OBSERVATION

INDEX

Name : A. NITHISH Std. : Sec. :

Roll No. : 192121068 Sub :

S.no	Date	Title	Page no.	Teacher's Sign
27	27/03	Booth Algorithm	33	
28	27/03	Register Transition program	34	
29	27/03	Single Bus organisation	35	
30	27/03	multiple Bus organisation	36	
31	28/03	Two Stage pipelining	37	
32	28/03	Four stage pipelining	38	
33	28/03	Static prediction	39	
34	28/03	Data Hazards	40	
35	28/03	Instruction Hazards	41	
36	28/03	Structural Hazards	42	
37	28/03	Super Scalar processing	43	
38	28/03	Dynamic prediction	44	
39	28/03	decimal to Hexadecimal	45	
40	28/03	Octal to Decimal Binary	46-47	

Completed

28/3/24

Experiment- 1

AIM:

To write an Assembly language program to implement 8 bit addition using 8085 processor.

ALGORITHM:

1. Start the program by looking the first data into the accumulator.
2. move the data to a register.
3. Get the second data to load it into the accumulator.
4. Add the two register contents.
5. check the carry.
6. Store the value of sum & carry in the memory location.
7. Halt.

PROGRAM:

```
LDA 8500
MOV B,A
LDA 8501
ADD B
STA 8502
RST 1
```

Experiment - 2

AIM:

To write an assembly language program to implement 8-bit subtraction using 8085 processor.

ALGORITHM:

1. Start the program by loading the first data into the accumulator.
2. move the data to register.
3. Get the second data to load it into the accumulator.
4. subtract the two register contents.
5. check for borrow.
6. store the difference to borrow in memory location.

PROGRAM

```
LDA 8000
MOV B,A
LDA 8001
SUB B
STA 8002
RST 1
```

Experiment - 3

AIM:

To write an program to implement 8 bit multiplication.

ALGORITHM:

1. Start the program.
2. move data to a register.
3. Get second data & load
4. Add two register contents.
5. Increment value
6. check.
7. Store the value.
8. Halt.

PROGRAM:

```
LDA 8500
MOV B, A
LDA 8001
MOV C, A
JZ LOOP
XRA A
LOOP1: ADDE
DCR C
JZ LOOP
LOOP : STA 8002
RST 1
```

INPUT:

8500 06 H

8501 02H

OUTPUT:

8502 0CH

RESULT:

Thus the program was executed successfully using 8085 microprocessor.

Experiment - 4

AIM:

To write an assembly language to implement 8 bit division

ALGORITHM:

1. Start the program by loading a register pair.
2. move the data to a register.
3. Get the second data to load.
4. Subtract two register contents.
5. Increment value.
6. check.
7. Store the value.
8. Halt.

PROGRAM:

```
LDA 8501
MOV B,A
LDA 8500
MVI C,00
LOOP: CMP B
JC LOOP1
SUB B
INR C
JMP LOOP
STA 8503
DCR C
MOV A,C
LOOP1: STA 8502
RST 1
```

INPUT:

8500 06H

8501 02H

OUTPUT:

8502 03H (quotient)

8503 00H (remainder)

RESULT:

Thus the program was executed successfully using 8085 processor.

Experiment - 5

AIM:

To write an assembly language program to implement 16 bit addition using 8085 processor.

ALGORITHM:

1. Start the program to load the data.
2. move data to registers.
3. Load second number to register.
4. Add two register pairs.
5. Check the carry.
6. Store the value to halt program.

PROGRAM:

```
LDA 3050
MOV B,A
LDA 3051
ADD B
STA 3052
LDA 3053
MOV B,A
LDA 3054
ADC B
STA 3055
HLT
```

INPUT:

1200 13H . 120113H

1202 14H . 120314H

OUTPUT:

1300 . 27H

1301 27H

RESULT:

Execution of 16 bit addition is done successfully.

