N.G.F COLLEGE OF ENGINEERING AND TECHNOLOGY

PALWAL, HARYANA

COMPUTER ORGANIZATION & ARCHITECTURE LAB

CODE: PCC-CS-405

Submitted to:

Mrs. Bhawna

(Assistant Professor)

Submitted by:

NAME:-

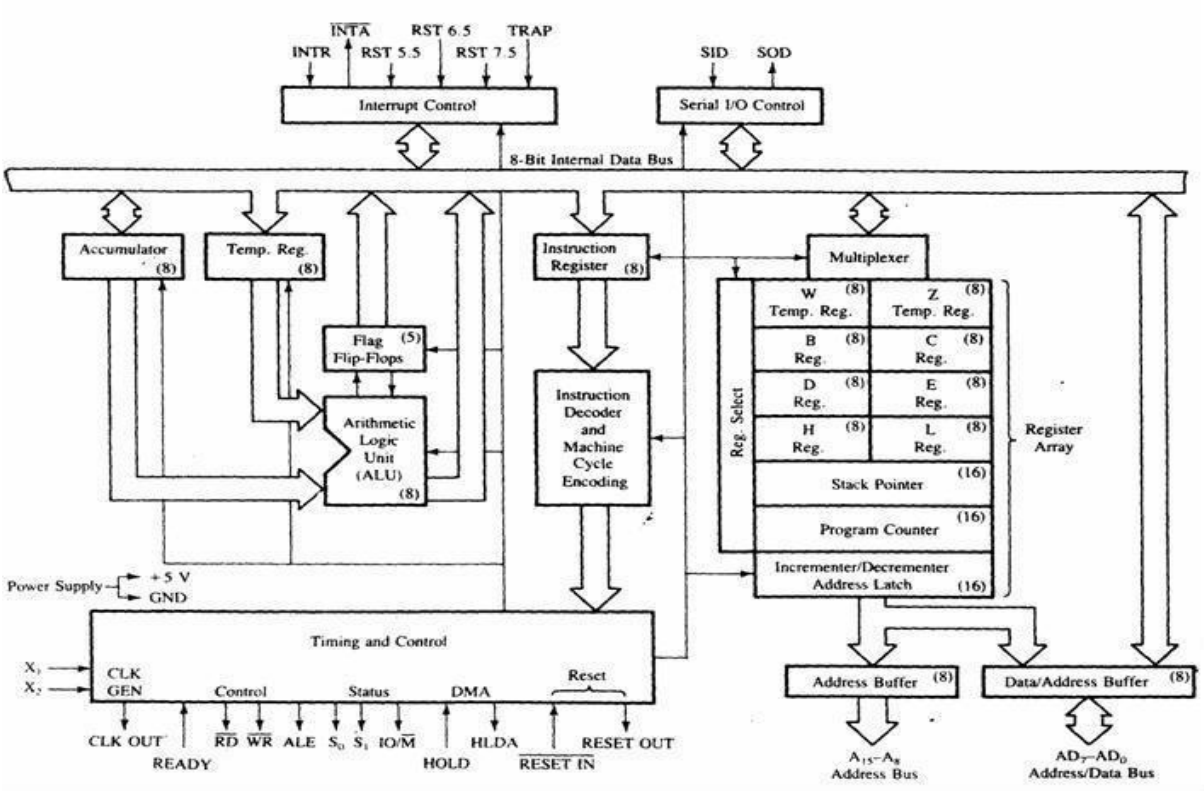
Roll No.

INDEX

PROGRAM-1

Write a Program to study working of GNU Simulator and basic architecture of 8085 along with small introduction.

ARCHITECTURE OF MICROPROCESSOR 8085:



List of registers used in 8085 to perform various operations:

Accumulator:-It is a 8-bit register which is used to perform arithmetical and logical operation. It stores the output of any operation. It also works as registers for i/o accesses.
It can be one of the operand in the instruction.

Temporary Register:-It is a 8-bit register which is used to hold the data on which the accumulator is computing operation. It is also called as operand register because it provides operands to ALU. **Registers**:-These are general purposes registers. Microprocessor consists 6 general purpose registers of 8-bit each named as B,C,D,E,H and L. Generally theses registers are not used for storing the data permanently. It carries the 8-bits data. These are used only during the execution of the instructions. These registers can also be used to carry the 16 bits data by making the pair of 2 registers. The valid register pairs available are BC,DE HL. We cannot use other pairs except BC,D,Eand HL.

