**Experiment 6**

**Project by**

|  |  |
| --- | --- |
| **Name:** | **Roll No:** |
| 1: Tanmay Dhanu | 19102A2003 |

**Non Restoring Algorithm**

|  |  |
| --- | --- |
| **Equipments Required:** | Computer, compiler |
| **Duration:** | 2 Hours |
| **Objectives:** | Design Non Restoring Algorithm using virtual lab simulator |
| **Description:** | To code and implement a program to calculate the results of the non restoring algorithm |
| **Theory/History:** | Non-restoring division uses the digit set {−1, 1} for the quotient digits instead of {0, 1}. The algorithm is more complex, but has the advantage when implemented in hardware that there is only one decision and addition/subtraction per quotient bit; there is no restoring step after the subtraction, which potentially cuts down the numbers of operations by up to half and lets it be executed faster. The basic algorithm for binary (radix 2) non-restoring division of non-negative numbers is: |
| **Algorithm/Flowchart:** | R := N  D := D << n -- R and D need twice the word width of N and Q  for i = n − 1 .. 0 do -- for example 31..0 for 32 bits  if R >= 0 then  q[i] := +1  R := 2 \* R − D  else  q[i] := −1  R := 2 \* R + D  end if  end    -- Note: N=Numerator, D=Denominator, n=#bits, R=Partial remainder, q(i)=bit #i of quotient. |
| **Code:** | class nonrestore:  def \_\_init\_\_(self,a,q,m):  self.a = a  self.q = q  self.m = m  self.cm = self.comp(m)  def leftshift(self):  for i in range(len(a)-1):  self.a[i] = self.a[i+1]  self.a[-1] = self.q[0]  for i in range(len(a)-1):  self.q[i] = self.q[i+1]  self.q[-1] = 0  def comp(self,m):  carry = 0  j=-1  for i in range(len(m)):  if m[i] == 0:  m[i] = 1  else:  m[i] = 0  while m[j] == 1:  m[j] = 0  j = j - 1  m[j] = 1  return m  def add(self):  carry = 0  for i in range(-1,-len(self.a)-1,-1):  self.a[i]=int((self.a[i] + self.m[i] + carry) % 2)  carry=int((self.a[i] + self.m[i] + carry) / 2)  def sub(self):  carry = 0  for i in range(-1,-len(self.a)-1,-1):  temp = (self.a[i]+self.cm[i]+carry)  if ((self.a[i]+self.cm[i]+carry)%2) == 0:  self.a[i] = 0  else:  self.a[i] = 1  if temp >= 2:  carry = 1  else:  carry = 0  def calc(self):  print("Intermediate States:")  for i in range(len(self.a),0,-1):  self.leftshift()  self.display1(i)  if self.a[0] == 0:  self.sub()  if self.a[0] == 0:  self.q[-1] = 1  else:  self.q[-1] = 0  self.display1(i)  else:  self.add()  if self.a[0] == 0:  self.q[-1] = 1  else:  self.q[-1] = 0  self.display1(i)  if self.a[0] == 1:  self.add()  def display(self):  rem = 0  quo = 0  for i in range(-1,-len(self.a)-1,-1):  rem += self.a[i]\*(2\*\*(abs(i)-1))  for i in range(-1,-len(self.q)-1,-1):  quo += self.q[i]\*(2\*\*(abs(i)-1))  print(f"Remainder = {rem} Quotient = {quo}")  def display1(self,count):  print(f"SC = {count} A = {self.a} Q = {self.q}")  a = [0,0,0,0]  q = [1,0,1,0]  m = [0,0,1,1]  print(f"A = {a}\nQ = {q}\nM = {m}")  d = nonrestore(a,q,m)  d.calc()  print(f"Final Result:")  d.display() |
| **Application:** | In computers to calculate the division |
| **Conclusion:** | We have successfully implemented non restoring division algorithm. |
| **References:** | Wikipedia |

|  |
| --- |
| **Output** |
|  |