Experiment - 6

AIM:

To write an assembly language program to implement 16 bit subtraction using 8085 processor.

ALGORITHM:

1. Start the program by loading a register pair.
2. Copy the data to another pair.
3. Load the second number to first pair.
4. Subtract & check the borrow.
5. End.

PROGRAM:

```
LHLD 2050
XCHG
LHLD 2052
MVI C,00
MOV A,E
SUB L
STA 2054
MOV A,D
SUB H
STA 2055
HLT.
```

Input:

1200 08H
1202 04H

OUTPUT:

1300 04H
1301 04H

RESULT:

Execution of 16 bit subtraction is done successfully.

Experiment - 7

AIM:

To write a assembly Language program to implement 16-bit multiplication using 8085 processor.

ALGORITHM:

1. Load the HL pair. Move to stack pointer.
2. Load the second pair data in the HL pair move to DE.
3. Make the H8500H & L8500L.
4. ADD HL to SP.
5. Carry increment.
6. The move E to A perform an operation.
7. The value option is zero.

PROGRAM:

```
LHLD 2500
SRHL
LHLD 2502
XCHG
LXI H,0000H
LXI L,0000L
Again: DAD SP
JNC START
INX B
START: DCX D
MOV A,E
ORA D
HLT
```

INPUT:

8501 04H
8502 02H

OUTPUT:

8503 08H
8504 08H

RESULT:

The execution of 16-bit multiplication has done successfully.

Experiment - 8

AIM:

To write a language program of assembly for 16 bit division using 8086 program.

ALGORITHM:

1. Read division & dividend, count < 8.
2. Left shift dividend & subtract the upper.
3. If CS = 1 go to 9.
4. Restore dividend & increment & lower.
5. Count < count - 1
6. Store upper 8 bit as dividend as remainder & lower 8 bit.

PROGRAM:

```
LDA 8501
MOV B,A
LDA 8500
MVI C,00
LOOP: CMPB
JC LOOP1
SUB B
INR C
JMP LOOP
STA 8503
DCR C
mov A,C
LOOP1: STA 8502
RST1.
```

INPUT:

AX 0006H
CX 0004H

OUTPUT:

1300 01H
1301 00H
1302 02H

RESULT:

The execution of 16 bit division has done successfully.

Experiment - 9

AIM:

To complete various logical operations using 8085 microprocessor.

ALGORITHM:

1. Load the data to accumulator, load data to register.
2. Perform logical operation like AND, OR & XOR Gates.
3. Store the result in specified memory location.

AND operation

```
MVI A,06
MVI B,04
ANA B
STA 2500
HLT
```

OR Operation

```
MVI A,01
MVI B,06
STA 800
HLT
```

XOR Operation

```
MVI A,03
MVI B,04
XRA B
STA 2000
HLT
```

OUTPUTS:

AND

Address hora	Address	data
D9C4	2500	4

OR

AH	A	D
0700	2000	7

XOR

AH	A	D
0700	2000	7

RESULT:

The program has been successfully has done.

Experiment - 10

AIM:

To design & implement the two bit half adder using Logsim simulator.

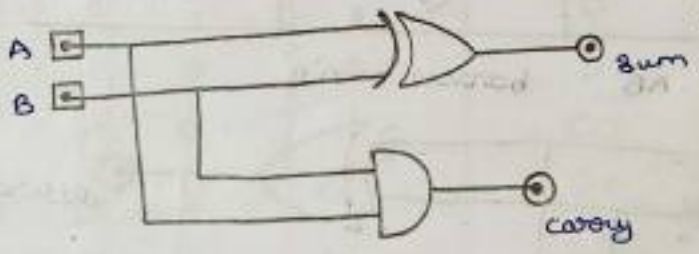
ALGORITHM:

1. pick & place the necessary gates.
2. Insert 2 inputs into Canva.
3. Connect the inputs to XOR gates & AND gates.
4. Insert 2 outputs into canvas.
5. make connections using the connection wires.
6. verify the truth table.

