CS321-322 Project 2020 Assembler and Emulator

Name : Chandrawanshi Mangesh Shivaji
Roll No. : 1801CS16
Date : 21 Nov. 2020

Task: The aim of this class project is to write a two pass assembler for an extended SIMPLE instruction set. Then write and test programs in SIMPLE assembly. A final part is to write an emulator for the SIMPLE machine

ASSUMPTIONS:

- The Pass1 of the assembler outputs no code and does not fail on undefined labels.
- If a value is in Hexadecimal it would always start with '**0x**'.
- If a value is in Octal it would always start with '**0**'.
- The size of DATA and SET instructions is assumed to be 32 bits and all 32 bits are used for representing value.
- All DATA and SET instructions must be written after HALT for avoiding unexpected faults.
- SET should have only postive integer values as operand
- Emulator also detect errorneous programs if they have accessed memory location out of the boundary, or instruction opcode does not exists
- I am assuming all SET and data instructions are after HALT in the provided assembly code

SUBMISSION:

- The name of the assembler source file is **asm.cpp** and emulator source file **is emu.cpp**
- The assembler file with various .example asm, .lst, .obj and .o files iare present in the zip along with emulator file and corresponding .txt files(for trce output)
- The **claims.txt** is also present in the zip.

- ***<u>Assembler:</u>***

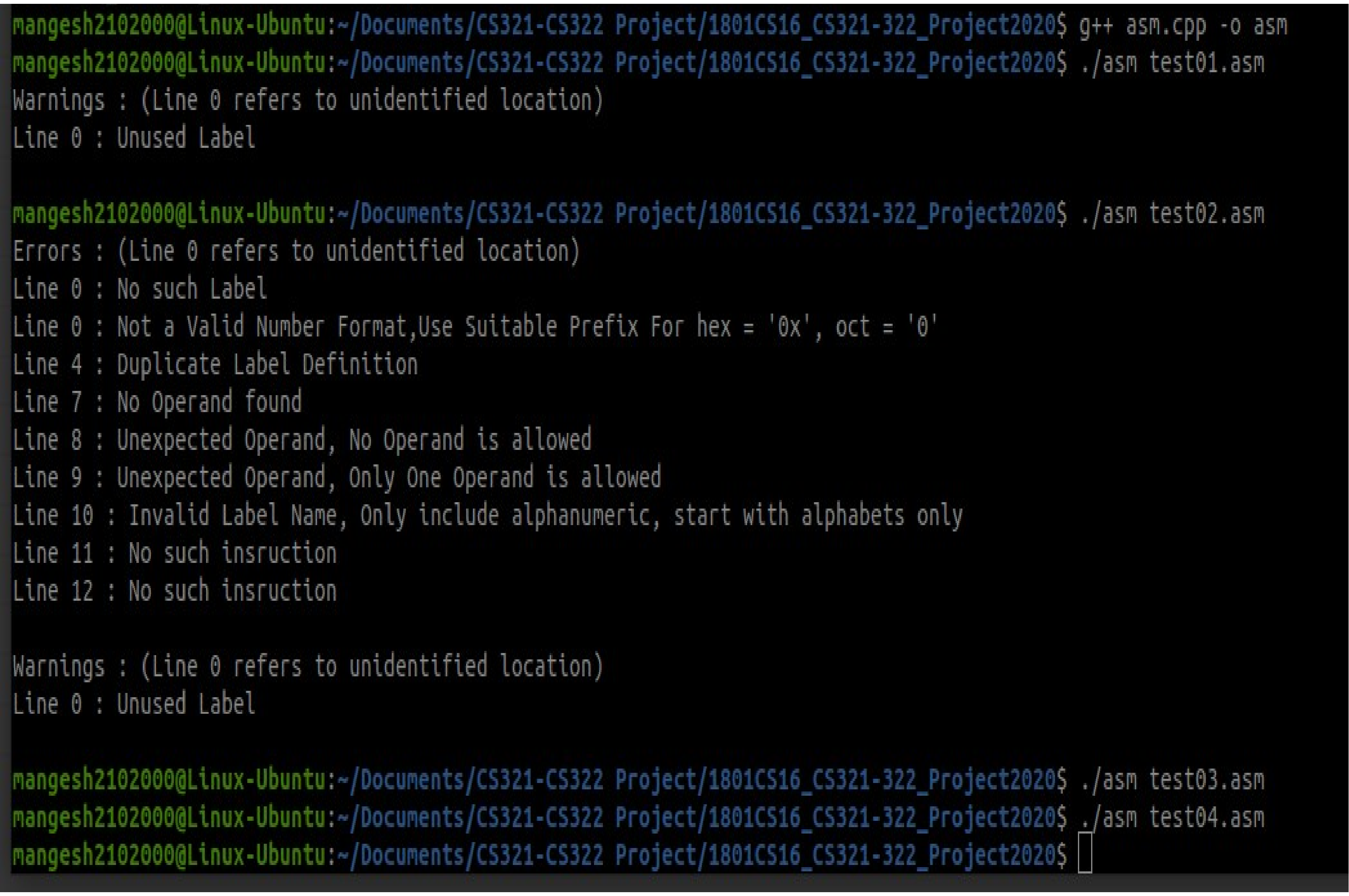
Instuctions to complie and run test files (Please keep all files in the same folder)

First Complie the Assembler using following instruction:

To Compile : **g++ asm.cpp -o asm**

To Run : **./asm test.asm**

All given test files are executed : **(test01.asm,test02.asm,test03.asm,test04.asm)**

