NORTH SOUTH UNIVERSITY

Department of Electrical and Computer Science Engineering (ECE).

Computer Organization and Architecture
Course Code: CSE332.
Section:01.
Project Report.

Submitted To:
Mohammad Abdul Qayum.
Assistant Professor.
Department of Electrical and Computer Science Engineering.
North South University.

Wardun Islam.
Lab instructor.
North South University.

Submitted By:
MD. Abu Yousuf Neshad.
ID:2212517042.

Instruction Format

1. **Operation Code (Opcode):** Specifies the operation or instruction to be executed. It tells the CPU what operation to perform, such as addition, subtraction, loading a value from memory, or branching to a different part of the program.
2. **Source Register(s):** This field identifies the source of the data or the registers involved in the operation. It could be one or more registers or memory locations, depending on the instruction.
3. **Destination Register:** This field identifies the destination where the result of the operation should be stored. This can be a register or memory location.
4. **Immediate Value:** Some instructions may include an immediate value or constant as one of their operands. This value is directly encoded within the instruction itself, rather than being fetched from a register or memory.
5. **Address/Offset:** For memory-related instructions, an address or offset field may be included to specify a memory location. This field might be an absolute memory address or an offset relative to a base address.

R-Type Instructions (Register Type):

- **Opcode:** Specifies the operation to be performed (e.g., add, subtract).
- **Source Register 1(Rs):** The first source register.
- **Source Register 2(Rt):** The second source register.
- **Destination Register (Rd):** The Register where the result is stored.
- **Shift Amount (Shmt):** Some R-type instructions may include a field for specifying the shift

Opcode	Rs	Rt	Rd	Shmt
(24-21)	(20-16)	(15-11)	(10-6)	(5-0)

I-Type Instructions (Immediate Type):

- **Opcode:** Specifies the operation (e.g., load, store, addi).
- **Source Register:** The source register.
- **Destination Register:** The destination register (for results).
- **Immediate Value:** A constant or immediate value used in the operation.

Opcode	Rs	Rd	immediate
(24-21)	(20-16)	(15-11)	(10-0)

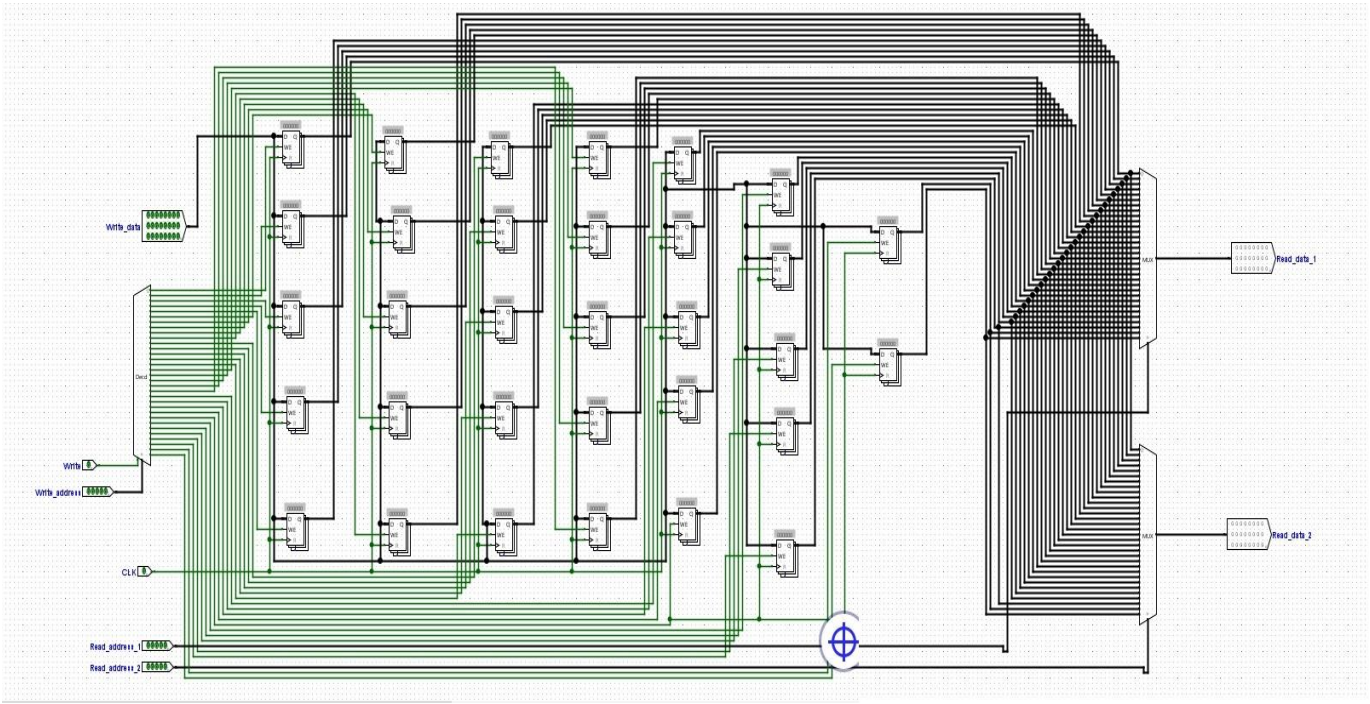
J-Type Instructions (Jump Type):