Truth Table:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

OUTPUTS:



RESULT:

The 2-bit half adder has been successfully designed & implemented successfully using Logsim simulator.

Experiment - 11

AIM:

To design & implement the two bit half subtractor using algorithm simulator.

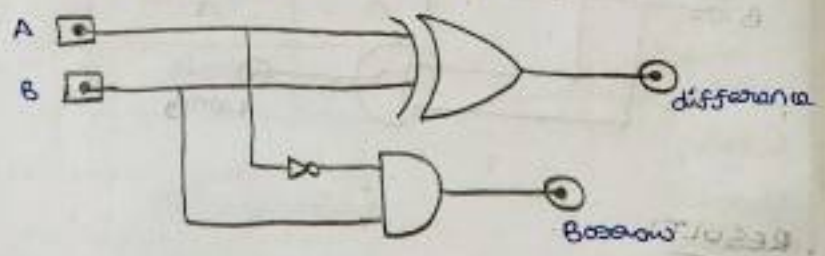
ALGORITHM:

1. Pick & place the necessary gates.
2. Insert 2 inputs in canvas.
3. Connect the inputs to OR gate, AND gates & NOT gate.
4. Insert the outputs into canvas.
5. make the connections using the connecting wires.
6. verify the truth table.

Truth Table

Inputs		outputs	
A	B	difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

diff = $A'B + AB'$ borrow = $A'B$



RESULT:

The 2 bit half subtractor has been done designed & implemented has done successfully using logics simulator.

Experiment - 12

AIM:

To design and implement two full adder using Logism simulator.

ALGORITHM:

1. Pick & place two necessary gates.
2. Insert input.
3. Connect the input to the XOR gates, AND gates & OR gate.
4. Insert two outputs into the convas.
5. make the connections using the connecting wires.
6. verify the truth table.

Truth Table:

Inputs			Outputs	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum = $(A \oplus B) \oplus Cin$ Carry = $A \cdot B + (A \oplus B)$

RESULT:

The Full adder has been designed successfully.

Experiment - 13

AIM:

To design & implement the Full Subtractor using logism simulator.

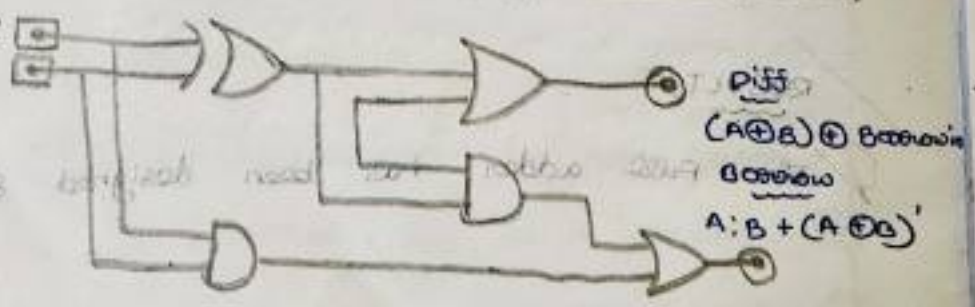
PROCEDURE:

1. pick up & place two necessary gates.
2. insert 3 input into two canvas.
3. Connect two input to two XOR gates, AND & OR gates.
4. insert 2 outputs into the canvas.
5. make the connections using the connecting wires.
6. verify the truth.

Truth Table

Inputs			Outputs	
A	B	Borrow	Diff.	Borrow
0	0	0	0	0
0	0	1	1	1
0	0	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	0	0	0	0
1	1	1	1	1

OUTPUT:



RESULT:

Three full subtractor been designed & implemented successfully.

Experiment - 14

AIM:

To write a C program to implement cpu performance measures.

ALGORITHM:

1. Start.
2. Declare the necessary variables.
3. Initialize the cpu array elements to.
4. prompt the user to enter the no. of processes.
5. Store the value of 10 in p.
6. Start a loop from 0 to p-1.
7. Start max as the first element of two cpu.
8. Display the processes with the largest execution first.
9. Exit the program.

