Write a program to implement two pass assembler.

ASM CODE:

```
START 100
L1  MOVER AREG,=5
    MOVEM BREG X
    SUB   AREG,=2
    LTORG
    MOVER AREG Y
    BC any,L1
    ADD CREG,4
X   DC  5
Y   DS  2
    END
```

SOURCE CODE:

```
#include <bits/stdc++.h>

struct MOTtable
{
char Mnemonic[6];
int Class;
char Opcode[3];
};

static struct MOTtable MOT[28] = {
{"STOP", 1, "00"},
{"ADD", 1, "01"},
{"SUB", 1, "02"},
{"MULT", 1, "03"},
{"MOVER", 1, "04"},
{"MOVEM", 1, "05"},
{"COMP", 1, "06"},
{"BC", 1, "07"},
{"DIV", 1, "08"},
{"READ", 1, "09"},
{"PRINT", 1, "10"},

{"START", 3, "01"},
{"END", 3, "02"},
{"ORIGN", 3, "03"},
{"EQU", 3, "04"},
{"LTORG", 3, "05"},

{"DS", 2, "01"},
{"DC", 2, "02"},

{"AREG", 4, "01"},
{"BREG", 4, "02"},
{"CREG", 4, "03"},

{"EQ", 5, "01"},
{"LT", 5, "02"},
{"GT", 5, "03"},
{"LE", 5, "04"},
{"GE", 5, "05"},
{"NE", 5, "06"},
{"ANY", 5, "07"}};

struct symboltable
{
char Symbol[8];
int Address;
int Size;
} ST[20];

struct intermediatecode
{
int LC;
int Code1, Type1;
int Code2, Type2;
int Code3, Type3;
} IC[30];

int nMOT = 28; // Number of entries in MOT
int LC = 0; // Location counter
int iST = 0; // Index of next entry in Symbol Table
int iIC = 0; // Index of next entry in intermediate code table/
char s1[8], s2[8], s3[8], label[8];
int tokencount; // total number of words in a statement

int searchST(char symbol[])
{
int i;
for (i = 0; i < iST; i++)
if (strcmp(ST[i].Symbol, symbol) == 0)
```

```
return (i);

return (-1);
}

int searchMOT(char symbol[])
{
int i;
for (i = 0; i < nMOT; i++)
if (strcmp(MOT[i].Mnemonic, symbol) == 0)
return (i);

return (-1);
}

int insertST(char symbol[], int address, int size)
{
strcpy(ST[iST].Symbol, symbol);
ST[iST].Address = address;
ST[iST].Size = size;
iST++;

return (iST - 1);
}

void imperative() // Handle an executable statement
{
int index;
index = searchMOT(s1);
IC[iIC].Type1 = IC[iIC].Type2 = IC[iIC].Type3 = 0; // intialize
IC[iIC].LC = LC;
IC[iIC].Code1 = index;
IC[iIC].Type1 = MOT[index].Class;
LC = LC + 1;
if (tokencount > 1)
{
index = searchMOT(s2);
if (index != -1)
{
IC[iIC].Code2 = index;
IC[iIC].Type2 = MOT[index].Class;
}
else
{ // It is a variable
index = searchST(s2);
if (index == -1)
index = insertST(s2, 0, 0);
IC[iIC].Code2 = index;
IC[iIC].Type2 = 7; // VALUE 7 IS FOR VARIABLES
}
}
if (tokencount > 2)
{
{
index = searchST(s3);
if (index == -1)
index = insertST(s3, 0, 0);
IC[iIC].Code3 = index;
IC[iIC].Type3 = 7; // VALUE 7 IS FOR VARIABLES
}
}
iIC++;
}