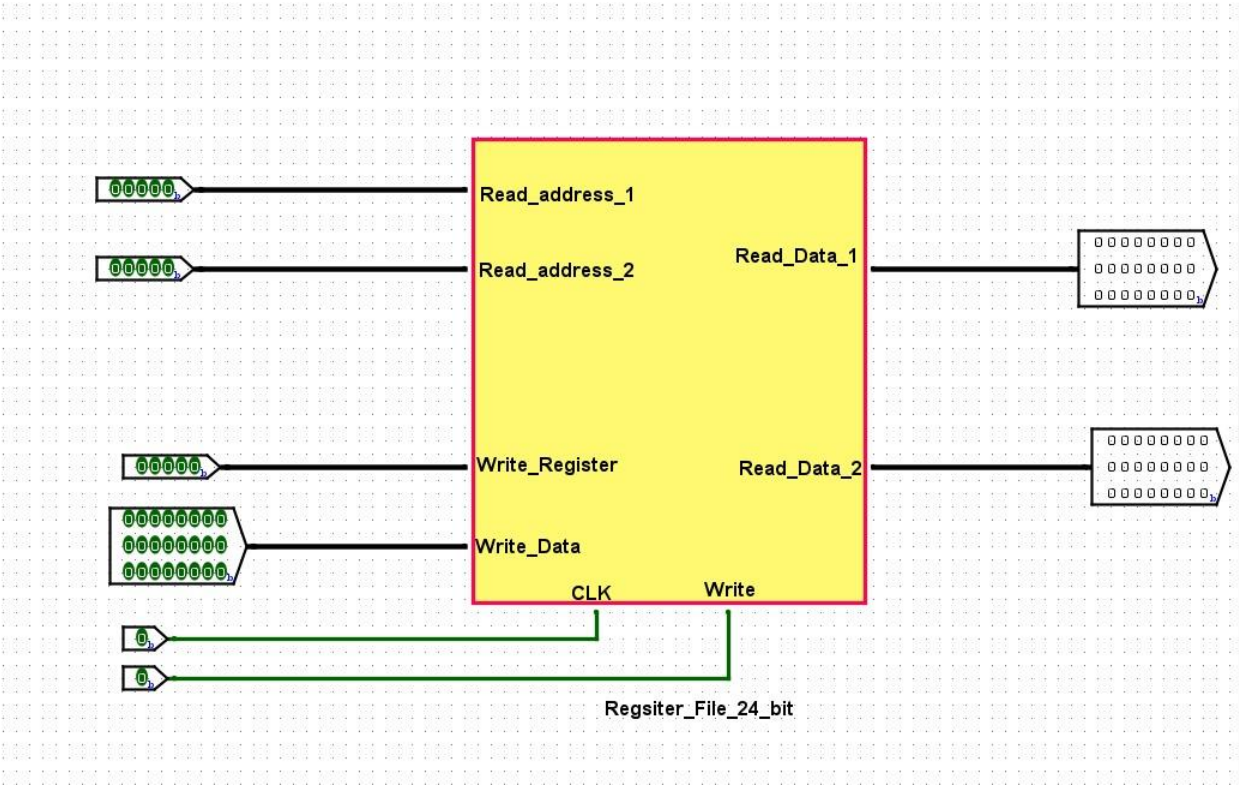
- **Opcode:** Specifies the operation (e.g., jump, branch).
- **Target Address:** The target address to which the program should jump or branch.