PROGRAM:

```
#include <stdio.h>
int main()
{
 float ch;
 int p, p1;
 Float cpu (5);
 Float cpu, ct, max;
 int n= 1000;
 for (i=10; i<=n; i++)
 {
   cpu (5) = 0;
 }
   print ("\n Enter the no. of processes : ");
   Scanf ("%d", &p);
   P1 = p;
   For (i=0; i<p; i++)
   {
```

```
printf ("\n Enter the cycles for instruction of
        processor: ");
scanf ("%f", &cr);
ct = 1000 * cp/cr;
printf (" the CPU time is "%f", ct);
cpu [i] = ct;
}
max = cpu [0];
for (i=0; i<p; i++)
{
 if (cpu (i) <= max)
 max = cpu (i);
}
print ("\n the processor has lowest execution time is
        %f");
return 0;
}
```

OUTPUT:

Enter the no. of processor : 2

Enter the cycle for instruction of processor : 1.5

Enter the clock rate in GHz : 3

CPU time : 500'000 00

Enter the cycles per instruction of processor 2.2

Enter the clock rate in GHz : 4 CPU time is

650,000000

RESULT:

Thus the program was executed successfully using C++,

Experiment - 15

AIM:

To write a program to implement to integer addition.

PROGRAM:

```
# include <stdio.h>
Int main()
{
  Int num1, num2, sum = 0;
printf("Enter two numbers num1 & num 2: \n");
scanf ("%d %d", &num1, num2);
 sum = num1 + num2;
Printf ("Sum of num1 & num2 is; %d" sum);
return 0;
}
```

Input:

Enter the num1 : 5

Enter the num2 : 6

OUTPUT:

Sum of num1 & num2 = 1

RESULT:

Thus program was executed successfully using Dev C++.

Experiment - 16

AIM:

To implement C program of two integer subtraction.

PROGRAM:

```
#include <stdio.h>
int main()
{
  int num1, num2, difference = 0
  printf ("Enter num1 & num 2 : \n");
  Scanf ("%d%d", &num1 or num2)
  different = num1 - num2;
  printf (" different of num1 & num2 is %d", differnce);
  return 0;
}
```

Input:

Enter the num1 : 4

Enter the num2 : 2

Output:

different of two number is 2

RESULT:

Thus the program was executed successfully using per C++.

Experiment - 17

AIM:

To write & implement C program of two integer multiplication.

PROGRAM:

```
# include <stdio.h>
int main()
{ int num1, num2, product;
printf ("Enter the num1 value & num2 value ");
scanf ("%d %d", num1, num2);
product = num1 * num2
printf ("The product of %d & %d num1, num2,
product");
return 0;
}
```

Input:

Enter the num1 value : 4

Enter the num2 value : 6

Output:

The product of num1 & num2 value 4, 6, 24.

RESULT:

Thus the program was executed successfully using Ter C++.

Experiment - 18

AIM:

To implement a C program for Integer Arithmetic Division.

PROGRAM:

```
# include <stdio.h>
int main() {
 int dividend = 10;
 int divisor = 2;
 int quotient;
 quotient = dividend / divisor;
 printf ("The quotient of %d divided by %d is %d \n",
          dividend, divisor, quotient);
 return 0;
}
```

ALGORITHM:

1. Initialize quotient to 0.
2. Repeat until dividend is greater than or equal to divisor.
   (a) Subtract divisor from dividend
   (b) increment quotient by 1.
3. Return the quotient & the remainder.

Output:

num1: 6
num2: 3
Div: 2

RESULT:

To write a program for integer Arithmetic Division was written & executed successfully.

Experiment - 19

AIM:

To write a c program to implement single precision format.