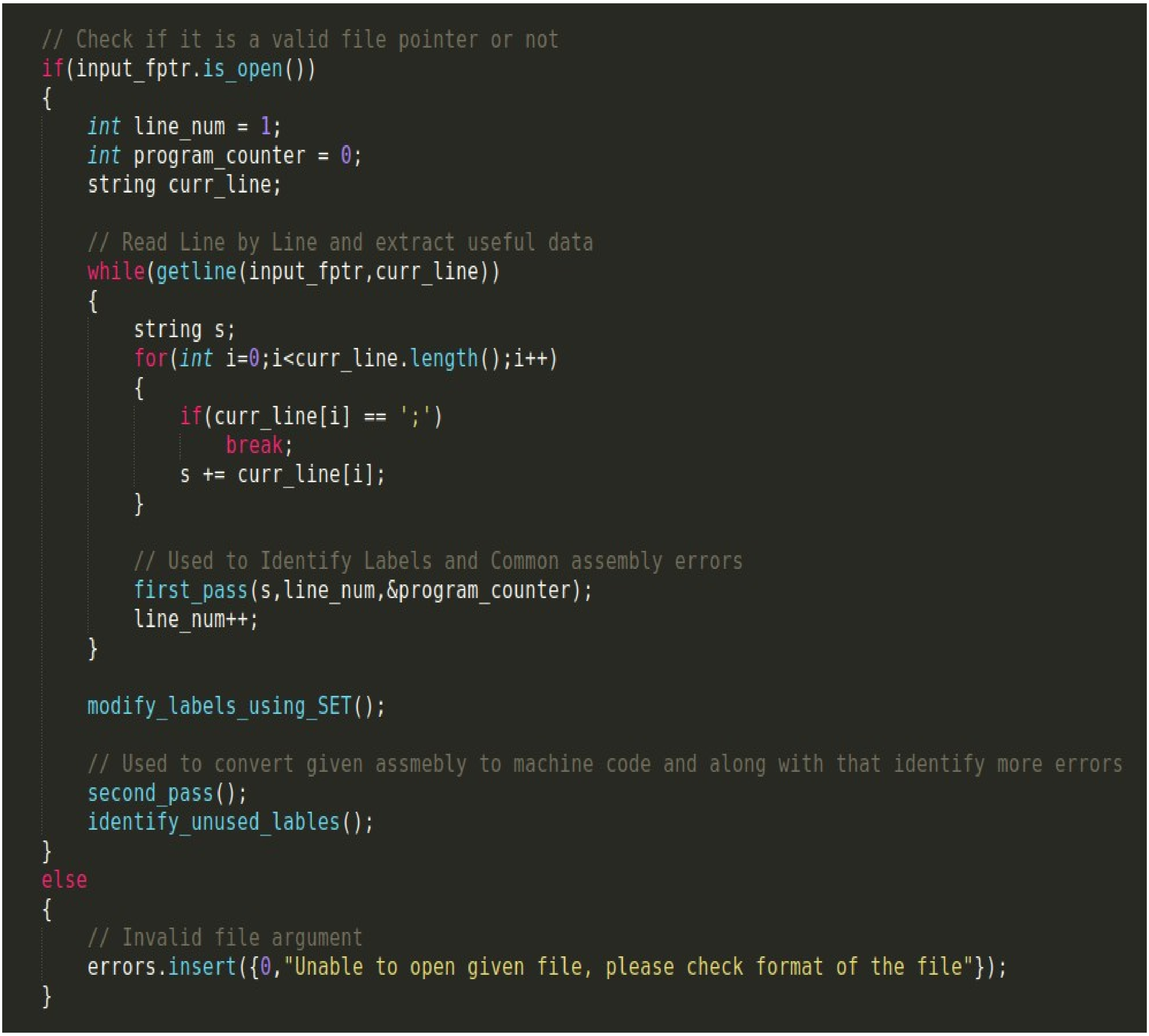


```
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ g++ asm.cpp -o asm
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ ./asm test01.asm
Warnings : (Line 0 refers to unidentified location)
Line 0 : Unused Label

mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ ./asm test02.asm
Errors : (Line 0 refers to unidentified location)
Line 0 : No such Label
Line 0 : Not a Valid Number Format,Use Suitable Prefix For hex = '0x', oct = '0'
Line 4 : Duplicate Label Definition
Line 7 : No Operand found
Line 8 : Unexpected Operand, No Operand is allowed
Line 9 : Unexpected Operand, Only One Operand is allowed
Line 10 : Invalid Label Name, Only include alphanumeric, start with alphabets only
Line 11 : No such insruction
Line 12 : No such insruction

Warnings : (Line 0 refers to unidentified location)
Line 0 : Unused Label

mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ ./asm test03.asm
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ ./asm test04.asm
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ 
```

After that you will see, corresponding Errors and Warnings in the terminal (only if they exist). If the program assembles without any errors, object file named test.o will be created. Output for a valid input assembly program (.asm) will include : .log , .lst and .o files

.log file	=>	declared labels, used labels, errors and warnings
.lst file	=>	listing file (Advanced listing format)
.o file	=>	object file contains machine code

Two-Pass Assembler:



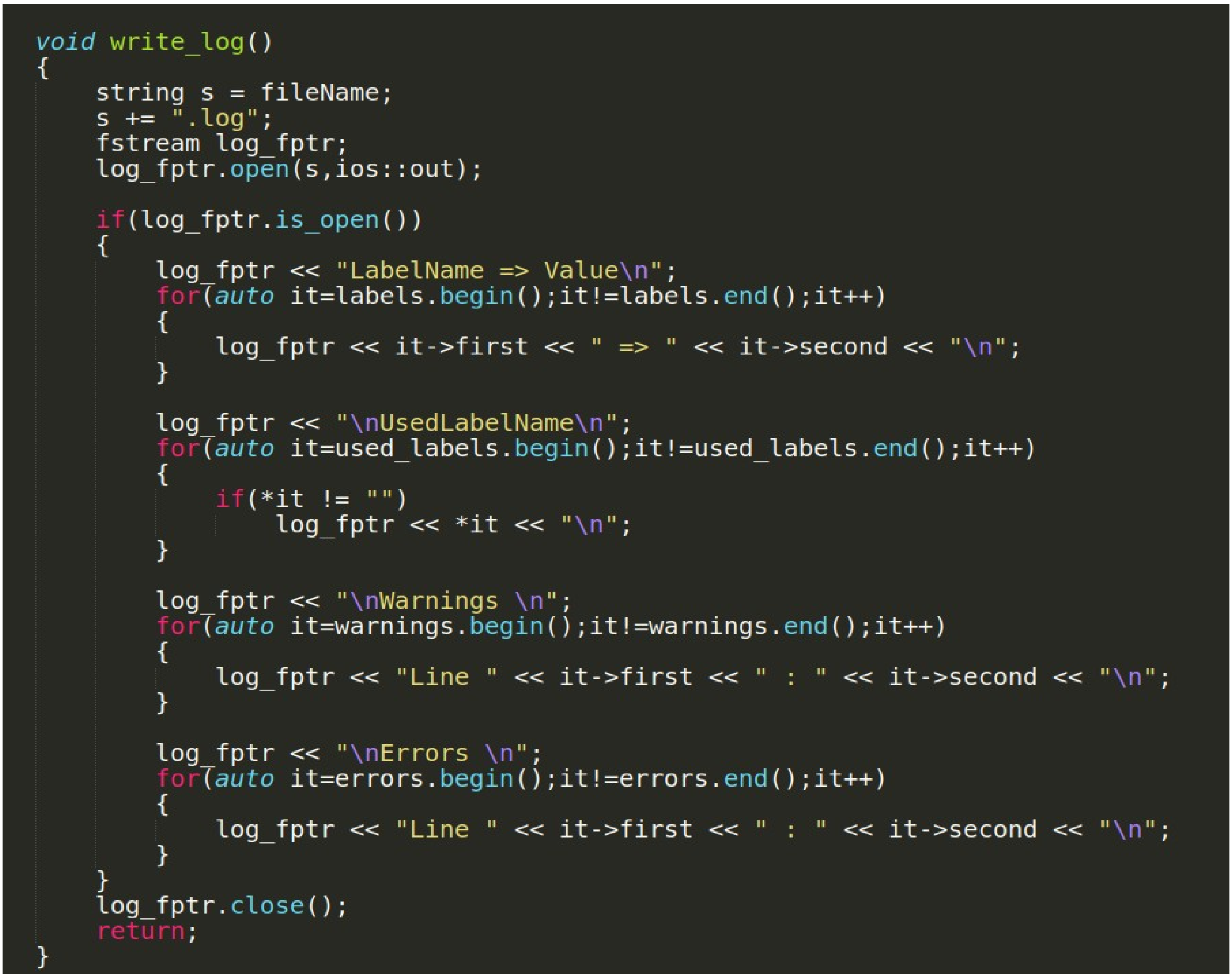
```
// Check if it is a valid file pointer or not
if(input_fptr.is_open())
{
    int line_num = 1;
    int program_counter = 0;
    string curr_line;

    // Read Line by Line and extract useful data
    while(getline(input_fptr,curr_line))
    {
        string s;
        for(int i=0;i<curr_line.length();i++)
        {
            if(curr_line[i] == ';')
                break;
            s += curr_line[i];
        }

        // Used to Identify Labels and Common assembly errors
        first_pass(s,line_num,&program_counter);
        line_num++;
    }

    modify_labels_using_SET();

    // Used to convert given assmebly to machine code and along with that identify more errors
    second_pass();
    identify_unused_lables();
}
else
{
    // Invalid file argument
    errors.insert({0,"Unable to open given file, please check format of the file"});
}
```

I have used two passes to assemble. In First Pass, I have identified all the labels and common assembly errors. In the Second Pass, I have obtained the corresponding machine code and identified additional errors if they exist.

