SUMMER PROJECT REPORT

Course: MEL G642 **Date**: 10th June 2023

Name: VLSI Architecture Design **Sem**: 2nd / HD

Submitter Detail:

Bharathi Shrinivasan T R

2022H1400182P (M.E. Embedded Systems)

Objective: Understand and comprehend the design philosophy of CISC based Processor architecture.

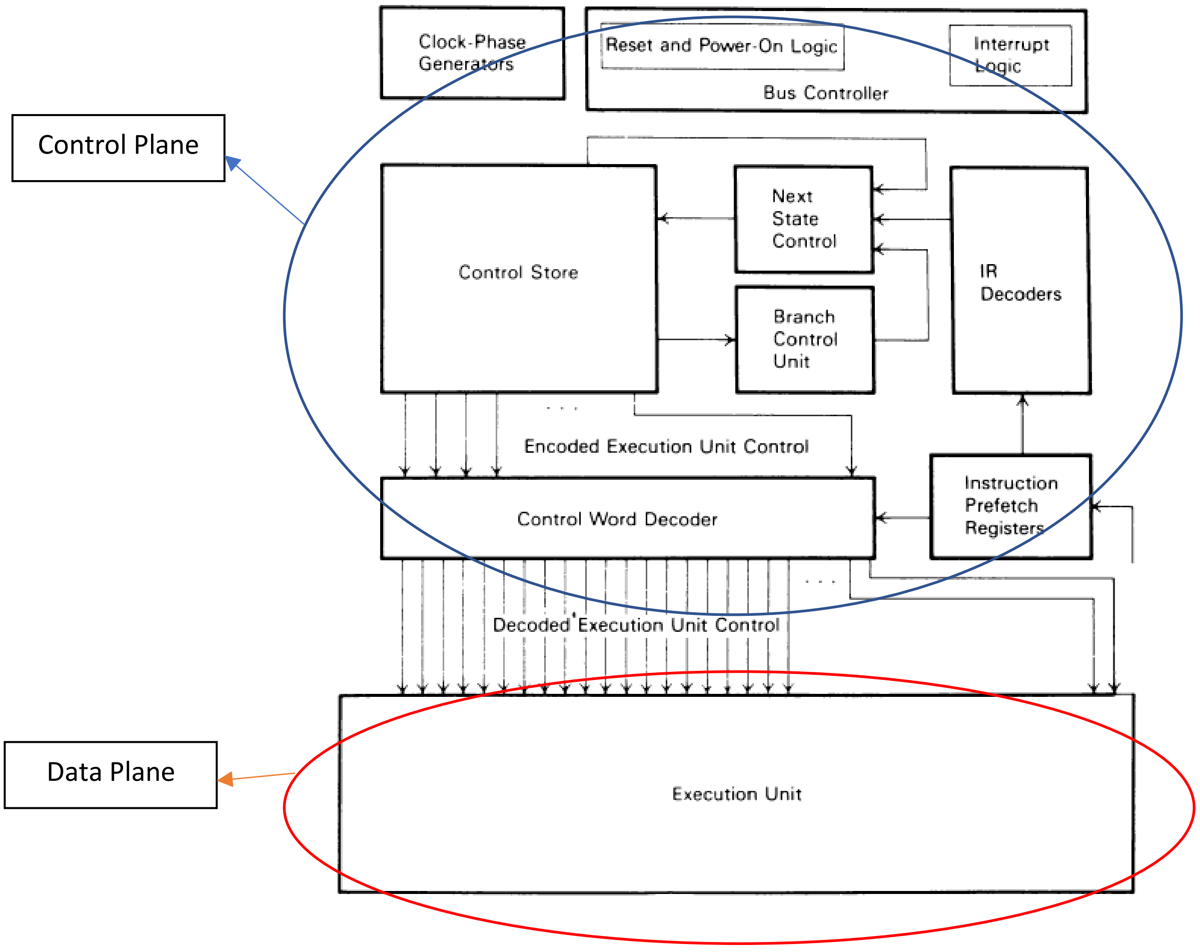
Reference: "*Microprocessor Logic Design: Flowchart Method*" - Nick Tredennick

Project:

1. Learn flowchart method-based development of CISC architecture design.
2. Develop a custom 16-bit MIN (Minimum) architecture prototype using HDL modelling of the logic circuits and implementing same on Xilinx's ZedBoard (Zynq 7000 series FPGA).
3. Machine-code a program to compute Fibonacci series using the implemented processor.

Project database links:

1. Complete Vivado (2020.2) source code file: Drive Link
2. Simulation results and report: Drive Link



1.1 Block diagram of a complete CISC architecture

Table of Contents

Design Workflow carried out during this project

The Design of microprocessor majorly has two phase – Data-path design, Control-path design. This section shows the water-flow model of the work carried in this complete design.

Define Programmer's Instruction set supported list. And its opcode encoding to 16-bit instruction format. List them with corresponding 16-bit opcode.

↓

Design Execution Unit components / Data path.

(i) Programmer's register bank {R0-R7}

(ii) Arithmetic Logic Unit (ALU) with all supporting operations as comitted previous.

(iii) Memory unit

(iv) Design a complete blueprint of EU with all the above componets integrated

↓

Define Decoded Control Word format. Redefine the EU components as neccessary to accomodate DCW.

↓

In a spreadsheet, get DCW for few micro-code (refering the flowchart diagram of MIN). Refine DCW and EU components if any required.

↓

Integrate the EU componets. Test the EU execution for few microcode/stateID.

↓

Design Control path componets.

(i) Design Control store with ROM programmed with control word of MIN microcode referring MIN flowchart and prepated control store list for all stateIDs.

(ii) Design Instruction Decoder

(ii) Design NextState Logic

↓

Devise adequate clocking strategy (clock phase) to individual componets in control path and execution path.

↓

Test the control path alone to various instructions and its control flow / stateIDs it traverse. Verify it with the MIN flowchart.

↓

Integrate both Datapath and Controlpath. Provide proper clock phase as planned. And feed the RAM memory for user program (Fibonnaci). Test the result.

Programmer’s Instruction format and list of supporting instructions

Programmer's Instruction	Operation
NOP	No Operation: No change in Reg/MEM/flag.
LDR Rx,Ry	Load: Rx <- Ry , Indirect mode Rx<-@Ry
STR Rx,Ry	Store: Rx->Ry, indirect mode Rx->@Ry
TST Ry	Test: @Rx is read in ALU and only flags are modified. No change in Reg/MEM/flag.
BZ Ry	If flag is Zero jumps to Ry pointing instruction. No change in Reg/MEM/flag.
ADD Rx,Ry	Add: Ry<-Rx+Ry
SUB Rx,Ry	Subract: Ry<-Rx-Ry
AND Rx,Ry	Bitwise AND: Ry<-Rx&Ry
OR Rx,Ry	Bitwise OR: Ry<-Rx|Ry
XOR Rx,Ry	Bitwise XOR: Ry<-Rx^Ry
PUSH Rx,Ry	Ry<-Ry-1, @Ry<-Rx (Fully Decending stack)
POP Rx,Ry	Rx<-@Ry, Ry<-Ry+1 (Fully Decending stack)

Each instruction is 16-bit in width, bit field anatomy is given as below. The supported list of programmer’s instruction along with custom assigned Op-Code is listed in the above figure.

	15-12	11-10	9-8	7	6-4	3	2-0
Programmer's	4-bit	2-bit	2-bit	1-bit	3-bit	1-bit	3-bit
Instruction	**Op Code**	**Future**	**Mode**	**Future**	**Rx**	**Future**	**Ry**
NOP	0	0	00: direct	0	R0	0	R0
LDR	1	0	01: indirect	0	R1	0	R1
STR	2	0	10: indirect+base	0	R2	0	R2
TST	3	0		0	R3	0	R3
BZ	4	0		0	R4	0	R4
ADD	5	0		0	R5	0	R5
SUB	6	0		0	R6	0	R6
AND	7	0		0	R7	0	R7
OR	8	0		0		0	
XOR	9	0		0		0	
PUSH	10	0		0		0	
POP	11	0		0		0	

Reference –

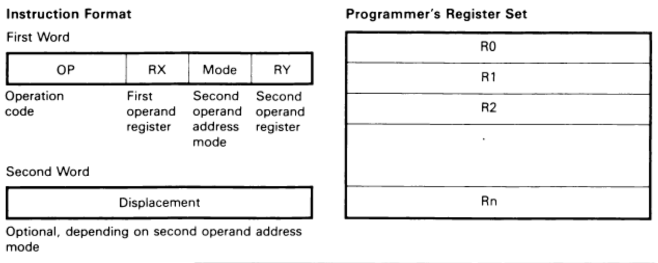
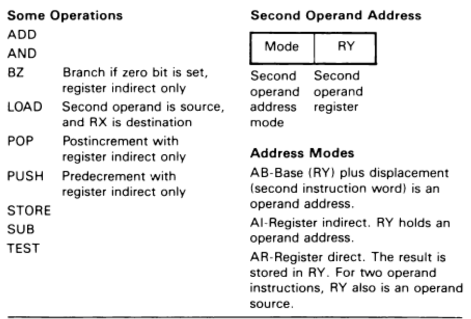


Figure 3.2 MIN instruction format and register set



Some Operations

ADD	
AND	
BZ	Branch if zero bit is set, register indirect only
LOAD	Second operand is source, and RX is destination
POP	Postincrement with register indirect only
PUSH	Predecrement with register indirect only
STORE	
SUB	
TEST	

Second Operand Address

Mode	RY
Second operand address mode	Second operand register

Address Modes

AB-Base (RY) plus displacement (second instruction word) is an operand address.

AI-Register indirect. RY holds an operand address.

AR-Register direct. The result is stored in RY. For two operand instructions, RY also is an operand source.

Figure 3.3 MIN instruction set summary

Few examples of instructions and it's opcode -

Instruction	**OpCode (Hex)**
LDR R5,R7	16'h1057
LDR R4,[R5]	16'h1145
LDR R6,[R7+10]	16'h1267
	16'h000A
BZ R6	16'h4006

The Op-code- 1st nibble corresponds to the operation/instruction, 2nd nibble is the addressing mode, 3rd is Rx selection, 4th is Ry selection.

Note: for Indirect addressing plus displacement mode, the Opcode generated is 16+16bits / 4byte long – the first 2byte is the instruction opcode, second 2byte is the displacement value.

Control Word Format -

Control Word		
26 bits	2 bits	6 bits
Decoded Control Word	**TY selection**	**Next Address**

The Control Store contains Control Word for each State-ID. Note, in this architecture we didn't use a dedicated Control Word decoder unit, so the ready decoded control word is stored in Control Word itself. The bit size and field anatomy is as above.

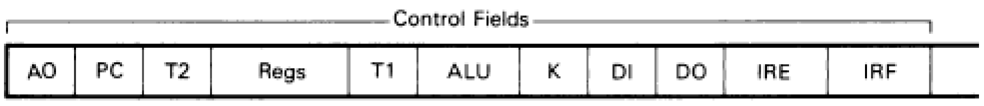
Decoded control word format used by Execution Unit –

		bit_IreIrf	bit_SrcAO	bit_WriteDO	bit_DesEdb	bit_SrcABus	bit_SrcBBus
bit position ->		25	23:24	22	20:21	17:19	14:16
No of bits ->		1	2	1	2	3	3
	State ID	**IRF->IRE**	**Memory Operation**			**Src -> A Bus**	**Src -> B Bus**
			Src -> AO	A->DO	MEM->dest		
ControlStore_Address	XXXX	0	A->AO : 01 B->AO : 10 None : 00		edb->DI : 01 edb->IRF : 10 None : 00	Rx->A -001 Ry->A -010 T1->A -011 T2->A -100 PC->A -101 None : 000	Rx->B -001 Ry->B -010 T1->B -011 T2->B -100 PC->B -101 DI->B -110 None : 000
0	NOP	0	00	0	00	000	000

		bit_SrcRx	bit_SrcRy	bit_SrcT2	bit_SrcPC	bit_2ndOperand	bit_IntExtOp	bit_flagUpdate	bit_T1Update	bit_EnableALU
bit position ->		12:13	10:11	8:9	6:7	4:5	3	2	1	0
No of bits ->		2	2	2	2	2	1	1	1	1
	State ID	**Bus Destination**				**ALU Operation**				
		Rx	Ry	T2	PC	2nd Operand	Int/Ext Operation	UpdateFlag	UpdateT1	EnableALU
ControlStore_Address	XXXX	A->Rx : 01 B->Rx : 10 None : 00	A->Ry : 01 B->Ry : 10 None : 00	A->T2 : 01 B->T2 : 10 None : 00	A->PC : 01 B->PC : 10 None : 00	B->ALU : 00 +1->ALU : 01 -1->ALU : 10 0->ALU : 11	External : 1 Internal/ADD : 0			
0	NOP	00	00	00	00	00	0	0	0	0

The above format is used in the project. The decoded control word is of 26-bit wide. The detail anatomy of each bit-field and its relation is provided.