These registers are programmed by user.

Flag Registers:-It consists of 5 flip flop which changes its status according to the result stored in an accumulator. It is also known as status registers. It is connected to the ALU. There are five flip-flops in the flag register are as follows:

The bit position of the flip flop in flag register is:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

All of the three flip flop set and reset according to the stored result in the accumulator.

1.**Sign**- If D7 of the result is 1 then sign flag is set otherwise reset. As we know that a number on the D7 always desides the sign of the number. if D7 is 1: the number is negative. if D7 is 0: the number is positive.

2.**Zeros(Z)**-If the result stored in an accumulator is zero then this flip flop is set as 1 otherwise it is reset and also if the result of any arithmetic or logical operation is zero its set as 1(The result of this operation can be in any registers).

3.**Auxiliary carry(AC)**-If any carry goes from D3 to D4 in the output then it is set otherwise it is reset.

4.**Parity(P)**-If the no of 1's is even in the output stored in the accumulator then it is set otherwise it is reset for the odd.

5.**Carry(C)**-If the result stored in an accumulator generates a carry in its final output then it is set otherwise it is reset.

Instruction registers(IR):-It is a 8-bit register. When an instruction is fetched from memory then it is stored in this register.

Description of other components of 8085 to:

Instruction Decoder: - Instruction decoder identifies the instructions. It takes the information from instruction register and decodes the instruction to be performed.

Program Counter:-It is a 16 bit register used as memory pointer. It stores the memory address of the next instruction to be executed. So we can say that this register is used to sequencing the program. Generally the memory have 16 bit addresses so that it has 16 bit memory. The program counter is set to 0000H. **Stack Pointer:**-It is also a 16 bit register used as memory pointer. It points to the memory location called stack. Generally stack is a reserved portion of memory where information can be stores or taken back together.

Timing and Control Unit:-It provides timing and control signal to the microprocessor to perform the various operation. It has three control signals. It controls all external and internal circuits. It operates with reference to clock signal. It synchronizes all the data transfers.

There are three control signal:

1.ALE- Airthmetic Latch Enable, It provides control signal to synchronize the components of microprocessor.

2.RD- This is active low used for reading operation.

3.WR-This is active low used for writing operation.

There are three status signal used in microprocessor S0, S1 and IO/M. It changes its status according the provided input to these pins.

Serial Input Output Control-

There are two pins in this unit. This unit is used for serial data communication. **Interrupt Unit**-There are 6 interrupt pins in this unit. Generally an external hardware is connected to these pins. These pins provide interrupt signal sent by external hardware to microprocessor and microprocessor sends acknowledgement for receiving the interrupt signal. Generally INTA is used for acknowledgement.

INTRODUCTION TO GNU Simulator 8085

8085 simulator is software on which instructions are executed by writing the programs in assembly language.
GNUSim8085 is a 8085 microprocessor simulator with following features.

- A simple editor component with syntax highlighting.
- A keypad to input assembly language instructions with appropriate arguments.
- Easy view of register contents.
- Easy view of flag contents.
- Hexadecimal <--> Decimal converter.
- View of stack, memory and I/O contents.
- Support for breakpoints for programming debugging.
- Stepwise program execution.
- One click conversion of assembly program to opcode listing.
- Printing support (known not to work well on Windows).
- UI translated in various languages.

Writing a program in assembly language:- Format of the instruction is as follows:-

Label	Operation	Operands	Comments
Its optional	Necessary	Necessary	Its optional

A basic assembly program consists of 4 parts.

1. Labels
2. Operations :- these operations can be specified as

Machine operations (mnemonics):-
They used to define operations in the form of opcode as mention in the instruction set of microprocessor 8085.

Pseudo operations (like preprocessor in C):-
these are assembly directives.