First Pass => first_pass(s,line_num,&program_counter);
Second Pass => second_pass();

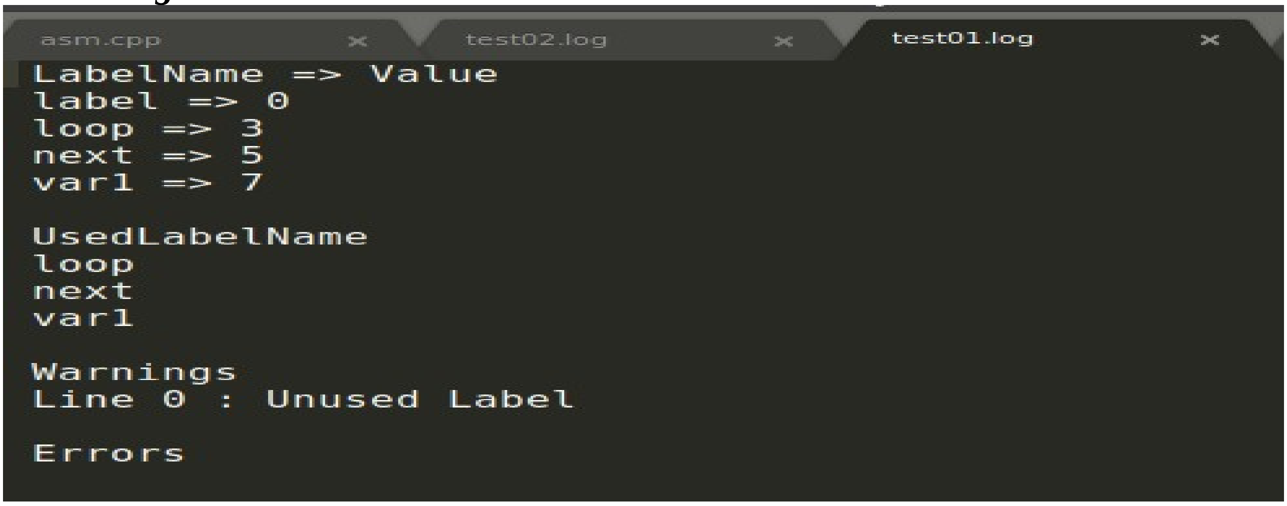
Labels,Used Labels,Errors,Warnings Detection : (Log Files)

function which writes to log file



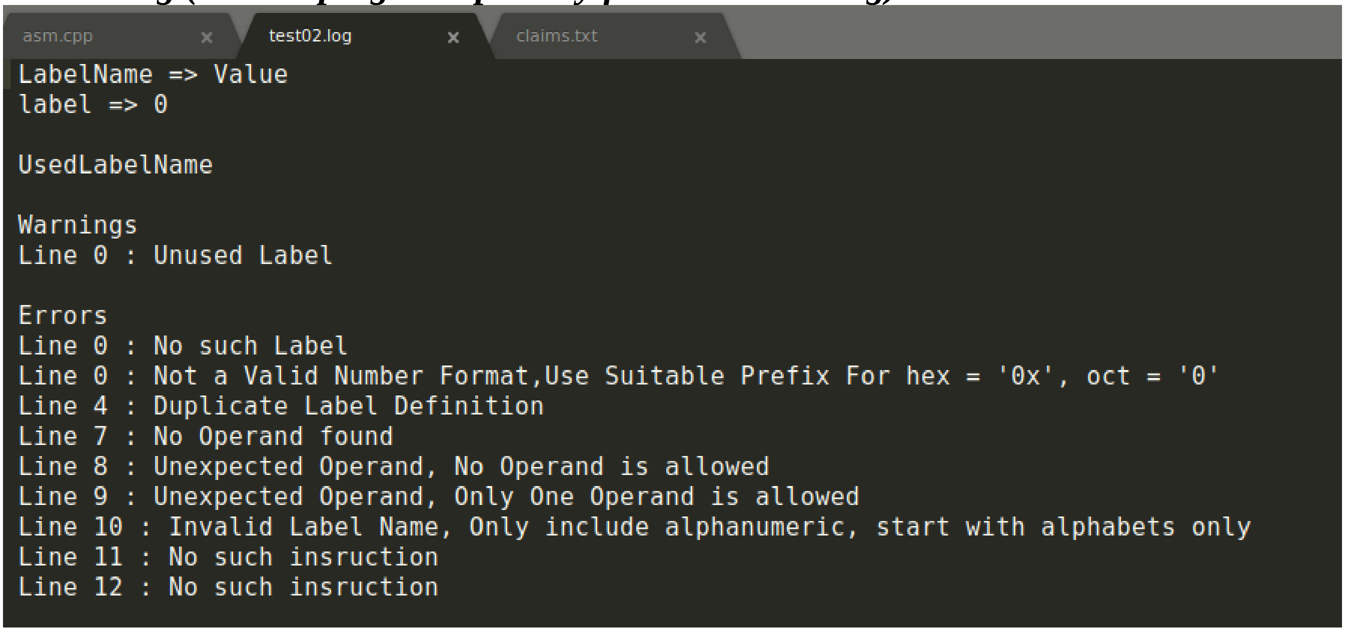
```
void write_log()
{
    string s = fileName;
    s += ".log";
    fstream log_fptr;
    log_fptr.open(s,ios::out);

    if(log_fptr.is_open())
    {
        log_fptr << "LabelName => Value\n";
        for(auto it=labels.begin();it!=labels.end();it++)
        {
            log_fptr << it->first << " => " << it->second << "\n";
        }

        log_fptr << "\nUsedLabelName\n";
        for(auto it=used_labels.begin();it!=used_labels.end();it++)
        {
            if(*it != "")
                log_fptr << *it << "\n";
        }

        log_fptr << "\nWarnings \n";
        for(auto it=warnings.begin();it!=warnings.end();it++)
        {
            log_fptr << "Line " << it->first << " : " << it->second << "\n";
        }

        log_fptr << "\nErrors \n";
        for(auto it=errors.begin();it!=errors.end();it++)
        {
            log_fptr << "Line " << it->first << " : " << it->second << "\n";
        }
    }
    log_fptr.close();
    return;
}
```

test01.log



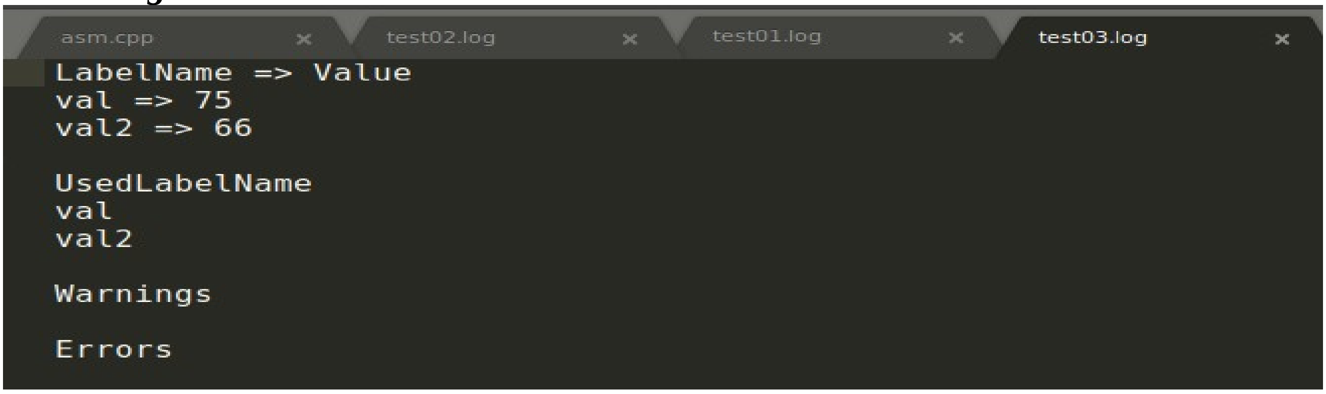
```
LabelName => Value
label => 0
loop => 3
next => 5
var1 => 7

UsedLabelName
loop
next
var1

Warnings
Line 0 : Unused Label

Errors
```

test02.log (it is the program specially for error handling)