PROGRAM:

```
# Include <stdio.h>
void print binary (int n , int i)
{
 int k;
 for (k = i-1 ; k >= 0; k --)
 {
  if (n>>k & 1)
  printf ("1");
  else
    printf ("0");
  }
}

type def union {
   float f;
   struct
   {
     unsigned int mantissa = 23;
     unsigned int exponent = 8;
     unsigned int sign = 1;
     }
   }
   my Float;
   void print IEEE (myFloat var)
   {
     printf ("%d", var.raw, sign);
     printf binary (var . raw . exponent, 8);
     Print ("\n");
     print binary (var, raw, exponent, 8);
     print ("\n");
```

```
print binary (var, row, mantissa, 23);
printf ("\n");
}
int main ()
{
my float var;
var f = 1259.125;
printf ("IEEE 754 representation of %f is \n", var);
printf (EEE(var) ;
return 0;
}
```

Output:

IEEE 754 representation of 1259.125000 is :

0/1000/001/00110101100100000 0000

RESULT:

Thus, the program was executed successfully using Dev C++.

Experiment - 20

AIM:

To write a c program to implament Double precision format.

PROGRAM:

```
# include <stdio.h>
# include <stdint.h>
void print_double_binary (double num) {
unit 64_t * ptr = (unint 64_t*) & num;
unit 64_t mask = NULL << 63.
printf ("Binary representation of %.15lf", num);
for (int i = 0; i< 64 ; i++){
printf ("%d", (* ptr & mask) != 0);
if (i == 0 || i == 11)
printf (" ");
mask >>= 1;
}
printf ("\n");
}
int main (){
double num = 3.14159 2653 5897 3238;
printf (double_binary (num));
return 0;
}
```

output: Double num = 3.141592653589732 38

Binary Representation.

3.141592653589793 : 0100000000001001 0010000111111010101000100010000101101000 11000.

RESULT:

Thus, the c program was Executed Successfully using Dev c++ in double precision.

Experiment - 21

AIM:

To implement a C program for floating point addition.

PROGRAM:

```
# include <stdio.h>
int main() {
float num1 = 2.5;
float num2 = 3.7;
float sum;
sum = num1 + num2;
printf("The sum of %f & %f is %f \n", num1,
                num2, sum);
return 0;
}
```

Input:

2.5 + 3.7

Output:

6.2

RESULT:

Thus, the C program was executed successfully using Dev C++ is addition using Float point programing.

Experiment - 22

AIM:

To implement a c program for floating point subtraction.

PROGRAM:

```
#include <stdio.h>
int main () {
float num1 = 5.8;
float num2 = 2.3;
Float difference;
difference = num1 - num2;
printf ("The difference of between %f and %f is %f
        \n", num1, num2, difference);
return 0;
}
```

Input:

5.8 - 2.3

Output: 3.5

RESULT:

Thus, the C program was executed successfully using Dev C++, subtraction using float point programing.

Experiment - 23

AIM:

To implement C program for floating point multiplication

PROGRAM:

```
# include <stdio.h>
int main() {
float num1 = 2.5;
float num 2 = 3.2;
float product;
Product = num1 * num 2;
printf ("the product of %f & %f is %f \n", num1,
        num2, product);
return 0;
}
```

Input:

6.3 * 5.5

Output:

34.65

RESULT:

Thus, the C program was executed successfully using Dev C++ is multiplication using float point programming.

Experiment - 24

AIM:

To implement C program for floating point Division.

PROGRAM:

```
# include <stdio.h>
int main() {
  float dividend = 10.0;
  float divisor = 3.0;
  float quotient;
quotient = dividend / divisor;
printf ("The quotient of %f dividend by %f is %f \n",
          dividend, divisor, quotient);
  return 0;
}
```

Input:

5.5 ÷ 6.3

Output:

0.873001

RESULT:

Thus, the C program was executed successfully using Dev C++, Division using float point programming.

Experiment - 25

AIM:

To write & implement C programming restoring division.