3. Operands
4. Comments

In addition, you have **constants** in an assembly program. Unless otherwise specified, a constant which is always numeric is in decimal form. If appended with a character h it is assumed to be in hexadecimal form. If a hex constant starts with an alpha-char don't forget to include the number 0 in the beginning, since that will help the assembler to differentiate between a label and a constant.

Labels:-
When given to any particular instruction/data in a program, takes the address of that instruction or data as its value. But it has different meaning when given to EQU directive. Then it takes the operand of EQU as its value. Labels must always be placed in the first column and must be followed by an instruction (no empty line). Labels must be followed by a : (colon), to differentiate it from other tokens.

Operations:-
As mentioned above the operations can be specified in two ways that are **mnemonics** and **pseudo operation**.
Pseudo operations can be defined by using following directives:- There are only 3 directives currently available in our assembly language.

1. DB - define byte (8 bits)

2. DS - define size (no. of bytes)
3. EQU - like minimalistic #define in C

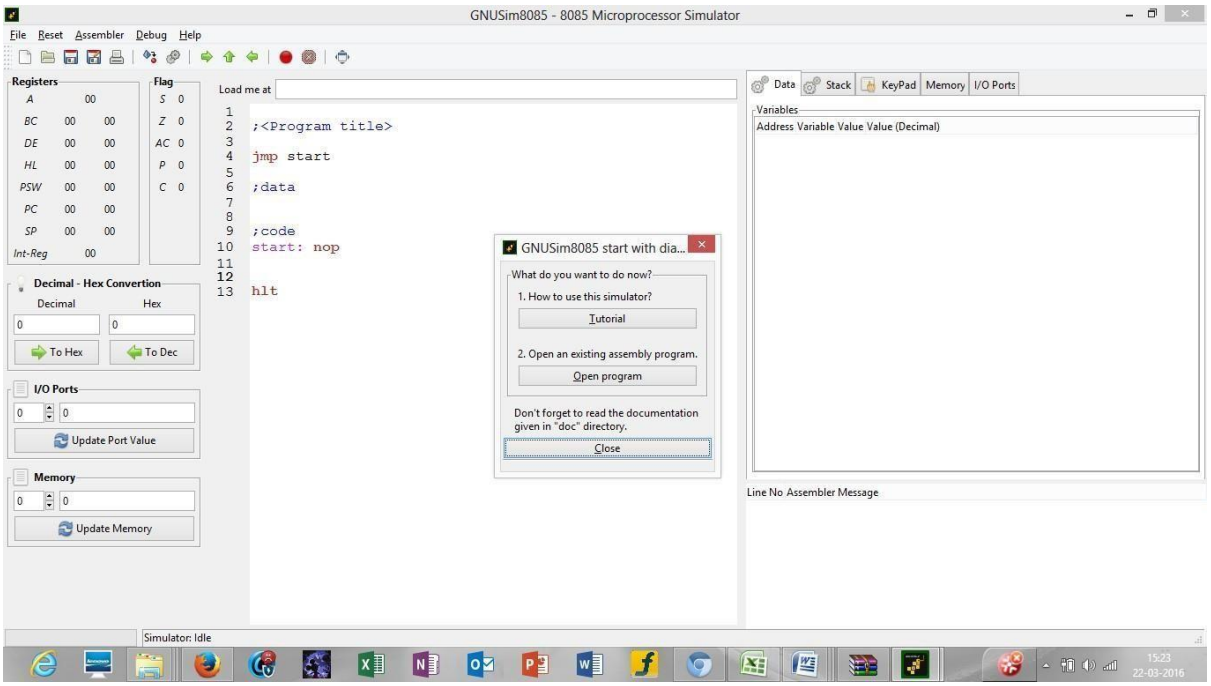
1. DB is used to define space for an array of values specified by comma separated list. And the label (if given to the beginning of DB) is assigned the address of the first data item.
2. DS is used to define the specified number of bytes to be assigned and initialize them to zero. To access each byte you can use the + or -operator along with label.
3. EQU behaves similar to #define in C. But it is simple. It can be used to give names only to numeric constants. Nesting of EQU is not allowed. You can use EQU only in operands for pseudo ops and mnemonics.