Opcode	Address
(24-21)	(20-0)

Register File

My register file is 24-bit. Since I have 5 bits in Rs, Rt, and Rd, the number of registers in the register file is 2^5=32.

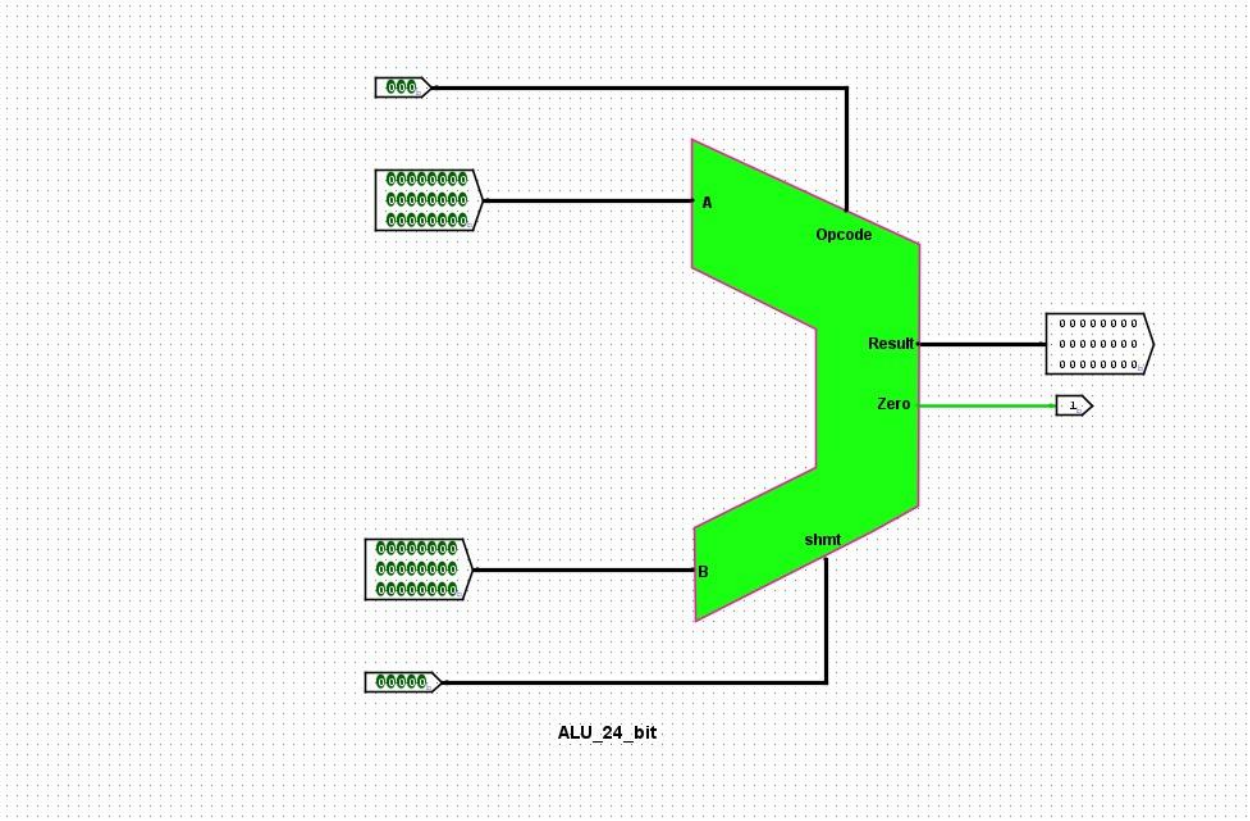