Reference -



Further note, each micro-code/StateID is executed by the EU as 3 stage-

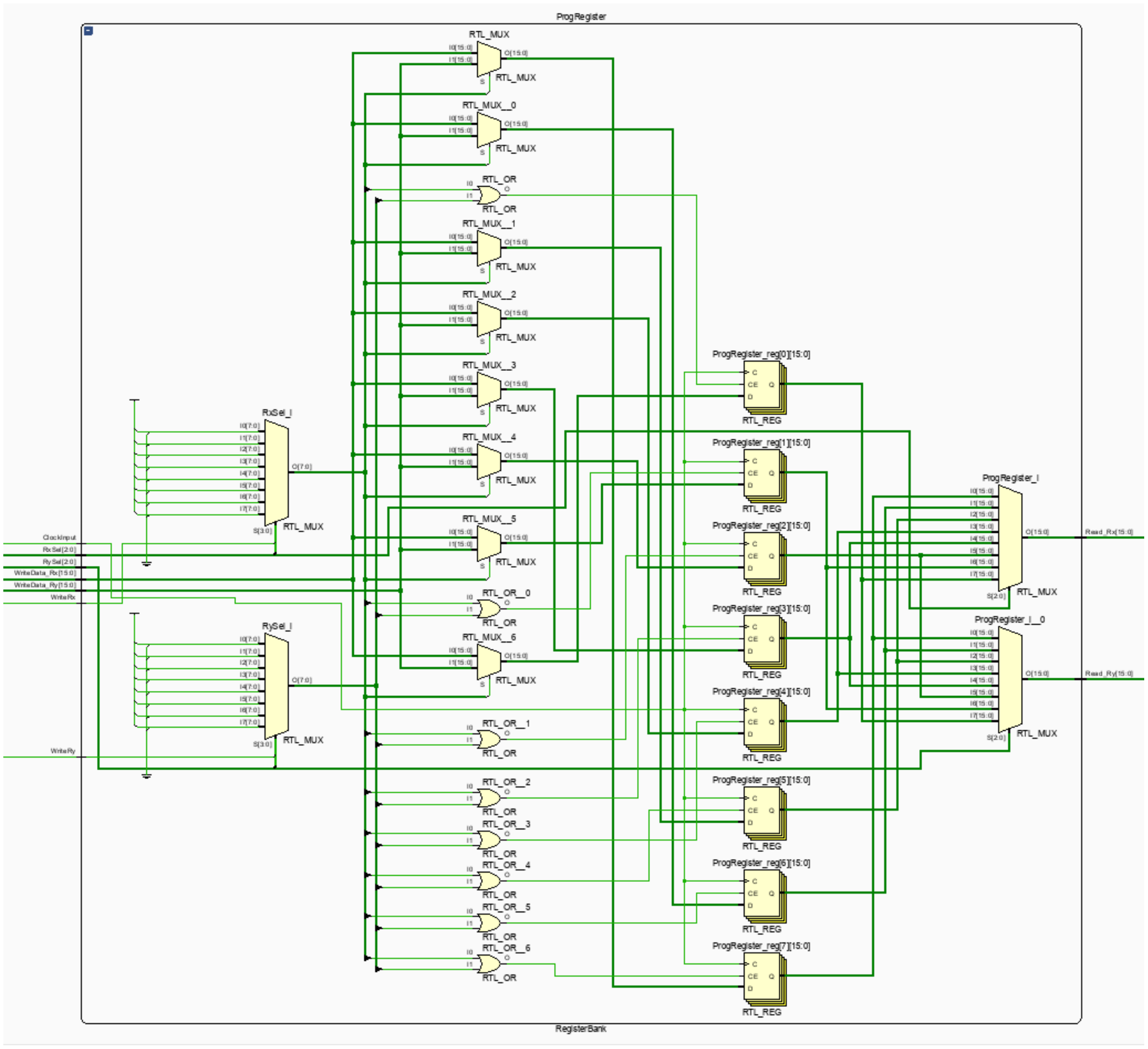
1. Buffer the bus with its coded source. (*Src->A Bus, Src->B Bus*)
2. Registers to latch the coded bus data, Perform ALU, Perform MEM address out. (*Bus destination*)
3. Read MEM data (*MEM -> reg*)

The EU execution is clocked by 3 phase clocks for each 3 stages.

DATA Plane development

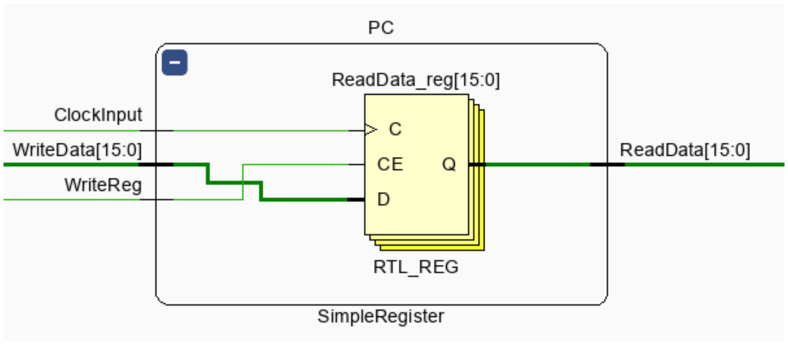
The Data path encompasses the whole Execution Unit, covering the ALU, MEM, Register Bank.

Programmer's Register Bank:



The Bank contains 8 (16-bit) registers R0-R7. Depending on the selected Rx, Ry, the corresponding registers can be independently accessed and written.

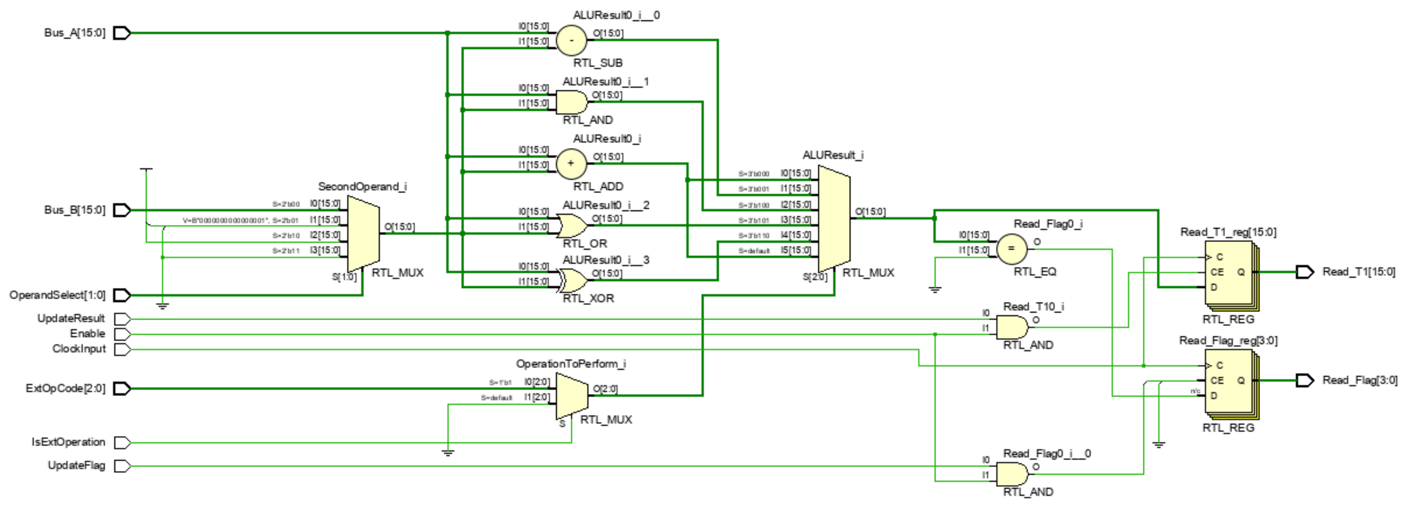
Simple Register module:



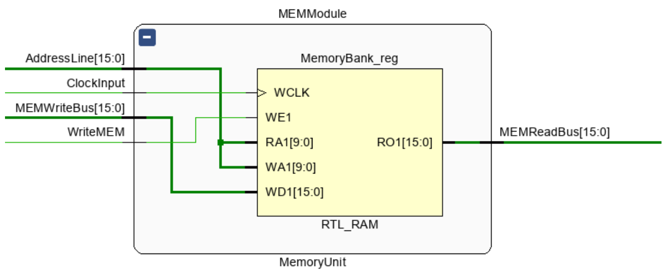
A 16-bit register with write signal gated. Used for internal registers – PC, T2, DI, DO, AO etc.,

ALU design:

a. ADD, SUB, AND, OR, XOR are the instructions supported
b. Condition code contains Zero flag only. (Future extendable to NEG, OV etc.,)
c. ALU operates on 16-bit operand (precisely two), and yields result and condition flags.
d. The ALU result is conditionally written into T1 register, condition code is conditionally written into Flag bits.



External Memory Unit:



This memory contains both the Program/code and Data segment together. The MIN architecture is Princeton architecture, both the code and data are from same memory/bus.

The user program is fed into this memory. Any external read/write is done to this unit.

Complete Execution Unit:

All the registers have window to read/write into two BUS (namely Bus A, Bus B). The writing is exclusive and governed by the microcode. Further EDB and EAB connects the EU to an external Memory Unit.

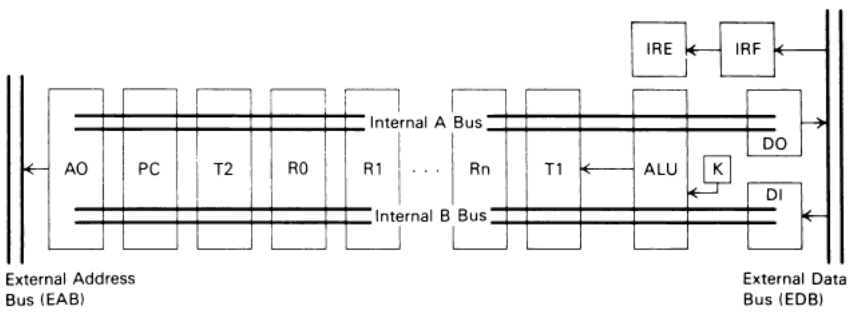
Note, each micro-code is executed within as a sequence –

1. The EU buses are wetted with the register content. (i.e., Rx->a, DI->b)
2. The destined registers read the new content from designated bus line. (i.e., A->PC, B->Rx,A->AO)
3. Finally, the memory related operation is performed. (i.e., EDB->DI, EDB->IRF, DO->EDB)

The EU is developed in vivado HDL (Verilog) and the elaborated design schematic is attached in the appendix.

The complete circuit RTL analysis is given in appendix.

Reference –



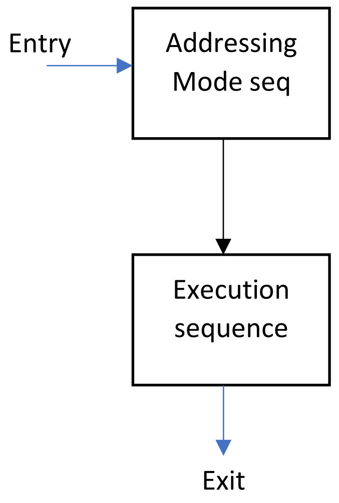
ALU	Arithmetic and Logic Unit	DO	Data Out buffer	K	Constant generator
AO	Address Out buffer	IRF	Instruction Register for Fetch	PC	Program Counter
DI	Data Input register	IRE	Instruction Register for Execution	R0-Rn	Programmer's registers
				T1, T2	Temporary registers

