SP ass 2:

Roll no: 38 Name: Yash Oswal

1. ALP:

```
     START 500
X  DC 65
     MOVER AREG, Y
     MOVEM BREG, ='11'
     ADD AREG, ='24'
     LTORG
     SUB BREG, ='15'
Y  DC 20
     ORIGIN 800
     LTORG
     BC DOWN
     MOVER AREG, NUM
DOWN  MOVER CREG, NUM
     ADD BREG, ='17'
NUM DS 5
END
```

2. Symbol Table

```
X    500
Y    511
DOWN 804
NUM  808
```

Literal Table

```
='11'     507
='24'     508
='15'     800
='17'     813
```

Pool Table

```
0
2
3
```

3. Intermediate Code:

```
     (AD,01)   (C,500)
500  (DL,02)   (C,65)
501  (IS,04)   (RG,1)(S,1)
503  (IS,05)   (RG,2)(L,1)
505  (IS,01)   (RG,1)(L,2)
507  (AD,05)   (DL,02)(C,1)
508  (AD,05)   (DL,02)(C,2)
509  (IS,02)   (RG,2)(L,3)
511  (DL,02)   (C,20)
512  (AD,03)   (C,800)
800  (AD,05)   (DL,02)(C,1)
801  (IS,07)   (S,2)
802  (IS,04)   (RG,1)(S,3)
804  (IS,04)   (RG,3)(S,3)
806  (IS,01)   (RG,2)(L,4)
808  (DL,01)   (C,5)
813  (AD,02)
```

4. Error Checking: replacing MOVER to MOVE on line 4

```
Checking line 1 for errors
[+] No errors at line 1

Checking line 2 for errors
[+] No errors at line 2

Checking line 3 for errors
[+] No errors at line 3

Checking line 4 for errors
[-] Invalid Instruction at line 4:      MOVE BREG, A
```