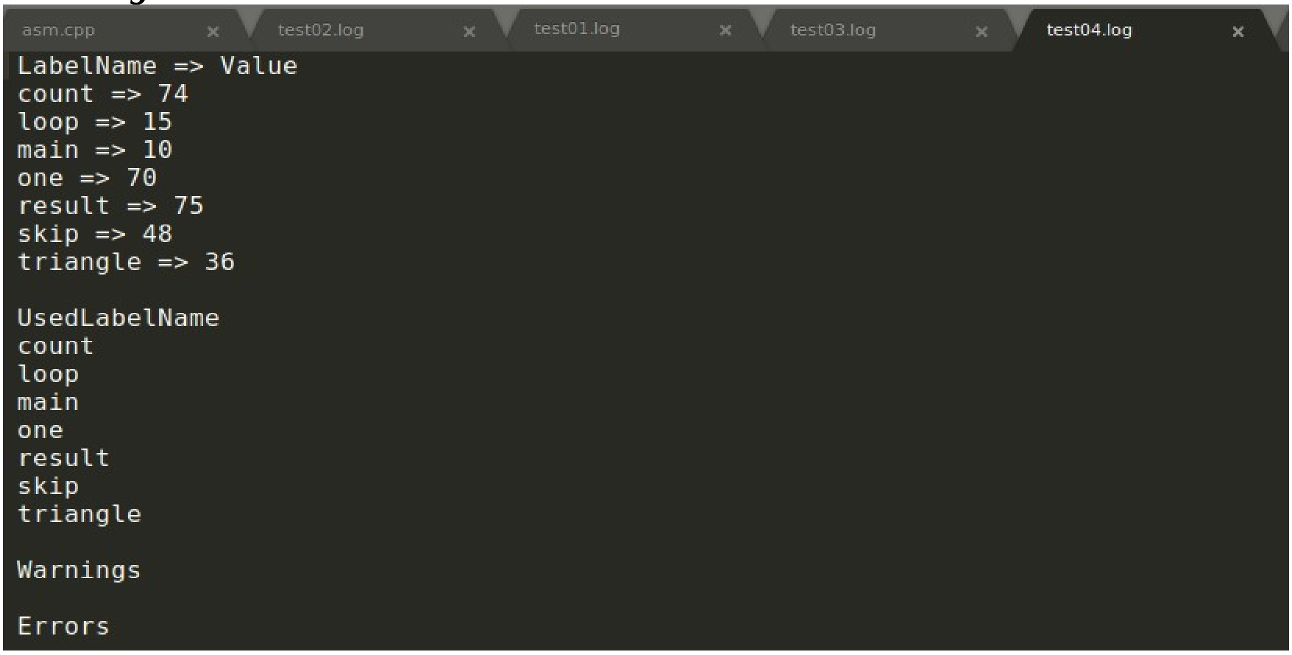
```
LabelName => Value
label => 0

UsedLabelName

Warnings
Line 0 : Unused Label

Errors
Line 0 : No such Label
Line 0 : Not a Valid Number Format,Use Suitable Prefix For hex = '0x', oct = '0'
Line 4 : Duplicate Label Definition
Line 7 : No Operand found
Line 8 : Unexpected Operand, No Operand is allowed
Line 9 : Unexpected Operand, Only One Operand is allowed
Line 10 : Invalid Label Name, Only include alphanumeric, start with alphabets only
Line 11 : No such insruction
Line 12 : No such insruction
```

test03.log



```
LabelName => Value
val => 75
val2 => 66

UsedLabelName
val
val2

Warnings

Errors
```

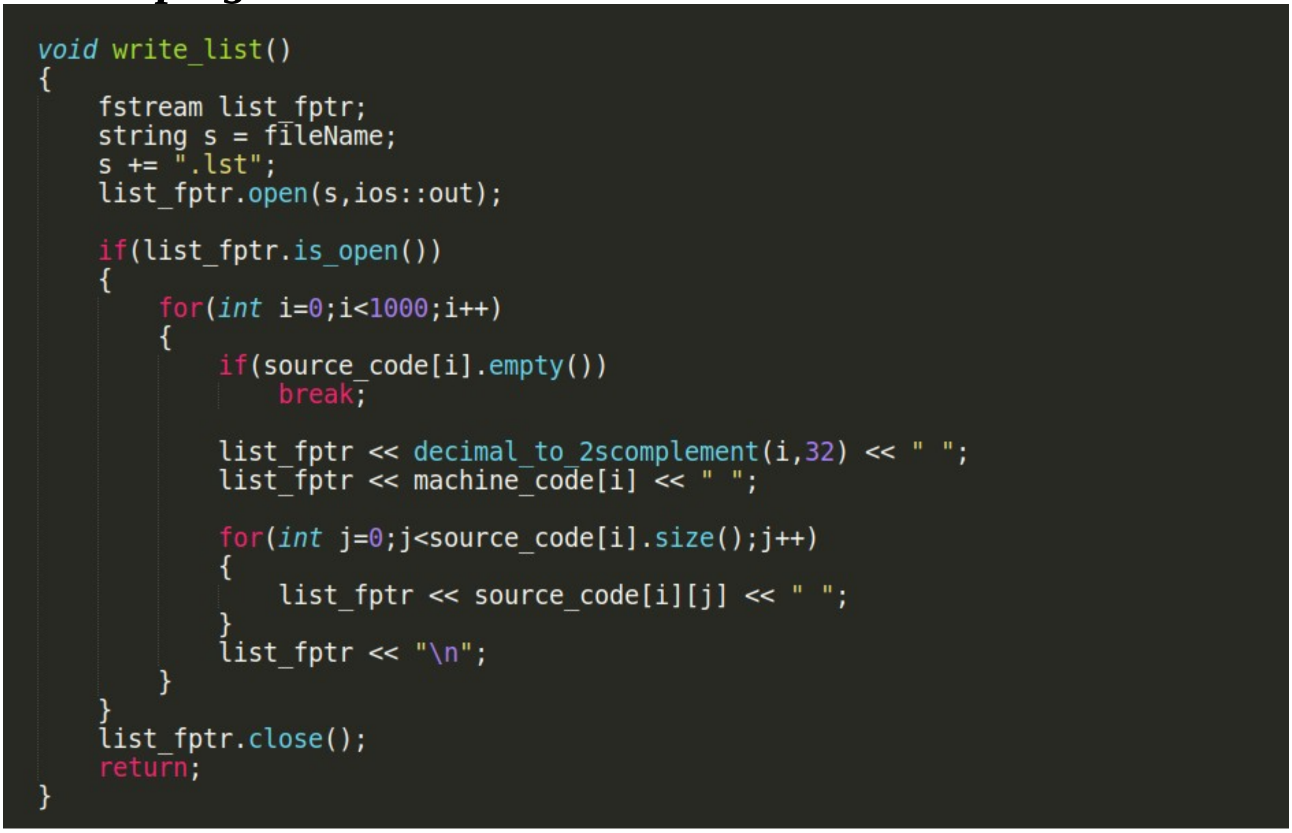
test04.log



```
LabelName => Value
count => 74
loop => 15
main => 10
one => 70
result => 75
skip => 48
triangle => 36

UsedLabelName
count
loop
main
one
result
skip
triangle

Warnings

Errors
```