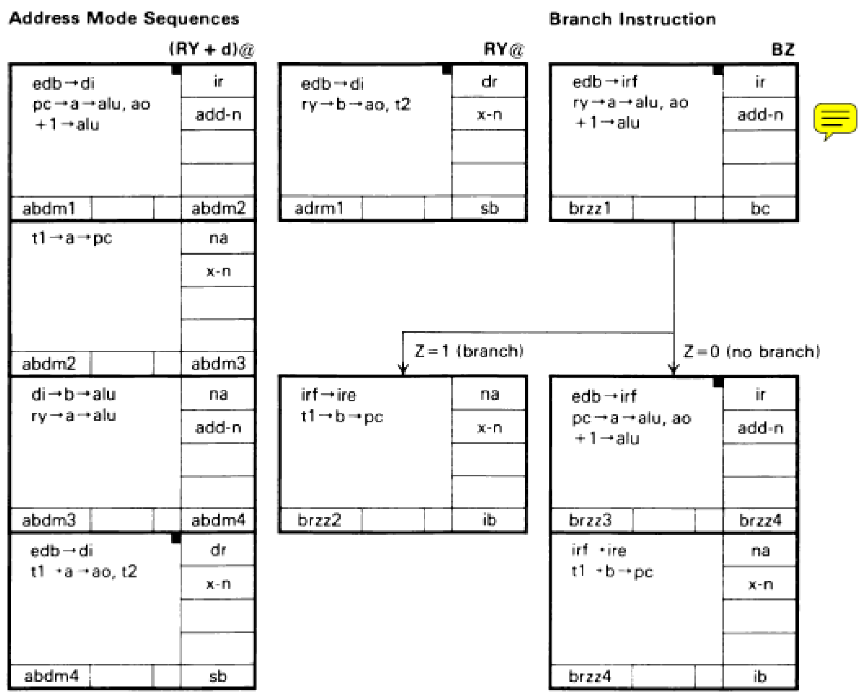
Example Rules of Operation

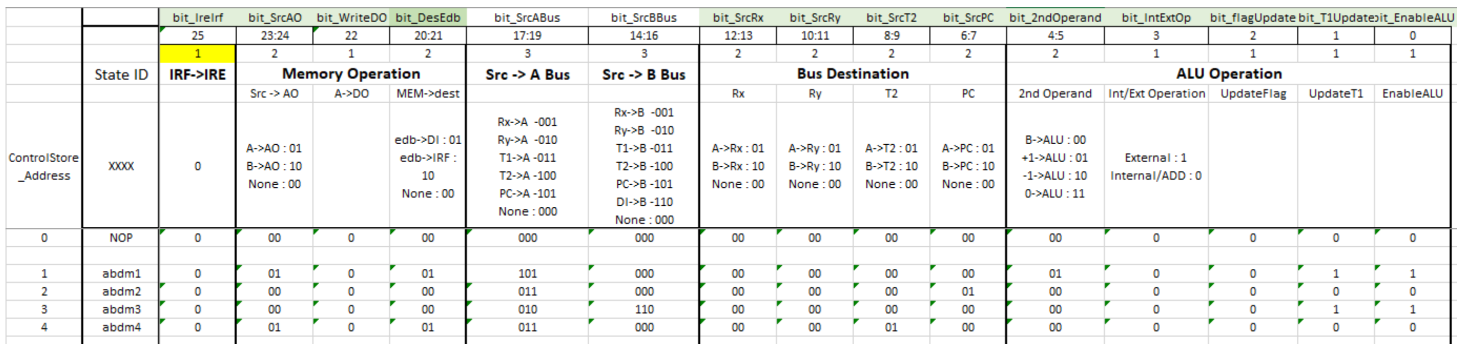
1. A transfer from source to bus to destination takes one state time.
2. A source can drive up to three destination loads.
3. Inputs to the ALU are from the A (internal) bus and either K (values 0, +1, −1) or the B (internal) bus.
4. When the ALU is a destination, T1 is automatically loaded from the ALU output.
5. A transfer to AO activates the on-chip external bus controller.

Figure 3.4 MIN execution unit block diagram

Flow chart used in the project / CISC micro-code:



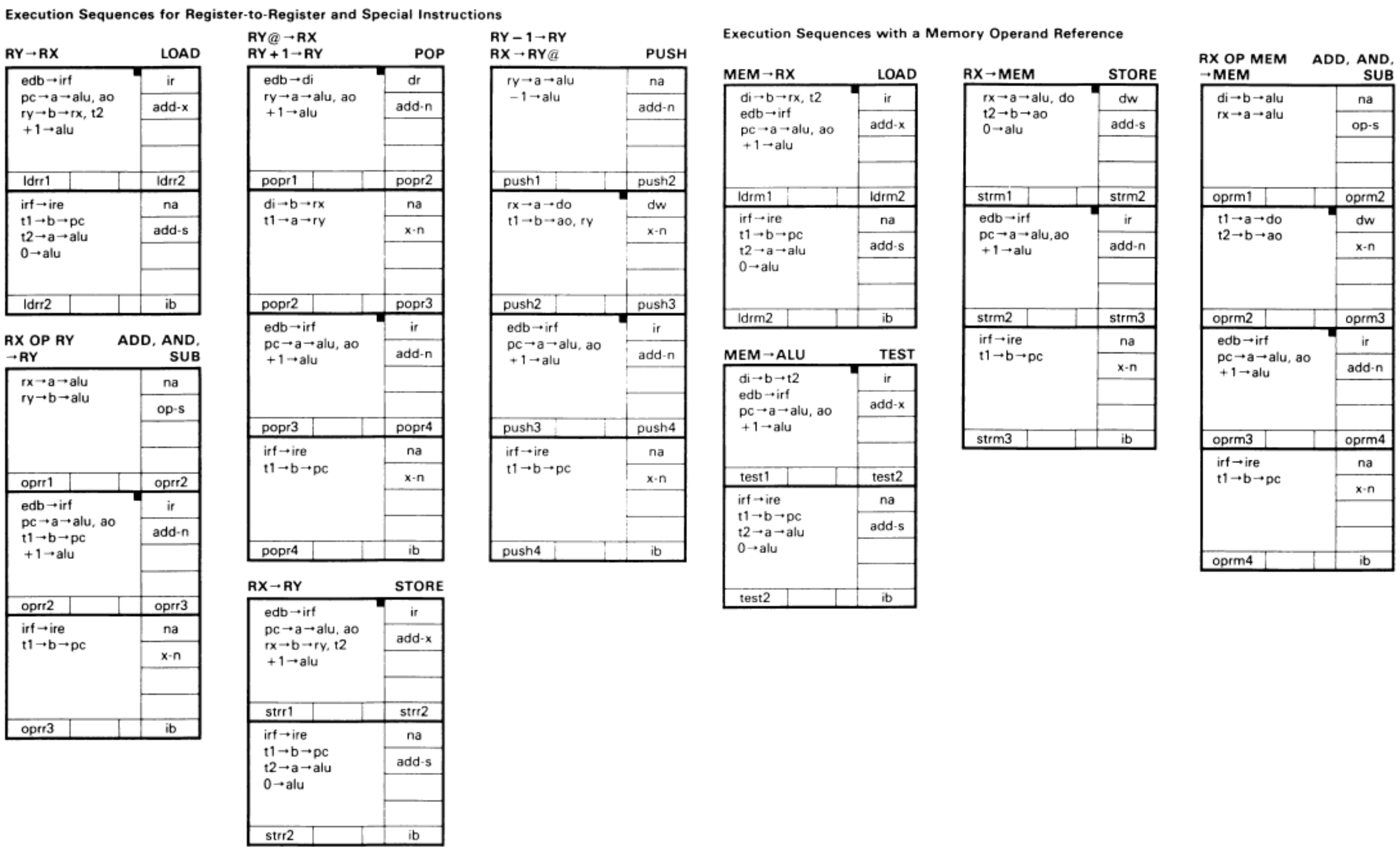




		bit_IreIrf	bit_SrcAO	bit_WriteDO	bit_DesEdb	bit_SrcABus	bit_SrcBBus	bit_SrcRx	bit_SrcRy	bit_SrcT2	bit_SrcPC	bit_2ndOperand	bit_IntExtOp	bit_flagUpdate	bit_T1Update	bit_EnableALU
		25	23:24	22	20:21	17:19	14:16	12:13	10:11	8:9	6:7	4:5	3	2	1	0
		1	2	1	2	3	3	2	2	2	2	2	1	1	1	1
	State ID	**IRF->IRE**	**Memory Operation**			**Src -> A Bus**	**Src -> B Bus**	**Bus Destination**				**ALU Operation**				
			Src -> AO	A->DO	MEM->dest			Rx	Ry	T2	PC	2nd Operand	Int/Ext Operation	UpdateFlag	UpdateT1	EnableALU
ControlStore _Address	XXXX	0	A->AO : 01 B->AO : 10 None : 00		edb->DI : 01 edb->IRF : 10 None : 00	Rx->A -001 Ry->A -010 T1->A -011 T2->A -100 PC->A -101 None : 000	Rx->B -001 Ry->B -010 T1->B -011 T2->B -100 PC->B -101 DI->B -110 None : 000	A->Rx : 01 B->Rx : 10 None : 00	A->Ry : 01 B->Ry : 10 None : 00	A->T2 : 01 B->T2 : 10 None : 00	A->PC : 01 B->PC : 10 None : 00	B->ALU : 00 +1->ALU : 01 -1->ALU : 10 0->ALU : 11	External : 1 Internal/ADD : 0			
0	NOP	0	00	0	00	000	000	00	00	00	00	00	0	0	0	0
1	abdm1	0	01	0	01	101	000	00	00	00	00	01	0	0	1	1
2	abdm2	0	00	0	00	011	000	00	00	00	01	00	0	0	0	0
3	abdm3	0	00	0	00	010	110	00	00	00	00	00	0	0	1	1
4	abdm4	0	01	0	01	011	000	00	00	01	00	00	0	0	0	0

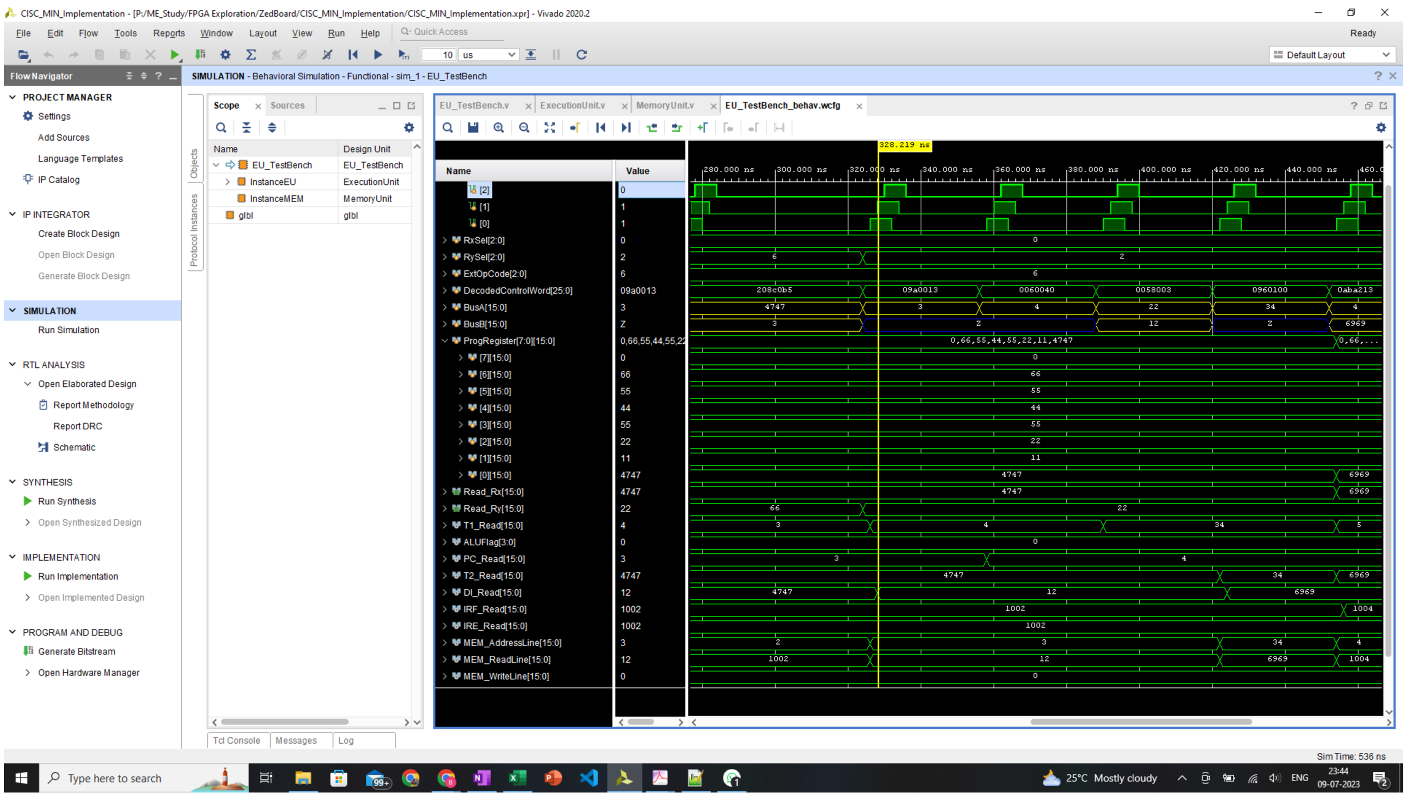
For each StateID, its corresponding micro-actions to take place inside EU to achieve one micro-instruction is elaborated in the spreadsheet as above. Note, the complete 36 StateIDs and its decoded control word is available in the project document (*BitFormat_ControlWordList.xlx* file).

Sequence micro-instruction flow chart –



For each StateID, its corresponding micro-actions to take place inside EU to achieve one micro-instruction is elaborated in the spreadsheet as above. Note, the complete 36 StateIDs and its decoded control word is available in the project document (*BitFormat_ControlWordList.xlx* file).