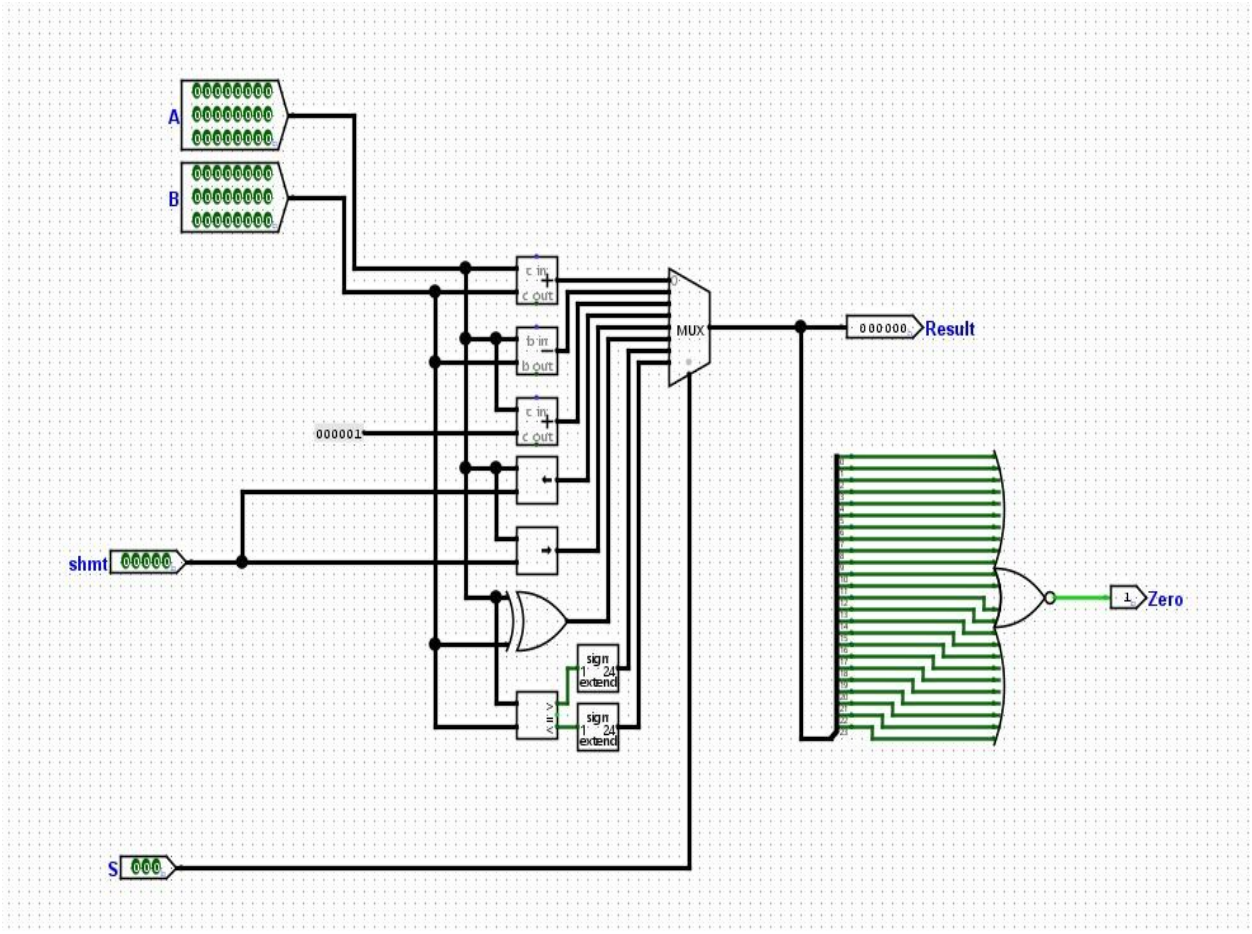
ALU

Alu Table

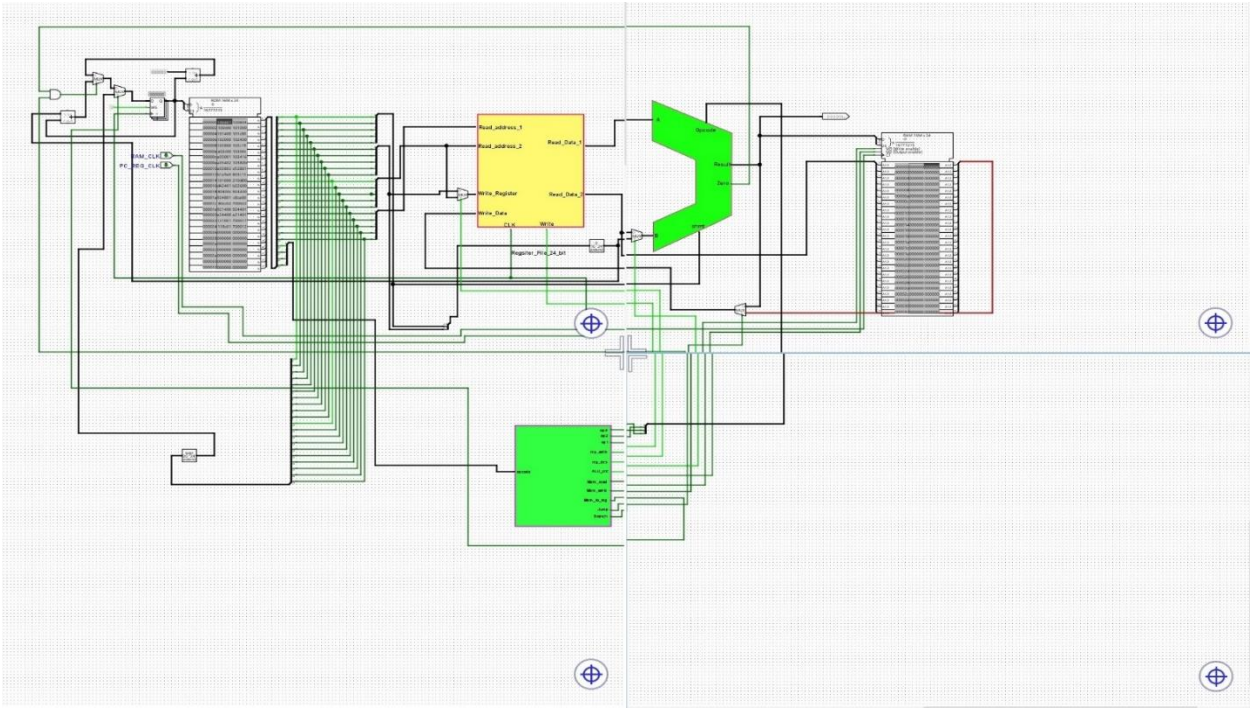
ALUOpCode			Operation
S2	S1	S0	
0	0	0	**ADD**
0	0	1	**SUB**
0	1	0	**INC**
0	1	1	**SHL**
1	0	0	**SHR**
1	0	1	**XOR**
1	1	0	**SGT**
1	1	1	**SLT**


000
00000000
00000000
00000000
A
Opcode
Result
00000000
00000000
00000000
Zero
1
shmt
00000000
00000000
00000000
B
00000

ALU_24_bit


A
B
c_in
c_out
b_in
b_out
000001
MUX
000000
Result
shmt
00000
1
Zero
sign
1
24
extend
S
000