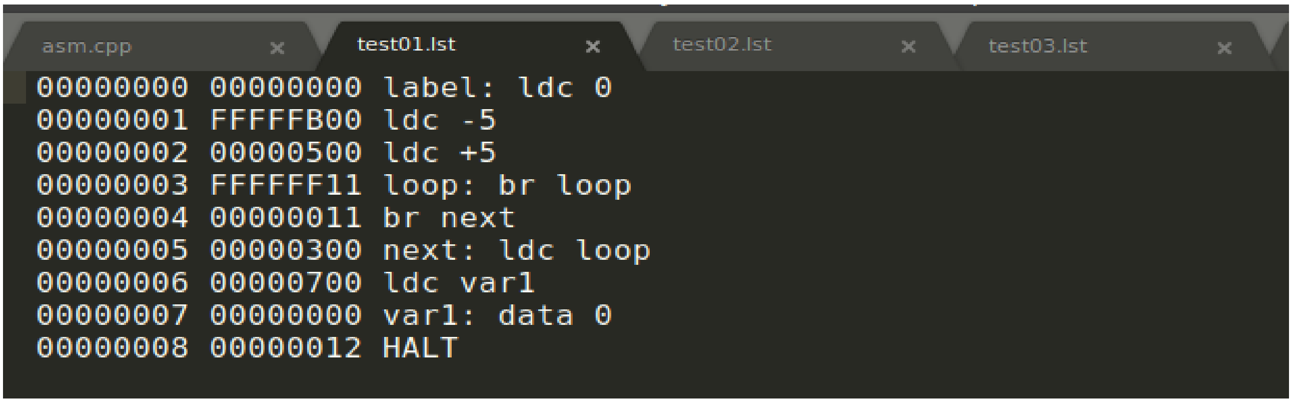
Advanced Lsiting Files : (List Files)

program counter machine code instruction



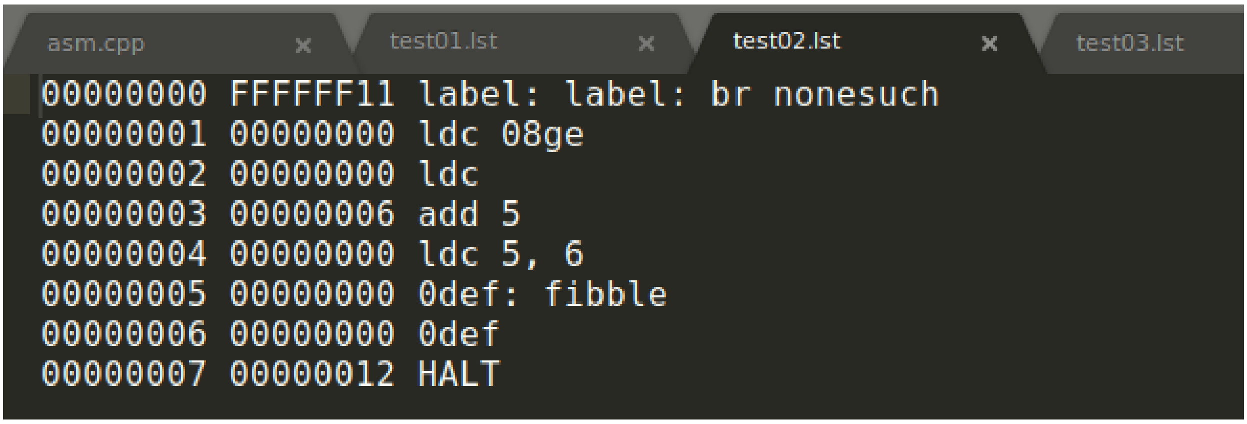
```
void write_list()
{
    fstream list_fptr;
    string s = fileName;
    s += ".lst";
    list_fptr.open(s,ios::out);

    if(list_fptr.is_open())
    {
        for(int i=0;i<1000;i++)
        {
            if(source_code[i].empty())
                break;

            list_fptr << decimal_to_2scomplement(i,32) << " ";
            list_fptr << machine_code[i] << " ";

            for(int j=0;j<source_code[i].size();j++)
            {
                list_fptr << source_code[i][j] << " ";
            }
            list_fptr << "\n";
        }
    }
    list_fptr.close();
    return;
}
```

test01.lst



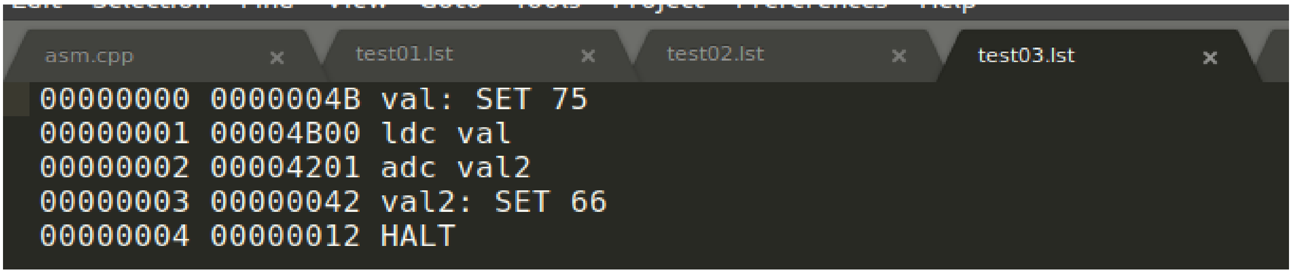
```
00000000 00000000 label: ldc 0
00000001 FFFFFB00 ldc -5
00000002 00000500 ldc +5
00000003 FFFFFF11 loop: br loop
00000004 00000011 br next
00000005 00000300 next: ldc loop
00000006 00000700 ldc var1
00000007 00000000 var1: data 0
00000008 00000012 HALT
```

test02.lst



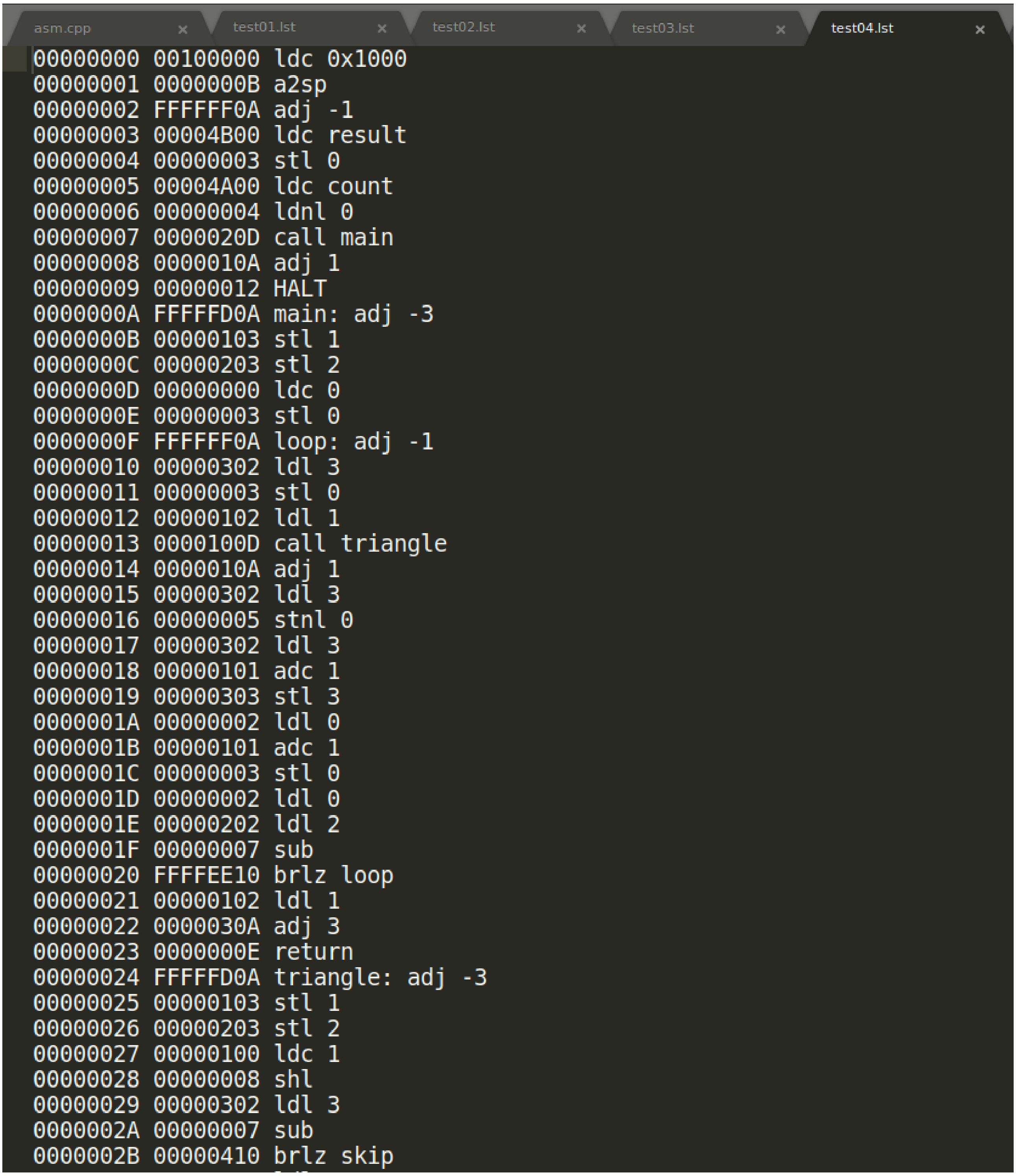
```
00000000 FFFFFF11 label: label: br nonesuch
00000001 00000000 ldc 08ge
00000002 00000000 ldc
00000003 00000006 add 5
00000004 00000000 ldc 5, 6
00000005 00000000 0def: fibble
00000006 00000000 0def
00000007 00000012 HALT
```