Operands:-

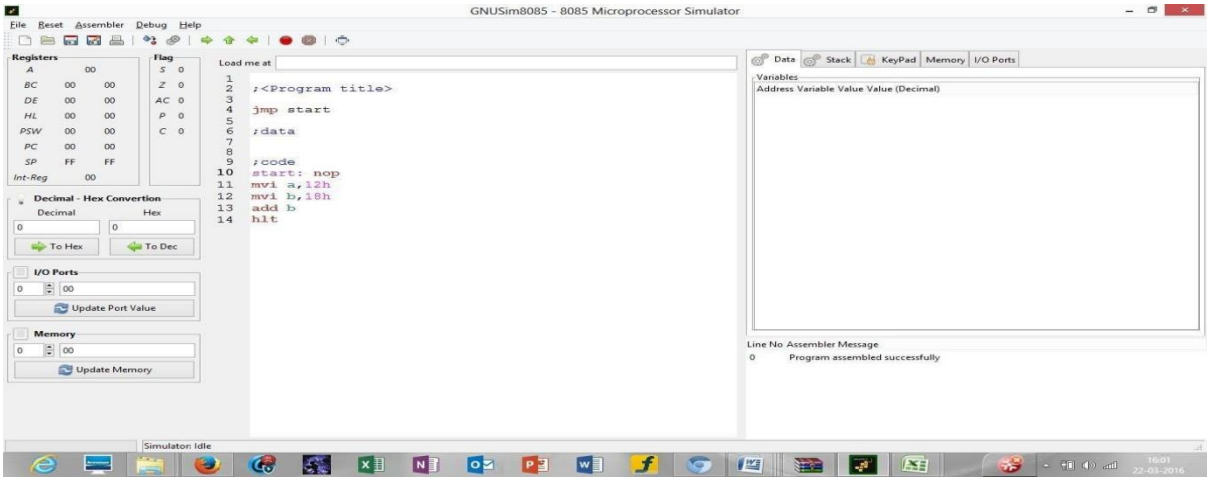
Operands are specified according to the user. The register set specified in the architecture of 8085 (A, B, C, D, H and L) are used to access and store data. These registers are specified as operand. In case of accessing data or storing data in the memory 'm' is specified as an operand and the address of this memory location is taken from the HL pair (data in HL pair).

