EXPERIMENT 2

**CLASS: TE CMPN A PID:182027**

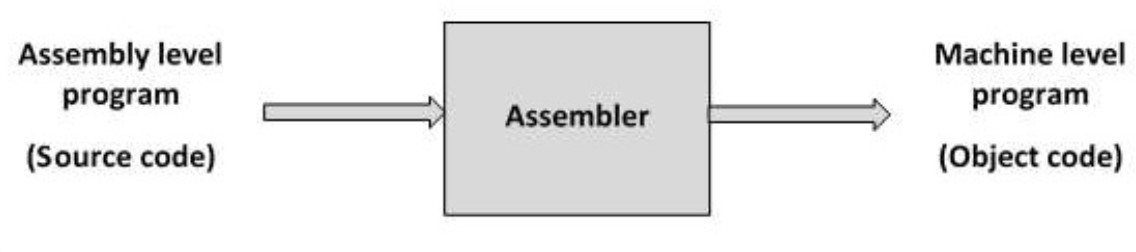
**NAME: REBECCA DIAS ROLL NO. : 19**

# AIM:

TO IMPLEMENT 2 PASS ASSEMBLER.

# THEORY:

1. Assembler?

Assembler is system software which converts an assembly languageThe input to the assembler is a source code written in assembly language (using mnemonics) and the output is an object code.

1. Features of an assembler
   * Translating mnemonic language to its equivalent object code
   * Assigning machine addresses to symbolic labels.
2. General design procedure (steps)
   * Specify the problem.
   * Specify data structures.
   * Deﬁne format of data structures.
   * Specify algorithm.
   * Look for modularity [capability of one program to be subdivided into independent programming units.].
   * Repeat 1 through 5 on modules.
3. Database tables

* Symbol Table:

The symbol table contains information to locate and relocate symbolic deﬁnitions and references. The assembler creates the symbol table section for the object ﬁle.It makes an entry in the symbol table for each symbol that is deﬁned or referenced in the input ﬁle and is needed during linking.

* Literal Table:

Literal table is used for keeping track of literals that are encountered in the programs.We directly specify the value, literal is used to give a location for the value.Literals are always encountered in the operand ﬁeld of an instruction.In pass 1, whenever a Literal is deﬁned and for entry is made in Literal table. In pass2, a Literal table is used for generating addresses of a Literal.

* Pseudo Opcode Table:

POT is the ﬁxed length table.In pass 1, using Pseudo Opcode, POT is consulted for processing some pseudo opcode like DS, DC, START, END, etc.In pass 2 using Pseudo Opcode, POT is consulted for processing some pseudo opcode like DS,DC,USING DROP.

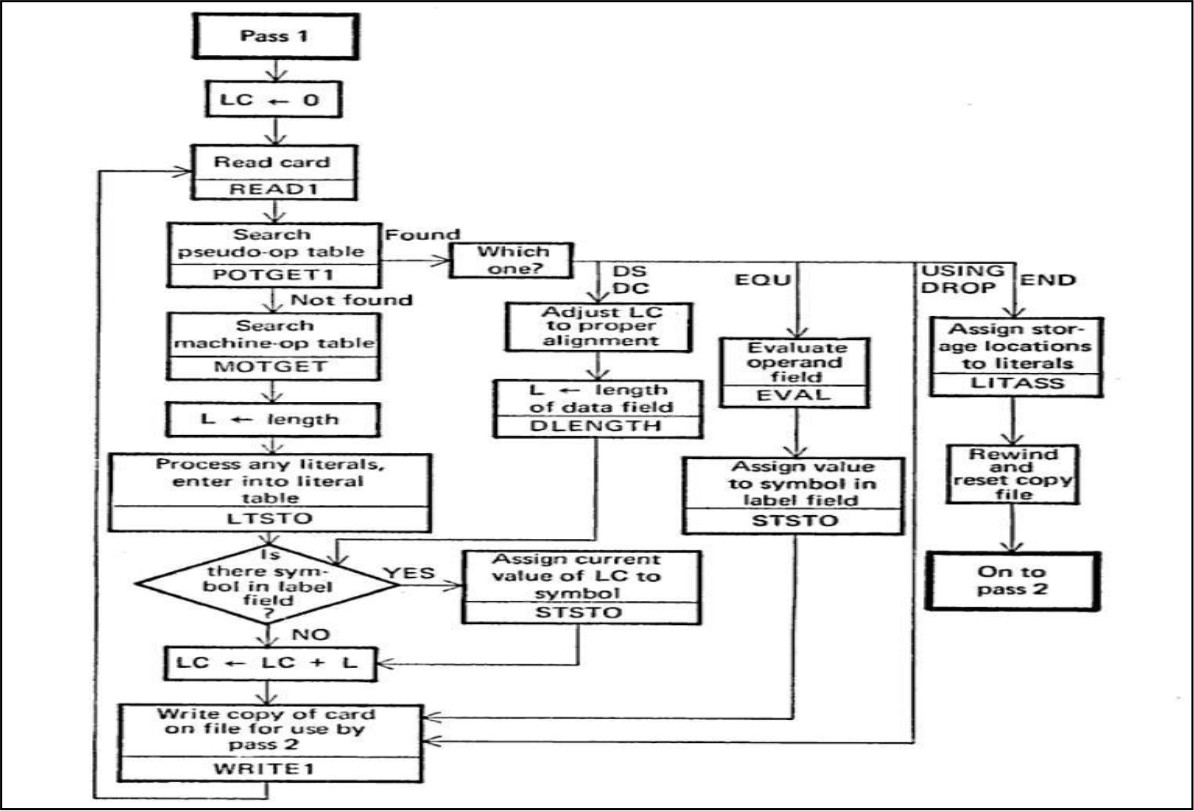
* Machine Opcode Table:

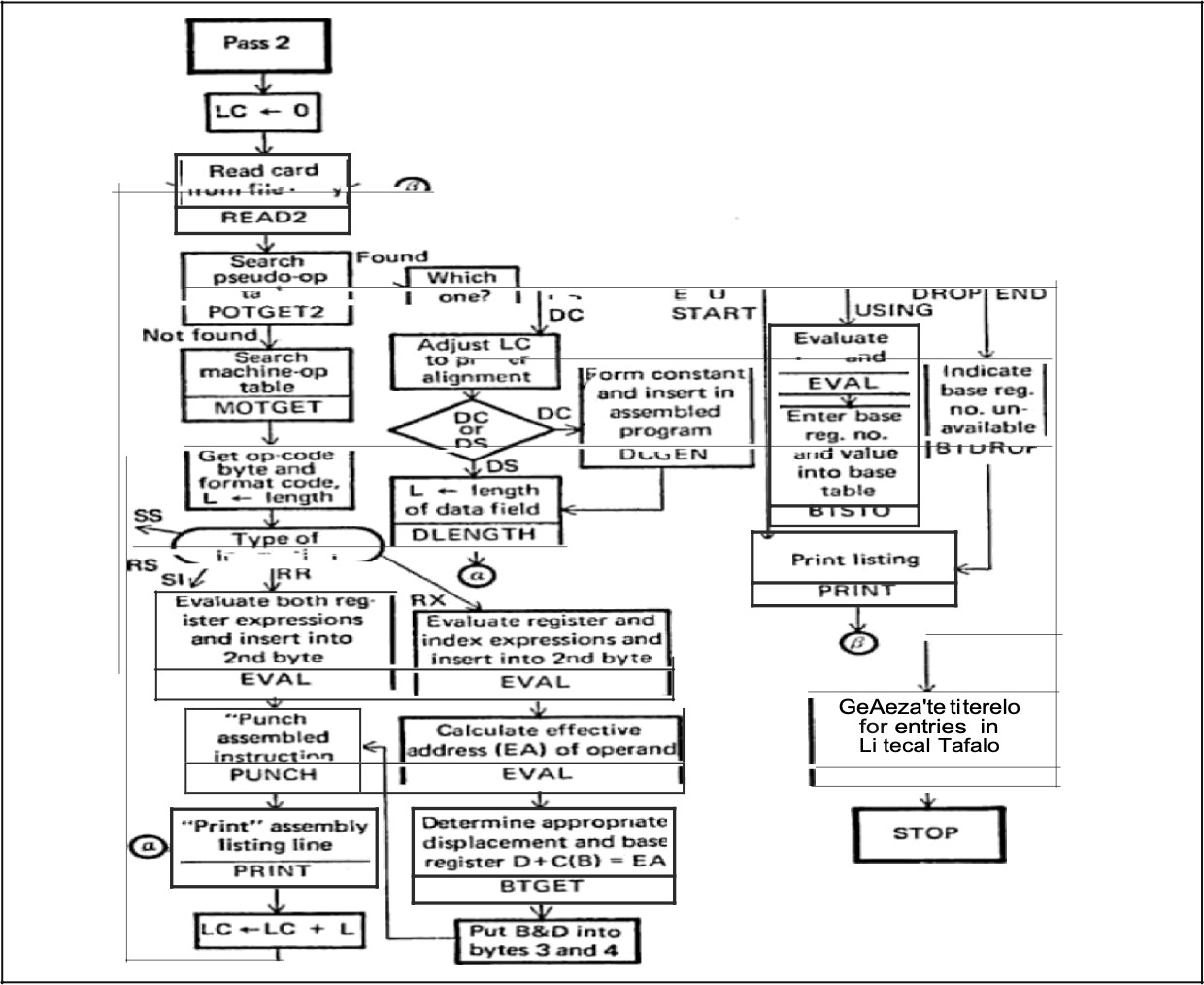
MOT is a ﬁxed length table i.e. we make no entry in either of the passes.It is used to accept the instructions and convert/gives its binary opcode.In pass 1, using mnemonic Opcode, MOT is consulted to update location Counter (LC).