test03.lst



```
00000000 0000004B val: SET 75
00000001 00004B00 ldc val
00000002 00004201 adc val2
00000003 00000042 val2: SET 66
00000004 00000012 HALT
```

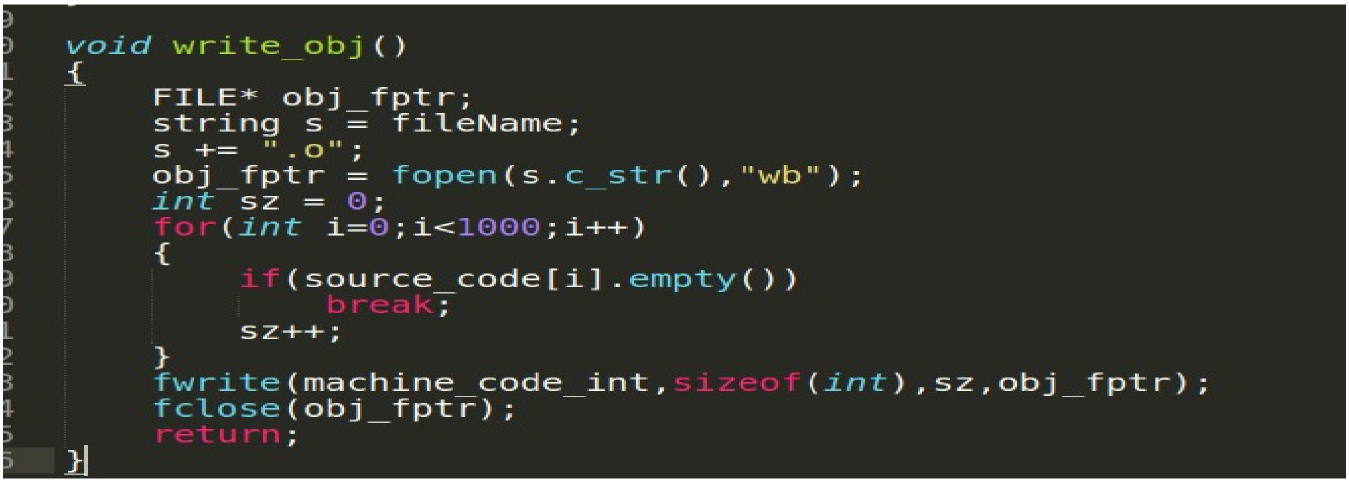
test04.lst



```
00000000 00100000 ldc 0x1000
00000001 0000000B a2sp
00000002 FFFFFF0A adj -1
00000003 00004B00 ldc result
00000004 00000003 stl 0
00000005 00004A00 ldc count
00000006 00000004 ldnl 0
00000007 0000020D call main
00000008 0000010A adj 1
00000009 00000012 HALT
0000000A FFFFFD0A main: adj -3
0000000B 00000103 stl 1
0000000C 00000203 stl 2
0000000D 00000000 ldc 0
0000000E 00000003 stl 0
0000000F FFFFFF0A loop: adj -1
00000010 00000302 ldl 3
00000011 00000003 stl 0
00000012 00000102 ldl 1
00000013 0000100D call triangle
00000014 0000010A adj 1
00000015 00000302 ldl 3
00000016 00000005 stnl 0
00000017 00000302 ldl 3
00000018 00000101 adc 1
00000019 00000303 stl 3
0000001A 00000002 ldl 0
0000001B 00000101 adc 1
0000001C 00000003 stl 0
0000001D 00000002 ldl 0
0000001E 00000202 ldl 2
0000001F 00000007 sub
00000020 FFFFEE10 brlz loop
00000021 00000102 ldl 1
00000022 0000030A adj 3
00000023 0000000E return
00000024 FFFFFD0A triangle: adj -3
00000025 00000103 stl 1
00000026 00000203 stl 2
00000027 00000100 ldc 1
00000028 00000008 shl
00000029 00000302 ldl 3
0000002A 00000007 sub
0000002B 00000410 brlz skip
```

