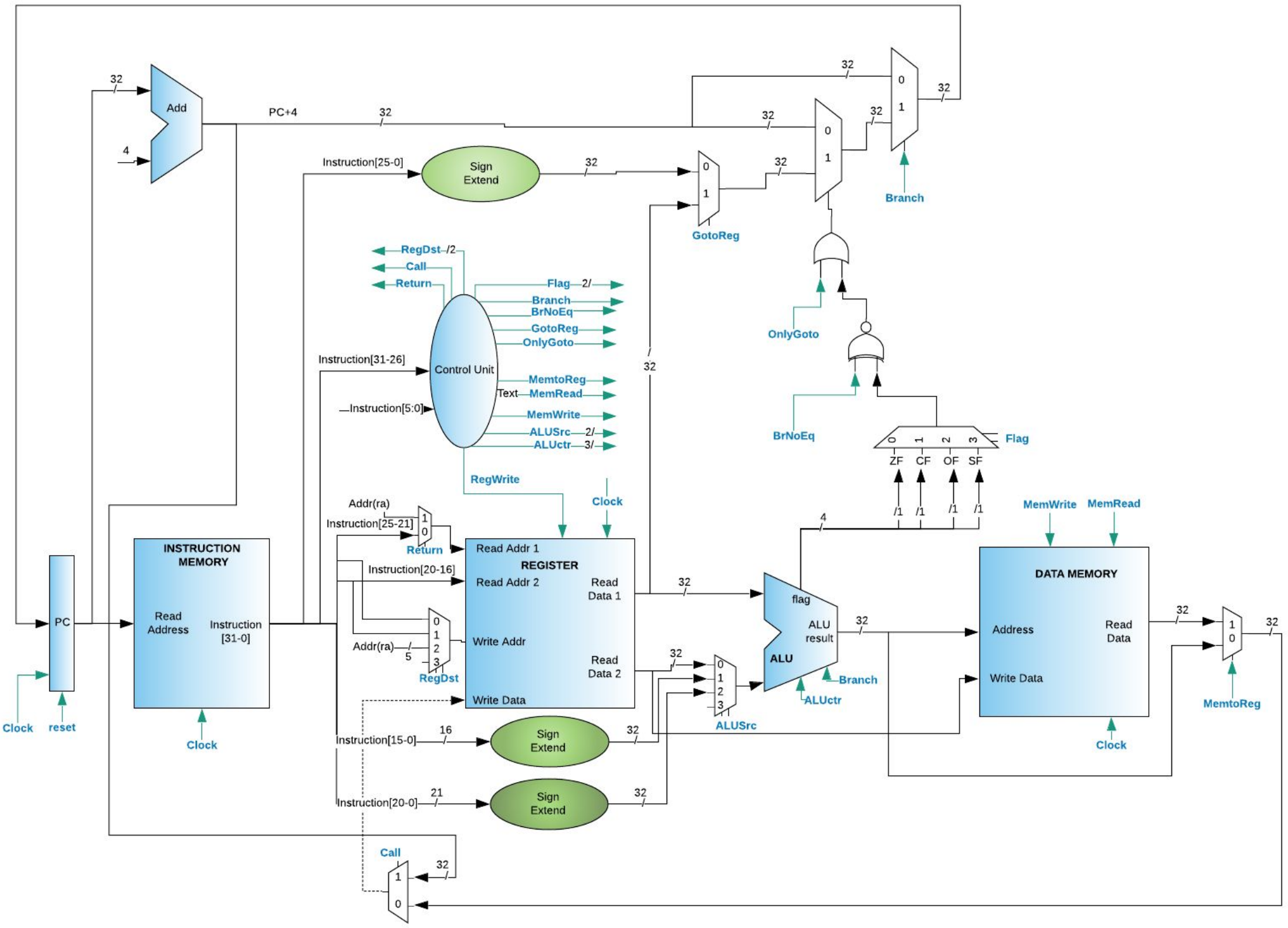
RISC PROCESSOR (SINGLE CYCLE)

VAIBHAV PODDAR (16CS10051)

Contents:

Architecture Diagram of the Processor:-

Vaibhav Poddar 16CS10051



Instruction Set Architecture of this Processor is as show below.