Testing Data path using vivado simulation –



Name	Design Unit
EU_TestBench	EU_TestBench
InstanceEU	ExecutionUnit
InstanceMEM	MemoryUnit
glbl	glbl

Name	Value
[2]	0
[1]	1
[0]	1
RxSel[2:0]	0
RySel[2:0]	2
ExtOpCode[2:0]	6
DecodedControlWord[25:0]	09a0013
BusA[15:0]	3
BusB[15:0]	Z
ProgRegister[7:0][15:0]	0,66,55,44,55,22
[7][15:0]	0
[6][15:0]	66
[5][15:0]	55
[4][15:0]	44
[3][15:0]	55
[2][15:0]	22
[1][15:0]	11
[0][15:0]	4747
Read_Rx[15:0]	4747
Read_Ry[15:0]	22
T1_Read[15:0]	4
ALUFlag[3:0]	0
PC_Read[15:0]	3
T2_Read[15:0]	4747
DI_Read[15:0]	12
IRF_Read[15:0]	1002
IRE_Read[15:0]	1002
MEM_AddressLine[15:0]	3
MEM_ReadLine[15:0]	12
MEM_WriteLine[15:0]	0

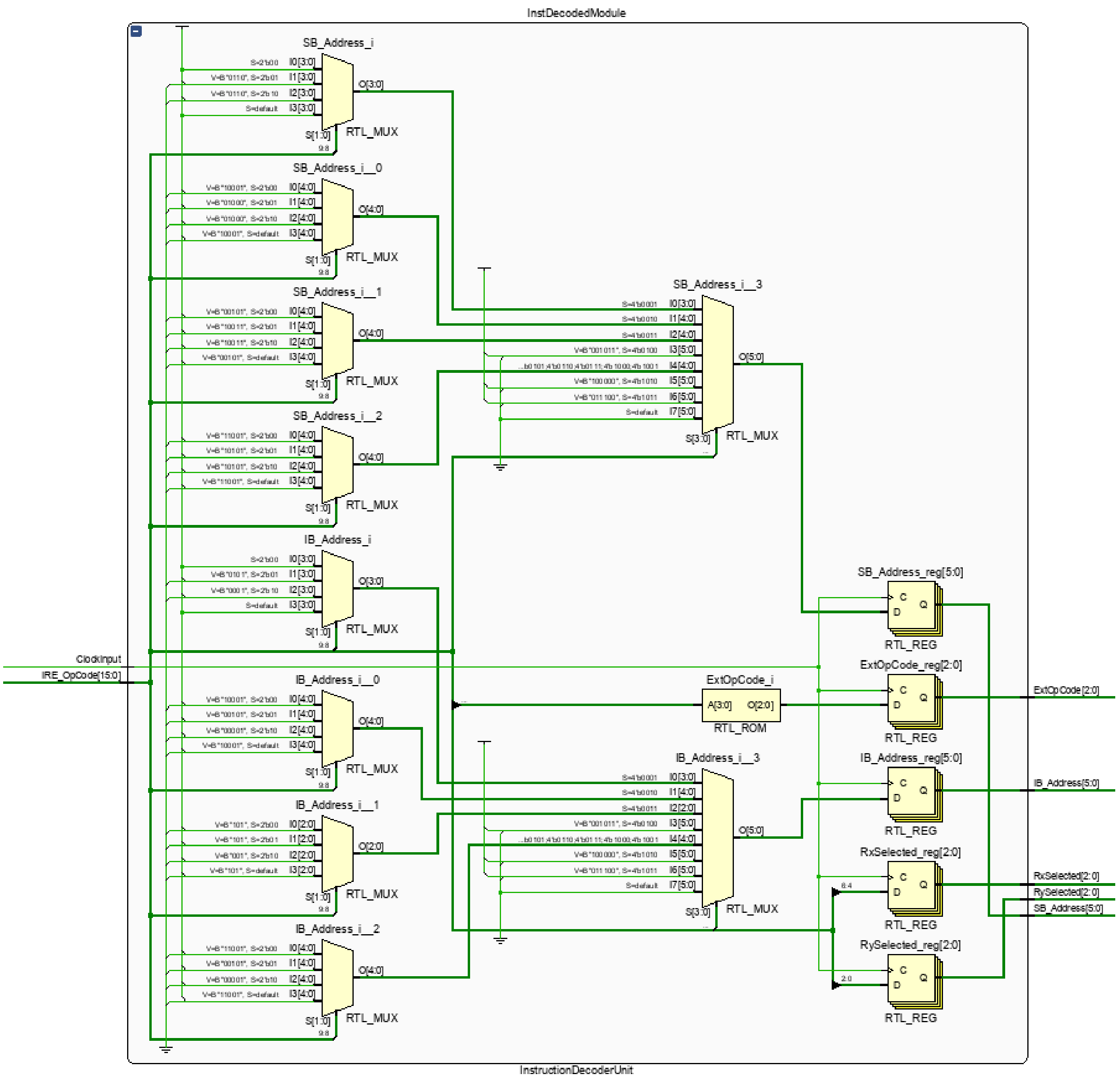
Control Plane Development –

The Control plane encompasses the control logic that orderly sequence the micro-code to the provided programmer's instruction, and sequence the system to read next instruction. It encompasses, Control Store Unit, Instruction decoder unit, Next State logic unit.

Instruction Decoder Unit-

This unit decodes the IRE Op-code to get details –

a. Programmer's operation to perform (ADD,LDR,TST etc.,)
b. Addressing mode (direct/indirect/indirect+base)
c. Selected Rx and Ry (R0-R7)



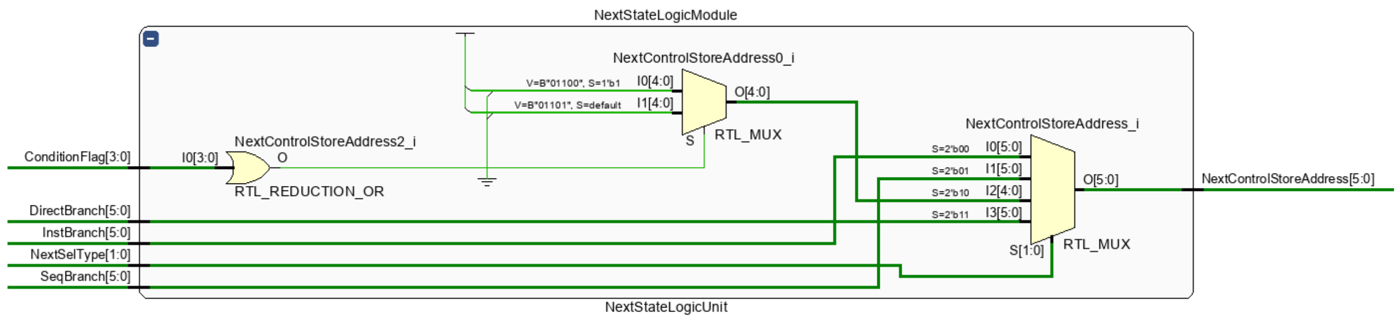
The circuit is designed in HDL and the synthesized RTL design is as above. IRE is decoded and decoded information is made available at every clock pulse.

			Inst Branch			Seq Branch			
Op Code	**Instr.**	**Modes applicable**	**0-direct**	**1-indirect**	**2-indirect+DP**	**0-direct**	**1-indirect**	**2-indirect+DP**	ExtOpCode
0	NOP	x	0	0	0	0	0	0	0
1	LDR	all	15	5	1	15	6	6	0
2	STR	all	17	5	1	17	8	8	0
3	TST	indirect only	5	5	1	5	19	19	0
4	BZ	x	11	11	11	11	11	11	0
5	ADD	all	25	5	1	25	21	21	ADD
6	SUB	all	25	5	1	25	21	21	SUB
7	AND	all	25	5	1	25	21	21	AND
8	OR	all	25	5	1	25	21	21	OR
9	XOR	all	25	5	1	25	21	21	XOR
10	PUSH	x	28	28	28	28	28	28	0
11	POP	x	32	32	32	32	32	32	0

Decoded parameters – Instruction branch, Sequence branch, External Operation, Rx and Ry selected. Above is the workout for each instruction and its corresponding IB and SB. Using this data, the Instruction decoder is designed to resolve same.

Next State Logic Unit –

This Unit is responsible to generate the Next State-ID the micro-code to move. The state could flow to either new Instruction Branch, or next execution sequence (SB), or direct branch as specified in the Control word of current micro-code, or a conditional branch i.e., BZ.



The Next State-ID / ControlStoreAddress is branched to either IB/SB/DB/conditional branch (bzrr2 if true, bzrr4 for false). Note, in this project we have zero flag CC and BZ is the only branching instruction. The detail control word will be discussed in next section.

Reference –

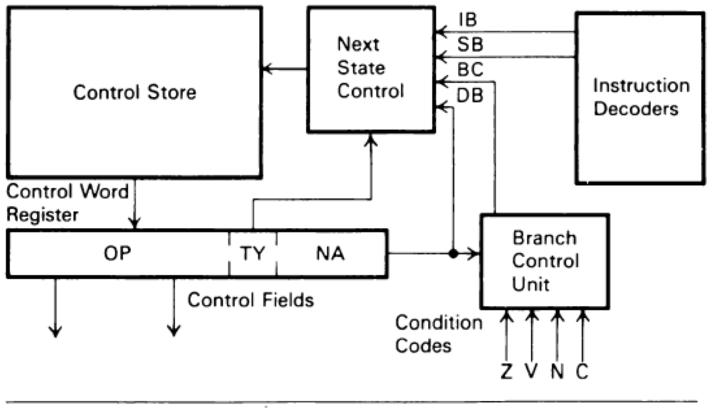
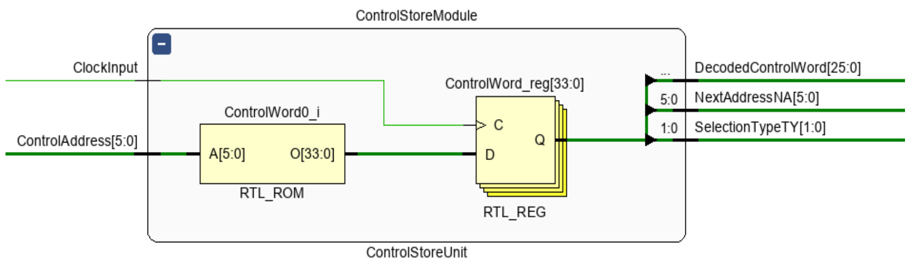


Figure 4.18 Control store branch control and next address select

Control Store Unit-



It’s a ROM memory that holds the Control Word (34 bits wide) for each State-ID/Control Address. Below is the anatomy

Control Word		
26 bits	2 bits	6 bits
Decoded Control Word	**TY selection**	**Next Address**

