → T_1	

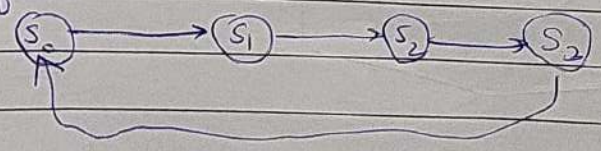


S_3

T_1 → RF_D3	RF_en
IR_{5-3} → RF_A3	



State Transition flow:



For ADI

Tasks

15 – 12	11 – 9	8 – 6	5 – 0
0001	RA	RB	Immediate

State S_0: Fetch Instruction
Update PC

State S_4: Read operand & Immediate

State S_5: Do operation

State S_6: Update Result

State S_0:

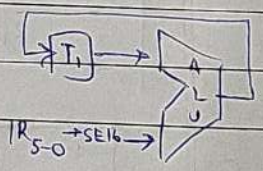
PC → Memory_Address	PC_en
→ ALU_A	Memory_Read
Memory_data → IR	Add
+1 → ALU_B	IR_en
ALU_C → PC	

State S_4:

IR_{11-9} → RF_A1	RF_enread
~~IR_{5-0} → SE16 → T2~~	T_1_en
RF_D1 → T1	~~T_2_en~~

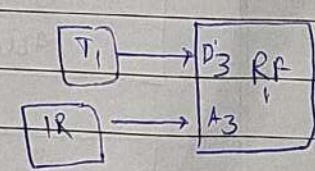
State S_5:

T_1 → ALU_A	ALU_Add select
IR_{5-0} → SE16 → ALU_B	T_1_en
ALU_C → T_1	

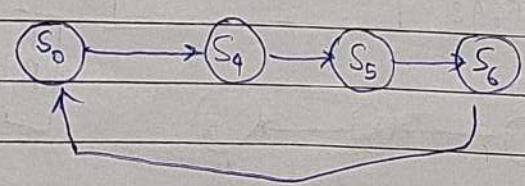


State S_6:

T_1 → RF_D3	RF_enwrite
IR_{8-6} → RF_A3	



State Transition Flow chart



Date: / /

Load Higher Immediate (LHI)

15 – 12	11 – 9	8 – 0
1000	RA	0 + 8 bit Imm

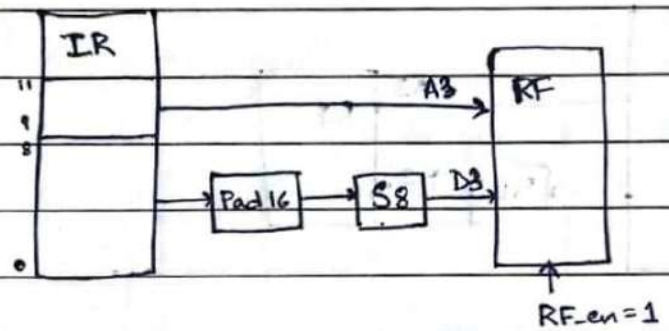
Tasks:

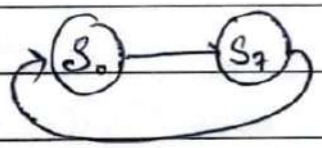
① Fetch Instruction
② Update PC
} → S_0

③ Understand Instruction
④ Update RF
} → S_7

S_7 :

$IR_{11-9} \rightarrow RF\_A3$	RF_en = 1
$IR_{8-0} \rightarrow Pad16 \rightarrow Shift8 \rightarrow RF\_D3$	





Date: / /

Load Lower Immediate (LLI)

15 … 12	11 … 9	8 … 0
1001	RA	0+8 bit Imm

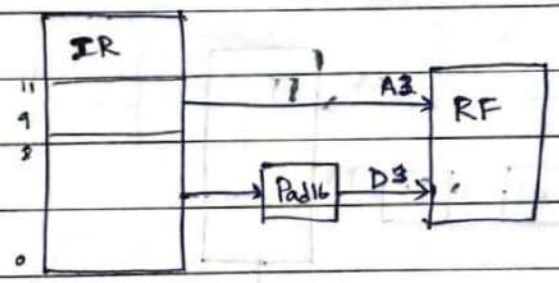
Tasks:

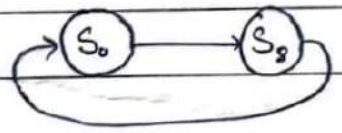
① Fetch Instouction
② Update PC
} → S_0

③ Understand Instouction
④ Update RF
} → S_8

S_8:

$IR_{11-9} \longrightarrow RF\_A3$	RF_en = 1
$IR_{8-0} \longrightarrow Pad\ 16 \longrightarrow RF\_D3$	





Date: / /

Load (LW)

15 … 12	11 … 9	8 … 6	5 … 0
1010	RA	RB	6 bit Imm.

Tasks:

① Fetch Instruction
② Update PC
} → S_0

③ Understand Instruction
④ Fetch Operand (Read RA)
} → S_9

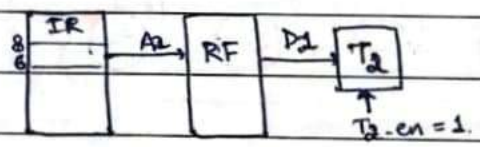
⑤ Compute Memory Address → S_{10}

⑥ Read Memory → S_{11}

⑦ Update RF → S_{12}

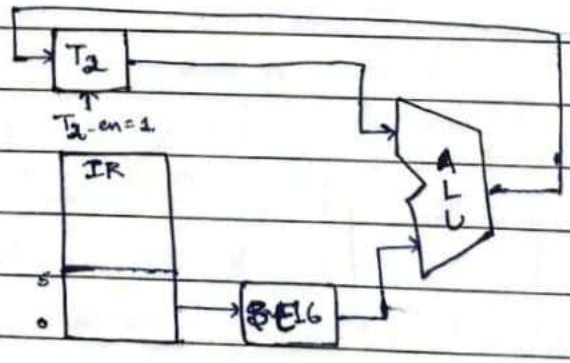
S_9 :

$IR_{8-6} \longrightarrow RF\_A2$	
$RF\_D2 \longrightarrow T_2$	$T_2\_en = 1$



S_{10} :

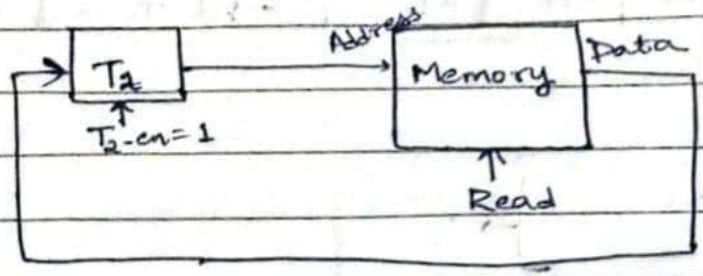
$T_2 \longrightarrow ALU\_A$	ADD
$IR_{5-0} \longrightarrow SE16 \longrightarrow ALU\_B$	$T_2\_en = 1$
$ALU\_C \longrightarrow T_2$	



Date: / /

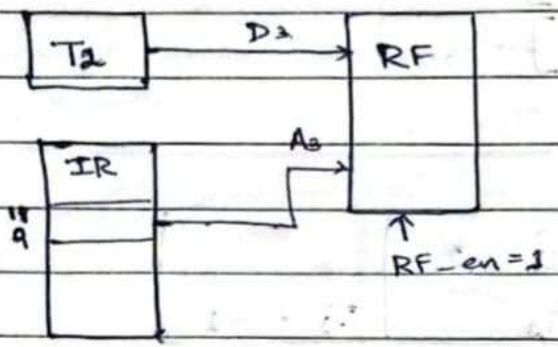
S_{11}:

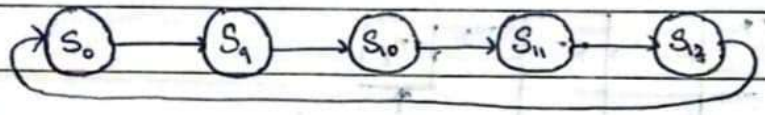
$T_2 \longrightarrow$ Mem_Add	Read
Mem_Data $\longrightarrow T_2$	T_2_en = 1



S_{12}:

$T_2 \longrightarrow$ RF_D3	RF_en = 1
$IR_{11-9} \longrightarrow$ RF_A3	





Date: / /

Store (SW)

15 – 12	11 – 9	8 – 6	5 – 0
1011	RA	RB	6-bit Imm.