ALGORITHM:

1. At each step, left shift the dividend by 1 position.
2. Subtract the divisor from A (A-M).
3. If the result is positive, then the step is said to be successful in this case, the Quotient bit will be "1" & the restoration is not required.
4. If the result is negative, then the step is said to be successful. In this case quotient bit will be "0" & restoration is required.
5. Repeat the above steps for all the bits of the dividend.

PROGRAM:

```
# include <stdio.h>
void restore_division (int dividend, int divisor, int* quotient)
{
* quotient = 0;
* remainder = 0;
For (int i=31 ; i>=0; i--){
    * remainder = (*remainder <<1) | (dividend >> i) & 1);
    if (* remainder >= divisor ) {
        * remainder -= divisor;
        * quotient |= (1<<i);
```

```
    }
  }
}
int main () {
    int dividend = 20;
    int divisor = 4;
    int quotient, remainder ;
    restore_division ( dividend, divisor, &quotient, &
                                        remainder);
    printf ("Quotient : %d \n", quotient);
    printf ("Remainder : %d \n", remainder);
    return 0;
}
```

Input:

Dividend = 20;

Divisor = 4;

Output:

Quotient = 5

remainder = 0.

RESULT:

Thus the program of restoring division is executed successfully using Dev c++.

Experiment - 26

AIM:

To write to implement C programming of Non-Restoring divisor.

ALGORITHM:

1. At each step, Left shift the dividend by 1 position.
2. Subtract the divisor from A (A0 - M)
3. If result is positive then its successful.
4. If result is negative then its unsuccessful
5. Repeat until the 1 to 4, for all divider.

PROGRAM:

```
# include <stdio.h>
# include <string.h>
void non-restoring_divison (int divisor,
                dividend, Quotient) {
int partial_remainder [N+1]
int borrow = 0;
memset (partial_remainder [i] = partial
                remainder [i-1] ;
partial_remainder [0] = dividend [i];
if (partial remainder [0] = dividend [i]
partial [i] += 2;
partial [i+1] = 1;
}
}
else {
for (int j = 0; j<N+1; j++) {
partial [i] += divisor [i];
if (partial_remainder [i] >= 2){
  Partial_remainder [i] = 2;
  partial [i+1] += 1
int main () { int div [N] =
        {1,1,0,1,0,1,0,1}
int divisor = {1,0,1,1,0,0,1,0}
for (int i=0 ; i<N ; i++) {
printf ("%d", quotient (i]); }
printf ("%N"); }
return 0; }
```

INPUT:

Enter a number = 15

Output:

Quotient = 00010

remainder = 00001

RESULT:

Thus the implementation of Non-restoring executed successfully.

Experiment - 27

Aim:

To write & implement C-program of booth Algorithm

PROGRAM:

```
# include <stdio.h>
void booth (int mult, int multiplier,
        int * result) {
* result = 0;
int mult = bit = 0;
int sign_bit = mult * 0x8000000;
while (multiplier != 0) {
int LS bit = multiple x0x1;
if (LS_bit != multiple) {
if (LS_bit == 1) {
* result = + multiplicand;
}
}
multiplicand = = 1;
int msb = multiplicand x0x80000;
multiple_bits = LS bit;
if (sign_bits != 0) {
* result = = * result;
}
}
int main() {
int multiplicand, multiple, product;
printf ("Enter the multiplicand");
scanf ("%d", &multiplicand);
printf ("Enter multiplier: ");
scanf ("%d", &multiplier);
booth (multiplicand, multiplier,
        & product);
printf ("product: %d", product);
return 0;
}
```

Input:

5 3

output:

15

RESULT:

Thus the implementation of Booth Algorithm in C-program is successfully done.

AIM:

To write & implement single bus organisation, in c-programming.