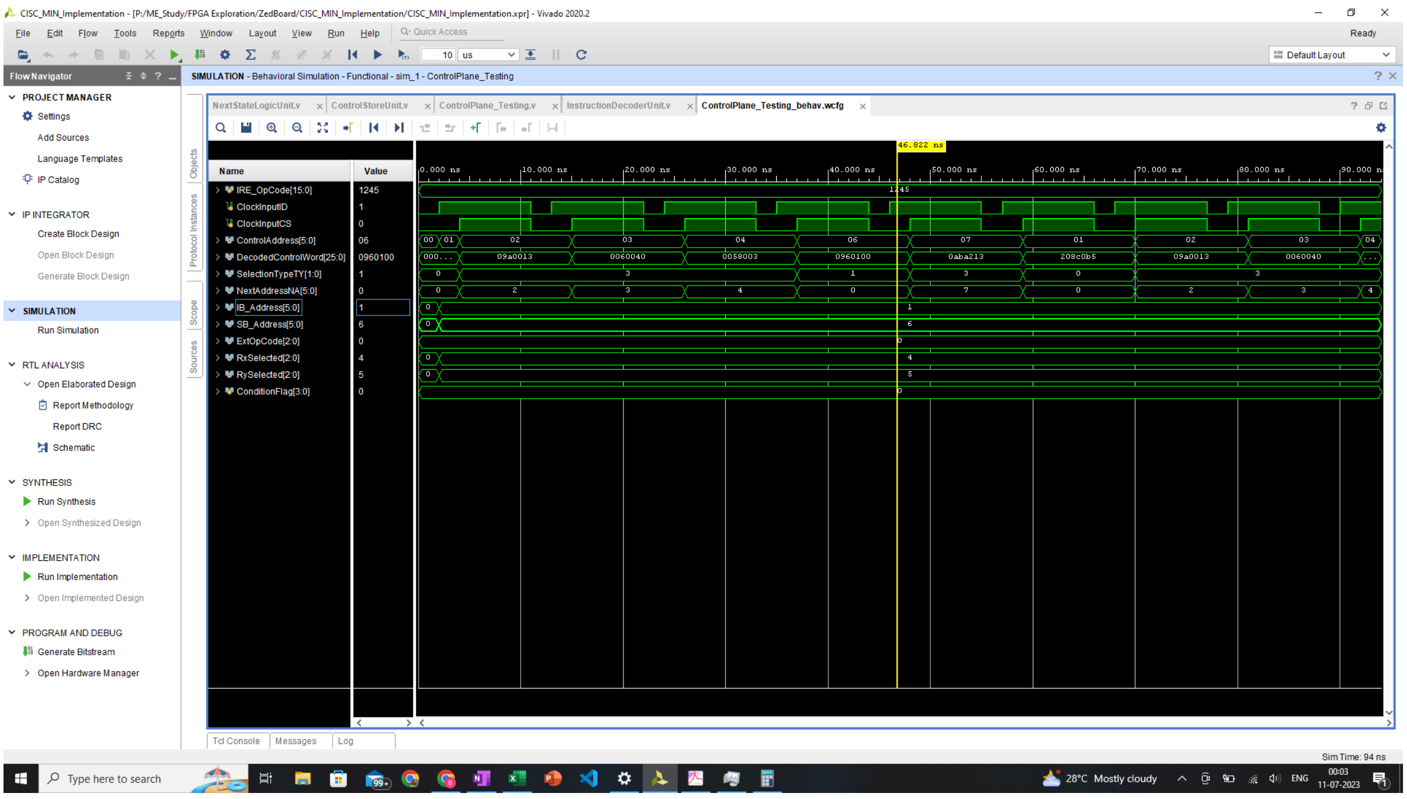
Content stored in the Control Store –

	bits position -	33-8	7-6	5-0
ControlStore_Address	StateID	DecodedControlWord 26 bits	NextAddressSelection **TY - 2 bit**	NextControlStoreAddress **NA - 6 bits**
0	NOP	0x0	11	13
1	abdm1	0x9a0013	11	2
2	abdm2	0x60040	11	3
3	abdm3	0x58003	11	4
4	abdm4	0x960100	01	0
5	adrm1	0x1108200	01	0
6	ldrm1	0xaba213	11	7
7	ldrm2	0x208c0b5	00	0
8	strm1	0x1430035	11	9
9	strm2	0xaa0013	11	10
10	strm3	0x200c080	00	0
11	brzz1	0xa40013	10	13
12	brzz2	0x200c080	00	0
13	brzz3	0xaa0013	11	14
14	brzz4	0x200c080	00	0
15	ldrr1	0xaaa213	11	16
16	ldrr2	0x208c0b5	00	0
17	strr1	0xaa4a13	11	18
18	strr2	0x208c0b5	00	0
19	test1	0xab8213	11	20
20	test2	0x208c0b5	00	0
21	oprm1	0x3800f	11	22
22	oprm2	0x1470000	11	23
23	oprm3	0xaa0013	11	24
24	oprm4	0x200c080	00	0
25	oprr1	0x2800f	11	26
26	oprr2	0xaac093	11	27
27	oprr3	0x200c080	00	0
28	popr1	0x940013	11	29
29	popr2	0x7a400	11	30
30	popr3	0xaa0013	11	31
31	popr4	0x200c080	00	0
32	push1	0x40023	11	33
33	push2	0x142c800	11	34
34	push3	0xaa0013	11	35
35	push4	0x200c080	00	0

Testing Control Plane using Vivado Simulation –

ASM code	Nibble-3	Nibble-2	Nibble-1	Nibble-0	OpCode	MicroCode (StateID) flow	TEST
NOP	0	0	-	-	16'h0000	0,13,14	Pass
LDR RX,RY	1	0	*Rx	*Ry	16'h1021	15,16	Pass
LDR RX,[RY]	1	1	*Rx	*Ry	16'h1121	5,6,7	Pass
LDR RX,[RY+DP]	1	2	*Rx	*Ry	16'h1221	1,2,3,4,6,7	Pass
BZ RY	4	0	-	*Ry	16'h4001	11,12(branch) or 11,13,14(no)	Pass
STR RX,RY	2	0	*Rx	*Ry	16'h2021	17,18	Pass
STR RX,[RY]	2	1	*Rx	*Ry	16'h2121	5,8,9,10	Pass
STR RX,[RY+DP]	2	2	*Rx	*Ry	16'h2221	1,2,3,4,8,9,10	Pass
TST [RY]	3	1	-	*Ry	16'h3101	5,19,20	Pass
TST [RY+DP]	3	2	-	*Ry	16'h3201	1,2,3,4,19,20	Pass
ADD RX,RY	5-9**	0	*Rx	*Ry	16'h5021	25,26,27	Pass
ADD RX,[RY]	5-9**	1	*Rx	*Ry	16'h5121	5,21,22,23,24	Pass
ADD RX,[RY+DP]	5-9**	2	*Rx	*Ry	16'h5221	1,2,3,4,21,22,23,24	Pass
PUSH RX,RY	A	0	*Rx	*Ry	16'hA021	32,33,34,35	Pass
POP RX,RY	B	0	*Rx	*Ry	16'hB021	28,29,30,31	Pass
	** ADD=5 SUB=2..XOR=9		*Rx = 2, Ry = 1				

LDR R4, [R5+DP] testing the sequence -



Name	Value
IRE_OpCode[15:0]	1245
ClockInputID	1
ClockInputCS	0
ControlAddress[5:0]	06
DecodedControlWord[25:0]	0960100
SelectionTypeTY[1:0]	1
NextAddressNA[5:0]	0
IB_Address[5:0]	1
SB_Address[5:0]	6
ExtOpCode[2:0]	0
RxSelected[2:0]	4
RySelected[2:0]	5
ConditionFlag[3:0]	0

Overall Unit integration and RTL analysis –

The detailed source code and testbench code are available in the drive resource.

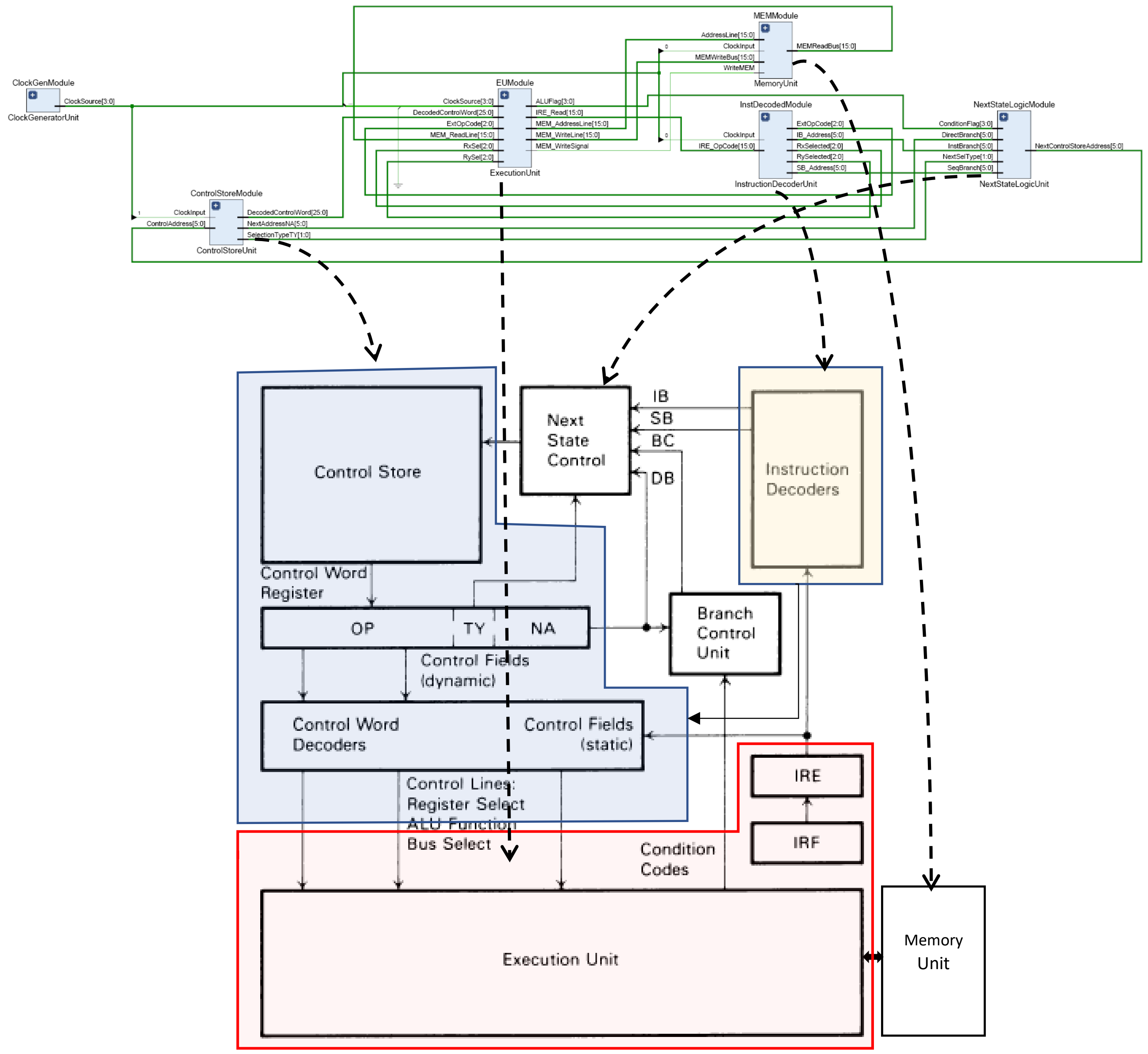
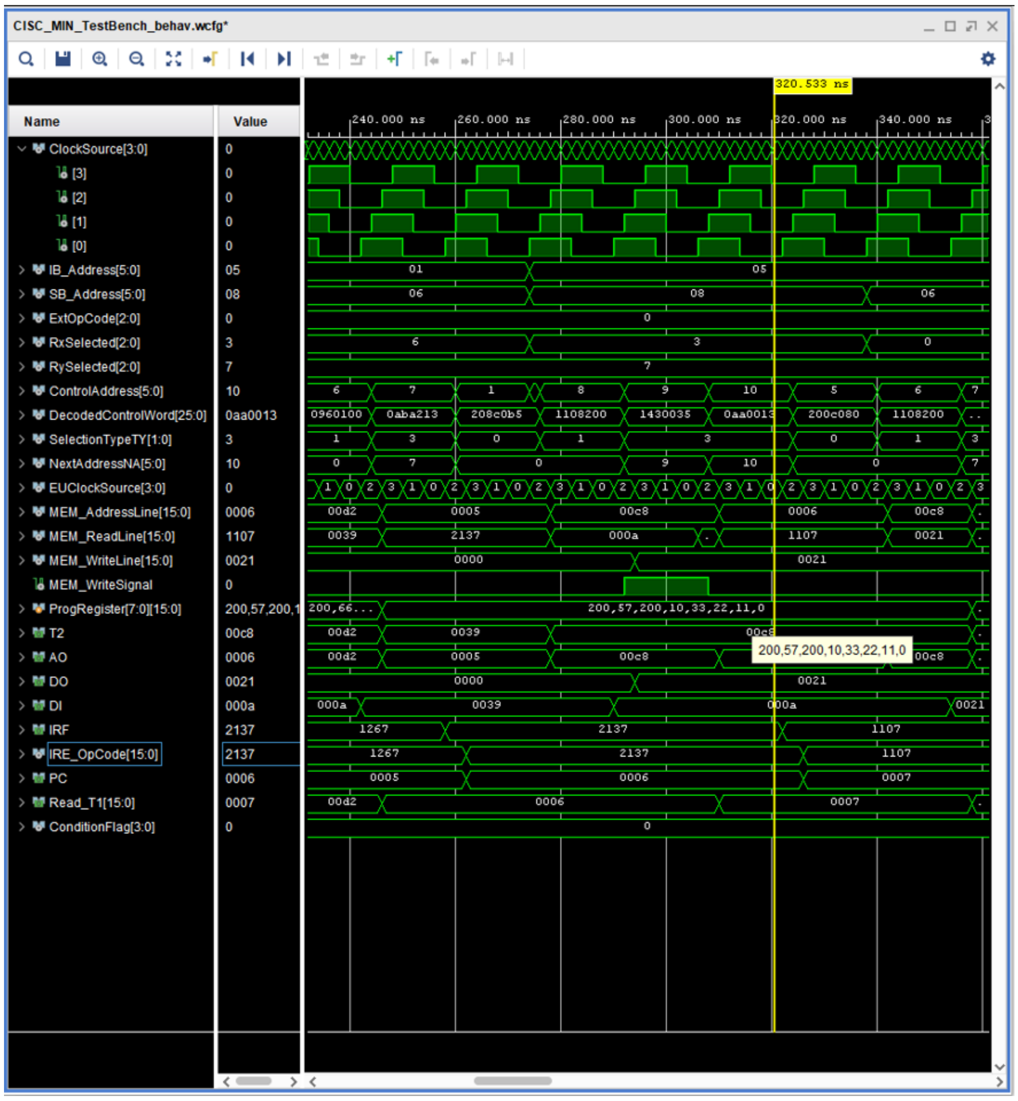