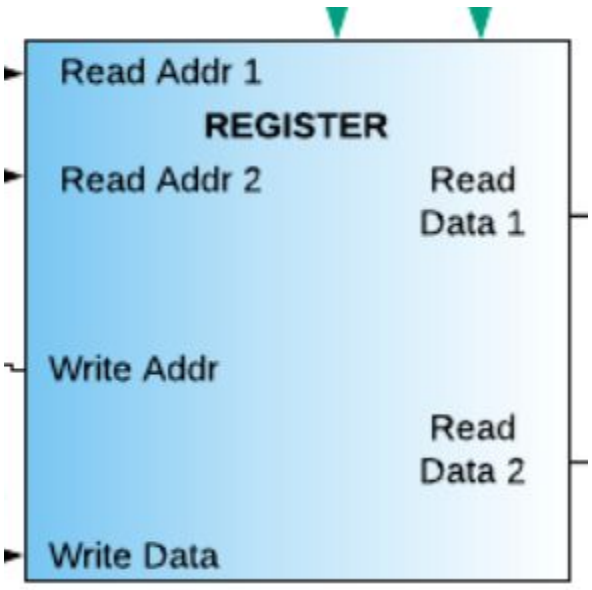
Class	Instruction	Usage	Meaning
Arithmetic	Add	add rs,rt	$rs \leftarrow (rs) + (rt)$
	Comp	comp rs,rt	$rs \leftarrow$ 2's Complement (rt)
	Add immediate	addi rs,imm	$rs \leftarrow (rs) + imm$
	Complement Immediate	compi rs,imm	$rs \leftarrow$ 2's Complement (imm)
Logic	AND	and rs,rt	$rs \leftarrow (rs) \wedge (rt)$
	XOR	xor rs,rt	$rs \leftarrow (rs) \oplus (rt)$
Shift	Shift left logical	shll rs, sh	$rs \leftarrow (rs)$ left-shifted by sh
	Shift right logical	shrl rs, sh	$rs \leftarrow (rs)$ right-shifted by sh
	Shift left logical variable	shllv rs, rt	$rs \leftarrow (rs)$ left-shifted by (rt)
	Shift right logical variable	shrlv rs, rt	$rs \leftarrow (rs)$ right-shifted by (rt)
	Shift right arithmetic	shra rs, sh	$rs \leftarrow (rs)$ arithmetic right-shifted by sh
	Shift right arithmetic variable	shrav rs, rt	$rs \leftarrow (rs)$ right-shifted by (rt)
Memory	Load Word	lw rt,imm(rs)	$rt \leftarrow mem[(rs) + imm]$
	Store Word	sw rt,imm,(rs)	$mem[(rs) + imm] \leftarrow (rt)$
Branch	Unconditional branch	b L	goto L
	Branch Register	br rs	goto (rs)
	Branch on zero	bz L	if $(zflag == 1)$ then goto L
	Branch on not zero	bnz L	if$(zflag == 0)$ then goto L
	Branch on Carry	bcy L	if $(carryflag == 1)$ then goto L
	Branch on No Carry	bncy L	if $(carryflag == 0)$ then goto L
	Branch on Sign	bs	if $(signflag == 1)$ then goto L
	Branch on Not Sign	bns L	if $(signflag == 0)$ then goto L
	Branch on Overflow	bv L	if $(overflowflag == 1)$ then goto L
	Branch on No Overflow	bnv L	if $(overflowflag == 0)$ then goto L
	Call	Call L	$ra \leftarrow$ (PC)+4 ; goto L
	Return	Ret	goto (ra)

Processor uses 32 bit address. Processor word size is 32 bits. It has total 32 registers.

Register File:

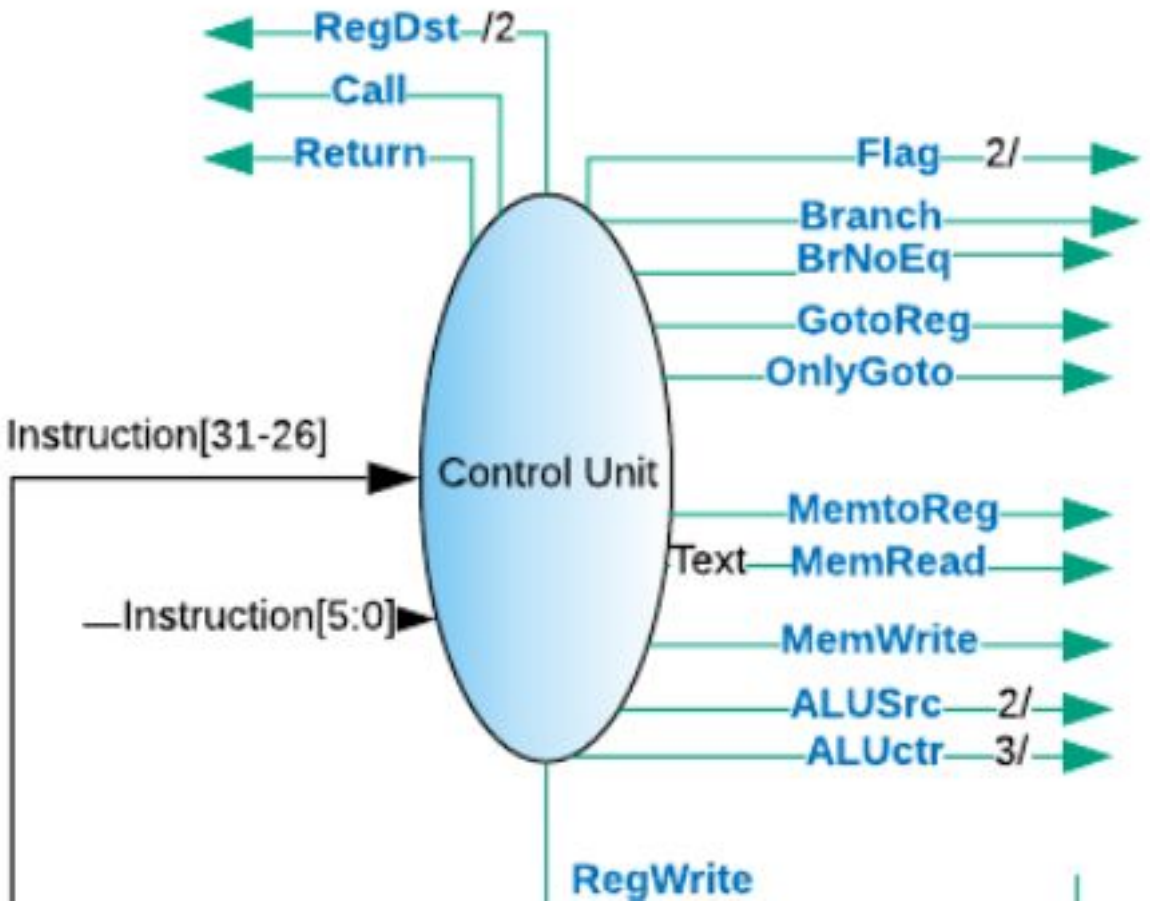
This block contains all 32 registers each of 32 bits.
It has :

