**GNUSIM - 8085**

1. **8-BIT ADDITION**

MVI C,00

LDA 100

MOV B,A

LDA 101

ADD B

JNC skip

INR C ; c=C+1

skip: STA 102

MOV A,C

STA 103

HLT

1. **8-BIT SUBTRACTION**

MVI C,00

LDA 100

MOV B,A

LDA 101

SUB B

JNC skip

INR C

skip: STA 102

MOV A,C

STA 103

HLT

1. **8-BIT MULTIPLICATION**

LXI H,100

MOV B,M

MVI A,00

MOV C,A

INX H

CONT: ADD M

JNC SKIP

INR C

SKIP: DCR B

JNZ CONT

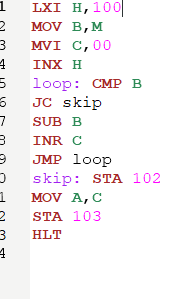
STA 102

MOV A,C

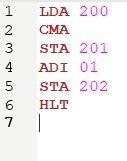
STA 103

HLT

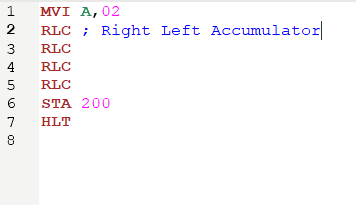
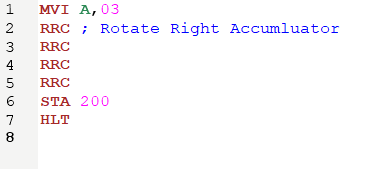
1. **8-BIT DIVISION**

****

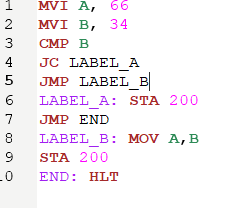
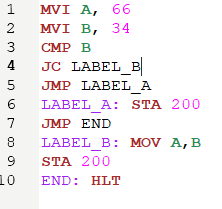
1. **1’S AND 2’S COMPLEMENT**

****

1. **ROTATE RIGHT AND ROTATE LEFT**

****

1. **FINDING LARGEST AND SMALLEST NUMBER**

****

**8086 EMULATOR**

1. **16-BIT ADDITION**

MOV CX, 000H

MOV AX, [1200H]

MOV BX, [1202H]

ADD AX, BX

JNC L1

INC CX

L1 MOV [1206H], CX

MOV [1204H], AX

HLT

1. **16-BIT SUBTRACTION**

MOV CX, 000H

MOV AX, [1200H]

MOV BX, [1202H]

SUB AX, BX

JNC SKIP

INC CX

NEG AX

SKIP MOV [1206H], CX

MOV [1204H], AX

HLT

1. **16-BIT MULTIPLICATION**

MOV DX, 000H

MOV AX, [1200H]

MOV BX, [1202H]

MUL BX

MOV [1206H], CX

MOV [1204H], DX

1. **16-BIT DIVISION**

MOV DX, 000H

MOV AX, [1200H]

MOV BX, [1202H]