INSTRUCTION SET OF MICROPROCESSOR 8085

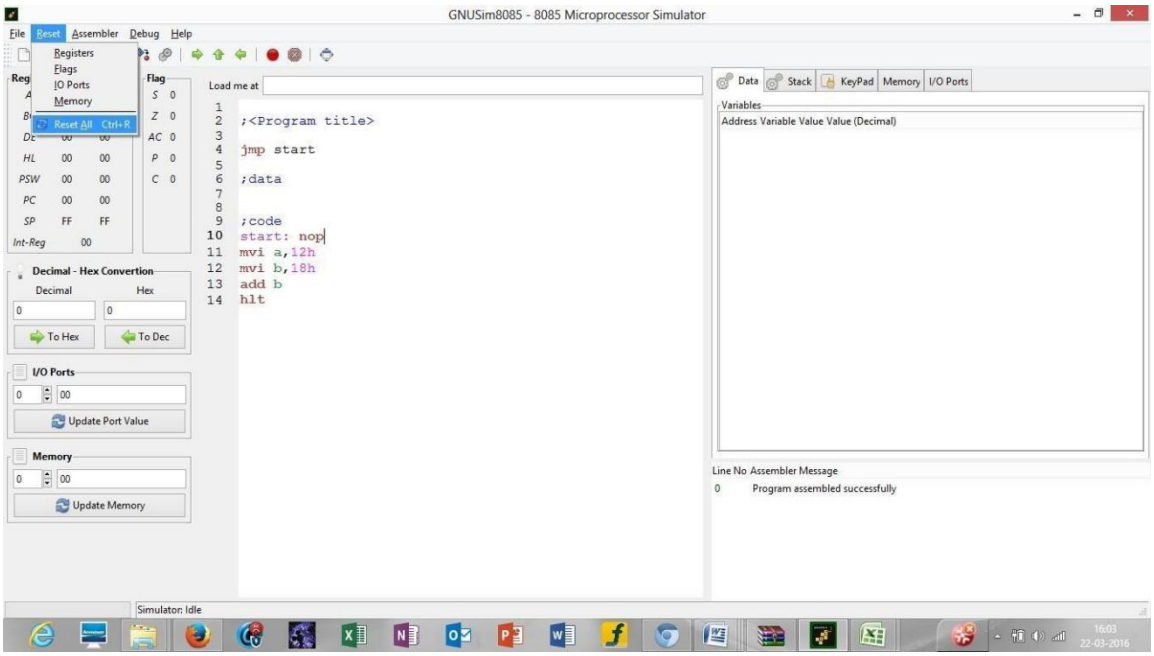
HOW TO START WITH GNU Simulator 8085



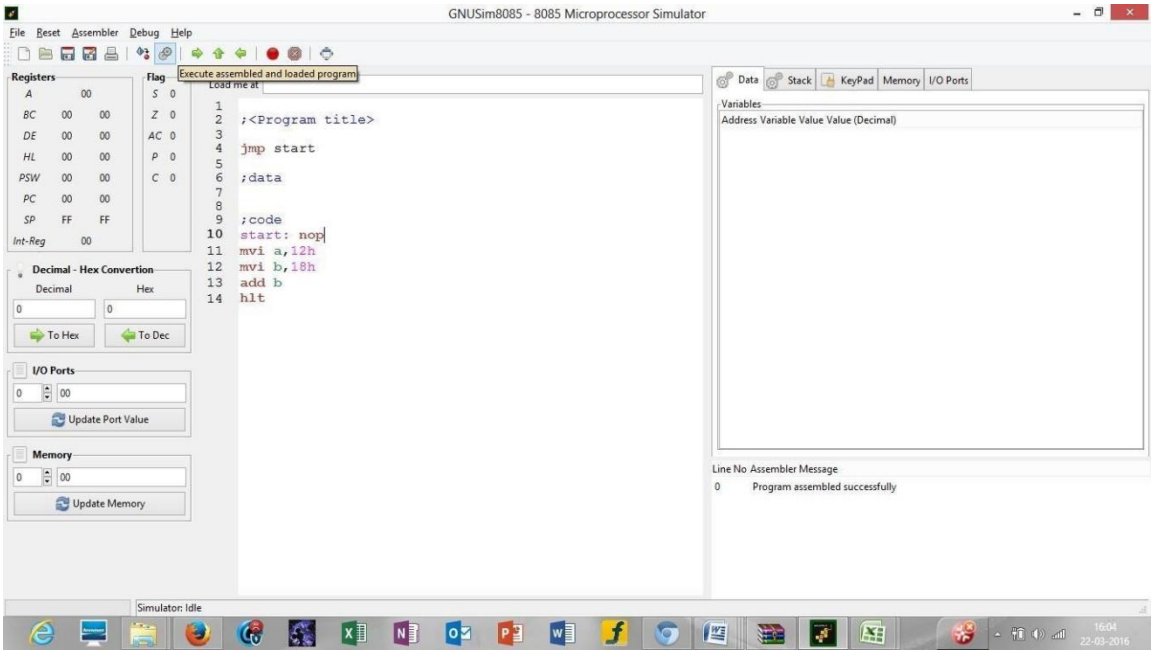
Step1:open GNU Sim 8085 above window will open. Now click on close button highlighted in the above screen shot.