- 2 Read Port (Read Addr 1, Read Addr 2)
- 2 Read Data line (Read Data 1, Read Data 2) , which gives the register content of Read Addr 1 and Read Addr 2 respectively.
- 1 Control line for writing into the register file.
- 1 clock
- 1 Write Addr port, where we want to write.
- 1 Write Data, what we want to write.
  Writing into Registers takes place only when “RegWrite” Signal is 1.



Control Unit:

This is the unit which controls all other modules in this processor. This is the heart and soul of this processor, which instructs what do when. This module sets proper signals to each unit as shown in the architecture diagram.
All green lines are output from the Control Unit.
Its input is Instruction Opcode and Fn.
Table1 for decoding this unit is given below for various kinds of Instructions.
RegWrite signal is also the output of from this Control Unit.



R Type:

	RegDst	Flag	Branch	BrNoEq	GotoReg	OnlyGoto	MemtoReg	MemRead	Memwrite	ALUSrc	ALUctr	RegWrite	Call	Return
Add rs,rt	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“00”	“001”	“1”	“0”	“0”
Comp rs,rt	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“00”	“010”	“1”	“0”	“0”
and rs,rt	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“00”	“011”	“1”	“0”	“0”
xor rs,rt	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“00”	“100”	“1”	“0”	“0”
shllv rs, rt	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“00”	“101”	“1”	“0”	“0”
shrlv rs,rt	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“00”	“110”	“1”	“0”	“0”
shrav rs, rt	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“00”	“111”	“1”	“0”	“0”

Immediate:

	RegDst	Flag	Branch	BrNoEq	GotoReg	OnlyGoto	MemtoReg	MemRead	Memwrite	ALUSrc	ALUctr	RegWrite	Call	Return
shra rs, imm	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“10”	“111”	“1”	“0”	“0”
shll rs, imm	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“10”	“101”	“1”	“0”	“0”
shrl rs, imm	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“10”	“110”	“1”	“0”	“0”
addi rs, imm	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“10”	“001”	“1”	“0”	“0”
compi rs,imm	“00”	“xx”	“0”	“x”	“x”	“x”	“0”	“x”	“0”	“10”	“000”	“1”	“0”	“0”

LOAD n STORE:

	RegDst	Flag	Branch	BrNoEq	GotoReg	OnlyGoto	MemtoReg	MemRead	Memwrite	ALUSrc	ALUctr	RegWrite	Call	Return
lw rt, imm(rs)	“01”	“xx”	“0”	“x”	“x”	“x”	“1”	“1”	“0”	“01”	“001”	“1”	“0”	“0”
sw rt, imm(rs)	“xx”	“xx”	“0”	“x”	“x”	“x”	“x”	“0”	“1”	“01”	“001”	“0”	“0”	“0”

BRANCH:

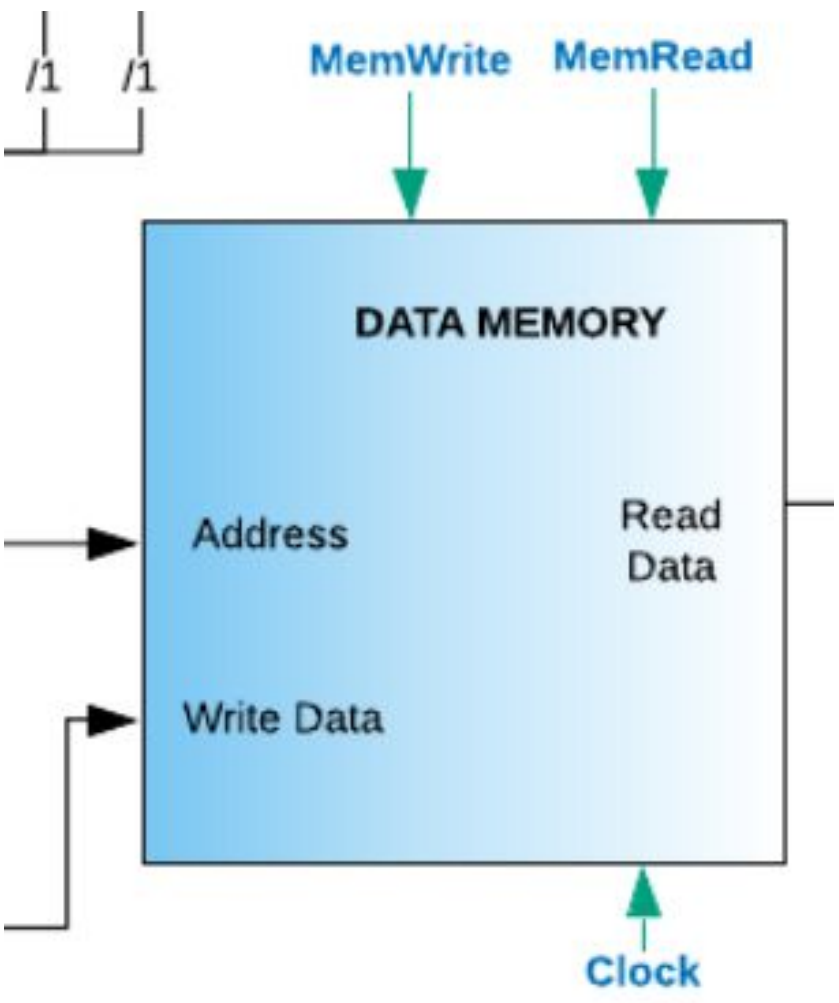
	RegDst	Flag	Branch	BrNoEq	GotoReg	OnlyGoto	MemtoReg	MemRead	Memwrite	ALUSrc	ALUctr	RegWrite	Call	Return
bz L	“xx”	“00”	“1”	“1”	“0”	“0”	“x”	“x”	“0”	“xx”	“xxx”	“0”	“x”	“0”
bnz L	“xx”	“00”	“1”	“0”	“0”	“0”	“x”	“x”	“0”	“xx”	“xxx”	“0”	“x”	“0”
bcy L	“xx”	“01”	“1”	“1”	“0”	“0”	“x”	“x”	“0”	“xx”	“xxx”	“0”	“x”	“0”
bncy L	“xx”	“01”	“1”	“0”	“0”	“0”	“x”	“x”	“0”	“xx”	“xxx”	“0”	“x”	“0”
bs L	“xx”	“11”	“1”	“1”	“0”	“0”	“x”	“x”	“0”	“xx”	“xxx”	“0”	“x”	“0”
bns L	“xx”	“11”	“1”	“0”	“0”	“0”	“x”	“x”	“0”	“xx”	“xxx”	“0”	“x”	“0”
bv L	“xx”	“10”	“1”	“1”	“0”	“0”	“x”	“x”	“0”	“xx”	“xxx”	“0”	“x”	“0”
bnv L	“xx”	“10”	“1”	“0”	“0”	“0”	“x”	“x”	“0”	“xx”	“xxx”	“0”	“x”	“0”
b L	“xx”	“xx”	“1”	“x”	“0”	“1”	“x”	“x”	“0”	“xx”	“xxx”	“0”	“x”	“0”
br rs	“xx”	“xx”	“1”	“x”	“1”	“1”	“x”	“x”	“0”	“xx”	“xxx”	“0”	“x”	“0”
Call L	“10”	“xx”	“1”	“x”	“0”	“1”	“x”	“x”	“0”	“xx”	“xxx”	“1”	“1”	“0”
Ret	“xx”	“xx”	“1”	“x”	“1”	“1”	“x”	“x”	“0”	“xx”	“xxx”	“0”	“x”	“1”

(Please refer to “Control_Signals.txt” file present in FILES folder)

TABLE1

DATA MEMORY:

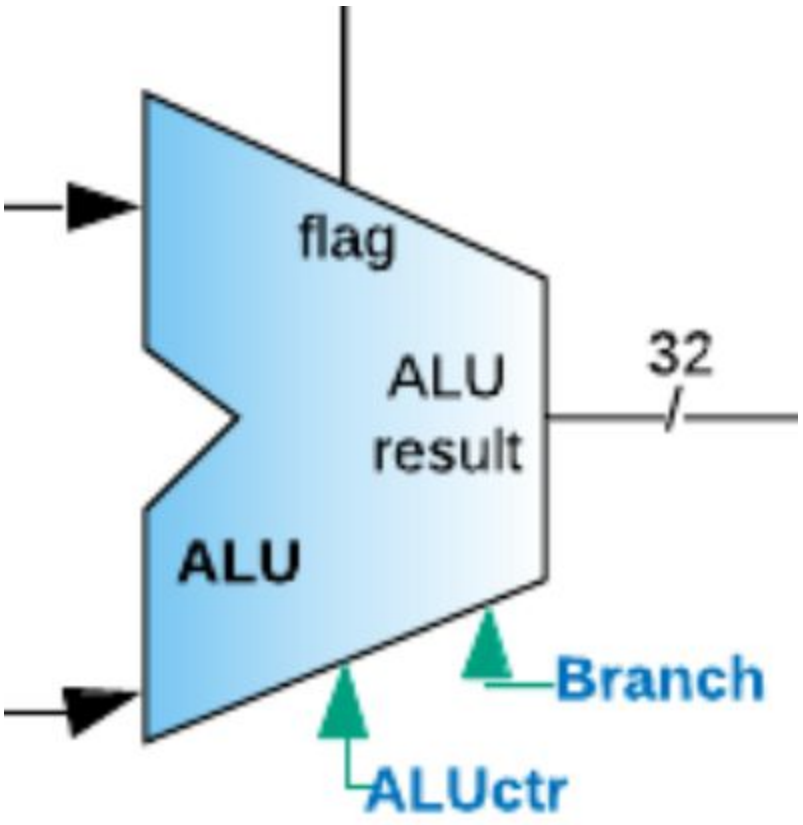
This is for storing data. Here data for sorting array by bubble sort is saved.
It is Memory Write and Memory Read Control Signal.
This is 1 port input 1 port output unlike Register File.