PROGRAM:

```
#include <stdio.h>
typedef struct {
int data;
int address;
} Bus;
typedef struct {
 Bus * bus;
} cpu;
typedef struct {
Bus * Bus
int data [100];
} memory;
int memory_a (memory*mem,
          int address)
return mem -> data [address];
}
void memory_w (memory mem, int
    add, int data) {
mem -> data [address] = data;
}
int main () {
Bus system_bus;
cpu cpu;
memory memory;
cpu.bus = system_bus;
memory.bus = & system bus;
memory_w (&memory, 0, 19);
int data_read = cpu_operation
          (& cpu, 0);
printf ("Data read by cpu:
        %d: data read);
return 0;
}
```

OUTPUT:

250: Data read by cpu.

RESULT:

Thus the implementation & Execution of single bus organisation is executed successfully.

Experiment - 29

AIM:

To write & implement the C programming for multiple Bus organisation.

PROGRAM:

```
#include <stdio.h>
int main ()
Bus data_bus;
Bus io_bus;
cpu CPU;
memory memory;
IO Devices io_device;
cpu.bus = & data_bus;
memory.bus= & data_bus;
io_bus= & io_bus;
memory_write (& memory, 0, 10);
int data_read = cpu_operation (& cpu, & memory, 0);
printf ("Data read by CPU : %d\n", data_read);
io_device_operation (& io_device, & memory, 1, 20);
data_read = cpu_operation (& cpu, & memory, 1);
return 0;
}
```

RESULT:

Thus the C program for multiple Bus organisation is implemented successfully using Dev C++.

Experiment - 30

AIM:

To write & implement the program of Two stage pipelining.

PROGRAM:

```
# include <stdio.h>
int main () {
int instruction_count = 0;
instruction current_instruction;
int execution_result;
for (int i=0 ; i<5 ; i++) {
Fetch_stage (& instruction_count & current_instruction);
Execute_stage (& current_instruction, & execution_result);
Printf ("cycle %d : Result = %d \n", i+1, execution_result);
}
return 0;
}
```

RESULT:

Thus the program of two stage pipelining was implemented successfully using dev c++.

Experiment - 31

AIM:

To write and implement C program of four stage pipelining.

PROGRAM:

```
#include <stdio.h>
int main () {
int instruction_count = 0;
instruction current_instruction;
pipeline register decode_reg, execute_reg;
for (int i=0; i<5 ; i++) {
fetch_stage (&instruction_count, &current_instruction);
decode_stage (&current_instruction, &decode_reg);
execute_stage (&decode_reg, &execute_reg);
writeback_stage (&execute_reg);
printf ("cycle %d; instruction opcode, %d,
operand 1 = %d, operand 2 = %d \n;
i+1, current_instruction.opcode, current_instruction
operand1, current_instruction operand 2);
}
return 0;
}
```

RESULT:

Thus the program has been executed successfully using Dev C++.

AIM:

To write & implement c program of static prediction.

PROGRAM:

```
#include <stdio.h>
# define TAKEN1
int static prediction (int instruction_address) {
if (instruction_address % 2 == 0){
return TAKEN ;
} else {
return NOT_TAKEN;
}
}
int main () {
    int main instruction address[] = {100,101,102,103,104};
int num_instructions = size of (instruction_address / size of (instruction
                                    +address)[0]);
printf (static prediction results : \n);
for (int i=0; i< num instruction;; i++){
int prediction= static_prediction (instruction_address[i]);
prediction == TAKEN ?"TAKEN":"NOTTAKEN");
}
return 0;
}
```

RESULT: Thus the program of static prediction was implemented successfully using Dev C++.

Experiment - 33

AIM:

To write & implement a program of Dynamic prediction.

PROGRAM:

```
#include <stdio.h>
# define T1
# define NTO
# define STS
# define SNTO
type def struct {
      int state;
} BP;
void int (BP* P){
      P -> state = SNT;
  }
int predict (BP* P){
    return P -> state >= 2 ? T : NT;
}
void update (BP* P, int act_out){
    if (act_out == T){
      if (P -> state < ST) P -> state ++;
    } else {
      if (P -> state > SNT) P -> state;
    }
  }
int main (){
BP P;
int (yP);
Printf ("Branch %d : prediction = %s\n", i+1, pred == T ? "Taken"
                                                : "NOT TAKEN");
  }
return 0;
}
```

RESULT: Thus the program of static prediction was implemented successfully using Dev C++.