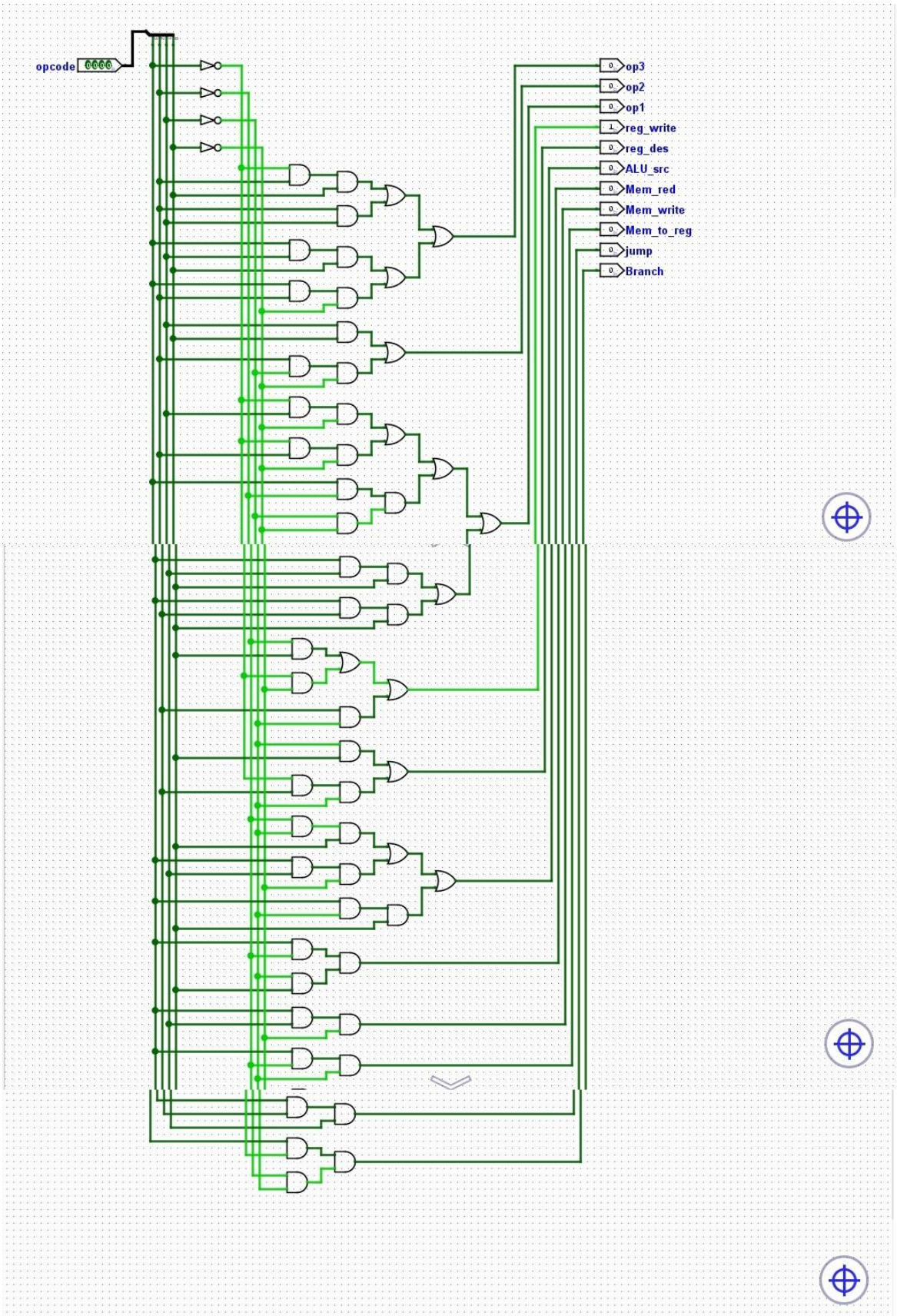
DATAPATH


Read_address_1
Read_address_2
Write_Register
Write_Data
Read_Data_1
Read_Data_2
CLK
WrEN
RAM_CLK
PC_MEM_CLK