Tasks:

① Fetch Instructions
② Update PC
} → S_0

③ Understand Instruction
④ Fetch Operand
} → S_{13}

⑤ Compute Memory Address → S_{14}

⑥ Transfer Data to Memory → S_{15}

S_{13}:

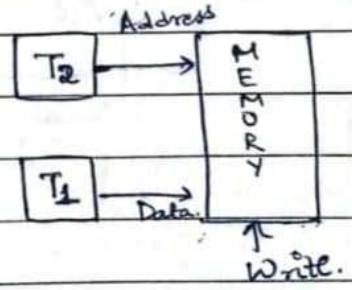
$IR_{11-9} \longrightarrow RF\_A1$	
$IR_{8-6} \longrightarrow RF\_A2$	
$RF\_D1 \longrightarrow T_1$	$T_1\text{-}en = 1$
$RF\_D2 \longrightarrow T_2$	$T_2\text{-}en = 1$

S_{14}:

$T_2 \longrightarrow ALU\_A$	ADD
$IR_{5-0} \longrightarrow SE16 \rightarrow ALU\_B$	
$ALU\_C \longrightarrow T_2$	$T_2\text{-}en = 1$

S_{15}:

$T_2 \longrightarrow Mem\_Add$	Write
$T_1 \longrightarrow Mem\_Data$	



Date: / /

Branch on Equality (BEQ)

1100	RA	RB	6-bit Imm

Tasks:

① Fetch Instruction
② Update PC
} → S_0

③ Understand Instruction
④ Fetch Operands (Read RA, RB)
} → S_{16}

⑤ Check for equality → S_{17}

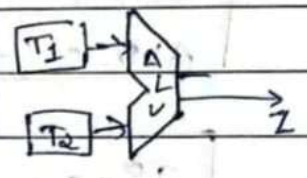
⑥ If $R_1 == R_2$, Update PC. → S_{18} & S_{19}

S_{16}:

$IR_{11-9} \rightarrow RF\_A1$	
$IR_{8-6} \rightarrow RF\_A2$	
$RF\_D1 \rightarrow T1$	$T1\_en = 1$
$RF\_D2 \rightarrow T2$	$T2\_en = 2$

S_{17}:

$T1 \rightarrow ALU\_A$	SUB
$T2 \rightarrow ALU\_B$	



S_{18}:

$PC \rightarrow ALU\_A$	SUB
$+1 \rightarrow ALU\_B$	
$ALU\_C \rightarrow T1$	

S_{19}:

$T1 \rightarrow ALU\_A$	ADD
$IR_{5-0} \rightarrow SE16 \rightarrow ALU\_B$	
$ALU\_C \rightarrow PC$	$PC\_en = 1$

Date: / /

Jump And Link (JAL)

1101	RA	9-bit Imm

Tasks:

1. Fetch Instruction → S_0
2. Update PC
3. Understand Instruction → S_{20}
4. Finding PC before updation
5. Storing PC in RF → S_{22}
6. PC = PC + Imm → S_{21}

S_{20}:

PC → ALU-A	SUB
+1 → ALU-B	
ALU-C → T1	

S_{21}:

T_1 → ALU-A	ADD
IR_{8-0} → SE16 → ALUB	
ALU-C → PC	PC_en = 1

S_{22}:

IR_{11-9} → RF-A3	RF_en = 1
T_1 → RF-D3	

Date: / /

Jump and Link Register (JLR)

1111	RA	RB	000000

Tasks:

① Fetch Instruction
② Update PC
} → S_0

③ Understand Instruction → S_{23}

④ Finding PC before updation → S_{24}

⑤ Store previous PC in RF
⑥ PC = PC + Imm.
} → S_{25}

S_{24}:

PC → ALU-A	SUB
+1 → ALU-B	
ALU-C → T_1	T_1-en

S_{25}:

IR_{11-9} → RF.A3	RF-en = 1
T_1 → RF-D3	
T_2 → PC	PC-en = 1

Jump Unconditionally (J)

15 – 12	11 – 9	8 – 0
1110	R_A	Immediate

Tasks:

1. Fetch instruction
2. Update PC
3. Understand
4. Compute new address
5. Update PC

S_1

$IR_{11-9} \rightarrow RF\_A1$	
$IR_{8-6} \rightarrow RF\_A2$	
$RF\_D1 \rightarrow T1$	$T_1\_en = 1$
$RF\_D2 \rightarrow T2$	$T_2\_en = 1$