* Base Table:

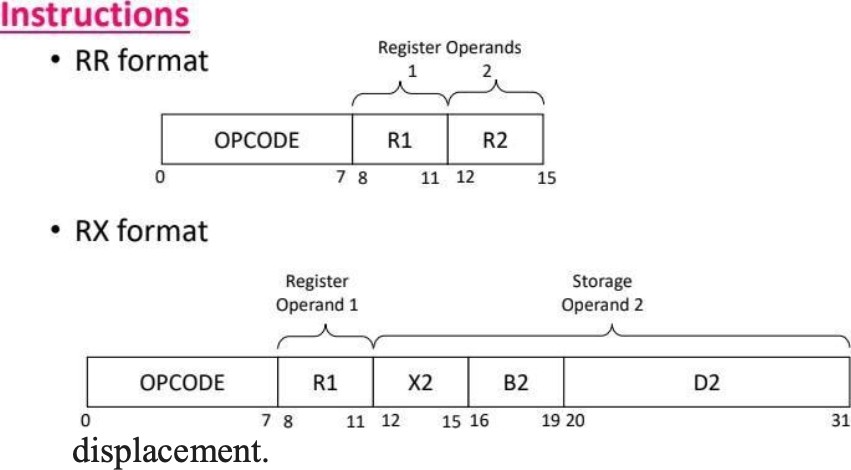
The base table(BT), that indicates which registers are currently speciﬁed as base registers by USING pseudo-ops and what the speciﬁed contents of these registers are.

1. Flowchart

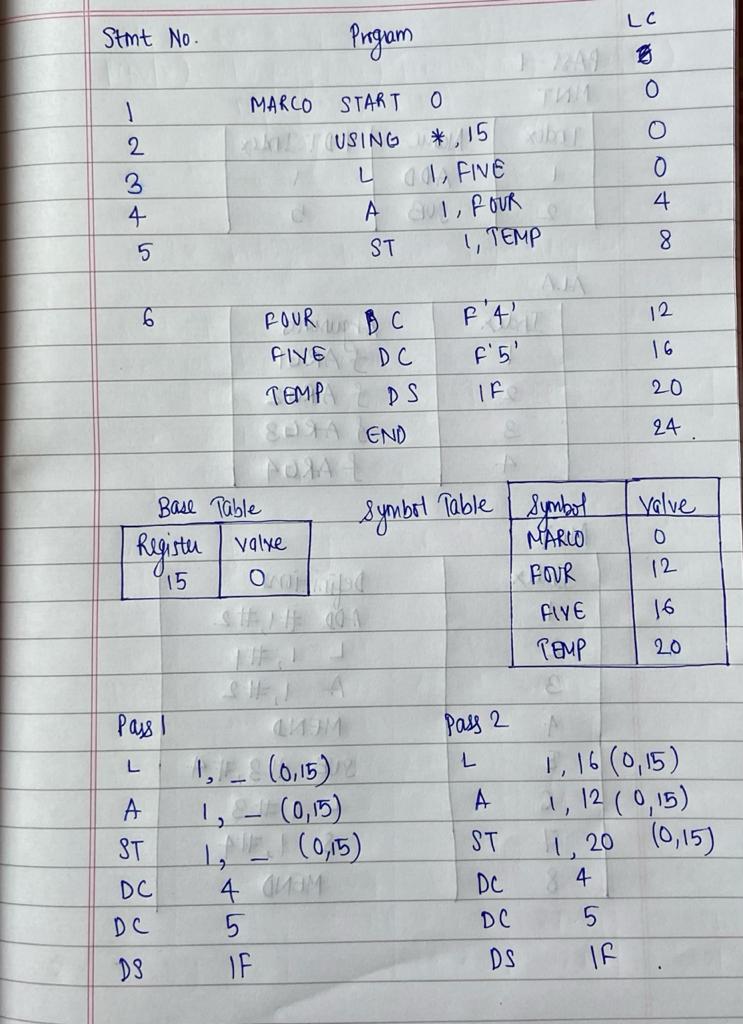




1. IBM360/370instructionformats

RR format.(two bytes):Generally byte 1 speciﬁes two 4-bit register numbers RX format (four bytes). Bits 0-3 of byte 1 specify either a register number or a modiﬁer; bits 4-7 of byte 1 specify the number of the general register to be used as an index; bytes 2-3 specify a base and displacement.