AIM:

To write & implement C program for Data Hazards.

PROGRAM:

```
#include <stdio.h>
int main() {
int a = 5 ;
int b = 10;
int c ;
the result is "c".
    c= a+b;
the result is "c"
    c= c*2;
printf ("Result! %d \n" c);
return 0;
}
```

RESULT:

Thus the program of Data Hazards is implemented successfully using Dev C++.

Experiment - 35

AIM:

To write and implement a program for instruction Hazards.

PROGRAM:

```
# include <stdio.h>
int main () {
int a = 5;
int b = 10;
int c;
int temp_a = a;
int temp_b = b;
c = temp_a + tem_b;
c = c * 2;
printf ("Result : %d \n" c);
return 0;
}
```

RESULT:

Thus the program of instruction Hazards is implemented successfully using Dev C++.

Aim:

To write and implement C program for structure Hazards

PROGRAM:

```
# include <stdio.h>
int main() {
int a = 5;
int b = 10;
int c;
int flag = (a>b);
if (flag) {
   c = (a+b);
} else {
   c = b;
}
c = c+2;
printf ("Result: %d\n", c);
return 0;
}
```

RESULT:

Thus the program of structure Hazards is implemented successfully using Dev C++.

Experiment - 07

Aim:

To write an implement a program for Super SCALAR processing.

PROGRAM:

```
# include <stdio.h>
int add (int a, int b){
return a+b;
}
int subtract (int a, int b){
return a-b;
}
int main (){
  int a= 5;
  int b= 10;
  int c,d;
  c= add (a,b);
  d = subtract (a,b)
printf ("Result of addition: %d\n", c);
printf ("Result of subtraction: %d\n", d);
return 0;
}
```

RESULT:

Thus the program of Super Scaler processing is implemented successfully using Dev C++.

Experiment - 38

AIM:

To write and implement C program for Register Transferring.

PROGRAM:

```
#include <stdio.h>
int main () {
 int a = 6;
 int b;
 b=a;
 printf ("value of b after register transfer : %d \n", b);
return 0;
}
```

RESULT:

Thus the program of Register transferring is implemented & successfully using Dev C++.

Experiment- 39

AIM:

To write a c program to implement decimal to hexadecimal conversion.

PROGRAM:

```
# include <stdio.h>
int main () {
  int n;
  printf ("Enter the decimal number");
  scanf ("%d \n", &n);
  printf ("The hexadecimal value is %d \n");
  return 0;
}
```

Input:

Enter the decimal number : $(894)_{10}$

Output:

The hexadecimal of value is $37E_{16}$

RESULT:

Thus the program was executed successfully using Dev C++.

Experiment - 40

AIM:

To write and implement C program converting octal to decimal.

PROGRAM:

```
#includ <stdio.h>
int main()
{
 char octal num [100];
 int i=0;
 printf ("Enter any octal numbers:");
 scanf ("%s", octal num);
 printf ("Equivalent binary values.");
 while (octal num[i])
 {
   switch (octal num[i])
   {
    case '0'
    printf ("000"); break
    case '1';
    printf ("001"); break
    case '2';
    printf ("010"); break
    case '3';
    printf ("011"); break
    case '4';
    printf ("100"); break;
    case '5';
    printf ("101"); break
```

```
    case '6';
    printf ("invalid octal digit");
    return 0;
  }
  i++;
  }
  return 0;
}
```

Input:

Enter any octal number $(2671)_8$

Output:

$(2671)_8 = (1465)_{10}$

RESULT:

Thus the program written is executed successfully.

Completed
28/3/24