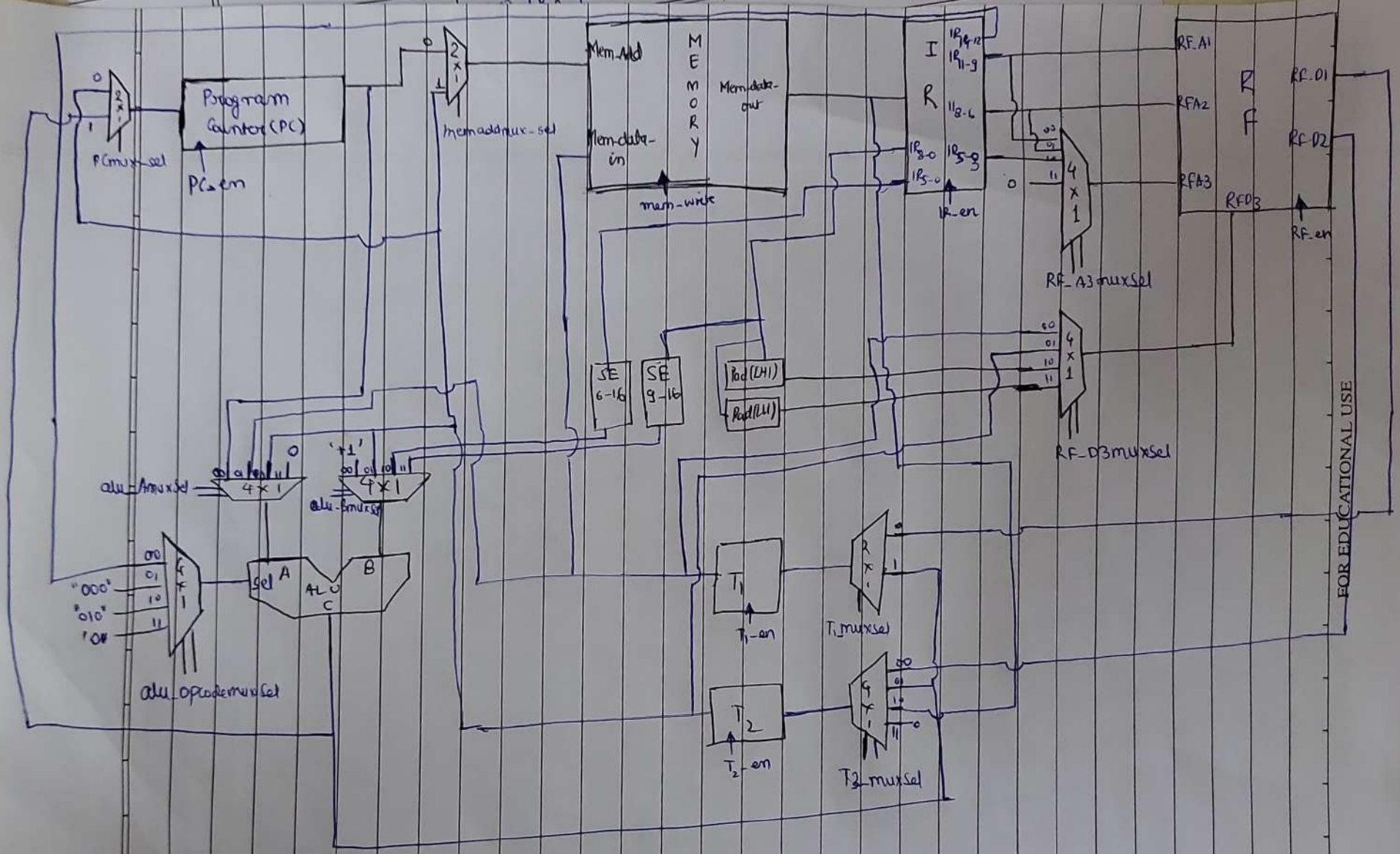
S_{26}

$PC \rightarrow ALU\_A$	ALU-select = SUB
$+1 \rightarrow ALU\_B$	~~PC-en~~ T_1-en
$ALU\_C \rightarrow T_1$	

S_{27}

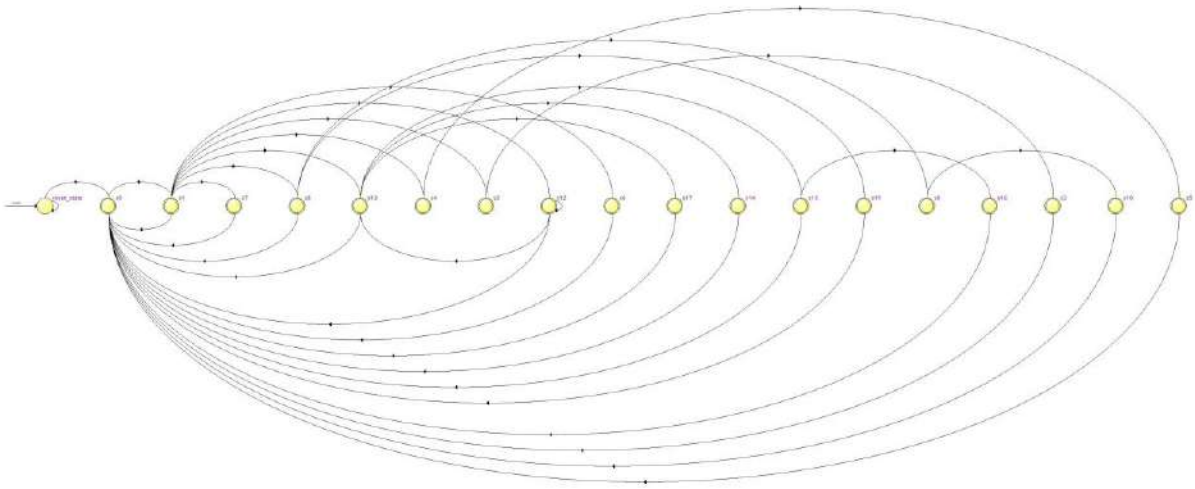
$T_1 \rightarrow ALU\_A$	ALU-select = ADD.
$IR_{8-0} \rightarrow \boxed{SE16} \rightarrow ALU\_B$	PC-en
$ALU\_C \rightarrow PC$	

$S_1 \rightarrow S_{26} \rightarrow S_{27}$


Program Counter (PC)
PCmux_sel
PC_en
memaddmux_sel
Mem_Add
Mem_data-in
MEMORY
Mem_data-out
mem_write
IR
IR_en
RF_A1
RFA2
RFA3
RF
RF_D1
RF_D2
RFD3
RF_en
RF_A3muxSel
RF_D3muxSel
SE 6-16
SE 9-16
Pad(LH1)
Pad(LH1)
alu_Amux Sel
alu_Bmux Sel
'+1'
alu_opcodemuxSel
"000"
"010"
"011"
A
B
ALU
C
T1
T1_en
T1_muxsel
T2
T2_en
T2_muxsel
FOR EDUCATIONAL USE

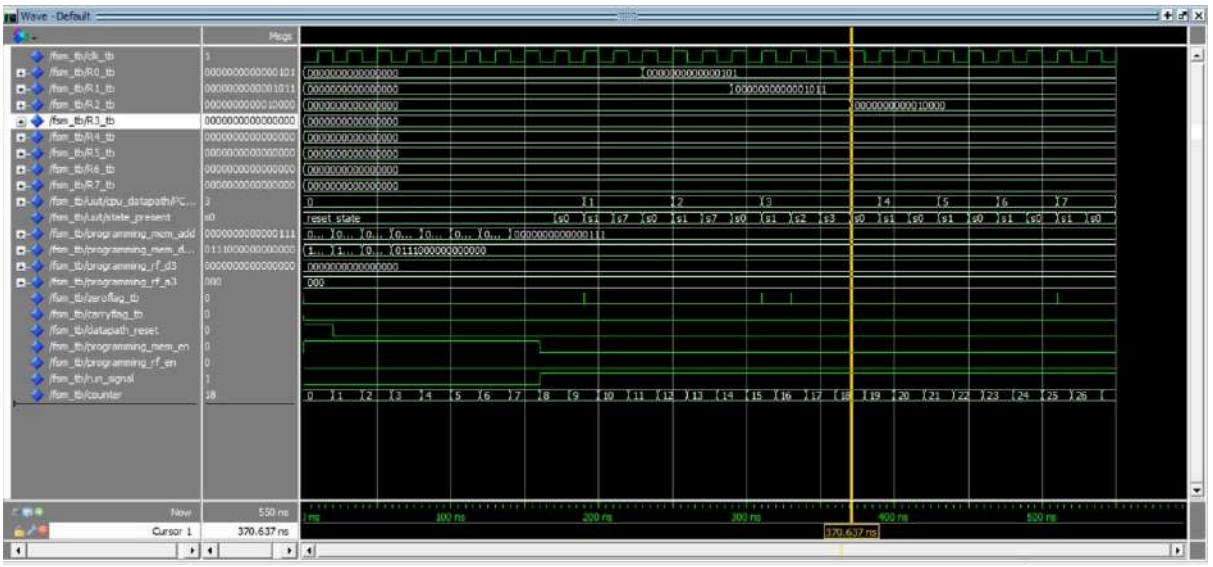
DATAPATH

CPU	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17
IR_en	1																	
T1_en		1	1		1									1				
T2_en		1							1	1								
PC_en	1														1	1		1
RF_en				1		1	1	1			1						1	1
mem_read																		
mem_write												1						
alu_sel(1 downto 0)	01	11	00	11	01				01				10	10	01	01		
alu_muxA_sel	00		01		01				10				01	00	01	01		
alu_muxB_sel	00		01		10				10				01	00	10	11		
T2_mux_sel		00							01	10								
T1_mux_sel		0	1		1									1				
PC_mux_sel	1														1	1		0
RF_D3_mux_sel				00		00	10	11			01						00	00
RF_A3_mux_sel				10		01	00	00			00						00	00
mem_address_mux_sel	0									1		1						
	Control Signals																	
	Note: Blank Columns represent Don't Care signal(X)																	



