**Experiment 6**

**Project by**

|  |  |
| --- | --- |
| **Name:** | **Roll No:** |
| 1: Akshay Prabhu | 18102A0058 |

**Restoring Algorithm**

|  |  |
| --- | --- |
| **Equipments Required:** | Computer, compiler |
| **Duration:** | 2 Hours |
| **Objectives:** | Design Restoring Algorithm using virtual lab simulator |
| **Description:** | To code and implement a program to calculate the results of the restoring algorithm |
| **Theory/History:** | Restoring division operates on fixed-point fractional numbers and depends on the assumption 0 < D < N.  The quotient digits q are formed from the digit set {0,1}.  The basic algorithm for binary (radix 2) restoring division 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  R := 2 \* R − D -- Trial subtraction from shifted value (multiplication by 2 is a shift in binary representation)  if R ≥ 0 then  q(i) := 1 -- Result-bit 1  else  q(i) := 0 -- Result-bit 0  R := R + D -- New partial remainder is (restored) shifted value  end  end  -- Where: N = Numerator, D = Denominator, n = #bits, R = Partial remainder, q(i) = bit #i of quotient |
| **Code:** | class restore:  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)  self.sub()  self.display1(i)  if self.a[0] == 0:  self.q[-1] = 1  self.display1(i)  else:  self.q[-1] = 0  self.add()  self.display1(i)  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 = [0,0,1,0]  print(f"A = {a}\nQ = {q}\nM = {m}")  d = restore(a,q,m)  d.calc()  print(f"Final Result:")  d.display() |
| **Application:** | In computers to calculate the division |
| **Conclusion:** | We have successfully implemented restoring division algorithm. |
| **References:** | Wikipedia |

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