**Name – Aumkar Gadekar**

**UID – 2017130023**

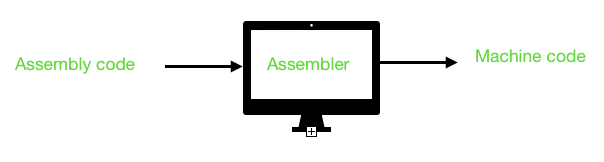
**Batch B**

**Experiment 2**

**Aim : To implement a two pass SIC assembler**

**Theory:**

# Assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.

data:image/svg+xml;charset=utf-8,%3csvg%20height=%22162px%22%20width=%22604px%22%20xmlns=%22http://www.w3.org/2000/svg%22%20version=%221.1%22/%3e

It generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code. Now, if assembler do all this work in one scan then it is called single pass assembler, otherwise if it does in multiple scans then called multiple pass assembler. Here assembler divide these tasks in two passes:

* **Pass-1:**
  1. Define symbols and literals and remember them in symbol table and literal table respectively.
  2. Keep track of location counter
  3. Process pseudo-operations
* **Pass-2:**
  1. Generate object code by converting symbolic op-code into respective numeric op-code
  2. Generate data for literals and look for values of symbols

**Input file:**

Assembler.txt​

COPY START 1000

FIRST STL RETADR

CLOOP JSUB RDREC

LDA LENGTH

COMP ZERO

JEQ ENDFIL

JSUB WRREC

J CLOOP

ENDFIL LDA EOF

STA BUFFER

LDA THREE

STA LENGTH

JSUB WRREC

LDL RETADR

RSUB

EOF BYTE C'EOF'

THREE WORD 3

ZERO WORD 0

RETADR RESW 1

LENGTH RESW 1

BUFFER RESB 4096

RDREC LDX ZERO

LDA ZERO

RLOOP TD INPUT

JEQ RLOOP

RD INPUT

COMP ZERO

JEQ EXIT

STCH BUFFER,X

TIX MAXLEN

JLT RLOOP

EXIT STX LENGTH

RSUB

INPUT BYTE X'F1'

MAXLEN WORD 4096

WRREC LDX ZERO

WLOOP TD OUTPUT

JEQ WLOOP

LDCH BUFFER,X

WD OUTPUT

TIX LENGTH

JLT WLOOP

RSUB

OUTPUT BYTE X'06'

END FIRST

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

typedef struct inst

{

char label[20];

char operand[20];

char opcode[20];

int count;

int loc;

}inst;

typedef struct symbol

{

char label[20];

int loc;

}symbol;

int main()

{

int label\_count=0;

char file\_name[25];

FILE \*fp,\*fp1,\*fp2,\*fp3,\*fp4,\*fp5,\*fp6;

char line[256];

fp = fopen("Ass1.txt", "r"); // read mode

fp1= fopen("Ass1.txt", "r"); // read mode

fp2= fopen("symtab.txt", "w");

fp3= fopen("optab.txt", "w");

fp5= fopen("optab.txt", "r");

fp6= fopen("output.txt","w");

char label[20],opcode[20],operand[20];

inst Instruct[300];

symbol SYMTAB[50];

fscanf(fp, "%s %s %s", label, opcode, operand);

fscanf(fp1, "%s %s %s", label, opcode, operand);

if(strcmp(opcode,"START")==0)

{

int loc=atoi(operand);

fgets(line, sizeof(line), fp);

int k=0;

while (fgets(line, sizeof(line), fp))

{

int i=0;

int t\_count=1;

while(line[i]=='\t')

i+=1;

while(i<strlen(line))

{

if(line[i]=='\t')

t\_count+=1;

i++;

}

if(t\_count==3)

{

fscanf(fp1, "%s %s %s", &label, &opcode, &operand);

fprintf(fp2, "%d\t%s\n",loc,label);

}

else if(t\_count==2)

{

fscanf(fp1, "%s %s", &opcode, &operand);

}

else

{

fscanf(fp1, "%s", &opcode);

}

fprintf(fp3,"%d\t%s",loc,line);

int flag=0;

int o;

char otp[20];

Instruct[k].loc=loc;

if(t\_count!=1)

strcpy(Instruct[k].operand,operand);

strcpy(Instruct[k].opcode,opcode);

Instruct[k].count=t\_count;

if(t\_count==3)

{

strcpy(SYMTAB[label\_count].label,label);

SYMTAB[label\_count].loc=loc;

label\_count+=1;

strcpy(Instruct[k].label,label);

}

FILE \*fp4;

fp4= fopen("opcode.txt", "r");

while(!feof(fp4))

{

fscanf(fp4,"%s %d",otp,&o);

if(strcmp(opcode,otp)==0)

{

loc=loc+3;

flag=1;

break;

}

}

if(flag==0)