void DC() // Handle declaration statement DC
{
int index;
index = searchMOT(s1);
IC[iIC].Type1 = IC[iIC].Type2 = IC[iIC].Type3 = 0; // intialize
IC[iIC].LC = LC;
IC[iIC].Code1 = index;
IC[iIC].Type1 = MOT[index].Class;
IC[iIC].Type2 = 6; // 6 IS TYPE FOR CONSTANTS
IC[iIC].Code2 = atoi(s2);
index = searchST(label);
if (index == -1)
index = insertST(label, 0, 0);
ST[index].Address = LC;
ST[index].Size = 1;
LC = LC + 1;
iIC++;
}
void DS() // Handle declaration statement DS
{
int index;
index = searchMOT(s1);
IC[iIC].Type1 = IC[iIC].Type2 = IC[iIC].Type3 = 0; // intialize
IC[iIC].LC = LC;
```

```
IC[iIC].Code1 = index;
IC[iIC].Type1 = MOT[index].Class;
IC[iIC].Type2 = 6; // 6 IS TYPE FOR CONSTANTS
IC[iIC].Code2 = atoi(s2);
index = searchST(label);
if (index == -1)
index = insertST(label, 0, 0);
ST[index].Address = LC;
ST[index].Size = atoi(s2);
LC = LC + atoi(s2);
iIC++;
}
void START() // Handle START directive
{
int index;
index = searchMOT(s1);
IC[iIC].Type1 = IC[iIC].Type2 = IC[iIC].Type3 = 0; // intialize
IC[iIC].LC = LC;
IC[iIC].Code1 = index;
IC[iIC].Type1 = MOT[index].Class;
IC[iIC].Type2 = 6; // 6 IS TYPE FOR CONSTANTS
IC[iIC].Code2 = atoi(s2);
LC = atoi(s2);
iIC++;
}
void declaration() // Handle a declaration statement
{
if (strcmp(s1, "DC") == 0)
{
DC();
return;
}
if (strcmp(s1, "DS") == 0)
DS();
}
void directive() // Handle an assembler directive
{
if (strcmp(s1, "START") == 0)
{
START();
return;
}
}
```

```
void intermediate() // Display intermediate code
{
int i;
char decode[9][3] = {" ", "IS", "DL", "AD", "RG", "CC", "C", "S"};
printf("\n\nIntermediate Code :");
for (i = 0; i < iIC; i++)
{
printf("\n%3d) (%s,%2s)", IC[i].LC, decode[IC[i].Type1], MOT[IC[i].Code1].Opcode);
if (IC[i].Type2 != 0)
{
if (IC[i].Type2 < 6)
printf(" (%s,%2s)", decode[IC[i].Type2], MOT[IC[i].Code2].Opcode);
else
printf(" (%s,%2d)", decode[IC[i].Type2], IC[i].Code2);
}
if (IC[i].Type3 != 0)
printf(" (%s,%2d)", decode[IC[i].Type3], IC[i].Code3);
}
}

void print_symbol() // Display symbol table
{
int i;
printf("\n*******symbol table ***********\n");
for (i = 0; i < iST; i++)
printf("\n%10s %3d %3d", ST[i].Symbol, ST[i].Address, ST[i].Size);
}
void print_opcode() // Display opcode table
{
int i;
printf("\nopcode table *************");
for (i = 0; i < nMOT; i++)
if (MOT[i].Class == 1)
printf("\n%6s %2s", MOT[i].Mnemonic, MOT[i].Opcode);
}

void mcode() // Generate machine code
{
int i;
printf("\n\nMachine Code :");
for (i = 0; i < iIC; i++)
{

if (IC[i].Type1 == 1)
{
