EXPERIMENT 2

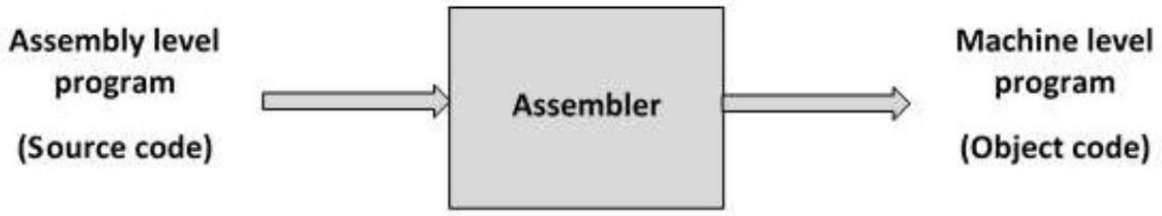
CLASS: TE CMPN A **PID:182027**
NAME: REBECCA DIAS **ROLL NO. : 19**

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.



2. Features of an assembler
   - Translating mnemonic language to its equivalent object code
   - Assigning machine addresses to symbolic labels.
3. 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.
4. 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.

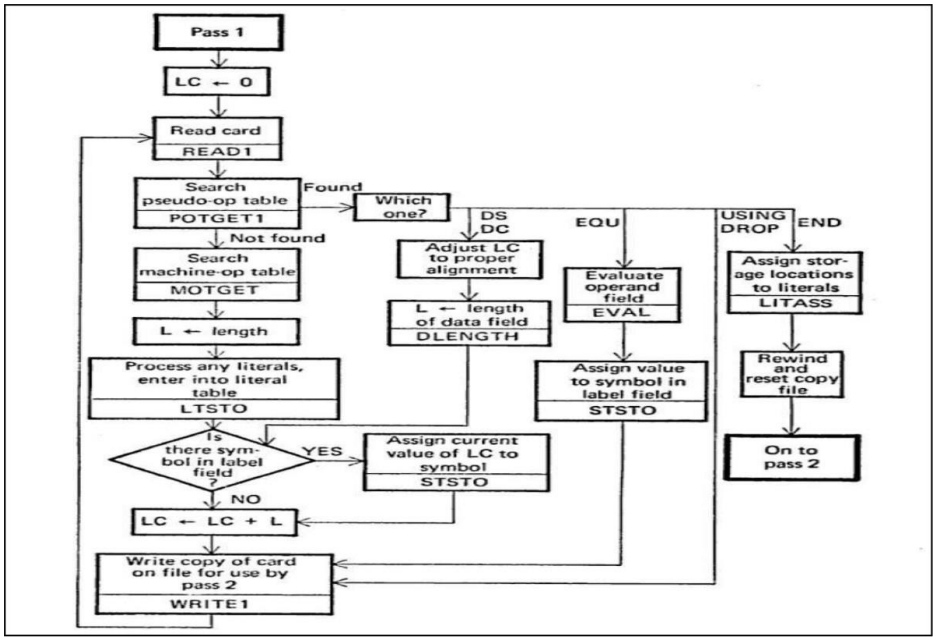
- Literal Table:
  Literal table is used for keeping track of literals that are encountered in the programs.We directly specify the value, literal is used to give a location for the value.Literals are always encountered in the operand field of an instruction.In pass 1, whenever a Literal is defined and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