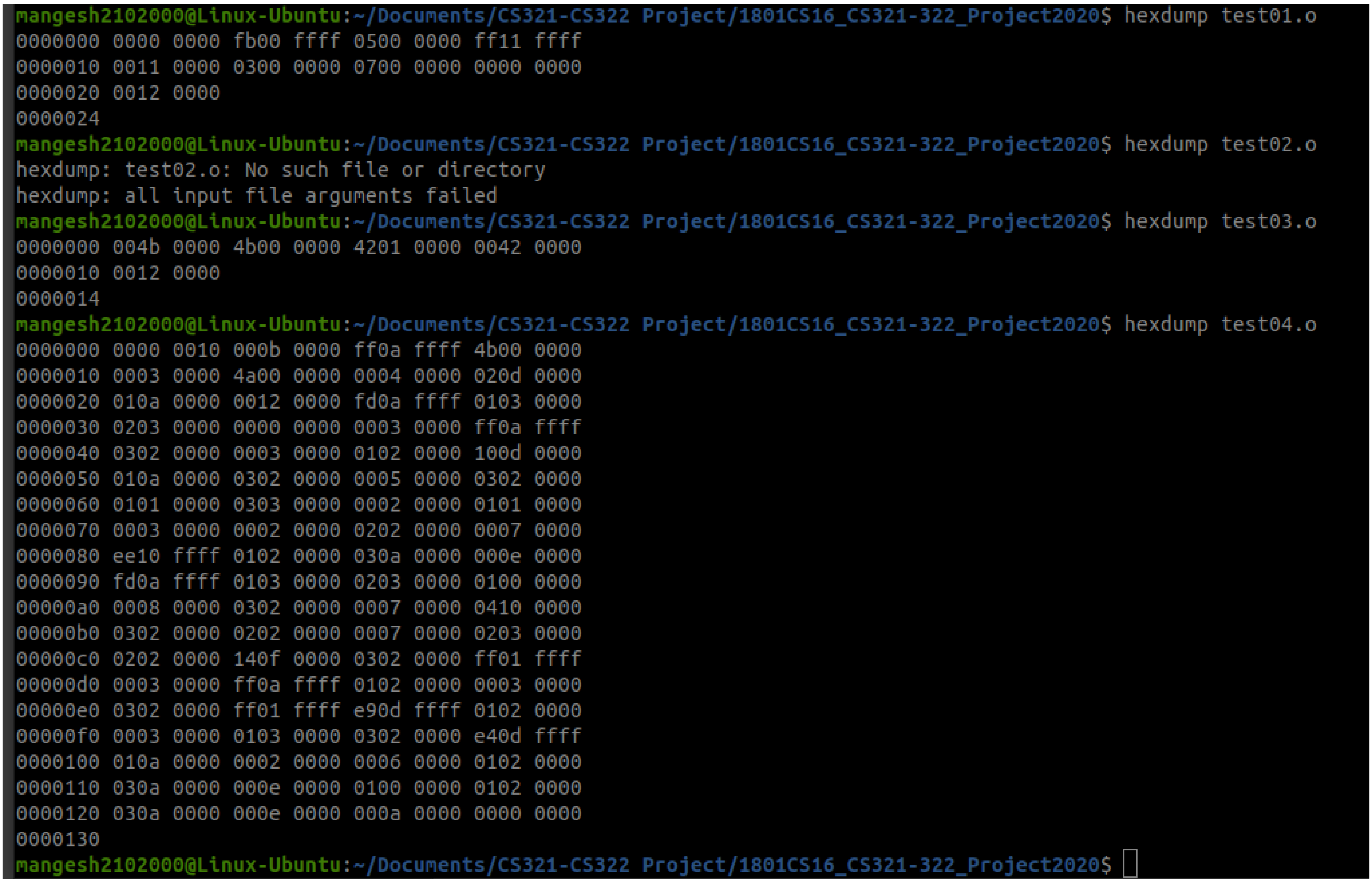
and continued ... (Please look test04.lst for full listing file)

Machine Code : (Object Files)
All the decimal machine codes are converted into corressponding binary format using fwrite.



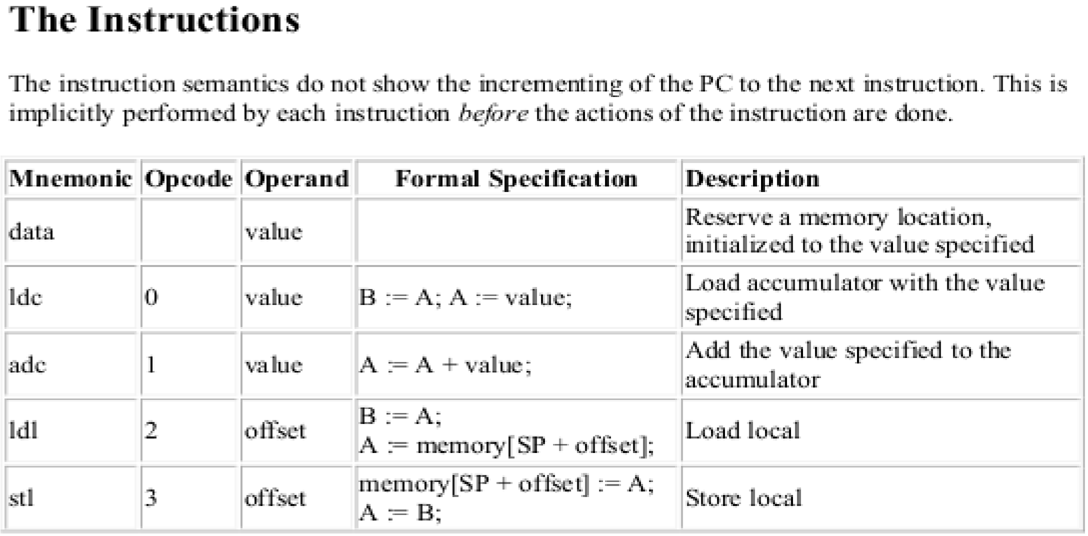
```
void write_obj()
{
    FILE* obj_fptr;
    string s = fileName;
    s += ".o";
    obj_fptr = fopen(s.c_str(),"wb");
    int sz = 0;
    for(int i=0;i<1000;i++)
    {
        if(source_code[i].empty())
            break;
        sz++;
    }
    fwrite(machine_code_int,sizeof(int),sz,obj_fptr);
    fclose(obj_fptr);
    return;
}
```

Using hexdump I have displayed the machine code for each of the files (test02.o does not exist as test02.asm contained errors). No object file is created for errorneous files.



```
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ hexdump test01.o
0000000 0000 0000 fb00 ffff 0500 0000 ff11 ffff
0000010 0011 0000 0300 0000 0700 0000 0000 0000
0000020 0012 0000
0000024
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ hexdump test02.o
hexdump: test02.o: No such file or directory
hexdump: all input file arguments failed
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ hexdump test03.o
0000000 004b 0000 4b00 0000 4201 0000 0042 0000
0000010 0012 0000
0000014
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ hexdump test04.o
0000000 0000 0010 000b 0000 ff0a ffff 4b00 0000
0000010 0003 0000 4a00 0000 0004 0000 020d 0000
0000020 010a 0000 0012 0000 fd0a ffff 0103 0000
0000030 0203 0000 0000 0000 0003 0000 ff0a ffff
0000040 0302 0000 0003 0000 0102 0000 100d 0000
0000050 010a 0000 0302 0000 0005 0000 0302 0000
0000060 0101 0000 0303 0000 0002 0000 0101 0000
0000070 0003 0000 0002 0000 0202 0000 0007 0000
0000080 ee10 ffff 0102 0000 030a 0000 000e 0000
0000090 fd0a ffff 0103 0000 0203 0000 0100 0000
00000a0 0008 0000 0302 0000 0007 0000 0410 0000
00000b0 0302 0000 0202 0000 0007 0000 0203 0000
00000c0 0202 0000 140f 0000 0302 0000 ff01 ffff
00000d0 0003 0000 ff0a ffff 0102 0000 0003 0000
00000e0 0302 0000 ff01 ffff e90d ffff 0102 0000
00000f0 0003 0000 0103 0000 0302 0000 e40d ffff
0000100 010a 0000 0002 0000 0006 0000 0102 0000
0000110 030a 0000 000e 0000 0100 0000 0102 0000
0000120 030a 0000 000e 0000 000a 0000 0000 0000
0000130
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$
```

Table of instruction names and expected operands :