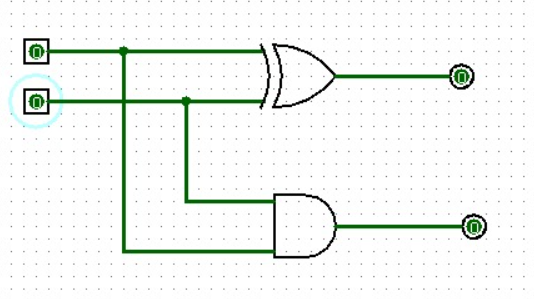
DIV BX

MOV [1206H], CX

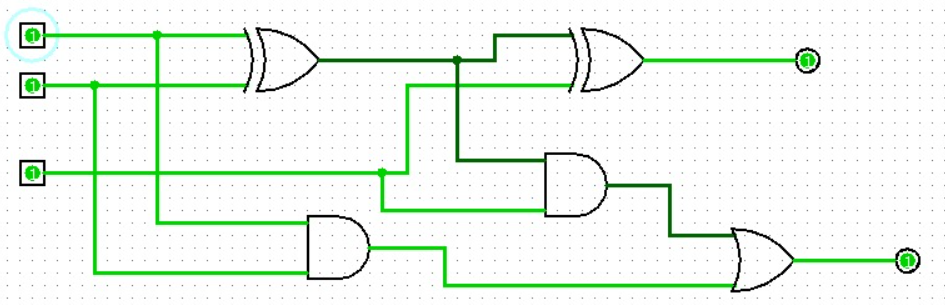
MOV [1204H], DX

**LOGISIM**

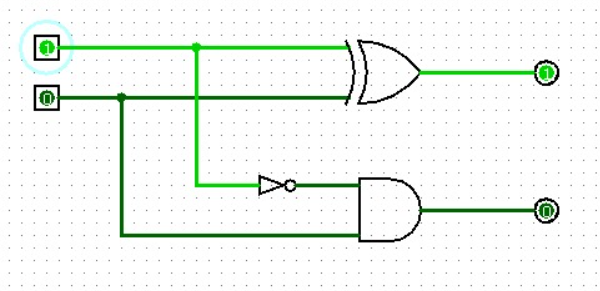
1. **HALF ADDER**



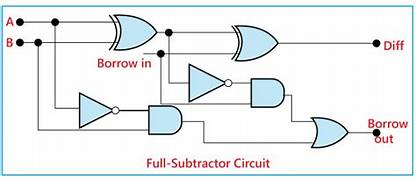
1. **FULL ADDER**



1. **HALF SUBTRACTOR**



1. **FULL SUBTRACTOR**



**C PROGRAM**

1. **2 STAGE PIPELINING**

#include <stdio.h>

int main() {

int a, b, choice, res;

int fetchCycles = 1;

int executeCycles = 1;

// Fetch Stage

printf("Enter number 1:");

scanf("%d", &a);

printf("Enter number 2:");

scanf("%d", &b);

// Execution Stage

printf("1-Addition:\n2-Subtraction:\n3-Multiplication:\n4-Division:");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Performing addition\n");

res = a + b;

break;

case 2:

printf("Performing subtraction\n");

res = a - b;

break;

case 3:

printf("Performing Multiplication\n");

res = a \* b;

break;

case 4:

printf("Performing Division\n");

res = a / b;

break;

default:

printf("Wrong input");

return 1;

}

printf("Result: %d\n", res);

printf("Total Cycles: %d\n", fetchCycles + executeCycles);

return 0;

}

1. **4 STAGE PIPELINING**

#include <stdio.h>

int main() {

int a, b, choice, res;

int fetchCycles = 1;

int decodeCycles = 1;

int executeCycles = 1;

int writeBackCycles = 1;

// Fetch Stage

printf("Enter number 1:");

scanf("%d", &a);

printf("Enter number 2:");

scanf("%d", &b);

// Decode Stage

printf("1-Addition:\n2-Subtraction:\n3-Multiplication:\n4-Division:");

scanf("%d", &choice);

// Execution Stage

switch (choice) {

case 1:

printf("Performing addition\n");

res = a + b;

break;

case 2:

printf("Performing subtraction\n");

res = a - b;

break;

case 3:

printf("Performing Multiplication\n");

res = a \* b;

break;

case 4:

printf("Performing Division\n");

res = a / b;

break;

default:

printf("Wrong input");

return 1;

}

// Write Back Stage

printf("Result: %d\n", res);

printf("Total Cycles: %d\n", fetchCycles + decodeCycles + executeCycles + writeBackCycles);

return 0;

}

1. **CPU PERFORMANCE**

#include <stdio.h>

int main()

{

float cr;

int p,p1,i;

float cpu[5];

float cpi,ct,max;

int n=1000;

for(i=0;i<=4;i++)

{

cpu[5]=0;