{

if(strcmp(opcode,"WORD")==0)

{

loc=loc+3;

}

else if(strcmp(opcode,"RESW")==0)

{

int opr=atoi(operand);

loc=loc+(3\*opr);

}

else if(strcmp(opcode,"BYTE")==0)

{

if(operand[0]=='X')

loc=loc+1;

else

{

int len=strlen(operand)-2;

loc=loc+len;

}

}

else if(strcmp(opcode,"RESB")==0)

{

int opr=atoi(operand);

loc=loc+opr;

}

}

fclose(fp4);

k++;

}

printf("PASS ONE IMPLEMENTED\n\n");

printf("SYMBOL TABLE\n");

for(int p=0;p<label\_count;p++)

{

printf("%d\t%s\n",SYMTAB[p].loc,SYMTAB[p].label);

}

printf("\n");

printf("PASS TWO\n");

int z=0;

while(z<k)

{

char out1[20];

char out2[20];

char o[5];

char otp[10];

fp4= fopen("opcode.txt", "r");

int f=0;

while(!feof(fp4))

{

fscanf(fp4,"%s %s",otp,o);

if(strcmp(Instruct[z].opcode,otp)==0)

{

f=1;

strcpy(out1,o);

break;

}

}

if(f==0)

{

if(strcmp(Instruct[z].opcode,"BYTE")==0)

{

int w=2,cn=0;

while(Instruct[z].operand[w]!='\'')

{

out1[cn]=Instruct[z].operand[w];

cn+=1;

w+=1;

}

if(strcmp(out1,"EOF")==0)

strcpy(out1,"454F46");

strcpy(out2,"");

}

else{

strcpy(out1,"--");

strcpy(out2,"----");

}

}

else{

for(int t=0;t<label\_count;t++)

{

if(strcmp(SYMTAB[t].label,Instruct[z].operand)==0)

{

itoa(SYMTAB[t].loc,out2,10);

break;

}

}

}

printf("%s%s%s\n",line,out1,out2);

fprintf(fp6, "%s%s\n",out1,out2);

z++;

}

}

else{

printf("ERROR ! FIRST OPERAND SHOULD ALWAYS BE START");

}

fclose(fp);

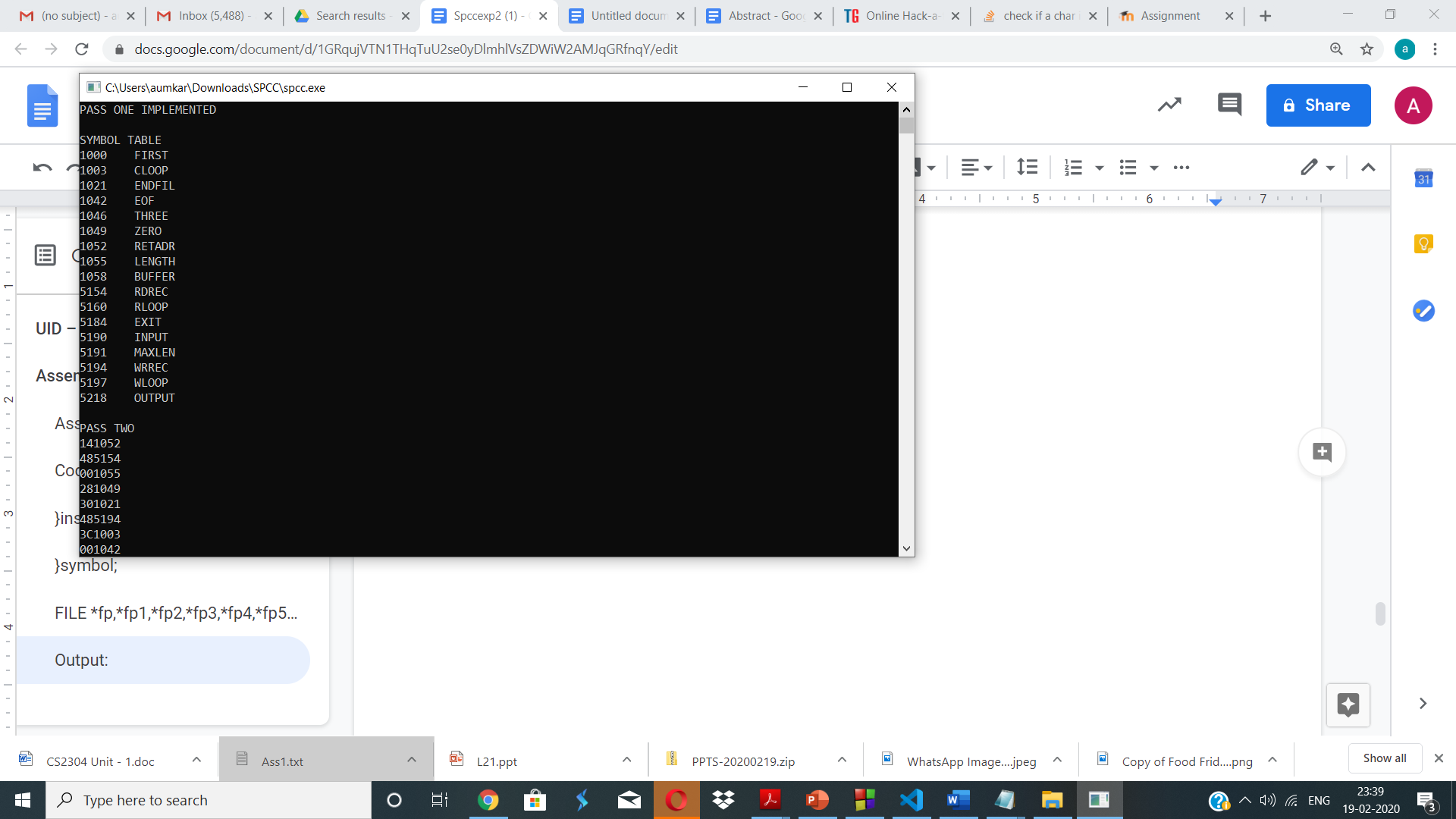
return 0;