Figure 4.1 Microprocessor block diagram (microcoded controller)

Testing Programmer’s instruction –

Address	Operation	at end	OpCode	
0	NOP	200,66,55,44,33,22,11,0	16'h0000	MemoryBank[0]=16'h0000;
1	LDR R5,R7	200,66,200,44,33,22,11,0	16'h1057	MemoryBank[1]=16'h1057;
2	LDR R4,[R5]	200,66,200,10,33,22,11,0	16'h1145	MemoryBank[2]=16'h1145;
3	LDR R6,[R7+10]	200,57,200,10,33,22,11,0	16'h1267	MemoryBank[3]=16'h1267;
4			16'hA	MemoryBank[4]=16'hA;
5	STR R3,[R7]	200,57,200,10,33,22,11,0	16'h2137	MemoryBank[5]=16'h2137;
6	LDR R0,[R7]	200,57,200,10,33,22,11,33	16'h1107	MemoryBank[6]=16'h1107;
7	STR R6,[R7+100]	200,57,200,10,33,22,11,33	16'h2267	MemoryBank[7]=16'h2267;
8			16'h64	MemoryBank[8]=16'h64;
9	NOP	200,57,200,10,33,22,11,33	16'h0000	MemoryBank[9]=16'h0000;
10	LDR R0,[R7+100]	200,57,200,10,33,22,11,57	16'h1207	MemoryBank[10]=16'h1207
11			16'h64	MemoryBank[11]=16'h64;
12	ADD R0,R3 : R3<-R0+R3	200,57,200,10,90,22,11,57	16'h5003	MemoryBank[12]=16'h5003
13	SUB R0,R4	200,57,200,47,90,22,11,57	16'h6004	MemoryBank[13]=16'h6004
14	SUB R0,[R7]	200,57,200,47,90,22,11,57	16'h6107	MemoryBank[14]=16'h6107
15	LDR R0,[R7]	200,57,200,47,90,22,11,24	16'h1107	MemoryBank[15]=16'h1107
16	XOR R0,[R7+100]	200,57,200,47,90,22,11,24	16'h9207	MemoryBank[16]=16'h9207
17			16'h64	MemoryBank[17]=16'h64;
18	LDR R0,[R7+100]	200,57,200,47,90,22,11,33	16'h1207	MemoryBank[18]=16'h1207
19			16'h64	MemoryBank[19]=16'h64;
20	LDR R0,R7	200,57,200,47,90,22,11,200	16'h1007	MemoryBank[20]=16'h1007
21	ADD R7,R0	200,57,200,47,90,22,11,400	16'h5070	MemoryBank[21]=16'h5070
22	PUSH R1,R0	200,57,200,47,90,22,11,399	16'hA010	MemoryBank[22]=16'hA010
23	PUSH R4,R0	200,57,200,47,90,22,11,398	16'hA040	MemoryBank[23]=16'hA040
24	POP R6,R0	200,47,200,47,90,22,11,399	16'hB060	MemoryBank[24]=16'hB060
25	POP R6,R0	200,11,200,47,90,22,11,400	16'hB060	MemoryBank[25]=16'hB060
26	SUB R0,R7	200,11,200,47,90,22,11,400	16'h6007	MemoryBank[26]=16'h6007
27	BZ R3	200,11,200,47,90,22,11,400	16'h4003	MemoryBank[27]=16'h4003
28	SUB R5,R7	0,11,200,47,90,22,11,400	16'h6057	MemoryBank[28]=16'h6057
29	BZ R3		16'h4003	MemoryBank[29]=16'h4003
30	NOP		16'h0000	MemoryBank[30]=16'h0000
90	LDR R0,R7	200,11,200,47,90,22,11,200	16'h1007	MemoryBank[90]=16'h1007


CISC_MIN_TestBench_behav.wcfg*

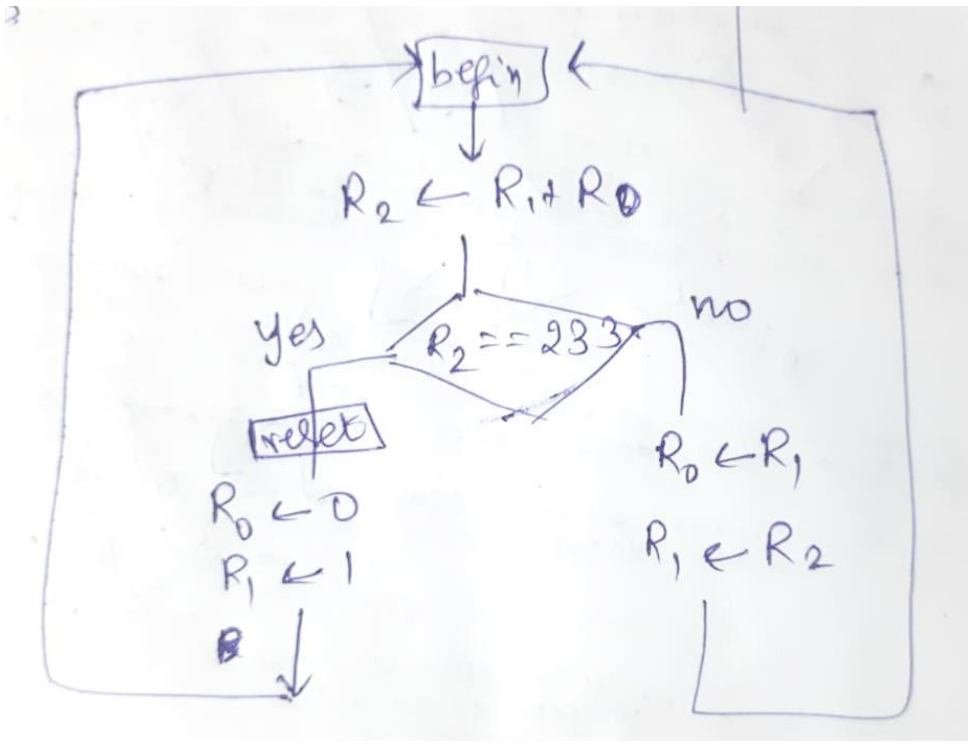
Name	Value
ClockSource[3:0]	0
[3]	0
[2]	0
[1]	0
[0]	0
IB_Address[5:0]	05
SB_Address[5:0]	08
ExtOpCode[2:0]	0
RxSelected[2:0]	3
RySelected[2:0]	7
ControlAddress[5:0]	10
DecodedControlWord[25:0]	0aa0013
SelectionTypeTY[1:0]	3
NextAddressNA[5:0]	10
EUClockSource[3:0]	0
MEM_AddressLine[15:0]	0006
MEM_ReadLine[15:0]	1107
MEM_WriteLine[15:0]	0021
MEM_WriteSignal	0
ProgRegister[7:0][15:0]	200,57,200,1
T2	00c8
AO	0006
DO	0021
DI	000a
IRF	2137
IRE_OpCode[15:0]	2137
PC	0006
Read_T1[15:0]	0007
ConditionFlag[3:0]	0

Coding Fibonacci series generator for the built microprocessor-

Address	**Program for Fibanacci series**	Comment	OpCode
0	NOP		16'h0000
1	<BEGIN>: ADD R0,R6	R6<-R0+R6	16'h5006
2	ADD R1,R6	R6<-R1+R6	16'h5016
3	LDR R2,R6	R2<-R6	16'h1026
4	SUB #233,R6		16'h6076
5	BZ <RESET>	If R6==233 MAX GOTO RESET	16'h4004
6	LDR R0,R1	R0=R1	16'h1001
7	LDR R1,R2	R1=R2	16'h1012
8	SUB R6,R6	R6=0	16'h6066
9	BZ <BEGIN>	JUMP TO BEGIN	16'h4003
10	NOP		16'h0000
11	<RESET>: LDR R1,#1	R1=1	16'h1015
12	SUB R0,R0	R0=0	16'h6000
13	BZ <BEGIN>		16'h4003

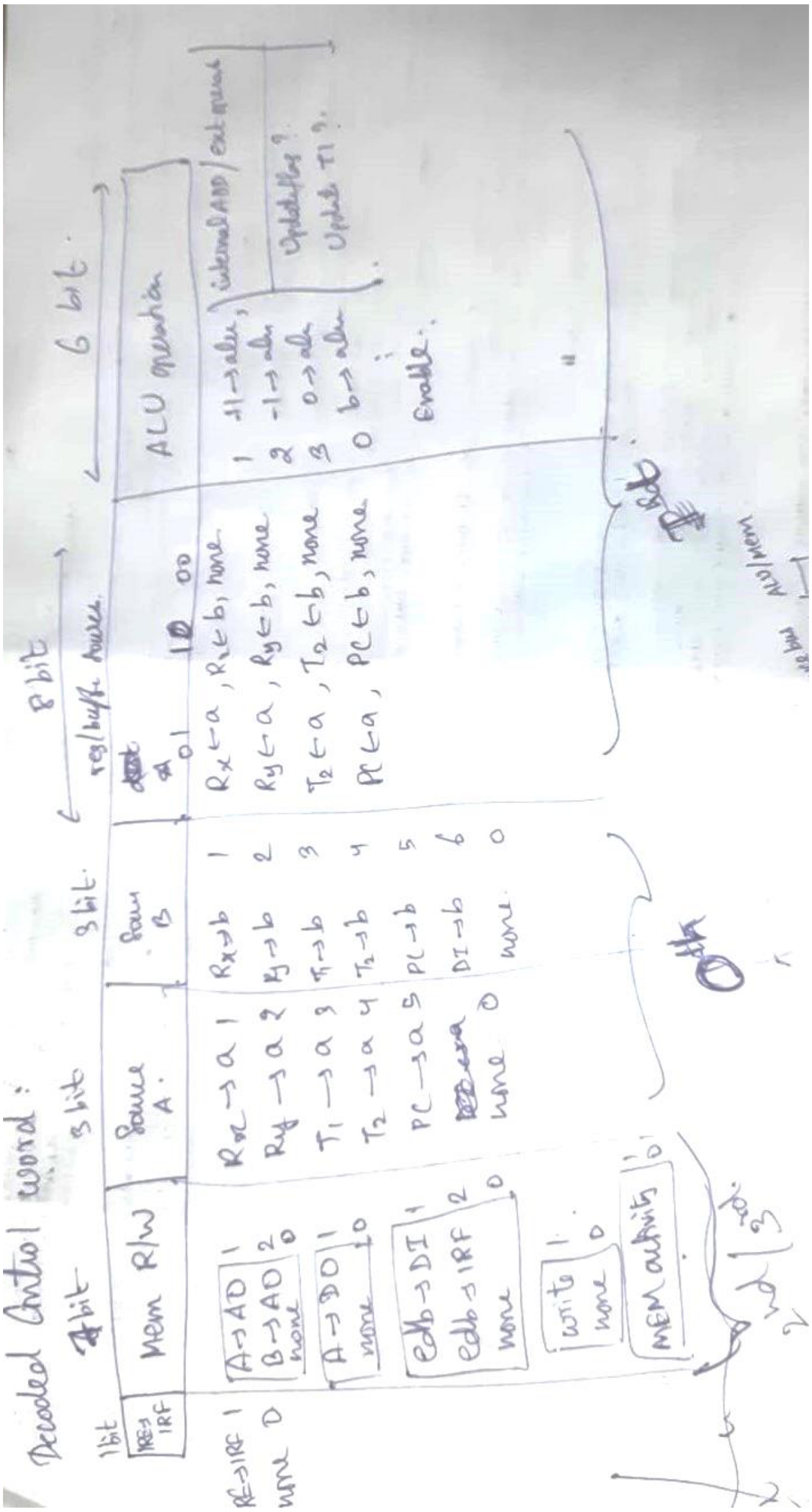
Initialise registers		
R0	0	<prev prev>
R1	1	<previous>
R2	1	<output>
R3	1	<BEGIN>
R4	11	<RESET>
R5	1	#1
R6	0	<TEMP VAR>
R7	233	#233

Flowchart-



Appendix –

Decoder Control Word discussion –



Decoded Control word :

1bit	~~4~~ 4bit	3 bit	3 bit	8 bit reg/buffer sources	6 bit
IRE→IRF	Mem R/W	Source A.	Sour B	~~dest~~ 01 · 10 · 00	ALU operation
RE→IRF 1 none 0	A→AD 1 B→AD 2 none 0	Rx→a 1	Rx→b 1	Rx←a , Rx←b, none	1 +1→alu,
	A→DO 1 none 0	Ry→a 2	Ry→b 2	Ry←a , Ry←b, none	2 -1→alu
	edb→DI 1 edb→IRF 2 none 0	T1→a 3	T1→b 3	T2←a , T2←b, none	3 0→alu
	write 1 none 0	T2→a 4	T2→b 4	PC←a , PC←b, none	0 b→alu
	MEM activity 1 0	PC→a 5	PC→b 5		:
		~~[illegible]~~	DI→b 6		Enable.
		none 0	none 0		internal ADD / ext operand Update flag ? Update T1 ?.