Various Programs to test 15 Instructions

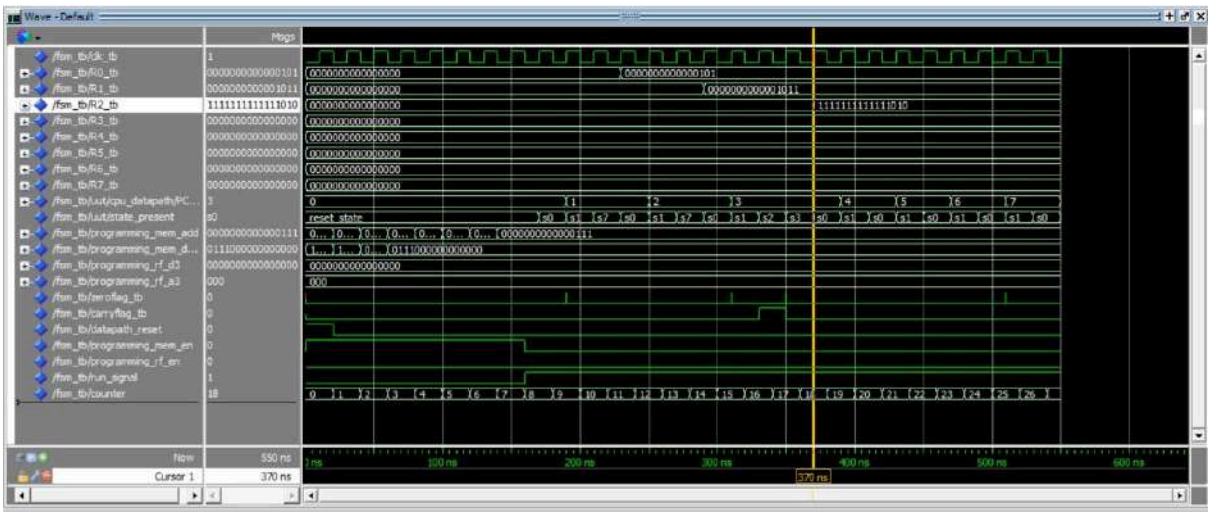
1. ADD:



```
C/C++
LLI R0,5
LLI R1,11
ADD R2,R0,R1
```

- Therefore R2 has 16 stored in it.

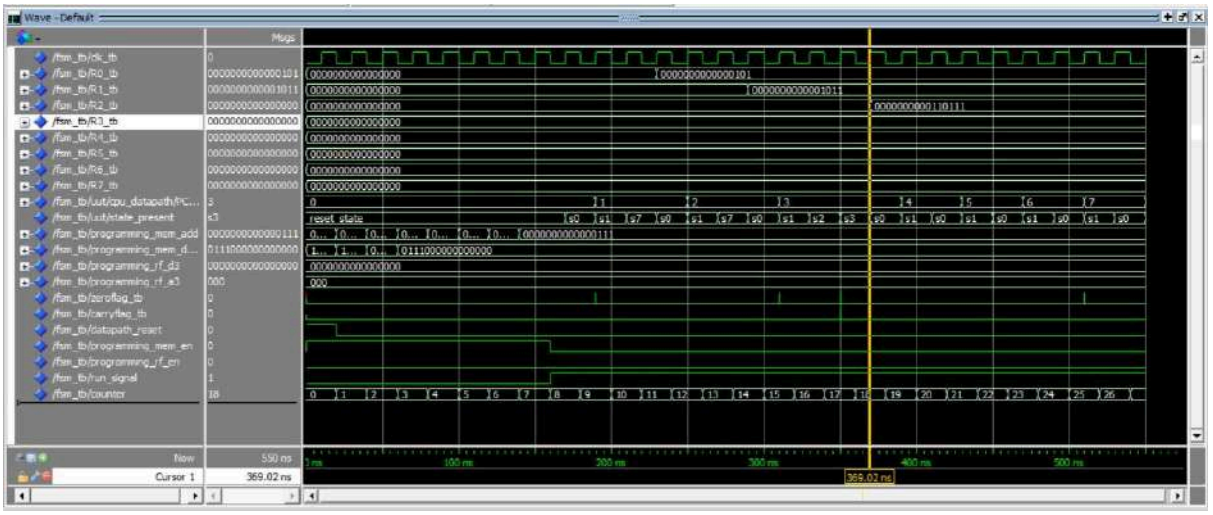
2. SUB:



```
C/C++
LLI R0,5
LLI R1,11
Sub R2,R0,R1
```

- Therefore R2 has -6 (in 2s complement) stored in it.

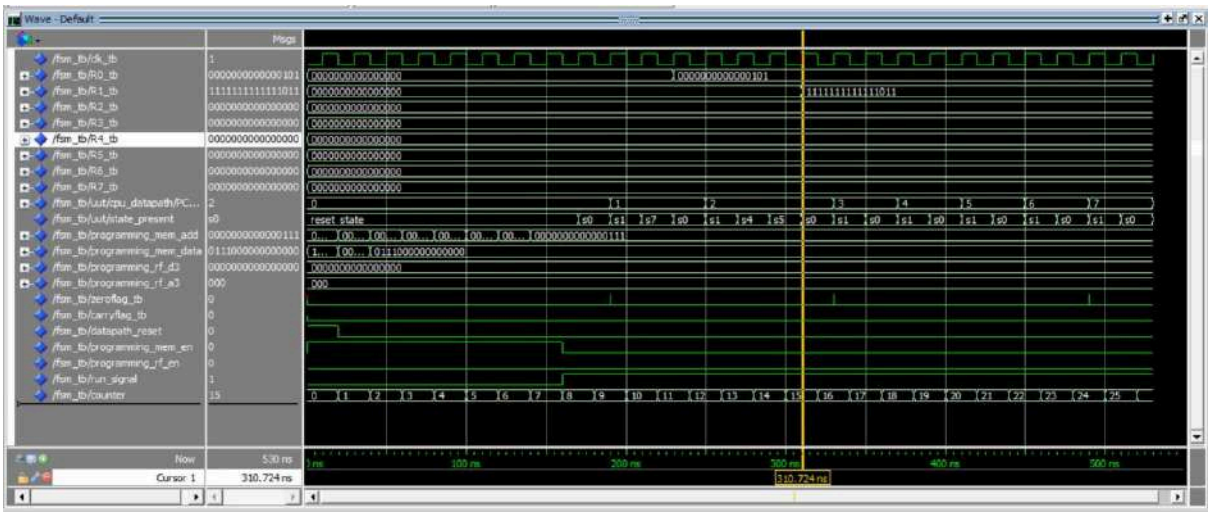
3. MUL:



```
C/C++
LLI R0,5
LLI R1,11
MUL R2,R0,R1
```

- Therefore R2 has 55 stored in it.