ALU:

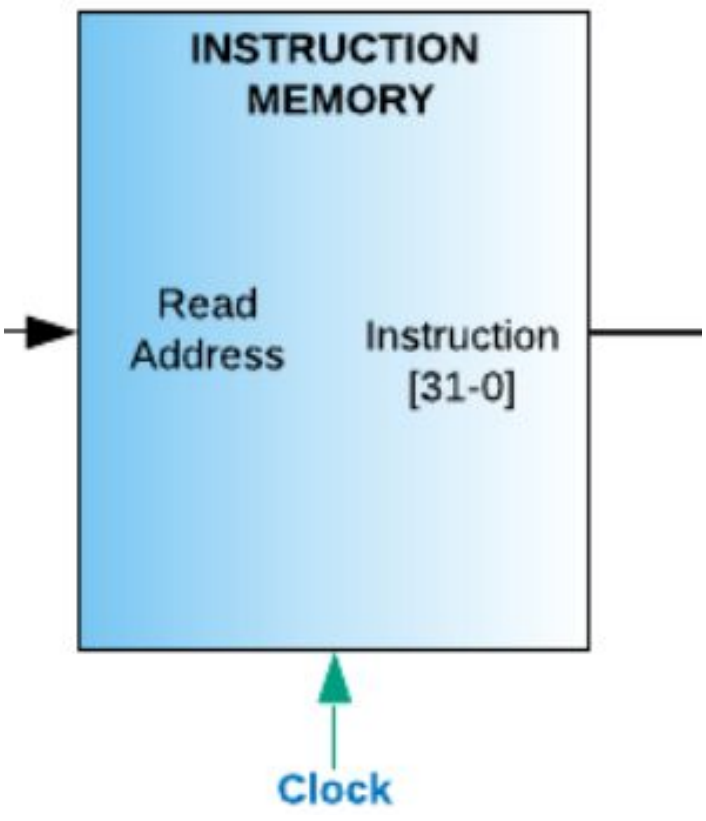
This is used for performing all Mathematical Operations like Additions, Shifting, 2's Complement, AND, XOR.
This is a combinational circuit.
It gives result based on what operation it has performed. It has gives 4 different kinds of flags:

- Zero Flag,
- Carry Flag,
- Overflow Flag and
- Sign Flag.



INSTRUCTION MEMORY:

Instruction Memory is used to store all the instructions which we want out processor to execute.
It takes the Address of the instruction and outputs the 32 bits instruction which is present is that input address.



Instruction Formats used while creating this processor are:

(Please refer to "Instruction_Format.txt" file present in FILES folder)

R-type instructions:

add rs, rt	000000	--rs-	--rt-	xxxxx	xxxxx	000001	ADD
comp rs, rt	000000	--rs-	--rt-	xxxxx	xxxxx	000010	2's COMP
and rs, rt	000000	--rs-	--rt-	xxxxx	xxxxx	000011	AND
xor rs, rt	000000	--rs-	--rt-	xxxxx	xxxxx	000100	XOR
shllv rs, rt	000000	--rs-	--rt-	xxxxx	xxxxx	000101	Shift L
shrlv rs, rt	000000	--rs-	--rt-	xxxxx	xxxxx	000110	Shift R
shrav rs, rt	000000	--rs-	--rt-	xxxxx	xxxxx	000111	Shift R with proper sign

Load Store instructions:

lw rt, imm(rs)	000001	--rs-	--rt-	------imm----------	ADD
sw rt, imm(rs)	000010	--rs-	--rt-	------imm----------	ADD

Immidiate instructions:

addi rs, imm	100001	--rs-	----------------imm----------		ADD
compi rs, imm	100010	--rs-	----------------imm----------		2's COMP
shll rs, sh	100011	--rs-	---------------sh-------------		Shift L
shrl rs, sh	100100	--rs-	---------------sh-------------		Shift R
shra rs, sh	100101	--rs-	---------------sh-------------		Shift R with proper sign

Branch instructions:

bz L	110001	------------------------L-------------		
bnz L	110010	------------------------L-------------		
bcy L	110011	------------------------L-------------		
bncy L	110100	------------------------L-------------		
bs L	110101	------------------------L-------------		
bns L	110110	------------------------L-------------		
bv L	110111	------------------------L-------------		
bnv L	111000	------------------------L-------------		
b L	111001	------------------------L-------------		
br rs	111010	--rs-	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	
Call L	111101	------------------------L-------------		
Ret	111100	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx		

Testing the Processor by running Bubble SORT code:

NOTE: Modified array is stored in Data Memory. For showing the Output of the sorted array, register $20 of register file is used.
Assuming array is stored in data Memory with starting address 0 and total 8 elements are present. Array is unsorted.