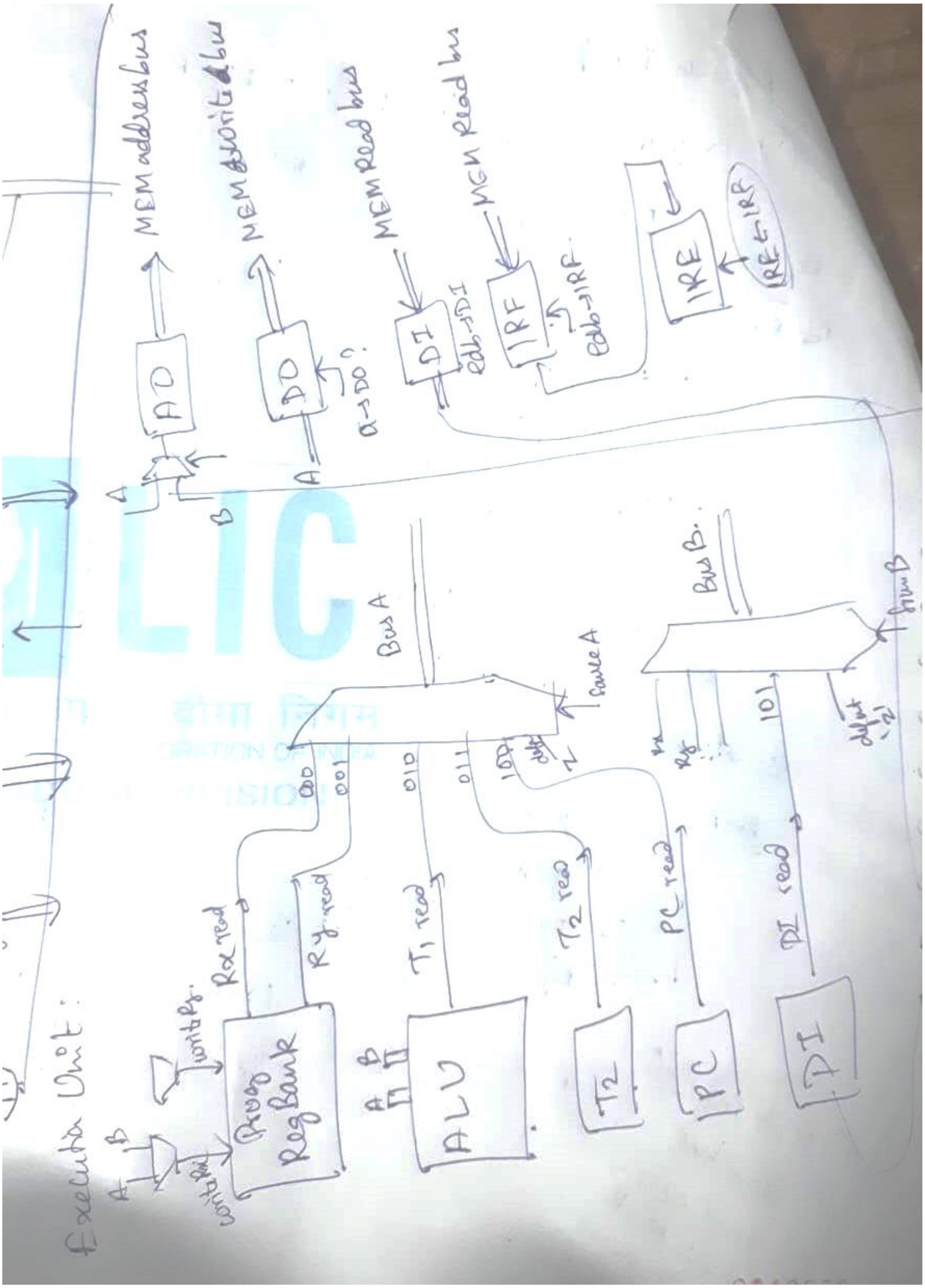
2nd | 3rd.