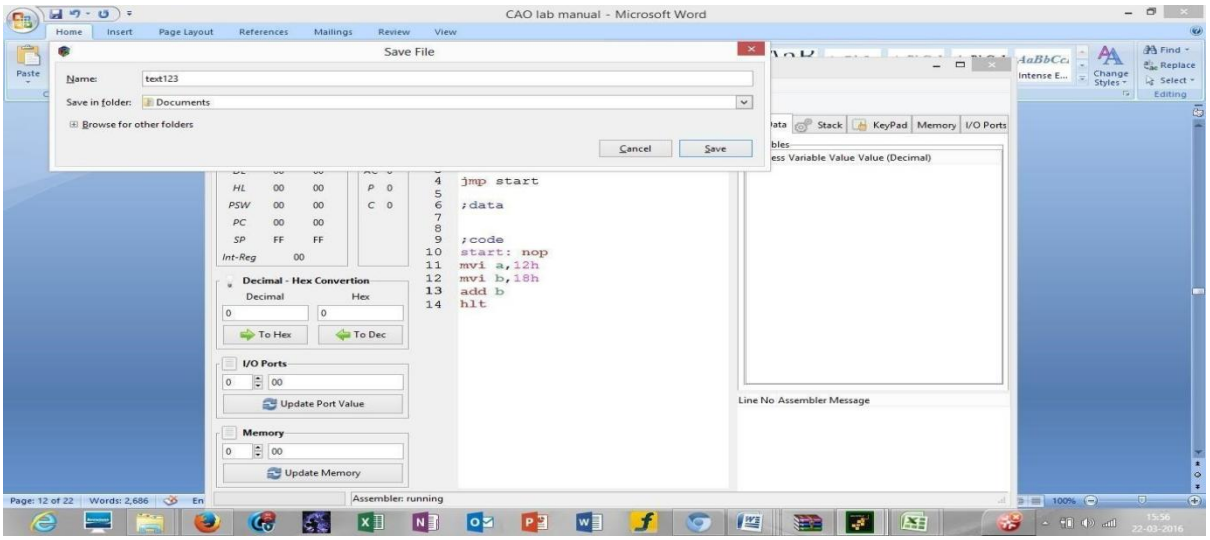
Step2: start writing the code after start: nop in load me at 10 that is at load me at 11.



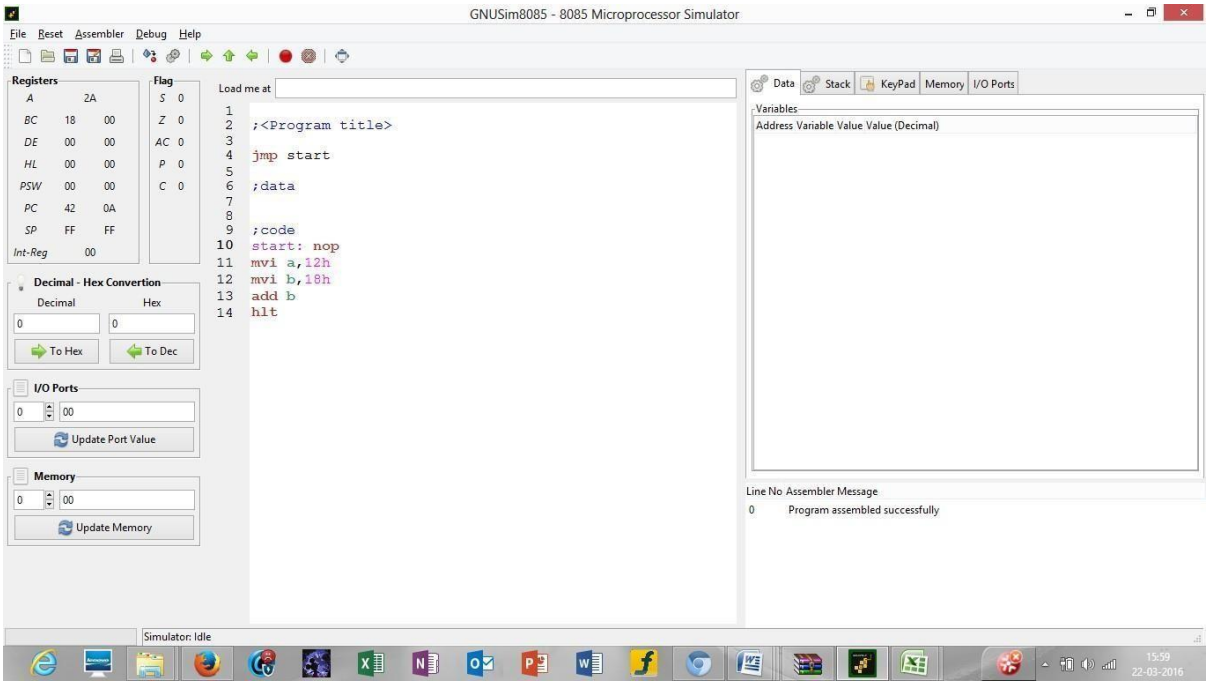
Step 3: click on reset and reset all the registers by clicking on reset all.