5. Source Code:

```
from io import TextIOWrapper

MOT={
        'STOP':('00','IS',0),
        'ADD':('01','IS',2),
        'SUB':('02','IS',2),
        'MUL':('03','IS',2),
        'MOVER':('04','IS',2),
        'MOVEM':('05','IS',2),
        'COMP':('06','IS',2),
        'BC':('07','IS',1),
        'DIV':('08','IS',2),
        'READ':('09','IS',1),
        '#print':('10','IS',1),
        'DEC':('11','IS',1),
        'START':('01','AD',1),
        'END':('AD',0),
        'ORIGIN':('03','AD',1),
        'EQU':('04','AD',2),
        'LTORG':('05','AD',0),
        'DS':('01','DL',1),
        'DC':('02','DL',1),
```

```
        'GT':('00','CC',0)
}

REG={
        'AREG':1,
        'BREG':2,
        'CREG':3,
        'DREG':4
}


class vars():
        LC=0
        ifp=open("tables/inter_code.txt",mode="a")
        ifp.truncate(0)
        lit=open("tables/literal_table.txt","a+")
        lit.truncate(0)
        tmp=open("tables/temp.txt","a+")
        tmp.truncate(0)
        symtab={}
        pooltab=[]
        words=[]
        symindex=0


def END():
        pool=0
        z=0
        vars.ifp.write("\t(AD,02)\n")
        vars.lit.seek(0,0)
        for x in vars.lit:
                if "**" in x:
                        pool+=1
                        if pool==1:
                                vars.pooltab.append(z)
                        y=x.split()
                        vars.tmp.write(y[0]+"\t"+str(vars.LC)+"\n")
                        vars.LC+=1
```

```
                else:
                        vars.tmp.write(x)
                z+=1
        vars.lit.truncate(0)
        vars.tmp.seek(0,0)
        for x in vars.tmp:
                vars.lit.write(x)
        vars.tmp.truncate(0)

def LTORG():
        pool=0
        z=0
        vars.lit.seek(0,0)
        x=vars.lit.readlines()
        i=0
        while(i<len(x)):
                f=[]
                if("**" in x[i]):
                        j=0
                        pool+=1
                        if pool==1:
                                vars.pooltab.append(z)
                        while(x[i][j]!="'"):
                                j+=1
                        j+=1
                        while(x[i][j]!="'"):
                                f.append(x[i][j])
                                j+=1
                        if(i!=len(x)-1):
                                vars.ifp.write("\t(AD,05)\t(DL,02)(C,"+str(f[0])+")\n")
                                y=x[i].split()
                                vars.tmp.write(y[0]+"\t"+str(vars.LC)+"\n")
                                vars.LC+=1
                                vars.ifp.write(str(vars.LC))
                        else:
                                vars.ifp.write("\t(AD,05)\t(DL,02)(C,"+str(f[0])+")\n")
                                y=x[i].split()
```

```
                        vars.tmp.write(y[0]+"\t"+str(vars.LC)+"\n")
                        vars.LC+=1
                else:
                        vars.tmp.write(x[i])
                z+=1
                i+=1
        vars.lit.truncate(0)
        vars.tmp.seek(0,0)
        for x in vars.tmp:
                vars.lit.write(x)
        vars.tmp.truncate(0)

def ORIGIN(addr):
        vars.ifp.write("\t(AD,03)\t(C,"+str(addr)+")\n")
        vars.LC =int(addr)

def DS(size):
        vars.ifp.write("\t(DL,01)\t(C,"+size+")\n")
        vars.LC=vars.LC+int(size)

def DC(value):
        vars.ifp.write("\t(DL,02)\t(C,"+value+")\n")
        vars.LC+=1

def OTHERS(KEY,k):
        z=MOT[KEY]
        vars.ifp.write("\t("+z[1]+","+z[0]+")\t")
        i=0
        y=z[-1]
        for i in range(1,y+1):
                vars.words[k+i]=vars.words[k+i].replace(",","")
                if(vars.words[k+i] in REG.keys()):
                        vars.ifp.write("(RG,"+str(REG[vars.words[k+i]])+")")
                elif("=" in vars.words[k+i]):
                        vars.lit.seek(0,2)
                        vars.lit.write(vars.words[k+i]+"\t**\n")
                        vars.lit.seek(0,0)
```

```
                x=vars.lit.readlines()
                vars.ifp.write("(L,"+str(len(x))+")")
            else:
                if(vars.words[k+i] not in vars.symtab.keys()):
                    vars.symtab[vars.words[k+i]]=("**",vars.symindex)
                    vars.ifp.write("(S,"+str(vars.symindex)+")")
                    vars.symindex+=1
                else:
                    w=vars.symtab[vars.words[k+i]]
                    vars.ifp.write("(S,"+str(w[-1])+")")
        vars.ifp.write("\n")
        vars.LC+=z[-1]

def detect_mn(k):
    if(vars.words[k]=="START"):
        vars.LC=int(vars.words[1])
        vars.ifp.write("\t(AD,01)\t(C,"+str(vars.LC)+')\n')
    elif(vars.words[k]=='END'):
        END()
    elif(vars.words[k]=="LTORG"):
        LTORG()
    elif(vars.words[k]=="ORIGIN"):
        ORIGIN(vars.words[k+1])
    elif(vars.words[k]=="DS"):
        DS(vars.words[k+1])
    elif(vars.words[k]=="DC"):
        DC(vars.words[k+1])
    else:
        OTHERS(vars.words[k],k)

def pass_one(alp:TextIOWrapper):
    lc=1
    for line in alp:
        error_handler(line,lc)
        lc+=1
        vars.words=line.split()
        if (vars.LC>0):
```

```
                vars.ifp.write(str(vars.LC))
        k=0
        if vars.words[0] in MOT.keys():
                val = MOT[vars.words[0]]
                detect_mn(k)
        else:
                if vars.words[k] not in vars.symtab.keys():
                        vars.symtab[vars.words[k]]=(vars.LC,vars.symindex)
                        #ifp.write("\t(S,"+str(symindex)+")\t")
                        vars.symindex+=1
                else:
                        x = vars.symtab[vars.words[k]]
                        if x[0] == "**":
                                #print("yes")
                                vars.symtab[vars.words[k]] = (vars.LC,x[1])
                k=1
                detect_mn(k)
    vars.ifp.close()
    vars.lit.close()
    vars.tmp.close()
    sym=open("tables/symbol_table.txt","a+")
    sym.truncate(0)
    for x in vars.symtab:
        sym.write(x+"\t"+str(vars.symtab[x][0])+"\n")
    sym.close()
    pool=open("tables/pool_table.txt","a+")
    pool.truncate(0)
    for x in vars.pooltab:
        pool.write(str(x)+"\n")
    pool.close()


def error_handler(line:str,lc:int):
    print(f"\nChecking line {lc} for errors")
    l=line.split()
    if l[0] in MOT.keys():
```

```
                op = MOT[l[0]]
                if (len(l)-1) < op[-1]:
                        print(f"[-] Error at line {lc}: Less operands than expcted")
                        exit(-1)
                elif (len(l)-1) > op[-1]:
                        print(f"[-] Error at line {lc}: More operands than expcted")
                        exit(-1)
                else:
                        print(f"[+] No errors at line {lc}")
        elif l[1] in MOT.keys():
                op = MOT[l[1]]
                if (len(l)-2) < op[-1]:
                        print(f"[-] Error at line {lc}: Less operands than expcted")
                        exit(-1)
                elif (len(l)-2) > op[-1]:
                        print(f"[-] Error at line {lc}: More operands than expcted")
                        exit(-1)
                else:
                        print(f"[+] No errors at line {lc}")
        else:
                print(f"[-] Invalid Instruction at line {lc}: {line}")
                exit(-1)


def getFile():
        fileName=input("Enter file name: ")
        alp = open(fileName,'r')
        return alp

if __name__=='__main__':
        alp=getFile()
        pass_one(alp)
```