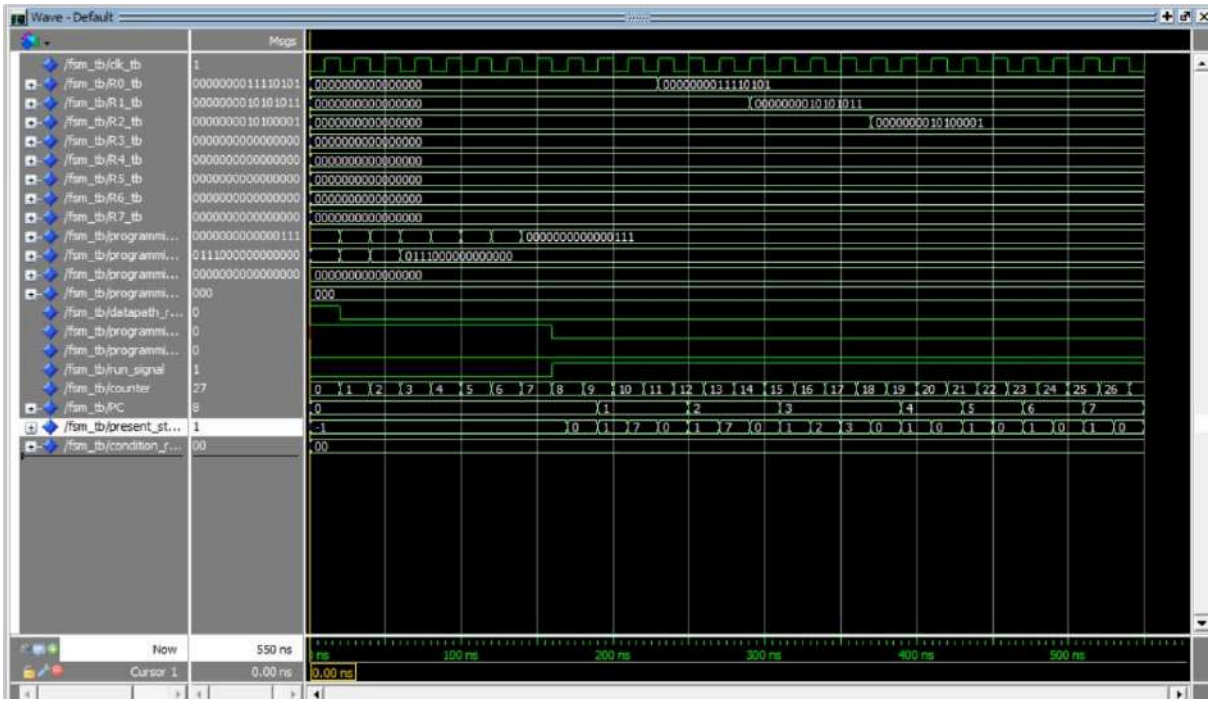
4. ADI:




```
C/C++
LLI R0,5
ADI R1,R0, -10
```

- Therefore R2 has -5 (in 2s complement) stored in it.

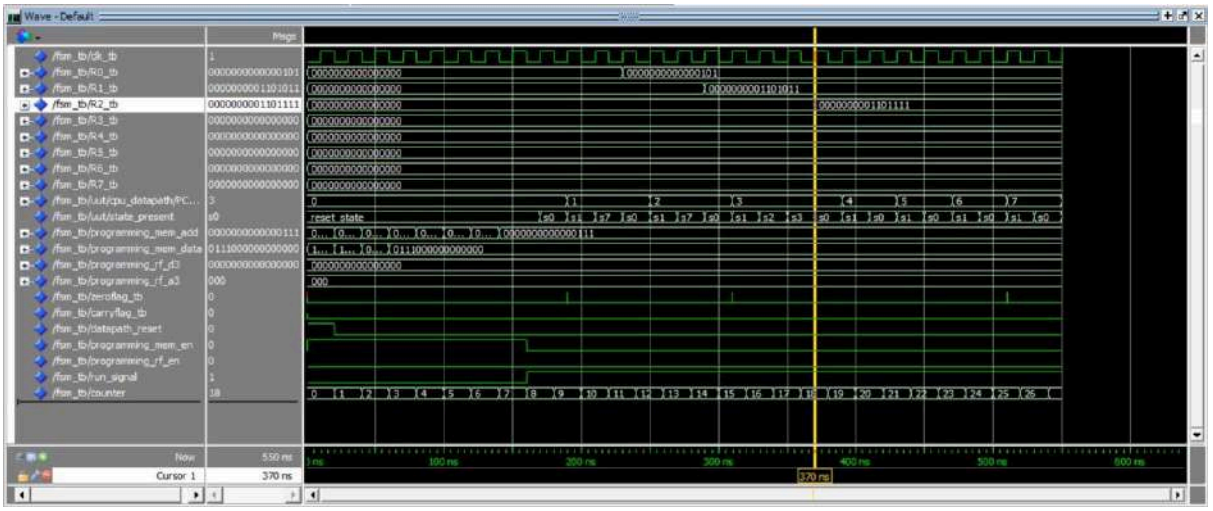
5. AND:



```
C/C++
LLI R0,011110101
LLI R1,010101011
AND R2,R0,R1
```

- Data is loaded in R0 and R1 and their logical AND is stored in R2

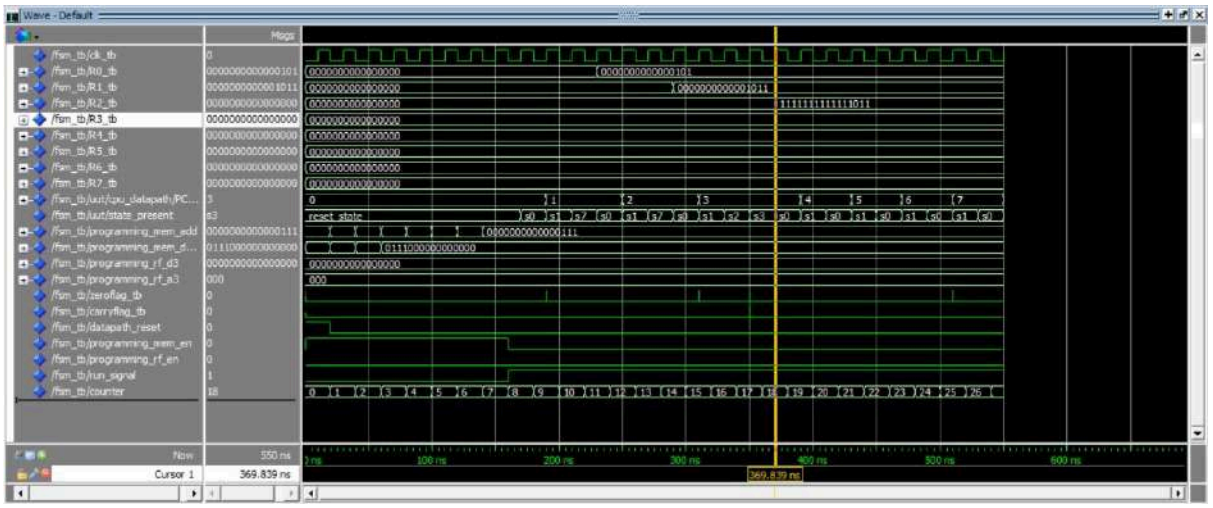
6. ORA:



```
C/C++
LLI R0,00000101
LLI R1,01101011
ORA R2,R0,R1
```

- Therefore R2 has 0000000001101111 stored in it.