1. Example



# IMPLEMENTATION:

with open("source.txt","r") as fi, open("destination.txt","w") as fo:

    fo.write(fi.read().replace("'"," "))

with open("destination.txt","r") as fileref:

    lines=fileref.readlines()

each\_line=[]

for line in lines:

    each\_line.append(line.strip().split(" "))

each\_line[0][0]="MARCO"

rx\_words=["L","A","ST"]

symbol\_table={}

location\_counter=[0]

base\_table={}

for line in each\_line:

     if (len(line)<4):

         diff=4-len(line)

     for \_ in range(diff):

         line.append("blank")

for i in range(len(each\_line)):

    if (each\_line[i][1]=="START"):

        symbol\_table[each\_line[i][0]]=location\_counter[-1]

        location\_counter.append(int(each\_line[i][2]))

    elif (each\_line[i][0]=="END"):

        break

    elif (each\_line[i][0]=="USING" and each\_line[i][1]=="\*"):

        base\_table[each\_line[i][3]]=location\_counter[-1]

        location\_counter.append(location\_counter[-1])

    elif (each\_line[i][0] in rx\_words):

        location\_counter.append(location\_counter[-1]+4)

        symbol\_table[each\_line[i][3]]=None

    elif (each\_line[i][0] in symbol\_table):

        symbol\_table[each\_line[i][0]]=location\_counter[-1]

        if (each\_line[i][1]=="DS"):

            num1=int(each\_line[i][2][0])

            type\_word=each\_line[i][2][1]

            if (type\_word=="F"):

                num2=4

            location\_counter.append(location\_counter[-1]+num1\*num2)

        elif (each\_line[i][1]=="DC"):

            location\_counter.append(location\_counter[-1]+4)

        else:

            continue

print("BASE TABLE")

print("-"\*20)

print("Base register\tValue")

for key,value in base\_table.items():

    print(f"{int(key)}\t\t{value}")

print("\nSYMBOL TABLE")

print("-"\*20)

print("Symbol\tValue")

for key,value in symbol\_table.items():

    print(f"{key}\t{value}")

print("\nLOCATION COUNTER")

print(location\_counter)

#Pass 1

print("\n\_\_\_\_\_\_\_\_\_\_\_\nPass 1:")

for line in each\_line:

    if (line==each\_line[0] or line==each\_line[1] or line[0]=="END"):

        continue

    elif (line[0] in symbol\_table and line[1]=="DC"):

        print("DC "+line[3])

    elif (line[0] in symbol\_table and line[1]=="DS"):

        print("DS "+line[2])

    else:

        for word in line:

            if (word in symbol\_table):

                print("\_",end=" ")

            elif (word=="blank"):

                continue

            else:

                print(word,end=" ")

        print(f"({base\_table['15']},{15})")

#Pass 2

print("\n\_\_\_\_\_\_\_\nPass 2:")

for line in each\_line:

    if (line==each\_line[0] or line==each\_line[1] or line[0]=="END"):

        continue

    elif (line[0] in symbol\_table and line[1]=="DC"):

        print("DC "+line[3])

    elif (line[0] in symbol\_table and line[1]=="DS"):

        print("DS "+line[2])

    else:

        for word in line:

            if (word in symbol\_table):

                print(symbol\_table[word],end=" ")

            elif (word=="blank"):

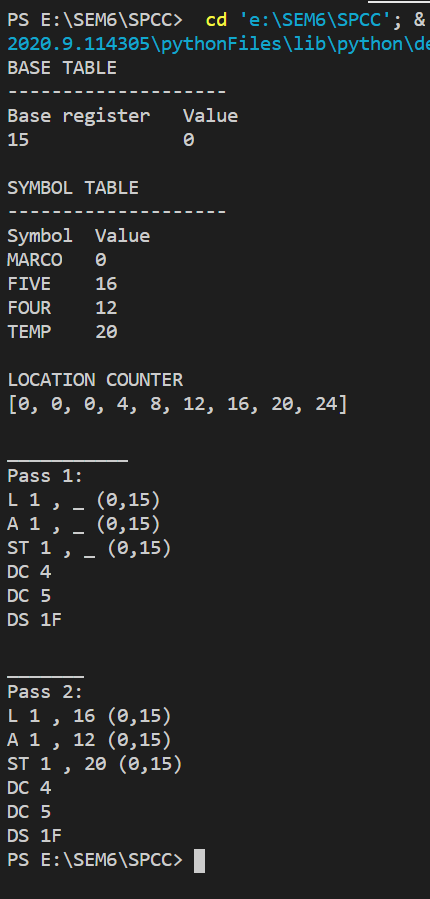
                continue

            else:

                print(word,end=" ")

        print(f"({base\_table['15']},{15})")

## OUTPUT:



CONCLUSION:

A two pass assembler for an IBM 360/370 microprocessor is implemented.