}

printf("\n Enter the number of processors:");

scanf("%d",&p);

p1=p;

for(i=0;i<p;i++)

{

printf("\n Enter the Cycles per Instrcution of processor:");

scanf("%f",&cpi);

printf("\n Enter the clockrate in GHz:");

scanf("%f",&cr);

ct=1000\*cpi/cr;

printf("The CPU time is: %f",ct);

cpu[i]=ct;

}

max=cpu[0];

for(i=0;i<p1;i++)

{

if(cpu[i]<=max)

max=cpu[i];

}

printf("\n The processor has lowest Execution time is: %f ", max);

return 0;

}

1. **FLOATING POINT ADDITION**

#include <stdio.h>

int main() {

// Declare variables

float num1, num2, sum;

// Get input from the user

printf("Enter the first number: ");

scanf("%f", &num1);

printf("Enter the second number: ");

scanf("%f", &num2);

// Perform addition

sum = num1 + num2;

// Display the result

printf("The sum of %f and %f is %f\n", num1, num2, sum);

return 0;

}

1. **FLOATING POINT SUBTRACTION**

#include <stdio.h>

int main() {

// Declare variables

float num1, num2, sub;

// Get input from the user

printf("Enter the first number: ");

scanf("%f", &num1);

printf("Enter the second number: ");

scanf("%f", &num2);

// Perform addition

sub = num1 - num2;

// Display the result

printf("The sub of %f and %f is %f\n", num1, num2, sub);

return 0;

}

1. **FLOATING POINT MULTIPLICATION**

#include <stdio.h>

int main() {

// Declare variables

float num1, num2, mul;

// Get input from the user

printf("Enter the first number: ");

scanf("%f", &num1);

printf("Enter the second number: ");

scanf("%f", &num2);

// Perform addition

mul = num1 \* num2;

// Display the result

printf("The mul of %f and %f is %f\n", num1, num2, mul);

return 0;

}

1. **FLOATING POINT DIVISION**

#include <stdio.h>

int main() {

// Declare variables

float num1, num2, div;

// Get input from the user

printf("Enter the first number: ");

scanf("%f", &num1);

printf("Enter the second number: ");

scanf("%f", &num2);

div = num1 / num2;

// Display the result

printf("The div of %f and %f is %f\n", num1, num2, div);

return 0;

}

1. **INTEGER ARI THEMATIC**

#include <stdio.h>

int main() {

int num1, num2;

// Input two integers

printf("Enter first integer: ");

scanf("%d", &num1);

printf("Enter second integer: ");

scanf("%d", &num2);

// Perform arithmetic operations

int sum = num1 + num2;

int difference = num1 - num2;

int product = num1 \* num2;

// Check if the second number is not zero before performing division

int quotient;

if (num2 != 0) {

quotient = num1 / num2;

} else {

printf("Cannot divide by zero.\n");

return 1; // Exit with an error code

}

// Print results

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

printf("Difference: %d\n", difference);

printf("Product: %d\n", product);

printf("Quotient: %d\n", quotient);

return 0; // Exit successfully

}

1. **SINGLE PRECISION REPRESENTATION**

#include <stdio.h>

int main() {

float num1 = 3.14159f; // Declare a single-precision variable

float num2 = 2.71828f;

float result;

result = num1 \* num2; // Perform a calculation

printf("Single-precision representation of num1: %.7f\n", num1);

printf("Single-precision representation of num2: %.7f\n", num2);

printf("Result of multiplication (single-precision): %.7f\n", result);

return 0;

}

1. **DOUBLE PRECISION REPRESENTATION**

#include <stdio.h>

int main() {

double num1 = 3.14159265358979323846; // Declare double-precision variables

double num2 = 2.71828182845904523536;

double result;

result = num1 \* num2; // Perform a calculation

printf("Double-precision representation of num1: %.16f\n", num1);

printf("Double-precision representation of num2: %.16f\n", num2);

printf("Result of multiplication (double-precision): %.16f\n", result);

return 0;