CONTROL TABLE

Control Table

Operation	Opcode	ALU_ operation	ALUOP	Reg_ write	Reg_ Dest	Alu_ src	Mem_ read	Mem_ write	MemTo_ reg	Jump	Branch
ADD	0000	ADD	000	1	0	0	0	0	0	0	0
ADDI	0001	ADD	000	1	1	1	0	0	0	0	0
SUB	0010	SUB	001	1	0	0	0	0	0	0	0
INC	0011	INC	010	1	0	0	0	0	0	0	0
SHL	0100	SHL	011	1	1	0	0	0	0	0	0
SHR	0101	SHR	100	1	1	0	0	0	0	0	0
XOR	0110	XOR	101	1	0	0	0	0	0	0	0
JUMP	0111	***	***	0	*	*	0	0	*	1	0
BRANCH	1000	SUB	001	0	*	0	0	0	*	0	1
LOAD	1001	ADD	000	1	1	1	1	0	1	0	0
STORE	1010	ADD	000	0	*	1	0	1	*	0	0
SLT	1011	SLT	111	1	0	0	0	0	0	0	0
SGT	1100	SGT	110	1	0	0	0	0	0	0	0
SUBI	1101	SUB	001	1	1	1	0	0	0	0	0


opcode
0000
op3
op2
op1
reg_write
reg_des
ALU_src
Mem_red
Mem_write
Mem_to_reg
jump
Branch

C code:

```
#include <stdio.h>
int main() {
   int n = 10; // The first number in the series
   int sum = 0;
   int i=0;
   int j=3;

  while(i!=n){
      sum += j; // Add the current number to the sum
      i+=1;
      j+=3;// Increment n by 3 to get the next number in the series
   }

   printf("The sum : %d\n",sum);

   return 0;
}
```

MIPS:

R1=sum

R2=n

R3=i

R4=j

0. addi r0, r2, 10

opcode rs rd im

0001 00000 00010 00000 01010

10080a

1. addi r0, r1, 0

opcode rs rd im

0001 00000 00001 00000 00000

100400

2. addi r0, r3, 0

opcode rs rd im

0001 00000 00011 00000 00000

100C00

3. addi r0, r4, 3

opcode rs rd im

0001 00000 00100 00000 00011

101003

4. beq r2, r3, 6

opcode rs rd im

1000 00010 00011 00000 00110

810C06

5. M[r3+0]<-r4

opcode rs rd im

1010 r3 r4 0

1010 00011 00100 00000 00000

a19000

6. Add r1, r4, r1

opcode rs rt rd shmt

00000 00001 00100 00001 00000

009020

7. Addi R3 R3 1

opcode rs rd im

0001 00011 00011 00000 00001

118C01

8. Addi R4 R4 3

opcode rs rd im

0001 00100 00100 00000 00011

121003

9. jump 4

opcode address

0111 00000 00000 00000 00100

700004

Hexadecimal

10080a 100400 100C00 101003 810C06 a19000 009020 118C01 121003 700004

Bubble sort in C:

```
#include <stdio.h>

int main() {
   int arr[] = {25, 12, 22, 11};
   int n = 4;
   int temp = 0;

   int i = 0;
   int a=0;
   int b=0;
   int c=0;
   int d=0;

   d=n-1;
   while (i < d) {
      int j = 0;
      b = n - i;
      c=b-1;
      while (j < c) {
         a = j + 1;
```

```
            if (arr[j] > arr[a]) {
                temp = arr[j];
                arr[j] = arr[a];
                arr[a] = temp;
            }
            j++;
        }
        i++;
    }

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Bubble sort in MIPS:

r2=n

r3=i;

r4=j

r5=temp

r7=a;

r8=b;

r9=c;

r10=d;

0. r1=r0-1

opcode rs rd im

subi r0 r1 1

1101 00000 00001 00000 00001

d00401

1. addi r0 r2 4

opcode rs rd im

0001 00000 00010 00000 00100

100804

2. addi r0 r3 0

opcode rs rd im

0001 00000 00011 00000 00000

100C00

3. addi r0 r4 0

opcode rs rd im

0001 00000 00100 00000 00000

101000

4. addi r0 r5 0

opcode rs rd im

0001 00000 00101 00000 00000

101400

5. addi r0 r7 0

opcode rs rd im

0001 00000 00111 00000 00000

101C00

6. addi r0 r8 0

opcode rs rd im

0001 00000 01000 00000 00000

102000

7. addi r0 r9 0

opcode rs rd im

0001 00000 01001 00000 00000

102400

8. addi r0 r10 0

opcode rs rd im

0001 00000 01010 00000 00000

102800

9.addi r0 r11 25

opcode rs rd im

0001 00000 01011 00000 11001

102c19

10. M[r0+0]<- r11

sw rs rd im

1010 00000 01011 00000 00000

a02c00

11.addi r0 r12 12

opcode rs rd im

0001 00000 01100 00000 01100

10300c

12. M[r0+1]<- r12

sw rs rd im

1010 00000 01100 00000 00001

a03001

13.addi r0 r13 22

opcode rs rd im

0001 00000 01101 00000 10110

103416

14. M[r0+2]<- r13

sw rs rd im

1010 00000 01101 00000 00010

a03402

15.addi r0 r14 11

opcode rs rd im

0001 00000 01110 00000 01011

10380B

16. M[r0+3]<- r14

sw rs rd im

1010 00000 01110 00000 00011

a03803

17. d=n-1;

subi r2 r10 1

1101 00010 01010 00000 00001

d12801

18. i < d

slt r3 r10 r15

opcode rs rt rd

1011 r3 r10 r15

1011 00011 01010 01111 00000

b1a9e0

19. beq r0 r15 19

1000 00000 01111 00000 10011

803c13

20.addi r0 r4 0

opcode rs rd im

0001 00000 00100 00000 00000

101000

21. sub r2 r3 r8

opcode rs rt rd shmt

0010 r2 r3 r8 0

0010 00010 00011 01000 00000

210d00

22. subi r8 r9 1

opcode rs rd im

1101 r8 r9 1

1101 01000 01001 00000 00001

d42401

23. j < c

slt r4 r9 r16

opcode rs rt rd

1011   r4 r9 r16

1011 00100 01001 10000 00000

b22600

24. beq r0 r16 12

beq r0 r16 12

1000 00000 10000 0000001100

80400C

25. r17<-M[j+0]

lw r4 r17 0

1001 00100 10001 00000 00000

924400

26.r18<-M[j+1]

lw r4 r18 1

1001 00100 10010 00000 00001

924801

27. arr[j] > arr[j+1]

sgt r17 r18 r19

1100 10001 10010 10011 00000

c8cA60

28. beq r1 r19 2

1000 00001 10011 00000 00010

80cc02

29. jump 34

0111 00000 00000 00001 00010

700022

30.r5<-M[j+0]

lw r4 r5 0

1001 00100 00101 00000 00000

921400

31. M[j+1]->r17

lw rs rd im

1001 r4 r17 1

1001 00100 10001 00000 00001

924401

32. r17->M[j+0]

sw rs rd im

1010 r4 r17 0

1010 00100 10001 00000 00000

a24400

33.M[j+1] <-temp

sw r4 r5 1

1010 00100 00101 00000 00001

a21401

34. j++;

r4=r4+1

addi r4 r4 1

0001 00100 00100 00000 00001

121001

35.jump 23

0111 00000 00000 00000 10111

700017

36.i=i+1

r3=r3+1

addi rs rd im

0001 00011 00011 00000 00001

118c01

37.jump 18

opcode address

0111 00000 00000 00000 10010

700012

38.

Bubble sort Hexadecimal:

d00401 100804 100C00 101000 101400 101C00 102000 102400 102800 102c19
a02c00 10300c a03001 103416 a03402 10380B a03803 d12801 b1a9e0 803c13
101000 210d00 d42401 b22600 80400C 924400 924801 c8ca60 80cc02 700022
921400 924401 a24400 a21401 121001 700017 118c01 700012