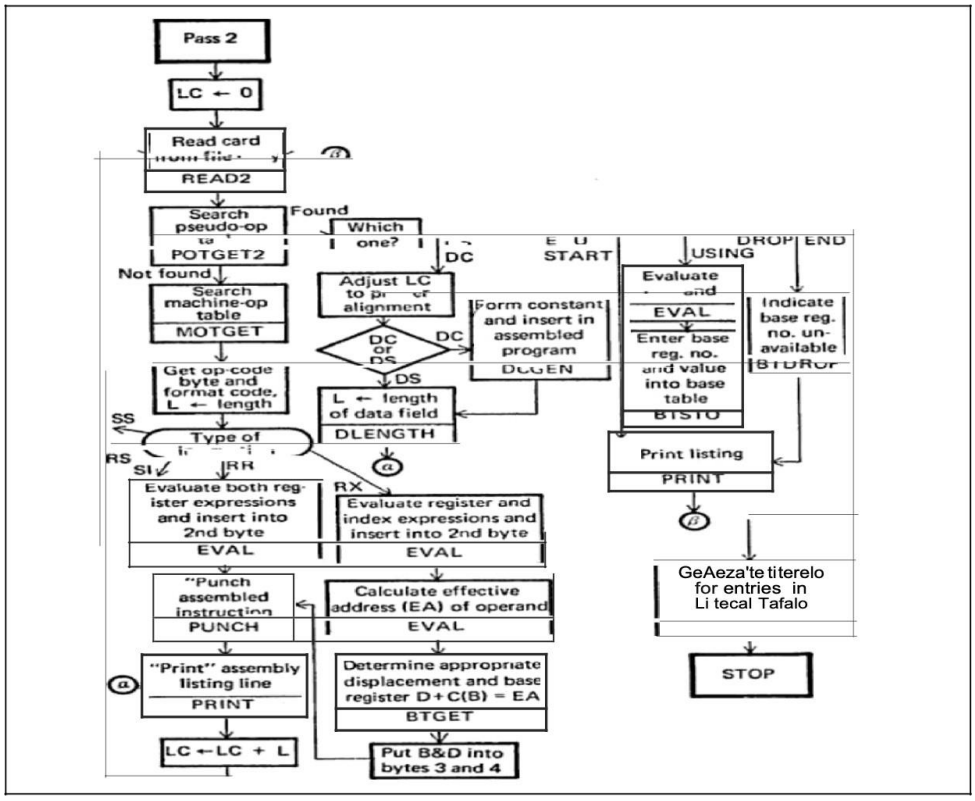
- Pseudo Opcode Table:
  POT is the fixed length table.In pass 1, using Pseudo Opcode, POT is consulted for processing some pseudo opcode like DS, DC, START, END, etc.In pass 2 using Pseudo Opcode, POT is consulted for processing some pseudo opcode like DS,DC,USING DROP.

- 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.

5. Flowchart



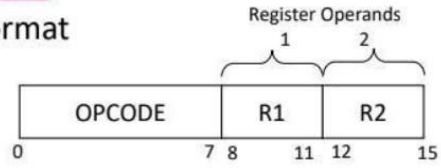


6. IBM360/370instructionformats

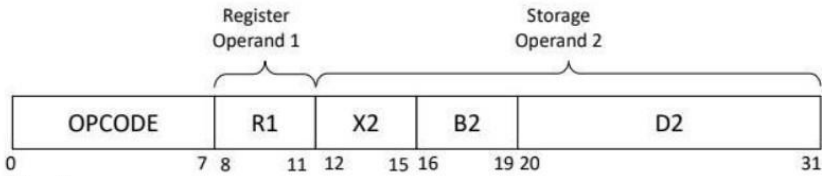
RR format.(two bytes):Generally byte 1 specifies two 4-bit register numbers RX format (four bytes). Bits 0-3 of byte 1 specify either a register number or a modifier; bits 4-7 of byte 1 specify the number of the general register to be used as an index; bytes 2-3 specify a base and displacement.