}

1. **READ ONLY MEMORY**

#include <stdio.h>

// Define a mock ROM data

const char romData[] = "Hello, this is data from ROM!";

int main() {

// Access and print data from ROM

printf("Data from ROM: %s\n", romData);

return 0;

}

1. **RANDOM ACCESS MEMORY**

#include <stdio.h>

#include <stdlib.h>

int main() {

// Allocate memory for an integer

int \*ptr = (int \*)malloc(sizeof(int));

if (ptr == NULL) {

printf("Memory allocation failed\n");

return 1;

}

// Write to and read from the allocated memory

\*ptr = 42;

printf("The value at the allocated memory location is: %d\n", \*ptr);

// Free the allocated memory

free(ptr);

printf("The value at the allocated memory location is: %d\n", \*ptr);

return 0;

}

1. **RESTORING DIVISION**

#include <stdio.h>

void RestoringDivision(int dividend, int divisor) {

int quotient = 0;

int remainder = 0;

for (int i = 0; i < 32; i++) { // Assuming 32-bit integers

remainder = (remainder << 1) | ((dividend >> 31) & 1);

dividend = dividend << 1;

remainder -= divisor;

if (remainder >= 0) {

quotient = (quotient << 1) | 1;

} else {

quotient = quotient << 1;

remainder += divisor;

}

}

printf("Quotient: %d\n", quotient);

printf("Remainder: %d\n", remainder);

}

int main() {

int dividend, divisor;

printf("Enter the dividend: ");

scanf("%d", &dividend);

printf("Enter the divisor: ");

scanf("%d", &divisor);

RestoringDivision(dividend, divisor);

return 0;

}

1. **STATIC PREDICTION**

#include <stdio.h>

// Simple linear regression formula: y = mx + c

float predict(float x, float m, float c) {

return m \* x + c;

}

int main() {

float m, c, x, predicted\_y;

// Let's assume some values for m (slope) and c (y-intercept)

m = 2.0;

c = 3.0;

// Get the value of x for which we want to predict y

printf("Enter the value of x: ");

scanf("%f", &x);

// Predict y based on the given x using linear regression formula

predicted\_y = predict(x, m, c);

printf("Predicted y = %.2f\n", predicted\_y);

return 0;

}

1. **CONVERT BINARY TO OCTAL, DECIMAL, HEXADECIMAL**

#include <stdio.h>

void main()

{

int num, binary\_num, decimal\_num = 0, base = 1, rem;

printf (" Enter a binary number\n");

scanf (" %d", &num);

binary\_num = num;

while ( num > 0)

{

rem = num % 10;

decimal\_num = decimal\_num + rem \* base;

num = num / 10;

base = base \* 2;

}

printf ( " The binary number is %d \t", binary\_num);

printf (" \n The decimal number is %d \t", decimal\_num);

}

#include <stdio.h>

void main()

{

long num, binary\_num, decimal\_num = 0, base = 1, rem;

printf (" Enter a binary number with the combination of 0s and 1s \n");

scanf (" %ld", &num);

binary\_num = num;

while ( num > 0)

{

rem = num % 10;

decimal\_num = decimal\_num + rem \* base;

num = num / 10;

base = base \* 2;

}

printf ( " The binary number is %ld \t", binary\_num);

printf (" \n The decimal number is %ld \t", decimal\_num);

int n=decimal\_num;

printf (" \n The decimal number is %x \t", n);

}

#include <stdio.h>

void main()

{

long num, binary\_num, decimal\_num = 0, base = 1, rem;

printf (" Enter a binary number with the combination of 0s and 1s \n");

scanf (" %ld", &num);

binary\_num = num;

while ( num > 0)

{

rem = num % 10;

decimal\_num = decimal\_num + rem \* base;

num = num / 10;

base = base \* 2;

}

printf ( " The binary number is %ld \t", binary\_num);

printf (" \n The decimal number is %ld \t", decimal\_num);

int n=decimal\_num;

printf (" \n The octal number is %o \t", n);