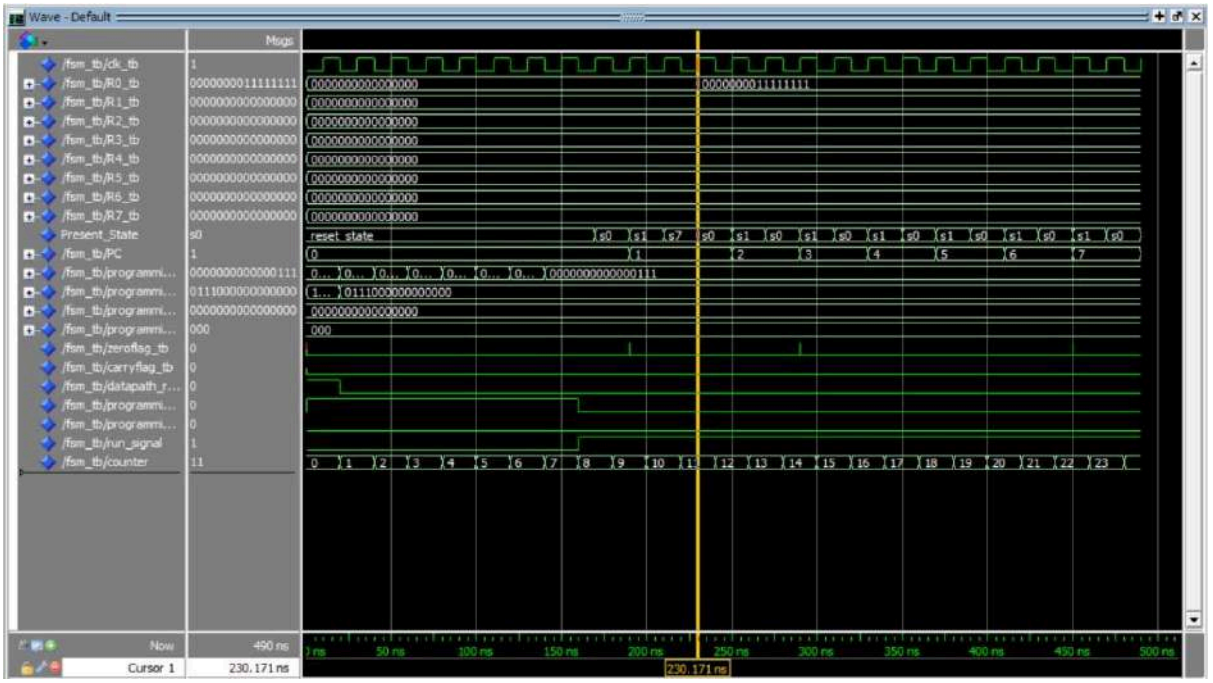
7. IMP:




```
C/C++
LHI R0,00000101
LHI R1,00001011
ADD R2,R0,R1
```

- Therefore R2 has 111111111111011 stored in it.

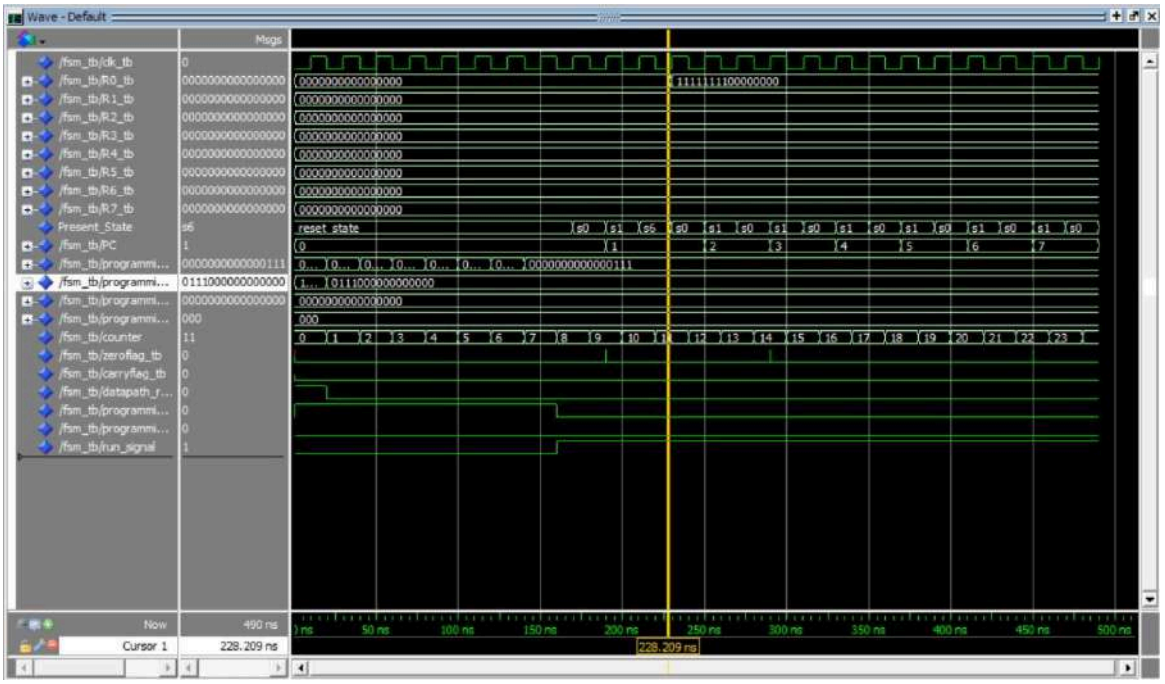
8. LLI:



```
C/C++
LLI R0, R1, (011111111)
```

- Therefore 11111111 loaded in 8 LSBs of R0

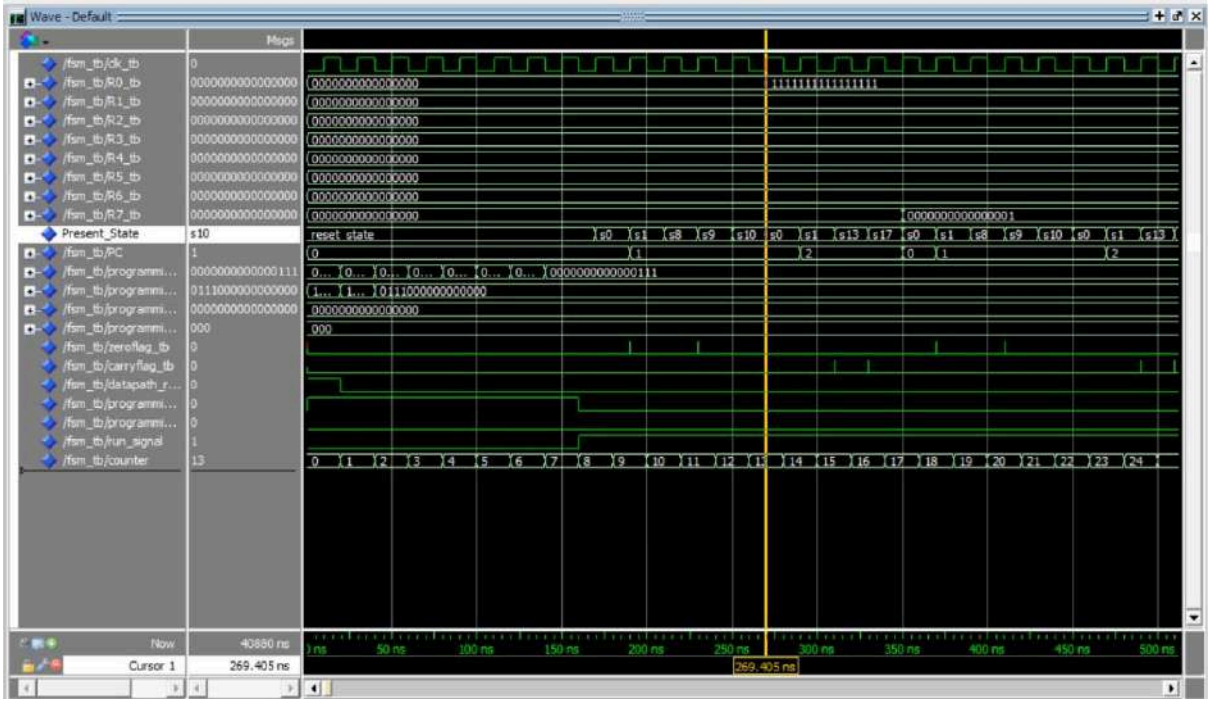
9. LHI:




```
C/C++
LHI R0, (011111111)
```