DATA in data memory : *Array containing numbers : [15 , 1 , 63 , 3 , 225 , 7 , 127, 31]*

(Please refer to "Bubble_Sort.txt" file present in FILES folder)

```
main:
   compi $1, 0
   b BubbleSort

BubbleSort:
      compi $3, -7        // $3 is storing -7
      add $3, $1
      bz Sortingover
      compi $2, 0              //Making $2 to zero for inner for loop
      b BubbleSortUtil

BubbleSortUtil:
      compi $5, -28       //Storing $5 = -28
      add $5, $2
      bz Increment
      lw $6, 0($2)
      compi $7,0
      add $7, $2
      addi $7, 4          // $7 = j+1
      lw $8, 0($7)
      compi $9,11111111111111111111111111111111
      xor $9, $8
      addi $9, 1          //$9 = -$8
      add $9, $6
      bns Swap
      addi $2, 4
      b BubbleSortUtil

Increment:
      addi $1,1
      b  BubbleSort

Swap:
   sw $6, $7(A)
   sw $8, $2(A)
   addi $2, 4
   b BubbleSortUtil

SortingOver:
      compi $10, 0
      lw $20, 0($10)
      lw $20, 4($10)
      lw $20, 8($10)
      lw $20, 12($10)
      lw $20, 16($10)
      lw $20, 20($10)
      lw $20, 24($10)
      lw $20, 28($10)
      //termination of the program

//$20 will store the final result.
```

****

Above is converted to a new code so that no labels are present:

1. *compi $1, 0*
2. *b BubbleSort*
3. *compi $3, -7*
4. *add $3, $1*
5. *bz Sortingover*
6. *compi $2, 0*
7. *b BubbleSortUtil*
8. *compi $5, -28*
9. *add $5, $2*
10. *bz Increment*
11. *lw $6, 0($2)*
12. *compi $7,0*
13. *add $7, $2*
14. *addi $7, 4*
15. *lw $8, 0($7)*
16. *compi $9,11111111111111111111111111111111*
17. *xor $9, $8*
18. *addi $9, 1*
19. *add $9, $6*
20. *bns Swap*
21. *addi $2, 4*
22. *b BubbleSortUtil*
23. *addi $1,1*
24. *b BubbleSort*
25. *sw $6, 0($7)*
26. *sw $8, 0($2)*
27. *addi $2, 4*
28. *b BubbleSortUtil*
29. *compi $10, 0*
30. *lw $20, 0($10)*
31. *lw $20, 4($10)*
32. *lw $20, 8($10)*
33. *lw $20, 12($10)*
34. *lw $20, 16($10)*
35. *lw $20, 20($10)*
36. *lw $20, 24($10)*
37. *lw $20, 28($10)*

<u>This code is converted into Machine Code and the converted code is given below:</u>

<u>(Please refer to “Bubble_Sort_Instructions.coe” file present in FILES folder)</u>

This is initialised to Instruction Memory.

Size of Instruction Memory is ***256 bytes.***

10001000001000000000000000000000,
11100100000000000000000000001000,
10001000011111111111111111111001,
00000000011000010000000000000001,
11000100000000000000000001110000,
10001000010000000000000000000000,
11100100000000000000000000011100,
10001000101111111111111111100100,
00000000101000100000000000000001,
11000100000000000000000001011000,
00000100010001100000000000000000,
10001000111000000000000000000000,
00000000111000100000000000000001,

10000100111000000000000000000100,
00000100111010000000000000000000,
10001001001111111111111111111111,
00000001001010000000000000000100,
10000101001000000000000000000001,
00000001001001100000000000000001,
11011000000000000000000001100000,
10000100010000000000000000000100,
11100100000000000000000000011100,
10000100001000000000000000000001,
11100100000000000000000000001000,
00001000111001100000000000000000,
00001000010010000000000000000000,
10000100010000000000000000000100,
11100100000000000000000000011100,
10001001010000000000000000000000,
00000101010101000000000000000000,
00000101010101000000000000000100,
00000101010101000000000000001000,
00000101010101000000000000001100,
00000101010101000000000000010000,
00000101010101000000000000010100,
00000101010101000000000000011000,
00000101010101000000000000011100;

<u>Data Memory information:</u>

<u>(Please refer to “DataMemory_Unsorted_Array.coe” file present in FILES folder)</u>

Size of Data Memory is ***64 bytes.***

00000000000000000000000000001111,
00000000000000000000000000000001,
00000000000000000000000000111111,
00000000000000000000000000000011,
00000000000000000000000011111111,
00000000000000000000000000000111,
00000000000000000000000001111111,
00000000000000000000000000011111,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000;

DATA in data memory : *Array containing numbers : [15 , 1 , 63 , 3 , 225 , 7 , 127, 31]*
Output After sorting : [1 , 3 , 7 , 15 , 31 , 63 , 127 , 225]

SIMULATION RESULTS

[See last figure for instant results.]