0th

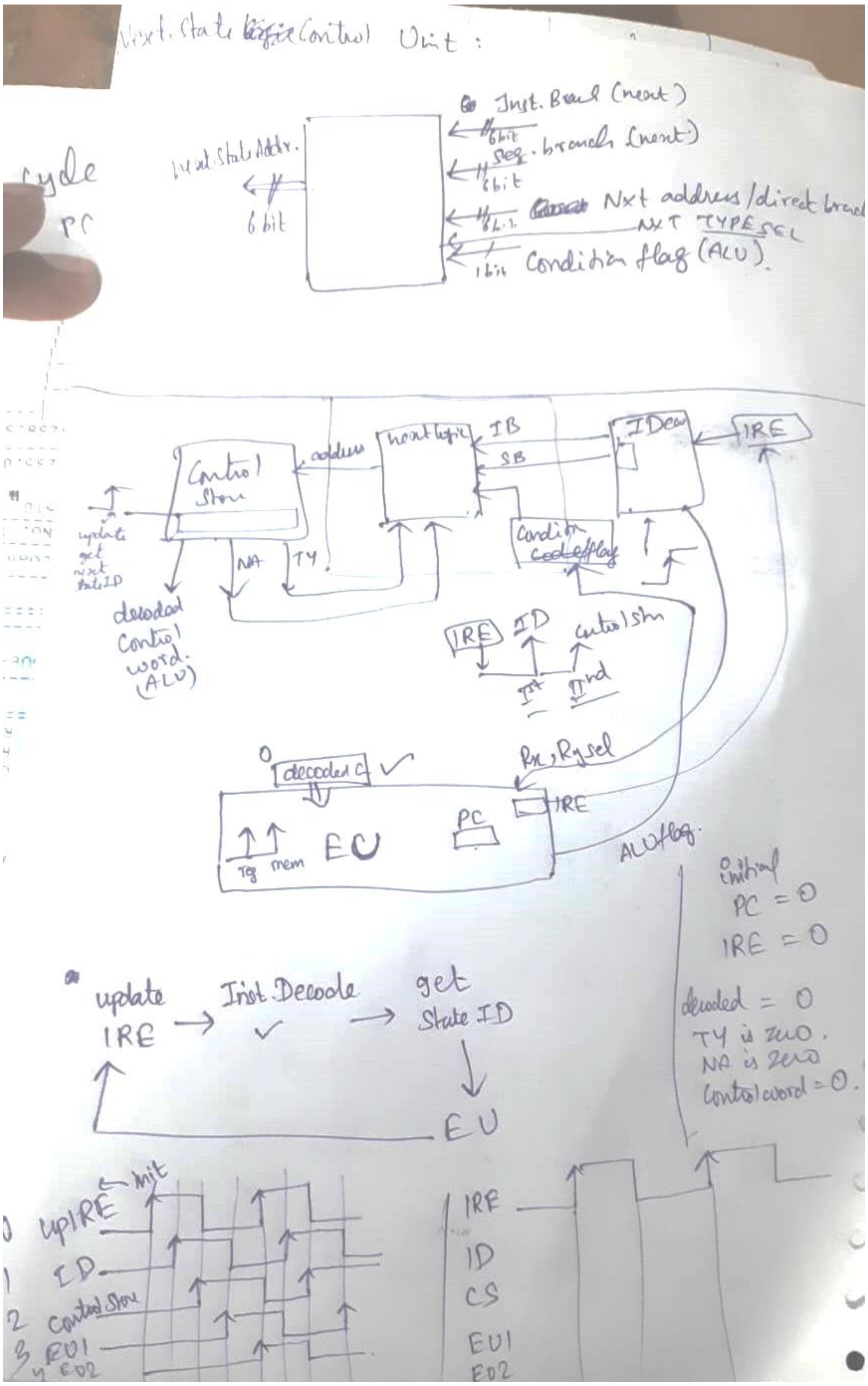
1st

ALU/MEM

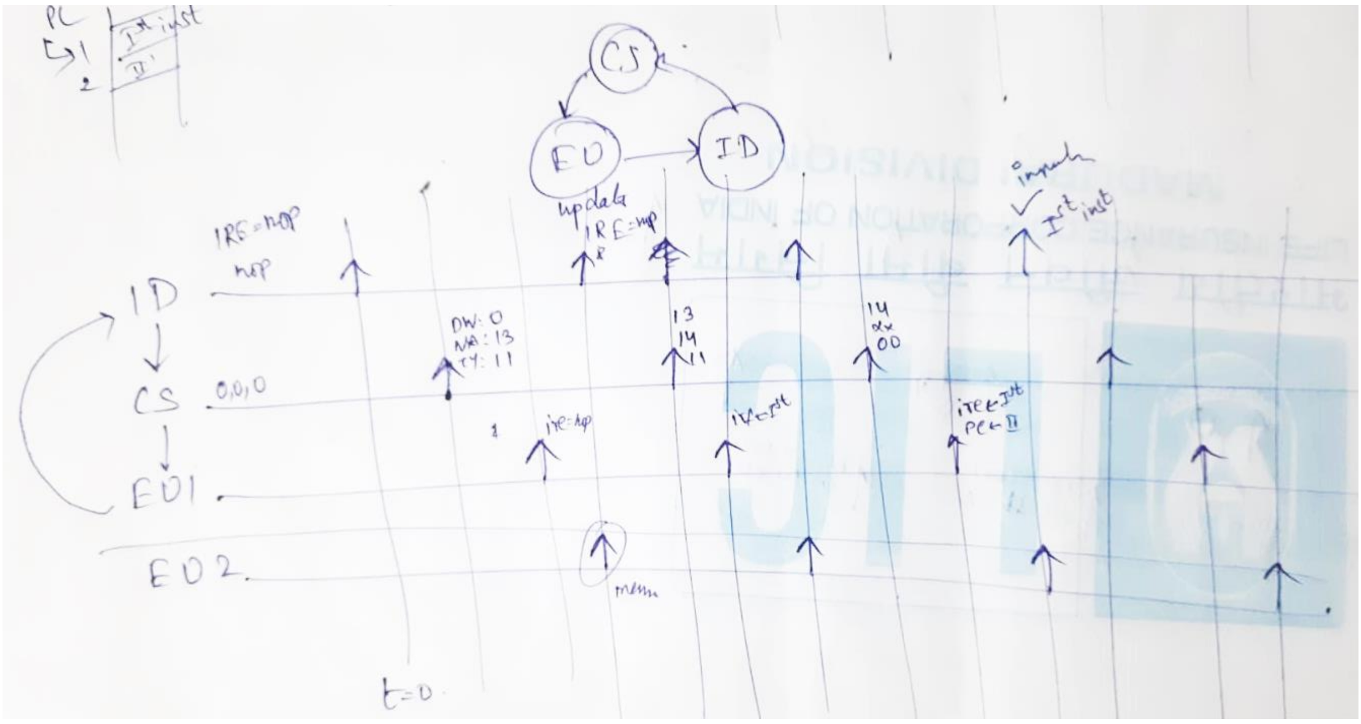
Execution Unit discussion –



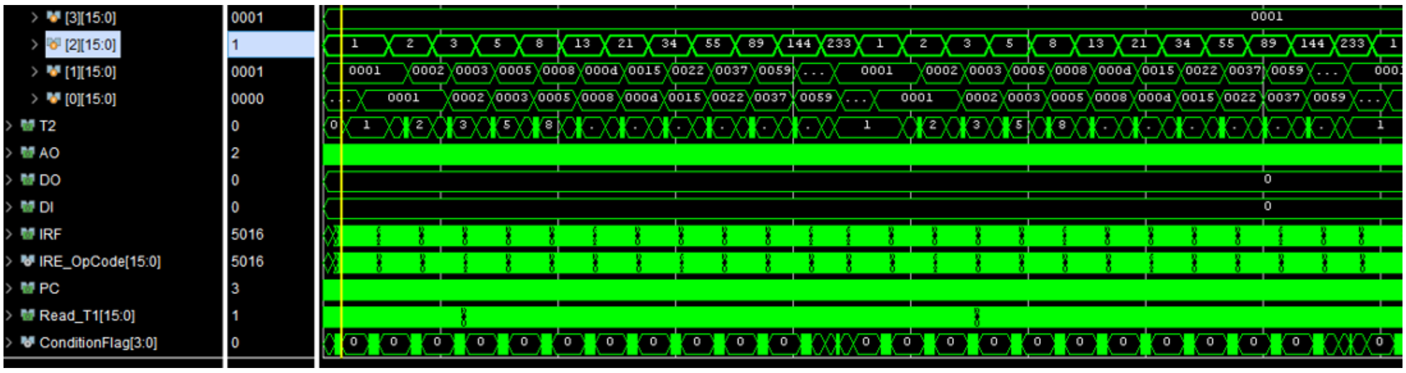
Next State logic discussion –



Clock Phase discussion –



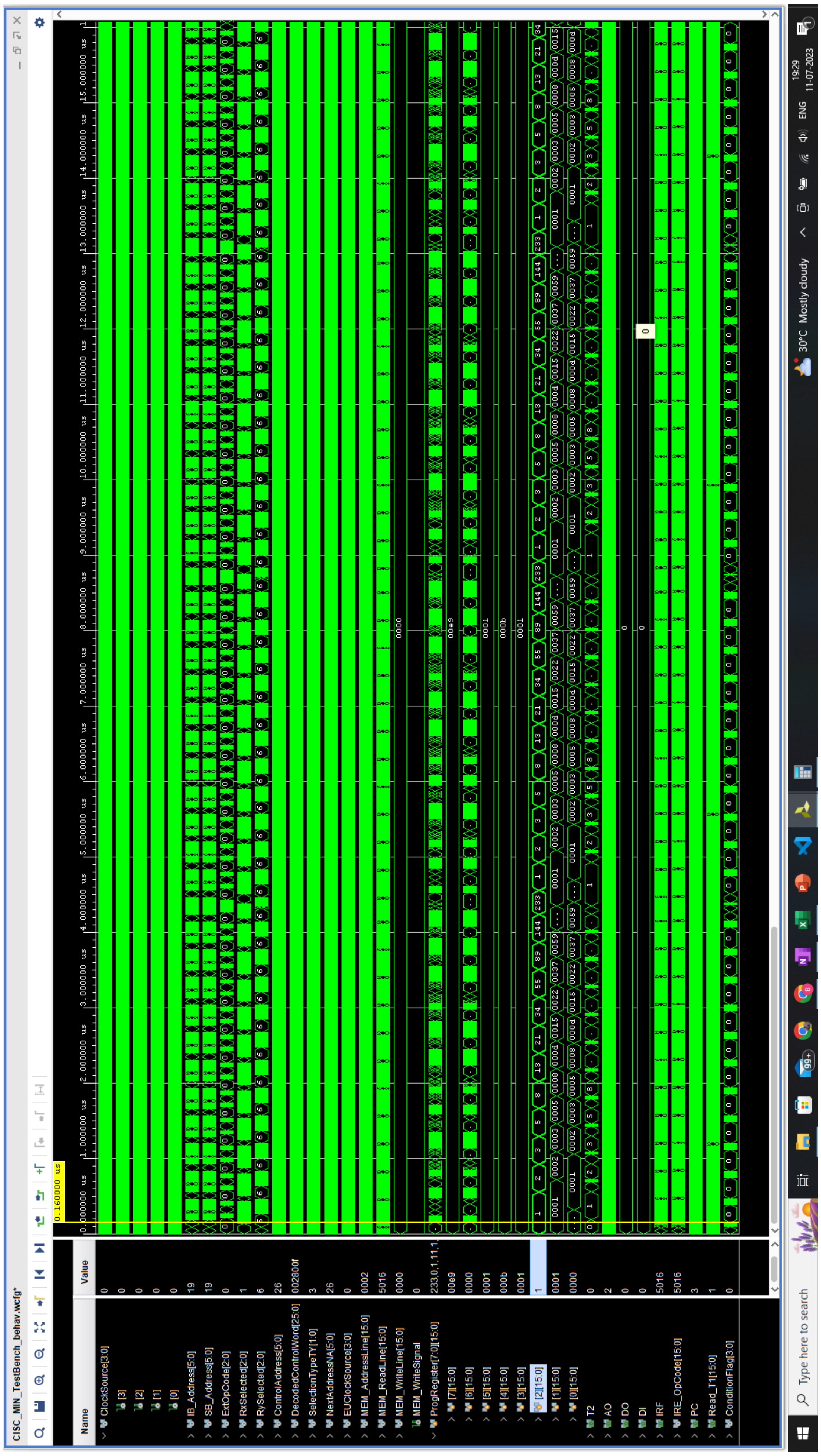
Fibonacci result –



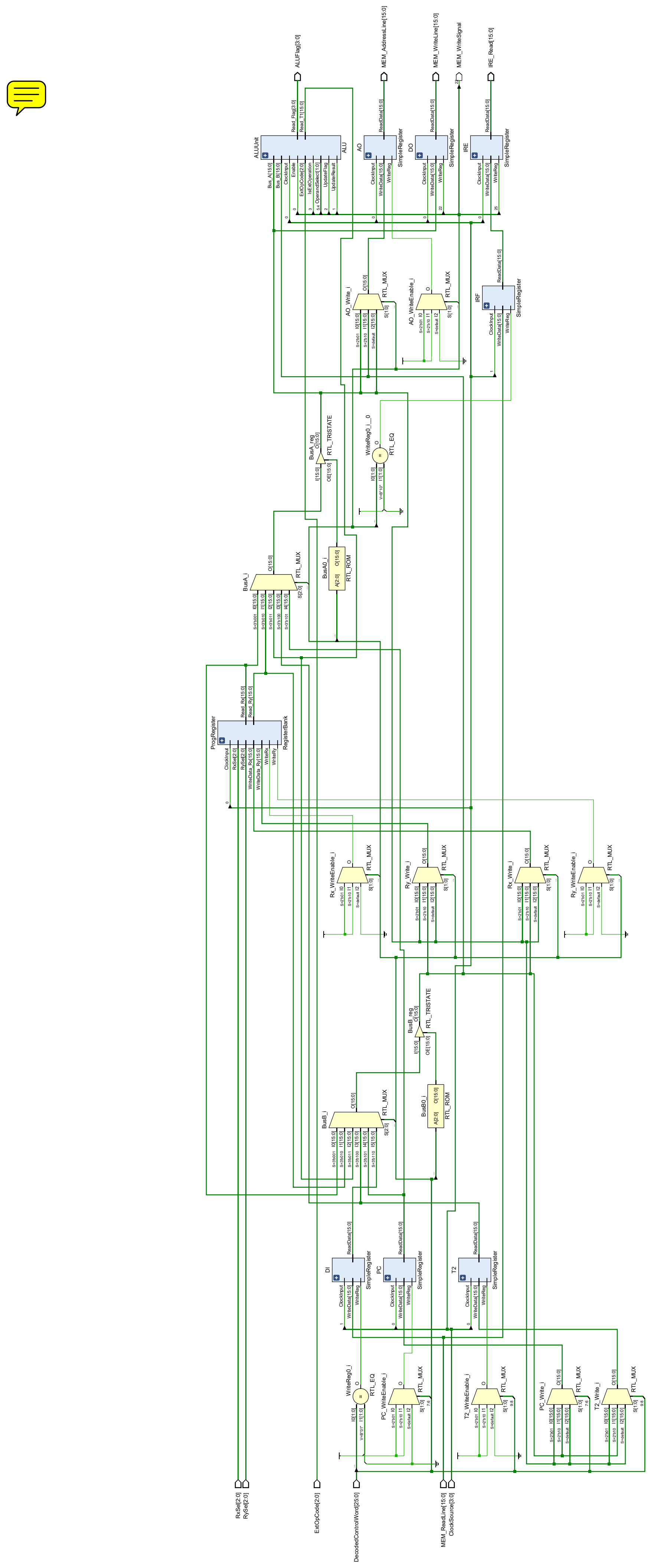
```
[3][15:0]           0001
[2][15:0]           1
[1][15:0]           0001
[0][15:0]           0000
T2                  0
AO                  2
DO                  0
DI                  0
IRF                 5016
IRE_OpCode[15:0]    5016
PC                  3
Read_T1[15:0]       1
ConditionFlag[3:0]  0
```

Note - R2 register holds the result (1,2,3,5…233).

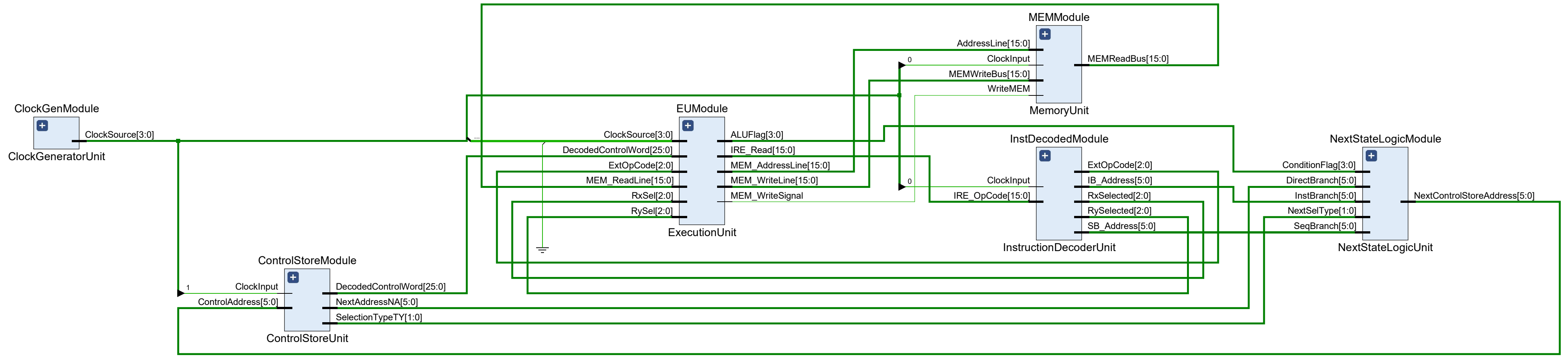
Overall MIN architecture simulation –



Name	Value
ClockSource[3:0]	0
[3]	0
[2]	0
[1]	0
[0]	0
IB_Address[5:0]	19
SB_Address[5:0]	19
ExtOpCode[2:0]	0
RxSelected[2:0]	1
RySelected[2:0]	6
ControlAddress[5:0]	26
DecodedControlWord[25:0]	002800f
SelectionTypeTY[1:0]	3
NextAddressNA[5:0]	26
EUClockSource[3:0]	0
MEM_AddressLine[15:0]	0002
MEM_ReadLine[15:0]	5016
MEM_WriteLine[15:0]	0000
MEM_WriteSignal	0
ProgRegister[7:0][15:0]	233,0,1,11,1,
[7][15:0]	00e9
[6][15:0]	0000
[5][15:0]	0001
[4][15:0]	000b
[3][15:0]	0001
[2][15:0]	1
[1][15:0]	0001
[0][15:0]	0000
T2	0
AO	2
DO	0
DI	0
IRF	5016
IRE_OpCode[15:0]	5016
PC	3
Read_T1[15:0]	1
ConditionFlag[3:0]	0



ALUFlag[3:0]
MEM_AddressLine[15:0]
MEM_WriteLine[15:0]
MEM_WriteSignal
IRE_Read[15:0]
ALUUnit
ALU
AO
SimpleRegister
DO
IRE
IRF
AO_Write_i
AO_WriteEnable_i
RTL_MUX
BusA_reg
RTL_TRISTATE
WriteReg0_i__0
RTL_EQ
BusA_i
BusA0_i
RTL_ROM
ProgRegister
RegisterBank
Rx_WriteEnable_i
Ry_Write_i
Rx_Write_i
Ry_WriteEnable_i
BusB_reg
BusB_i
BusB0_i
DI
PC
T2
WriteReg0_i
PC_WriteEnable_i
T2_WriteEnable_i
PC_Write_i
T2_Write_i
RxSel[2:0]
RySel[2:0]
ExtOpCode[2:0]
DecodedControlWord[25:0]
MEM_ReadLine[15:0]
ClockSource[3:0]


ClockGenModule
ClockSource[3:0]
ClockGeneratorUnit
ControlStoreModule
1
ClockInput
ControlAddress[5:0]
DecodedControlWord[25:0]
NextAddressNA[5:0]
SelectionTypeTY[1:0]
ControlStoreUnit
EUModule
ClockSource[3:0]
DecodedControlWord[25:0]
ExtOpCode[2:0]
MEM_ReadLine[15:0]
RxSel[2:0]
RySel[2:0]
ALUFlag[3:0]
IRE_Read[15:0]
MEM_AddressLine[15:0]
MEM_WriteLine[15:0]
MEM_WriteSignal
ExecutionUnit
MEMModule
0
AddressLine[15:0]
ClockInput
MEMWriteBus[15:0]
WriteMEM
MEMReadBus[15:0]
MemoryUnit
InstDecodedModule
0
ClockInput
IRE_OpCode[15:0]
ExtOpCode[2:0]
IB_Address[5:0]
RxSelected[2:0]
RySelected[2:0]
SB_Address[5:0]
InstructionDecoderUnit
NextStateLogicModule
ConditionFlag[3:0]
DirectBranch[5:0]
InstBranch[5:0]
NextSelType[1:0]
SeqBranch[5:0]
NextControlStoreAddress[5:0]
NextStateLogicUnit