- Therefore 11111111 loaded in 8 MSBs of R0

10. LW:

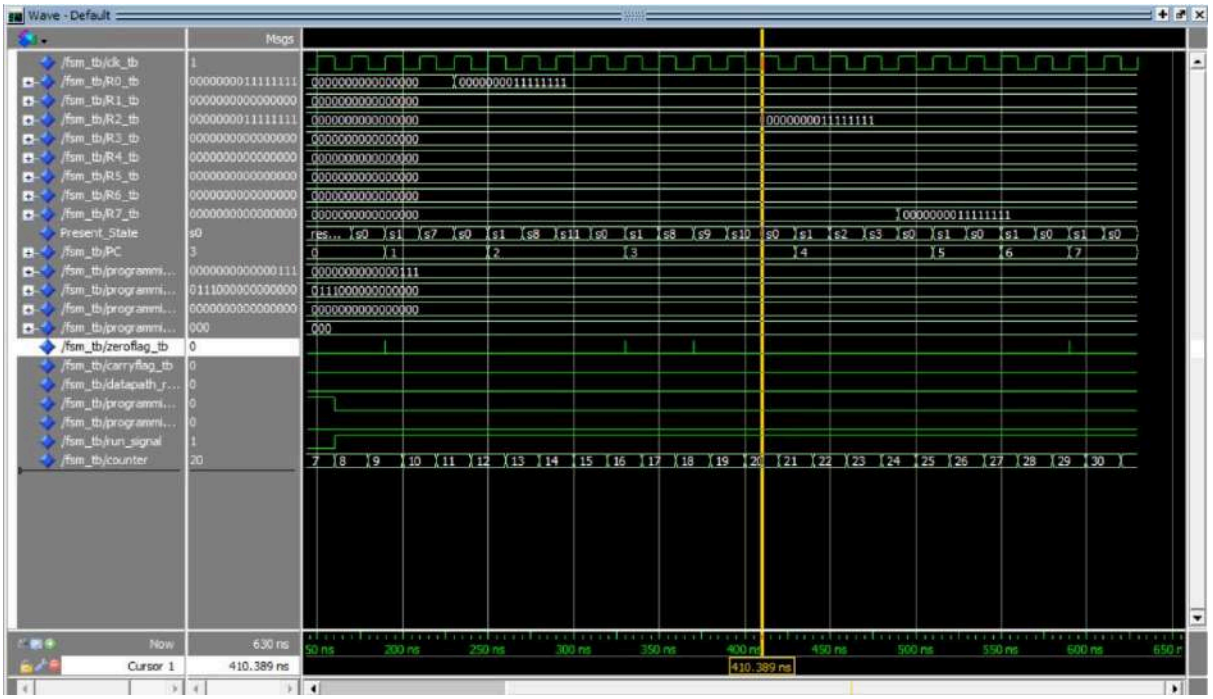



```
C/C++

LW R0, R1, 1
1111111111111111
```

- Therefore 1111111111111111 loaded in R0. (R1 had 0 and offset is 1)

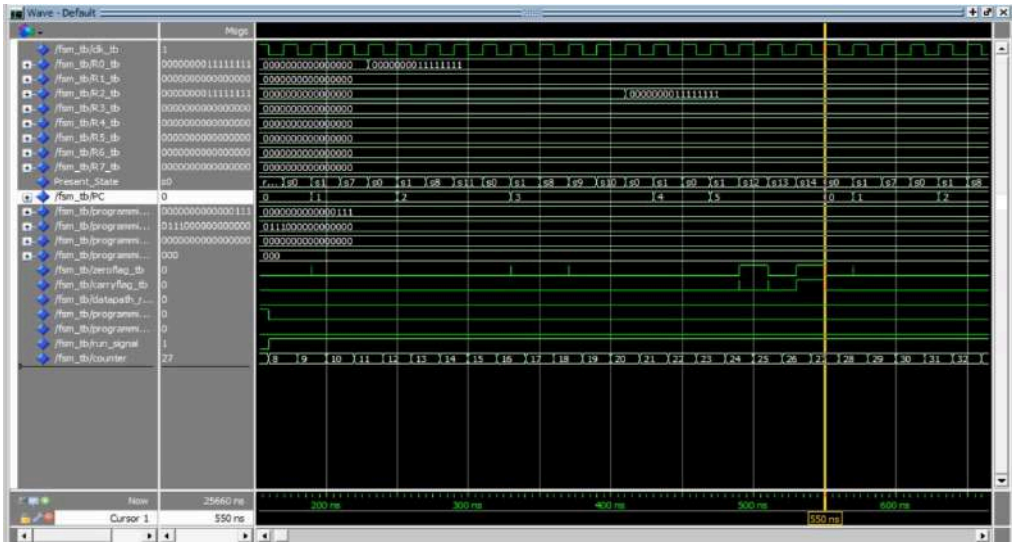
11. SW



```
C/C++
LLI  R0, 011111111
SW R0, R1, 3
LW R2, R1, 3
```

- First load a data into R0
- Store it at address R1+3
- Load the data at R1+3 into R2

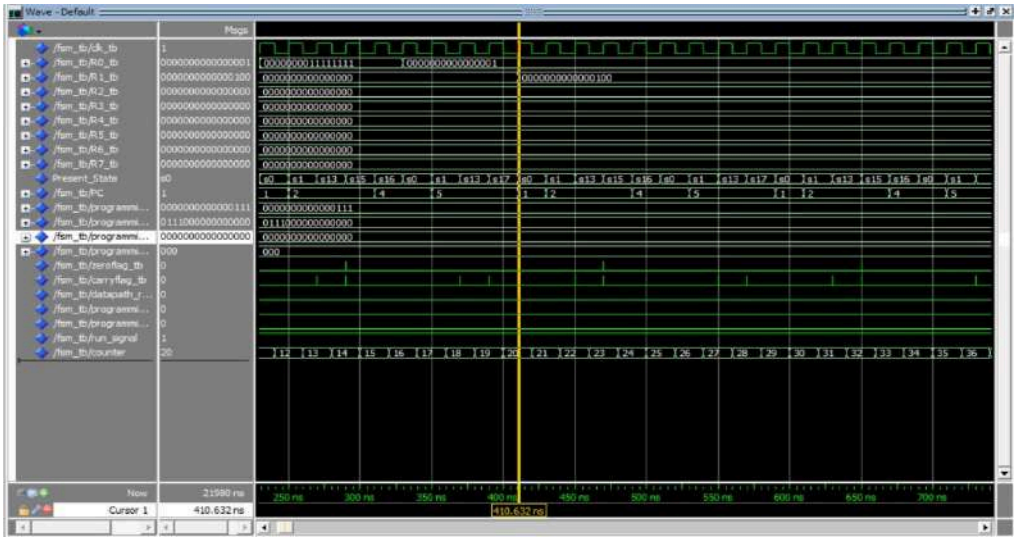
12. BEQ:



```
C/C++
LLI  R0, 011111111
SW R0, R1, 3
LW R2, R1, 3
-
BEQ R0, R2, -4
```