printf("\n%3d) %s ", IC[i].LC, MOT[IC[i].Code1].Opcode);
if (IC[i].Type2 == 0)
printf("00 000");

else if (IC[i].Type2 > 6) // No Register Operand
printf("00 %3d", ST[IC[i].Code2].Address);
else
{
printf("%2s ", MOT[IC[i].Code2].Opcode);
if (IC[i].Type3 == 7)
printf("%3d", ST[IC[i].Code3].Address);
}
}
else if (IC[i].Type1 == 2 && strcmp(MOT[IC[i].Code1].Mnemonic, "DC") == 0)
{
printf("\n%3d) ", IC[i].LC);
printf("00 00 %3d", IC[i].Code2);
}
}
}

int main()
{
char nextline[80];
int len, i, j, temp, errortype;
FILE *ptr1;
ptr1 = fopen("source.asm", "r");

while (!feof(ptr1))
{
// Read a line of assembly program and remove special characters
i = 0;
nextline[i] = fgetc(ptr1);
while (nextline[i] != '\n' && nextline[i] != EOF)
{
if (!isalnum(nextline[i]))
nextline[i] = ' ';
else
nextline[i] = toupper(nextline[i]);
i++;
nextline[i] = fgetc(ptr1);
}
nextline[i] = '\0';
```

```
sscanf(nextline, "%s", s1); // read from the nextline in s1

if (strcmp(s1, "END") == 0) // if the nextline is an END statement
break;

// if the nextline contains a label
if (searchMOT(s1) == -1)
{
if (searchST(s1) == -1)
insertST(s1, LC, 0);
// separate opcode and operands
tokencount = sscanf(nextline, "%s%s%s%s", label, s1, s2, s3);
tokencount--;
}
else
// separate opcode and operands
tokencount = sscanf(nextline, "%s%s%s", s1, s2, s3);

if (tokencount == 0) // blank line
continue; // goto the beginning of the loop

i = searchMOT(s1);

if (i == -1)
{
printf("\nWrong Opcode .... %s", s1);

continue;
}
switch (MOT[i].Class)
{
case 1:
imperative();
break;
case 2:
declaration();
break;
case 3:
directive();
break;
default:
printf("\nWrong opcode ...%s", s1);
break;
}
```

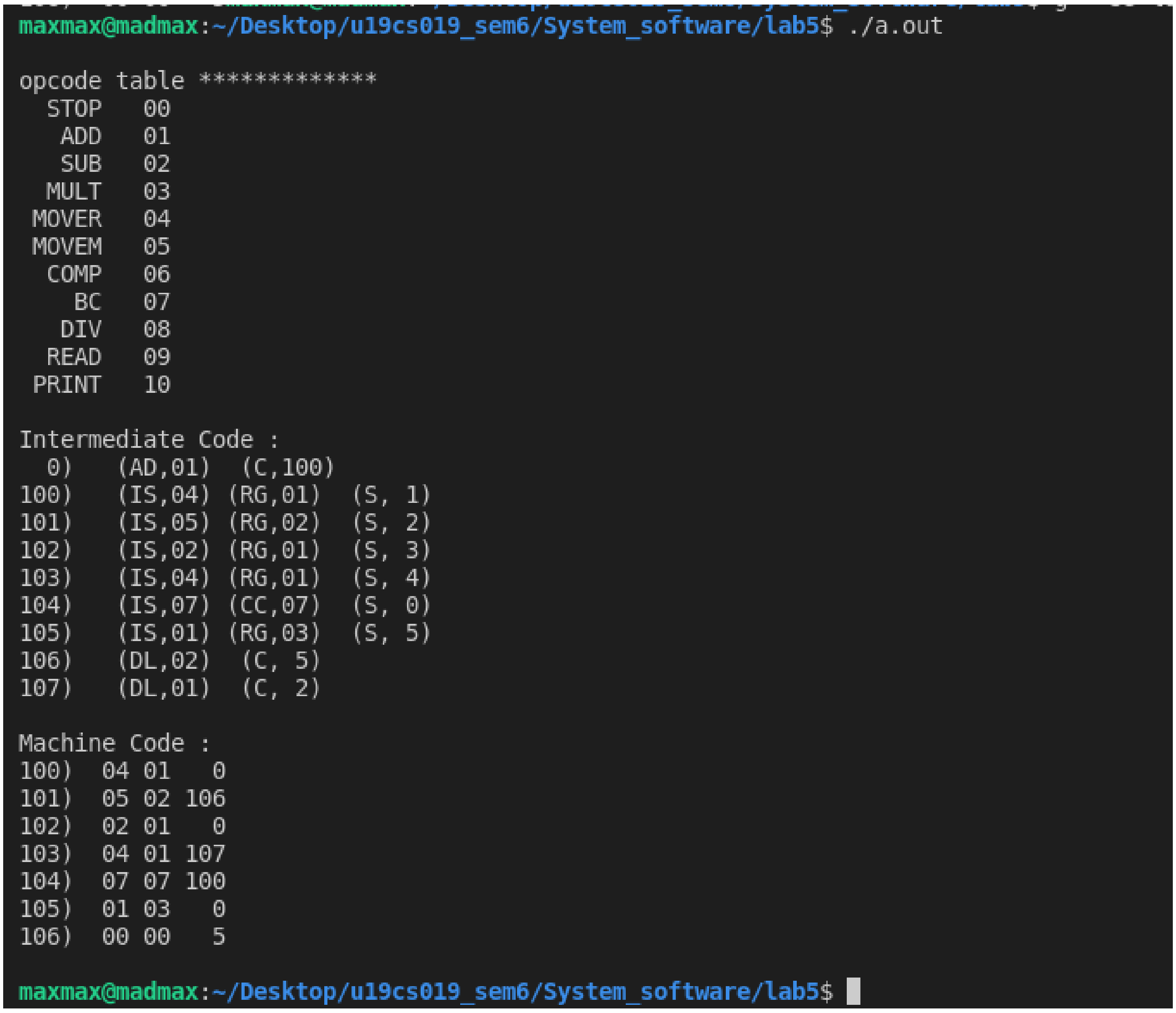
```
}

print_opcode();

intermediate();

mcode();
printf("\n\n");

return 0;
}
```



```
maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab5$ ./a.out

opcode table *************
  STOP   00
   ADD   01
   SUB   02
  MULT   03
 MOVER   04
 MOVEM   05
  COMP   06
    BC   07
   DIV   08
  READ   09
 PRINT   10

Intermediate Code :
  0)   (AD,01)  (C,100)
100)   (IS,04) (RG,01)  (S, 1)
101)   (IS,05) (RG,02)  (S, 2)
102)   (IS,02) (RG,01)  (S, 3)
103)   (IS,04) (RG,01)  (S, 4)
104)   (IS,07) (CC,07)  (S, 0)
105)   (IS,01) (RG,03)  (S, 5)
106)   (DL,02)  (C, 5)
107)   (DL,01)  (C, 2)

Machine Code :
100)  04 01   0
101)  05 02 106
102)  02 01   0
103)  04 01 107
104)  07 07 100
105)  01 03   0
106)  00 00   5

maxmax@madmax:~/Desktop/u19cs019_sem6/System_software/lab5$
```