```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

enum State {LOAD=1, ADD=2, STORE=3, SUB=4, IN=5, OUT=6, END=7, JMP=8, SKIPZ=9};

void printVaribles();

//Creating Instruction structure
typedef struct
{
    int opCode;//to store opcode
    int deviceOrAddress;//address of the operation
}Instruction;

//Global variable
int CODESIZE = 13;

//Tiny Machine Architecture variables
int pc = 10;// for program counter
int ir = 0;//for instruction register
int mar = 0;//to store address of memory
int dataMemory[9]={0,0,0,0,0,0,0,0,0};
int mdr = 0;//to store data of memory
int ac = 0;//accumulator

//function for print the tinny architecuter varibles
void printVaribles()
{
    printf(" PC: %d | A: %d | MEM: [", pc, ac);

    //print data in memory
    for (int i = 0; i < 9; i++)

    {

        printf("%d,", dataMemory[i]);

    }

    printf("] \n");

}

//Function used to count the amount of lines in text asmCodefile
int getNumberOfLineInFile(FILE *asmCodefile)
{
    char asmCodeBuffer[10];
    int count = 0;
    if (asmCodefile == NULL)
    {
        printf("ASM code file is not opening...\n");
        exit(0);
```

```
    }

    //caluclate the number of line
    while (fgets(asmCodeBuffer, 10, asmCodefile) != NULL)
    {
        count++;//increament count
    }

    fclose(asmCodefile);

    return count;//return count

}

//Function used to parse instruction into machine code

void tinyMachineSimulator(int opCode, int b)
{
    switch (opCode)
    {
        case LOAD:
            printf(" /* Loading from address [%d]... */ \n", b);

            ir = b;
            mar = ir;
            mdr = dataMemory[mar];
            ac= mdr;

            //Print value in tinny macine varibles
            printVaribles();

            pc += 1;

            printf(" /* PC <- PC + 1 */\n");
            printf("/* PC <- PC + 1 */ \n");
            printf("/* MAR <- IR.ADDR */ \n");
            printf("/* MDR <- MEM[MAR] */ \n");
            printf("/* A <- MDR */ \n");
            break;

        case ADD:
            printf(" /Adding accumulator and value obtain from address [%d]/ \n",
b);
            ir = b;
            mar = ir;
            mdr = dataMemory[mar];
            ac += mdr;

            //Print value in tinny macine varibles
            printVaribles();
            pc += 1;
            break;

        case STORE:

            printf(" /* storing accumulator to memory location 0 */ \n");

            mdr = ac;
            ir = b;
```

```
            mar = ir;
            dataMemory[mar] = mdr;

            //Print value in tinny macine varibles
            printVaribles();
            pc += 1;
            break;

        case SUB:
            printf(" /* Subtracting memory address value [%d] from accumulator*/ \
n", b);

            ir = b;
            mar= ir;
            mdr = dataMemory[mar];
            ac -= mdr;

            //Print value in tinny macine varibles
            printVaribles();
            pc += 1;
            break;

        case IN:
            printf(" /Please input value:/ \n");
            scanf("%d", &ac);

            //Print value in tinny macine varibles
            printVaribles();
            pc += 1;
            break;

        case OUT:
            printf(" /*Accumulator current value = %d */ \n", ac);

            //Print value in tinny macine varibles
            printVaribles();
            pc += 1;
            break;

        case END:
            printf(" Program complete \n");
            exit(1);

        case JMP:
            // *Jump to address
            printf(" /Setting program counter to %d/ \n", b);
            pc = b;

            //Print value in tinny macine varibles
            printVaribles();
            break;

        case SKIPZ:
            //Check if accumulator is 0,
            printf(" /Skipping the next instruction/ \n");

            if (ac == 0)//if it is skip next instruction
            {
                pc += 2;//increament PC by 2
```

```
            }
            else//otherwise
            {
                pc += 1; //increament PC by 1
            }

            //Print value in tinny macine varibles
            printVaribles();
            break;
        default:
            printf(" /There was an error parsing that opcode! Exiting program/ \
n");
            exit(0);
    }
}

int main(int argc, char *argv[])
{

    //This is where we get the number of lines in the code
    CODESIZE = getNumberOfLineInFile(fopen(argv[1], "r"));

    //This creates array for storing instructions
    Instruction programMemory[CODESIZE];

    //Reading a asmCodefile line by line
    FILE *asmCodefile = fopen(argv[1], "r");
    char asmCodefileBuffer[10];

    if (asmCodefile == NULL)

    {
        printf("ASM code file is not opening...");
        exit(0);

    }

    //read line from input file and check wether each line contains two values
    while (fgets(asmCodefileBuffer, sizeof(CODESIZE), asmCodefile) != NULL)
    {
        //Check if new line and empty space to skip
        if (strcmp(asmCodefileBuffer, "\n") == 0 || strcmp(asmCodefileBuffer, "
") != 0 )
        {
            //check if it contains two digits or not
            if( isdigit(asmCodefileBuffer[0]) && (int)asmCodefileBuffer[2] != 0)
            {
                //store data in programMemory at posion i
                programMemory[i].opCode = atoi(&asmCodefileBuffer[0]);
                programMemory[i].deviceOrAddress = atoi(&asmCodefileBuffer[2]);
                i++;//increament
            }
        }
    }

    //Need to close the asmCodefile
    fclose(asmCodefile);

    //Display output
```

```
    printf(" Tiny Machine Simulator \n");
    printf("Assembling program... \n");
    printf("Program assembled Run. \n");

    //Print intial value in tinny macine varibles

    printVaribles();

    //Get the instruction from programMemory
    for (int i = (pc / 10) - 1; i<sizeof(programMemory); i += 1)
    {
        //call function and passing programMemory value into our parser function
        tinyMachineSimulator(programMemory[i].opCode,
programMemory[i].deviceOrAddress);
    }

    printf(" Program concluded... \n");

    //system("pause");
    return 0;

}
```