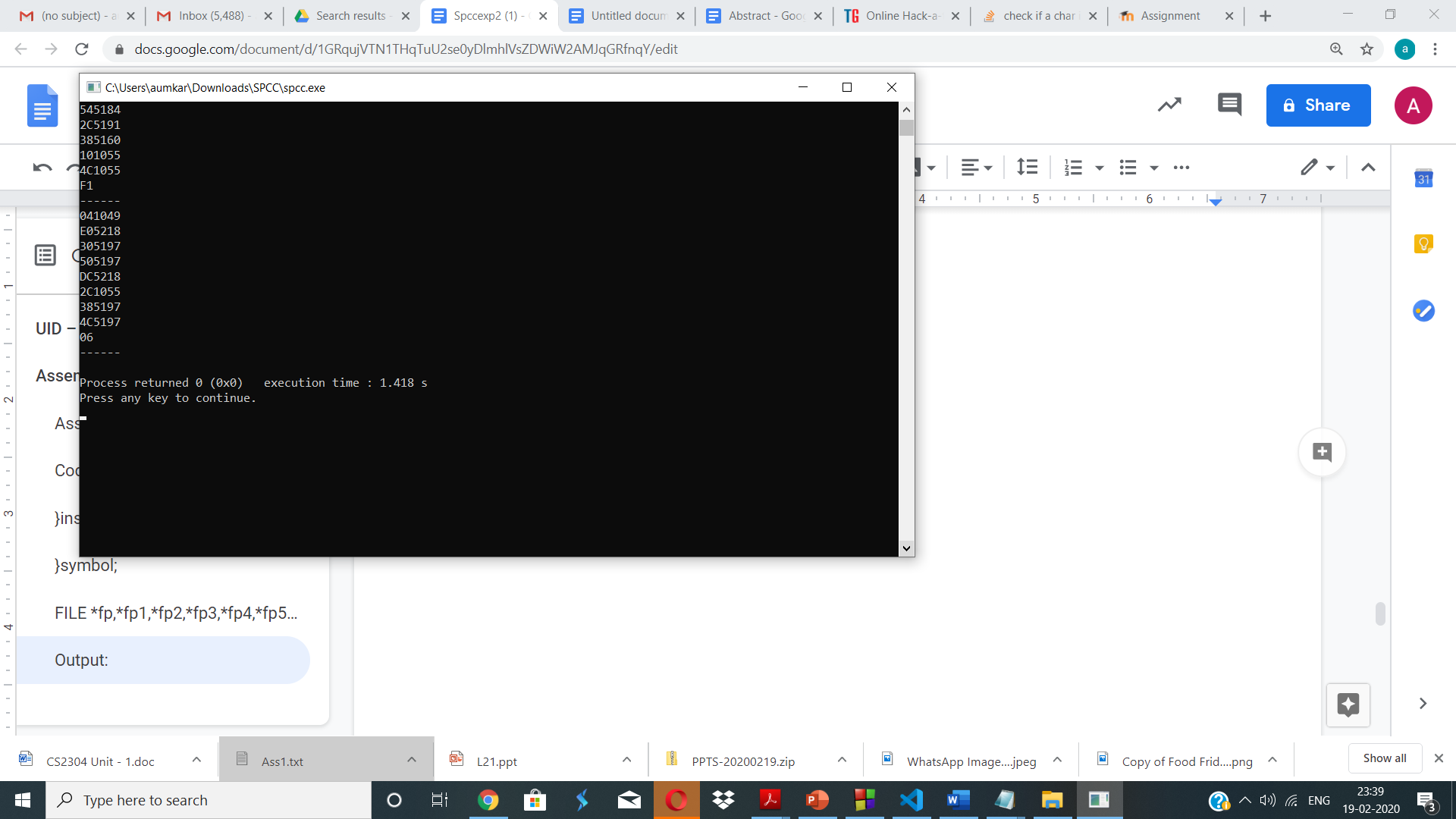
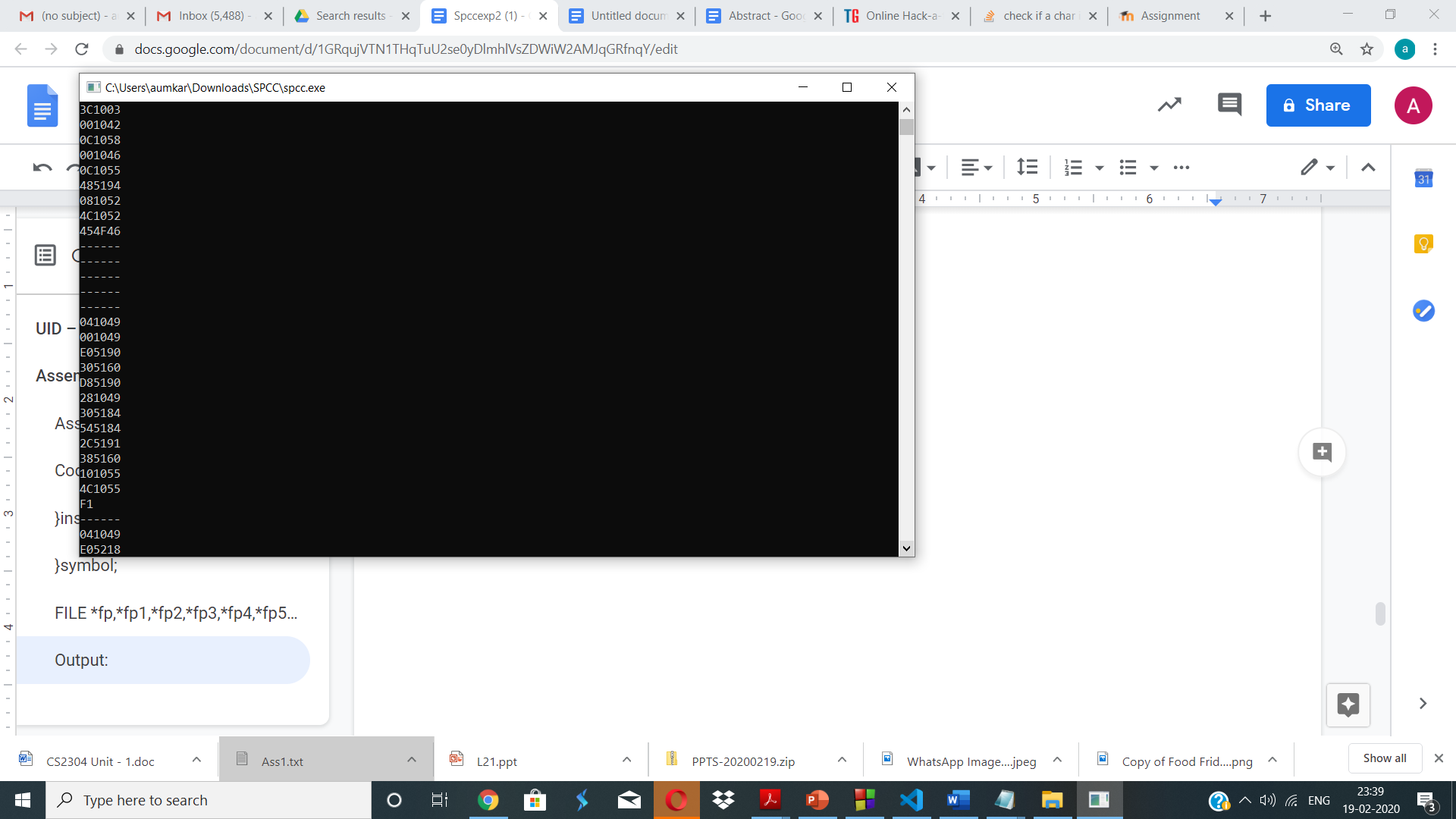
}

**Procedure:**

* Read the assembler.txt file, which contains the SIC assembly code. Removed all whitespaces and empty lines.
* In the first pass, the location\_counter is filled. The symbol table is also constructed.
* In the second pass, the output table is printed. Simultaneously, object code is evaluated before printing.
* No assumptions made.

**Output:**

****

****

**Conclusion:**

* The basic knowledge of how assemblers work was obtained i.e how object code is generated using a two pass assembler.
* This knowledge was applied by using python language and object code was generated for a given assembly code.