}

#include <stdio.h>

int main() {

long num, binary\_num, decimal\_num = 0, base = 1, rem;

printf("Enter a binary number with the combination of 0s and 1s:\n");

scanf("%ld", &num);

binary\_num = num;

while (num > 0) {

rem = num % 10;

decimal\_num = decimal\_num + rem \* base;

num = num / 10;

base = base \* 2;

}

printf("The hexadecimal number is %X\n", decimal\_num);

return 0;

}

1. **CONVERT OCTAL TO DECIMAL, HEXADECIMAL, BINARY**

#include <stdio.h>

int main()

{

char octalnum[100];

long i = 0;

printf("Enter any octal number: ");

scanf("%s", octalnum);

printf("Equivalent binary value: ");

while (octalnum[i])

{

switch (octalnum[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("110"); break;

case '7':

printf("111"); break;

default:

printf("\n Invalid octal digit ");

return 0;

}

i++;

}

return 0;

}

#include <stdio.h>

int main() {

int octal,hexa;

printf("Enter the octal value : ");

scanf("%o",&octal);

printf("The hexadecial of given octal number is : %x",octal);

}

#include <stdio.h>

int main() {

int octal;

printf("Enter the octal value : ");

scanf("%o",&octal);

printf("The decimal of given number is : %d",octal);

1. **CONVERT DECIMAL TO BINARY, OCTAL, HEXADECIMAL**

#include<stdio.h>

int main()

{

int n;

printf("enter the decimal number");

scanf("%d",&n);

printf("the hexa decimal value is:%x",n);

return 0;

}

#include<stdio.h>

int main()

{

int a[10],n,i;

printf("Enter the number to convert: ");

scanf("%d",&n);

for(i=0;n>0;i++)

{

a[i]=n%8;

n=n/8;

}

printf("\nOctal of Given Number is=");

for(i=i-1;i>=0;i--)

{

printf("%d",a[i]);

}

return 0;

}

#include<stdio.h>

int main()

{

int a[10],n,i;

printf("Enter the number to convert: ");

scanf("%d",&n);

for(i=0;n>0;i++)

{

a[i]=n%2;

n=n/2;

}

printf("\nBinary of Given Number is=");

for(i=i-1;i>=0;i--)

{

printf("%d",a[i]);

}

return 0;

}

1. **CONVERT HEXADECIMAL TO BINARY, OCTAL, DECIMAL**

#include<stdio.h>

int main()

{

int n;

printf("enter the hex decimal number");

scanf("%x",&n);

printf("the decimal value is:%d",n);

return 0;

}

#include <stdio.h>

int main() {

int n, a[10], m, i;

printf("Enter the hexadecimal number: ");

scanf("%x", &n);

m = n; // Save the decimal value in variable m

printf("Decimal value: %d\n", m);

for (i = 0; m > 0; i++) {

a[i] = m % 2;

m = m / 2;

}

printf("Binary of Given Number is: ");

for (i = i - 1; i >= 0; i--) {

printf("%d", a[i]);

}

return 0;

}

#include <stdio.h>

int main() {

int n, a[10], m, i;

printf("Enter the hexadecimal number: ");

scanf("%x", &n);

m = n; // Save the decimal value in variable m

printf("Decimal value: %d\n", m);

for (i = 0; m > 0; i++) {

a[i] = m % 8;

m = m / 8;

}

printf("Octal of Given Number is: ");

for (i = i - 1; i >= 0; i--) {

printf("%d", a[i]);

}

return 0;

}

#include <stdio.h>

int main() {

int n, a[10], m, i;

printf("Enter the hexadecimal number: ");

scanf("%x", &n);

m = n; // Save the decimal value in variable m

printf("Decimal value: %d\n", m);

for (i = 0; m > 0; i++) {

a[i] = m % 2;

m = m / 2;

}

printf("binary of Given Number is: ");

for (i = i - 1; i >= 0; i--) {

printf("%d", a[i]);