- Load data in R0 and R2 similar to previous code and BEQ if the the values are equal to start of the program

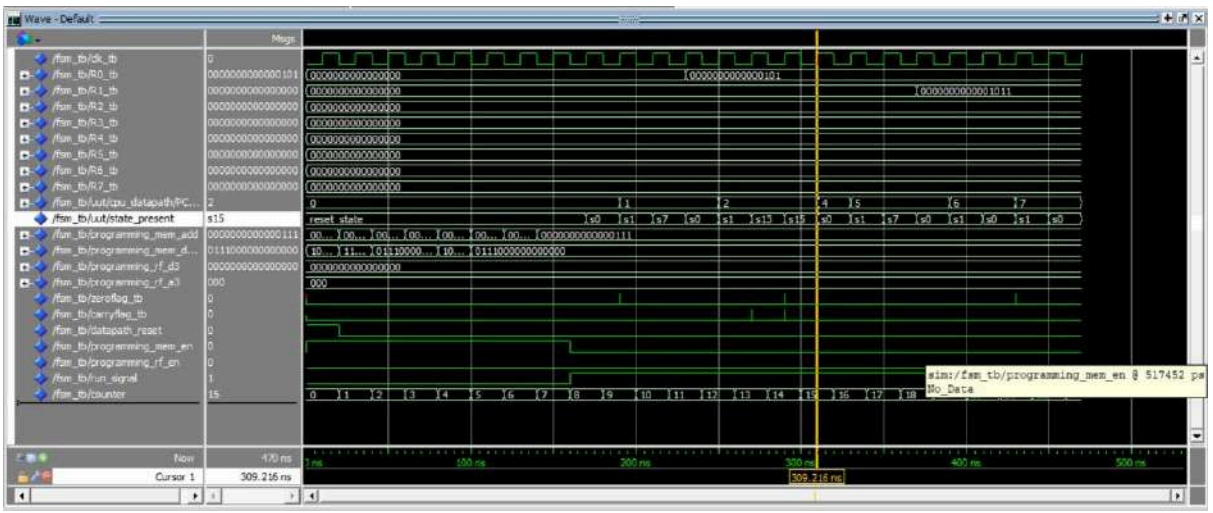
13&14. JAL and JLR:



```
C/C++
JAL R0, 3
-
-
JLR R1, R0, 0
```

- Jump to the 4th instruction and store current value of PC in R0
- Now jump to the PC stored in R0 and store this PC in R1

15. J



```
C/C++
LLI R0,5
```

```
J 3
-
-
-
LLI R1,11
```

- We have jumped from the PC=1 to PC=4 using the J 3 instruction.

Implementing a Simple Program-

- Pseudo code of program-

```
for(i=0;i<g;i++)
{
    if(isEven(arr[i])==0){
        arr[i] = arr[i]*2;
    }
    else{
        arr[i] = arr[i]-2;
    }
}
isEven(int a){
    if(a & 1 == 0){
      return 1;
    }
    return 0;
}
```

- Assembly code of the above program

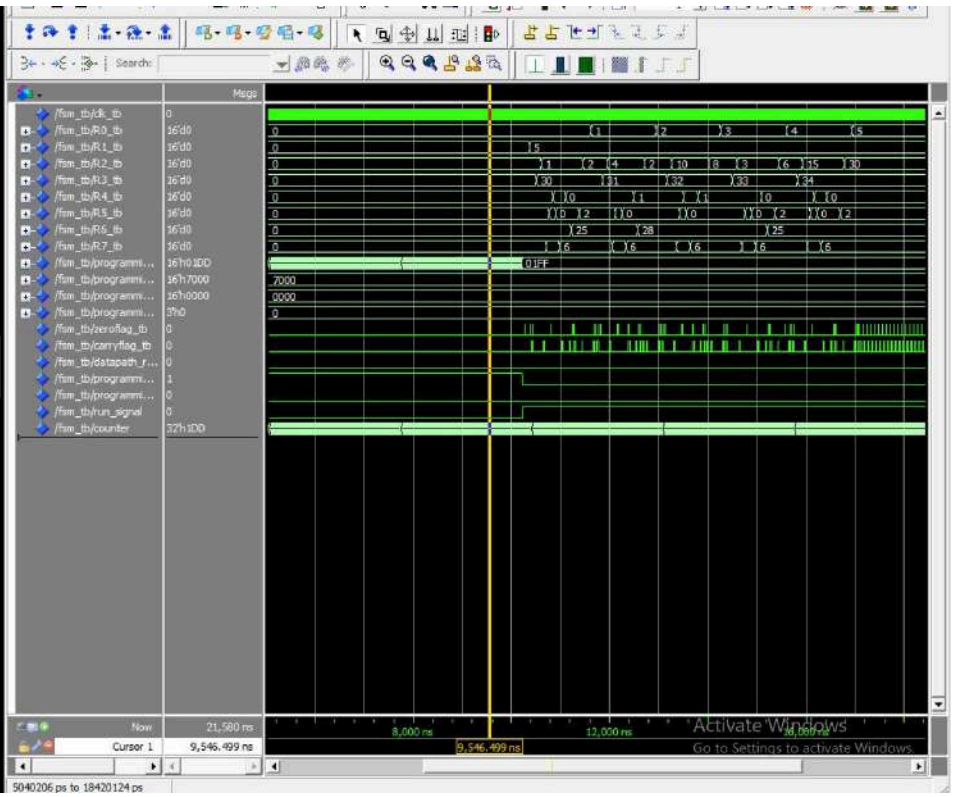
```
C/C++
lhi r0, 0
lli r1, 5
beq  r0, r1, 12
adi r3, r0, 31
lw   r2, r3, 0
```

```
jal  r7, 13
beq  r4, r5, 3
adi  r2, r2, -2
j 3
lli r5, 2
mul  r2, r2, r5
sw   r2, r3, 0
adi r0, r0, 1
j -11
j 0
-
-
-
lli  r5, 1
and r4, r2, r5
lli r5, 0
adi r7, r7, 1
beq  r4, r5, 5
lli  r4, 0
jlr r6, r7
-
-
lli  r4, 1
jlr  r6, r7
-
data 1
data 4
data 10
data 3
data 15
```

Implementation-

code.txt

```
lhi r0, 0
lli r1, 5
beq  r0, r1, 12
adi r0, r3, 30
lw   r2, r3, 0
jal  r7, 13
beq  r4, r5, 3
adi  r2, r2, -2
j 3
lli r5, 2
mul  r2, r2, r5
sw   r2, r3, 0
adi r0, r0, 1
j -11
j 0
nop
nop
nop
lli  r5, 1
and r4, r2, r5
lli r5, 0
adi r7, r7, 1
beq  r4, r5, 5
lli  r4, 0
nop
jlr r6, r7
nop
lli  r4, 1
jlr  r6, r7
nop
data 1
data 4
data 10
data 3
data 15
nop
```



Work Distribution—

Aniket	Apoorv	Kushaal	Shreya
• ALU • DataPath code • Small components code • Assembler script	• FSM code verification • DataPath flow diagram • Final documentation • Program Ideation and debugging	• FSM code • Testing and verification • Control flow instructions • Control Bits documentation	• Component codes • Report • pen paper designs • Individual instruction Program design