<u>Signals:</u>

- ***Result*** *: to show the sorted array.*
- ***Read_addr****: to show which instruction number is executing.*
- ***Instr****: binary representation of instruction in 32 bits.*
- ***Clock****: clock of this processor.*
- ***Reset****: Reset the whole processor.*

Fig (i) *at time 0 ns*

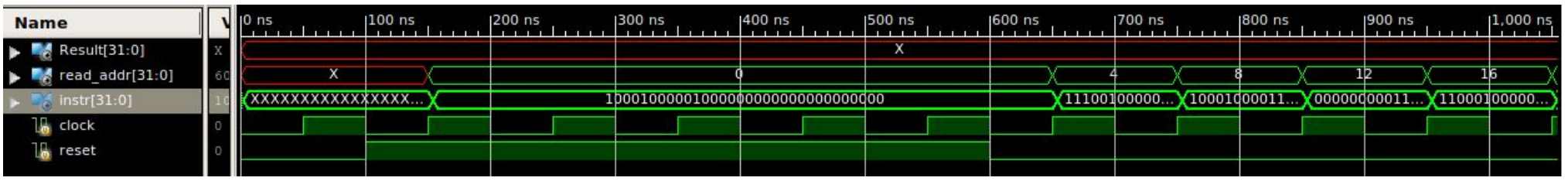


Fig (ii) *at time 10 ns*

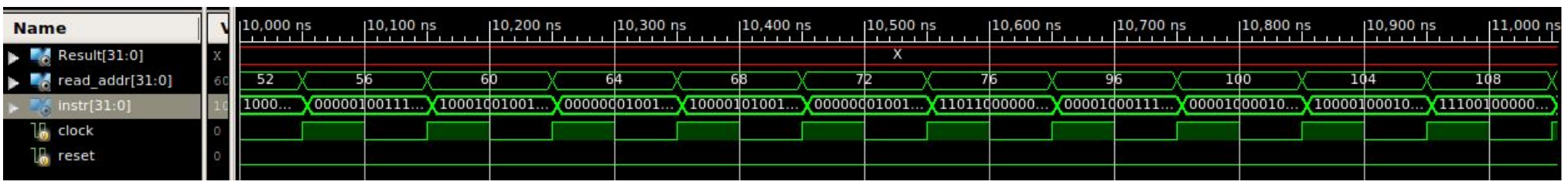


Fig (iii) *at time 20 ns*

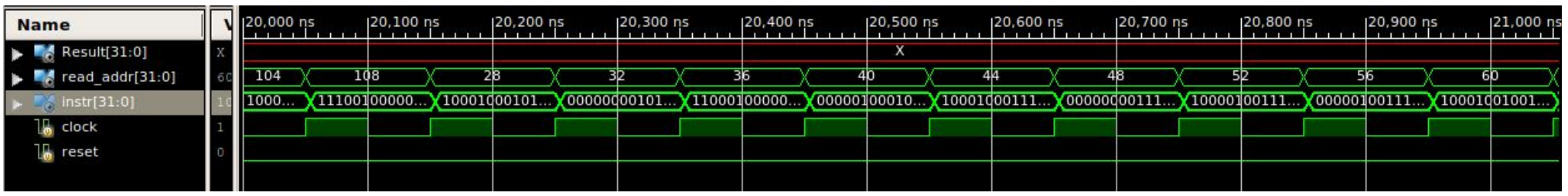


Fig (iv) *at time 30 ns*

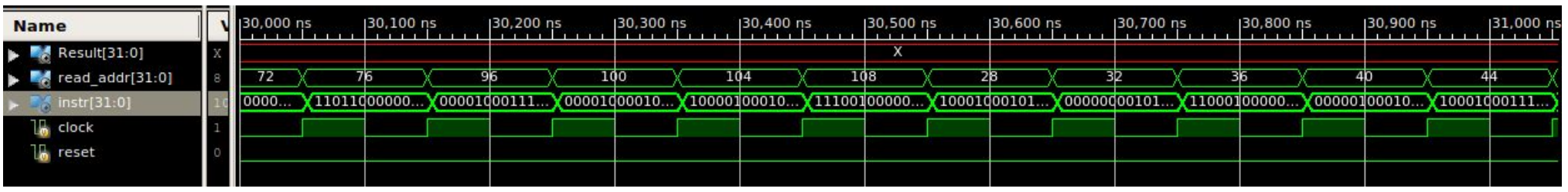


Fig (v) *at time 40 ns*

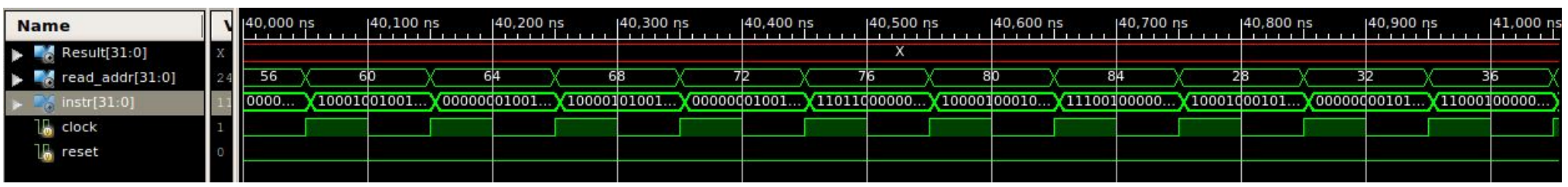


Fig (vi) *at time 50 ns*

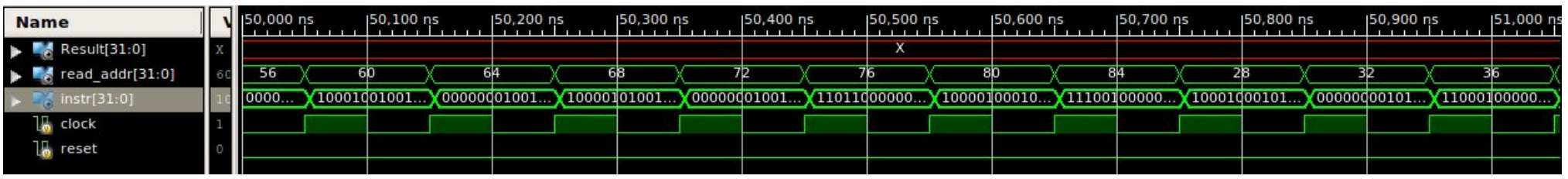


Fig (vii) *at time 60 ns*