Instructions

- RR format

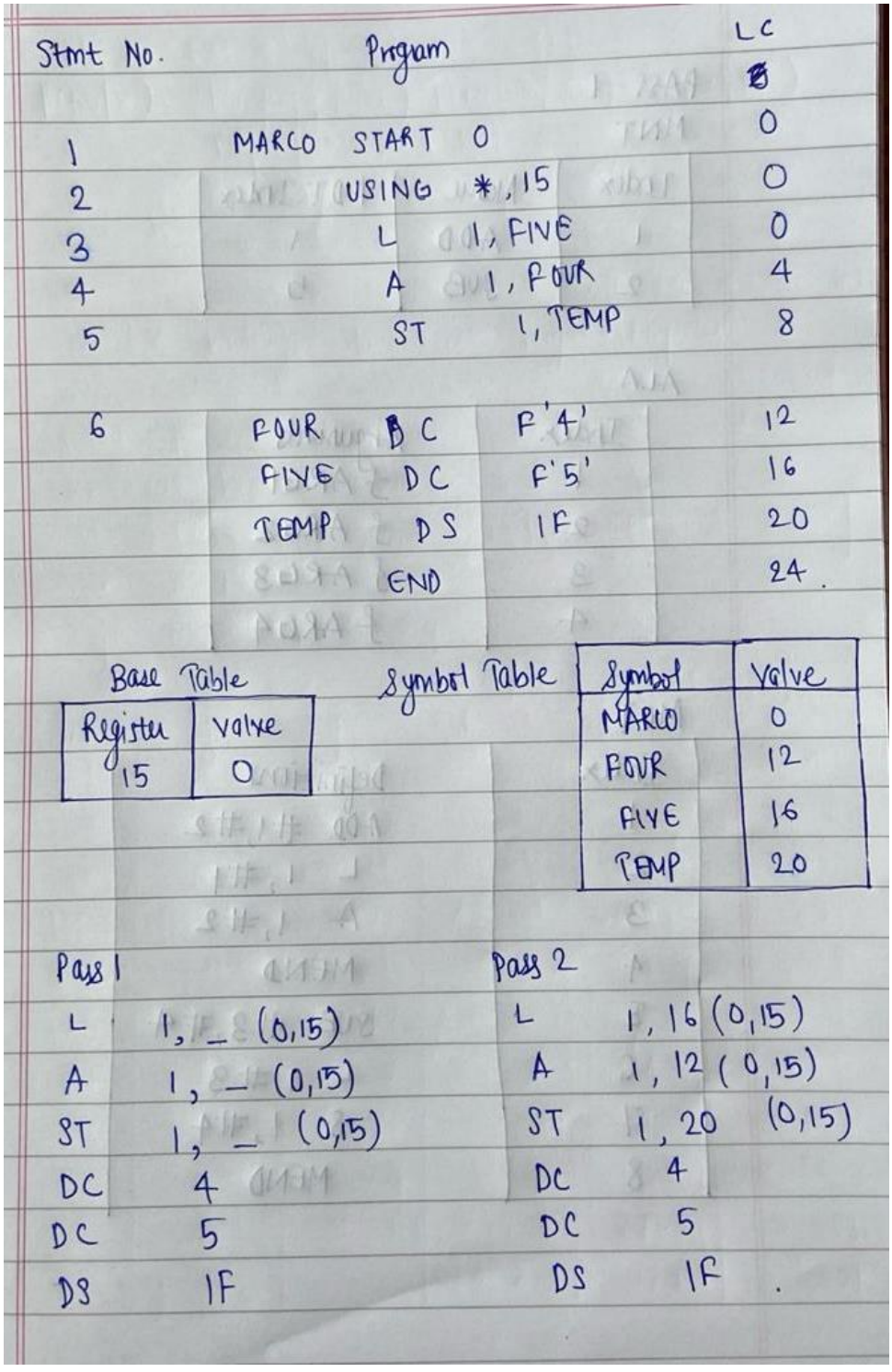


- RX format



displacement.

7. Example



Stmt No.	Program			LC
1	MARCO	START	0	0
2		USING	*,15	0
3		L	1, FIVE	0
4		A	1, FOUR	4
5		ST	1, TEMP	8
6	FOUR	DC	F'4'	12
	FIVE	DC	F'5'	16
	TEMP	DS	1F	20
		END		24

Base Table

Register	Value
15	0

Symbol Table

Symbol	Value
MARCO	0
FOUR	12
FIVE	16
TEMP	20

Pass 1

L	1, _	(0,15)
A	1, _	(0,15)
ST	1, _	(0,15)
DC	4	
DC	5	
DS	1F	

Pass 2

L	1, 16	(0,15)
A	1, 12	(0,15)
ST	1, 20	(0,15)
DC	4	
DC	5	
DS	1F	

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:

```
PS E:\SEM6\SPCC>  cd 'e:\SEM6\SPCC'; &
2020.9.114305\pythonFiles\lib\python\de
BASE TABLE
--------------------
Base register   Value
15              0

SYMBOL TABLE
--------------------
Symbol  Value
MARCO   0
FIVE    16
FOUR    12
TEMP    20

LOCATION COUNTER
[0, 0, 0, 4, 8, 12, 16, 20, 24]


___________
Pass 1:
L 1 , _ (0,15)
A 1 , _ (0,15)
ST 1 , _ (0,15)
DC 4
DC 5
DS 1F


_______
Pass 2:
L 1 , 16 (0,15)
A 1 , 12 (0,15)
ST 1 , 20 (0,15)
DC 4
DC 5
DS 1F
PS E:\SEM6\SPCC>
```

CONCLUSION:

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