Step 4: click on the highlighted button to execute the code



Step 5: after you execute the code mention the name your program by writing the name in the name section as mentioned in the screen shot in picture 5 and the drive where you want to save it. After that click on save.



PROGRAM-2

Aim: To study the ALU

INTRODUCTION:

Arithmetic Logic Unit:

In digital electronics, an arithmetic and logic unit (ALU) is a digital circuit that performs integer arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs.

Mathematician John von Neumann proposed the ALU concept in 1945, when he wrote a report on the foundations for a new computer called the EDVAC.

The inputs to the ALU are the data to be operated on (called operands) and a code from the control unit indicating which operation to perform. Its output is the result of the computation. One thing designers must keep in mind is whether the ALU will operate on big-endian or little- endian numbers.In many designs, the ALU also takes or generates inputs or outputs a set of condition codes from or to a status register. These codes are used to indicate cases such as carry-in or carry-out, overflow, divide-by-zero, etc.

A floating-point unit also performs arithmetic operations between two values, but they do so for numbers in floating-point representation, which is much more complicated than the two's complement representation used in a typical ALU. In order to do these calculations, a FPU has several complex circuits built-in, including some internal ALUs.

In modern practice, engineers typically refer to the ALU as the circuit that performs integer arithmetic operations (like two's complement and BCD). Circuits that calculate more complex formats like floating point, complex numbers, etc. usually receive a more specific name such as floating-point unit (FPU).

Conclusion: Thus we have studied ALU Successfully

PROGRAM-3

Write a program for multiplication of two binary numbers

```
#include<stdio.h>
int binaryAddition(int,int); int
main(){
long int binary1,binary2,multiply=0; int
digit,factor=1;

printf("Enter any first binary number: "); scanf("%ld",&binary1);
printf("Enter any second binary number: ");
scanf("%ld",&binary2);

   while(binary2!=0){ digit =
   binary2 %10;

      if(digit ==1){
           binary1=binary1*factor;
           multiply = binaryAddition(binary1,multiply);
      }
      else
        binary1=binary1*factor;

      binary2 = binary2/10; factor =
      10;
   }
   printf("Product of two binary numbers: %ld",multiply); return 0;
}
int binaryAddition(int binary1,int binary2){ int i=0,remainder = 0,sum[20];
   int binarySum=0;

   while(binary1!=0||binary2!=0){
      sum[i++] = (binary1 %10 + binary2 %10 + remainder ) % 2;
      remainder = (binary1 %10 + binary2 %10 + remainder ) / 2; binary1
      = binary1/10;
      binary2 = binary2/10;
   }

   if(remainder!=0) sum[i++] =
      remainder;
   --i;
   while(i>=0)
      binarySum = binarySum*10 + sum[i--];

   return binarySum;
}
```

Output:

```
Enter any first binary number: 1101 Enter any
second binary number: 11 Product of two binary
numbers: 100111
```

PROGRAM-4

Write a program for LRU page replacement algorithm.

PROGRAM:

```
#include<stdio.h>
main()
{
int     q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf("Enter no of pages:");
scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)
scanf("%d",&p[i]);    printf("Enter
no of frames:"); scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]); c++;
k++;
for(i=1;i<n;i++)
 { c1=0;
 for(j=0;j<f;j++)
 {
 if(p[i]!=q[j])
 c1++;
 }
 if(c1==f)
 { c++;
 if(k<f)
 {
q[k]=p[i]; k++;
for(j=0;j<k;j++)
printf("\t%d",q[j]);
printf("\n");
 }
 else
 {
 for(r=0;r<f;r++)
 { c2[r]=0;
 for(j=i-1;j<n;j--)
 {
 if(q[r]!=p[j]) else
 break;
 }
 }
 for(r=0;r<f;r++)
  b[r]=c2[r];


 for(r=0;r<f;r++)
  {
 for(j=r;j<f;j++)
 {
 if(b[r]<b[j])
 {
```

```
 t=b[r];
 b[r]=b[j];
 b[j]=t;
}
 }
  }
 for(r=0;r<f;r++)
 {
 if(c2[r]==b[0])
 q[r]=p[i];
 printf("\t%d",q[r]);
}
 printf("\n");
 }
  }
}
printf("\nThe no of page faults is %d",c);
}
```