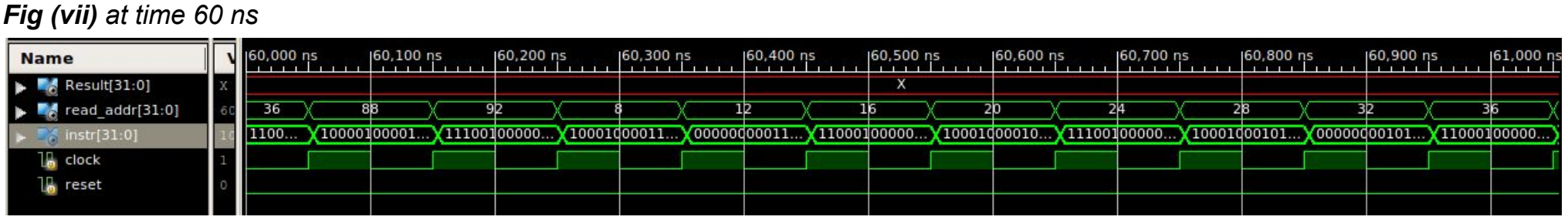


Fig (viii) *at time 70 ns*

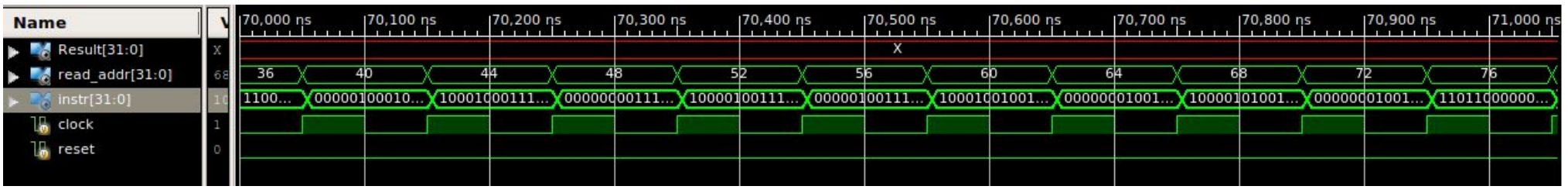


Fig (ix) *at time 80 ns*

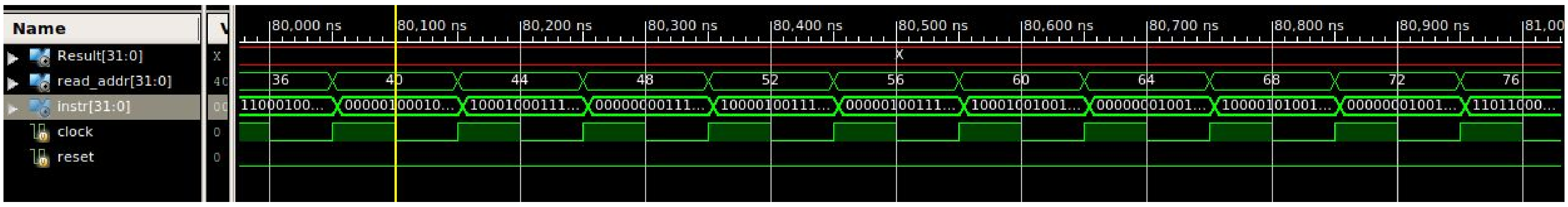
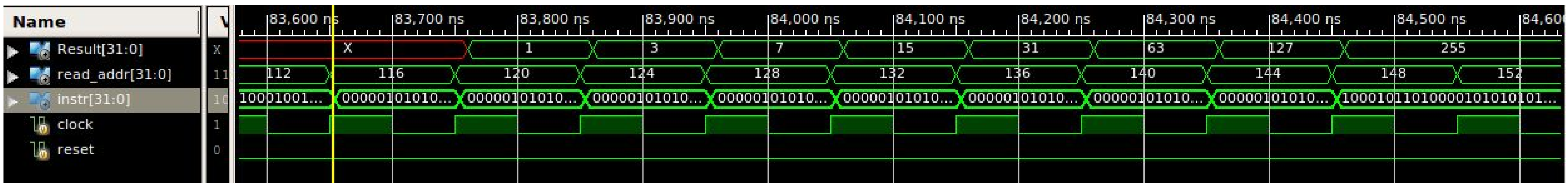


__Fig (x)__ at time 83 ns



HOW TO USE:

1. Import all the verilog files in the project.
    a. In case there is an issue with Instruction Memory, Double click on it and add the file “Bubble_Sort_Instructions.coe” in Instruction Memory.
    b. In case there is an issue with Data Memory, Double click on it and add the file “DataMemory_Unsorted_Array.coe” in Data Memory.
2. Run the “Test_Processor.v”.
3. Run the simulation up till ~83ns to 90 ns to see the output.

THE END