}

return 0;

}

1. **CACHE MEMORY**

#include <stdio.h>

#include <stdlib.h>

void simulateCache(int arraySize) {

for (int i = 0; i < arraySize; ++i) {

}

}

int main() {

printf("Simulating cache behavior...\n");

// Simulate cache behavior with different array sizes

simulateCache(1024); // Small array fits in the cache

simulateCache(2048); // Larger array, might experience cache misses

printf("Simulation complete.\n");

return 0;

}

1. **VIRTUAL MEMORY**

#include <stdio.h>

#include <stdlib.h>

#define PAGE\_SIZE 4096

#define NUM\_PAGES 1024

#define VIRTUAL\_MEMORY\_SIZE (PAGE\_SIZE \* NUM\_PAGES)

int main() {

srand(42);

for (int i = 0; i < 10; ++i) {

int virtual\_address = rand() % VIRTUAL\_MEMORY\_SIZE;

printf("Virtual Address: %d\n", virtual\_address);

}

return 0;

}

1. **BOOTH ALGORITHM**

#include <stdio.h>

int boothMultiplication(int multiplicand, int multiplier) {

int result = 0;

int a = multiplicand;

int b = multiplier;

int product = 0;

int count = 4;

while (count != 0) {

if (b & 1) {

product += a;

}

a <<= 1;

b >>= 1;

count--;

}

result = product;

return result;

}

int main() {

int multiplicand = 5; // Binary: 0101

int multiplier = 3; // Binary: 0011

int result = boothMultiplication(multiplicand, multiplier);

printf("Result of multiplication: %d\n", result);

return 0;

}

1. **HAZARDS**

#include <stdio.h>

int main() {

int a = 5, b = 10, c = 15, result;

// Data Hazard - Read After Write (RAW)

result = a + b; // Data dependency on 'a' and 'b'

printf("Data Hazard (RAW): %d\n", result);

// Control Hazard - Conditional Branch

if (c > 10) {

result = a + b;

} else {

result = a - b;

}

printf("Control Hazard: %d\n", result);

// Structural Hazard - Resource Conflict

int array1[5], array2[5];

for (int i = 0; i < 5; i++) {

array1[i] = i;

}

// Structural Hazard: Both loops trying to access the array simultaneously

for (int i = 0; i < 5; i++) {

array2[i] = array1[i] \* 2;

}

printf("Structural Hazard: Array values multiplied by 2.\n");

return 0;

}

1. **SINGLE BUS ORGANIZATION**

#include <stdio.h>

int main() {

int memory[10] = {0}; // Array to represent memory

// Write data to memory

int address = 5;

int data = 42;

memory[address] = data;

// Read data from memory

int readData = memory[address];

// Print the read data

printf("Data read from memory at address %d: %d\n", address, readData);

return 0;

}

1. **SUPER SCALAR PROCESSING**

#include <stdio.h>

// Function to simulate an instruction execution

void executeInstruction(int instructionNumber) {

printf("Executing instruction %d\n", instructionNumber);

// Simulated instruction execution logic

}

int main() {

// Simulated instruction stream

int instructionStream[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 1};

// Simulated superscalar execution with two pipelines

for (int i = 0; i < 10; i += 2) {

// Execute two instructions simultaneously using two pipelines

executeInstruction(instructionStream[i]);

executeInstruction(instructionStream[i + 1]);

}

return 0;

}

1. **MEMORY ALLOCATION**

#include <stdio.h>

#include <stdlib.h>

int main() {

// Allocate memory for an array of 5 integers

int \*array = (int \*)malloc(5 \* sizeof(int));

if (array == NULL) {

printf("Memory allocation failed\n");

return 1;

}

// Assign values to the allocated memory

for (int i = 0; i < 5; i++) {

array[i] = i \* 10;

}

// Print the values

for (int i = 0; i < 5; i++) {

printf("array[%d] = %d\n", i, array[i]);

}

// Free the allocated memory

free(array);

return 0;

}