OUTPUT:

```
Enter no of pages:10
Enter the reference string:7 5 9 4 3 7 9 6 2 1
Enter no of frames:3
 7
 7 5
 7 5 9
 4 5 9
 4 3 9
 4 3 7
 9 3 7
 9 6 7
 9 6 2
 1 6 2
```

PROGRAM-5

Main program to add two unsigned binary number [4 bit binary adder]

```
display('Enter First Binary Sequence') for i=1:4
a(i)=input(''); end
display('Enter Second Binary Sequence') for i=1:4
b(i)=input(''); end
carry=0; for i=4:-1:1
c(i)=xors(xors(a(i),b(i)),carry); carry=ors(ands(a(i),b(i)),ands(xors(a(i),b(i)),carry));
end
%*****************************************
fprintf('%d',a)
fprintf('\n')
fprintf('%d',b) fprintf('+\n')
display('====================')
fprintf('%d%d',carry,c) fprintf('\n')
```

Function for AND operation

```
function [res] = ands(a,b) if(a==1) && (b==1)
res=1;
else
res=0;
end end
```

Function for XOR operation

```
function [res]= xors(a,b) if (a~=b)
res=1;
else
res=0;
end end
```

Function for OR operation

```
function [res] = ors(a,b) if (a==0) && (b==0)
res=0;
else
  res=1;
end
end
```

PROGRAM-6

Write a program for binary multiplication.

```
#include<stdio.h>
int binaryAddition(int,int); int
main(){

  long int binary1,binary2,multiply=0; int
   digit,factor=1;

printf("Enter any first binary number: "); scanf("%ld",&binary1);
   printf("Enter any second binary number: ");
   scanf("%ld",&binary2);

   while(binary2!=0){ digit =
      binary2 %10;

      if(digit ==1){
           binary1=binary1*factor;
           multiply = binaryAddition(binary1,multiply);
      }
      else
        binary1=binary1*factor;

      binary2 = binary2/10; factor =
      10;
   }

   printf("Product of two binary numbers: %ld",multiply); return 0;
}

int binaryAddition(int binary1,int binary2){ int

   i=0,remainder = 0,sum[20];
   int binarySum=0;

   while(binary1!=0||binary2!=0){
      sum[i++] = (binary1 %10 + binary2 %10 + remainder ) % 2;
      remainder = (binary1 %10 + binary2 %10 + remainder ) / 2; binary1
      = binary1/10;
      binary2 = binary2/10;
   }

   if(remainder!=0) sum[i++] =
      remainder;
   --i;
   while(i>=0)




      binarySum = binarySum*10 + sum[i--];
```

```
  return binarySum;
}
```

Output:

Enter any first binary number: 1101
Enter any second binary number: 11 Product of two binary numbers: 100111

PROGRAM-7

AIM: Write a program for FIFO page replacement algorithm.

```
#include<stdio.h> int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
printf("\n ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)  scanf("%d",&a[i]);
printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&no); for(i=0;i<no;i++)
frame[i]= -1;
 j=0;
 printf("\tref string\t page frames\n"); for(i=1;i<=n;i++) {
printf("%d\t\t",a[i]);               avail=0;
for(k=0;k<no;k++)
                                    if(frame[k]==a[i])
 avail=1;
 if (avail==0)
{
 frame[j]=a[i]; j=(j+1)%no; count++;
 for(k=0;k<no;k++)
 printf("%d\t",frame[k]); }
 printf("\n");
}
 printf("Page Fault Is %d",count); return 0;
}
```

OUTPUT:

ENTER THE NUMBER OF PAGES: 20

ENTER THE PAGE NUMBER : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 ENTER THE NUMBER

OF FRAMES :3

ref string page frames 7 7 -1

-1

0 7 0 -1

1 7 0 1

2 2 0 1

0

3 2 3 1

0 2 3 0

4 4 3 0

2 4 2 0

3 4 2 3

0 0 2 3

3

2

1 0 1 3

2 0 1 2

0

1

7 7 1 2

0 7 0 2

1 7 0 1

Page Fault Is 15