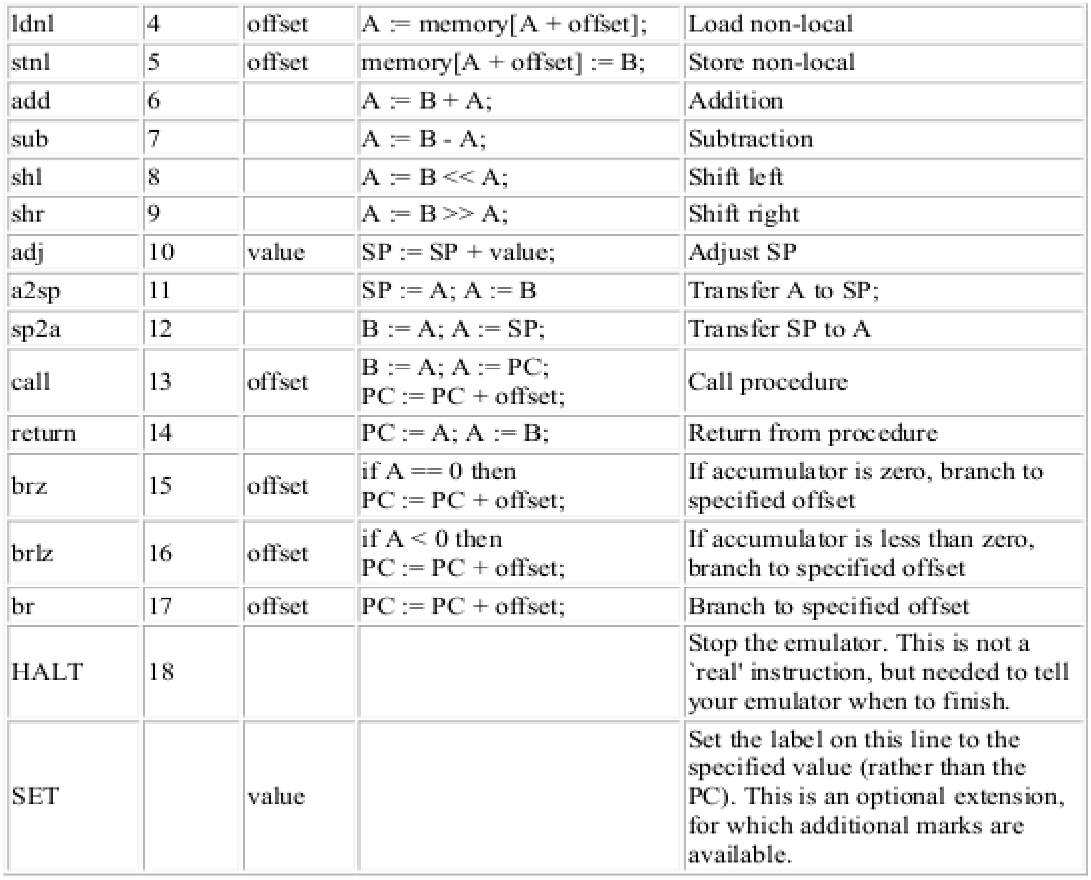
The Instructions

The instruction semantics do not show the incrementing of the PC to the next instruction. This is implicitly performed by each instruction *before* the actions of the instruction are done.

Mnemonic	Opcode	Operand	Formal Specification	Description
data		value		Reserve a memory location, initialized to the value specified
ldc	0	value	B := A; A := value;	Load accumulator with the value specified
adc	1	value	A := A + value;	Add the value specified to the accumulator

ldl	2	offset	B := A;
			A := memory[SP + offset];
				Load local

stl	3	offset	memory[SP + offset] := A;
			A := B;
				Store local



ldnl	4	offset	A := memory[A + offset];	Load non-local
stnl	5	offset	memory[A + offset] := B;	Store non-local
add	6		A := B + A;	Addition
sub	7		A := B - A;	Subtraction
shl	8		A := B << A;	Shift left
shr	9		A := B >> A;	Shift right
adj	10	value	SP := SP + value;	Adjust SP
a2sp	11		SP := A; A := B	Transfer A to SP;
sp2a	12		B := A; A := SP;	Transfer SP to A

call	13	offset	B := A; A := PC;
			PC := PC + offset;
				Call procedure

return	14		PC := A; A := B;	Return from procedure

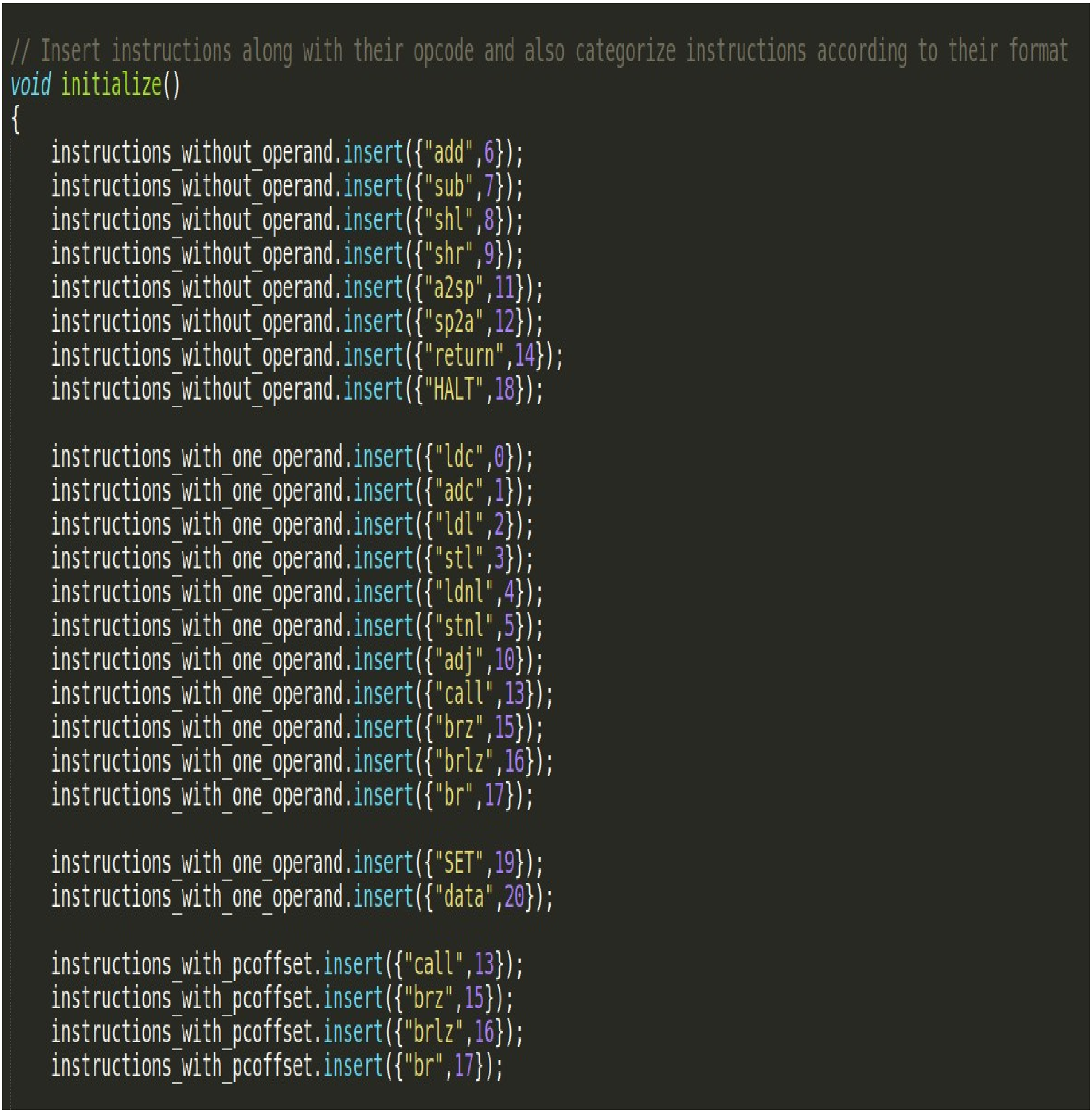
brz	15	offset	if A == 0 then
			PC := PC + offset;
				If accumulator is zero, branch to specified offset

brlz	16	offset	if A < 0 then
			PC := PC + offset;
				If accumulator is less than zero, branch to specified offset

br	17	offset	PC := PC + offset;	Branch to specified offset
HALT	18			Stop the emulator. This is not a `real' instruction, but needed to tell your emulator when to finish.
SET		value		Set the label on this line to the specified value (rather than the PC). This is an optional extension, for which additional marks are available.

I have used map data structure to store the following formats :

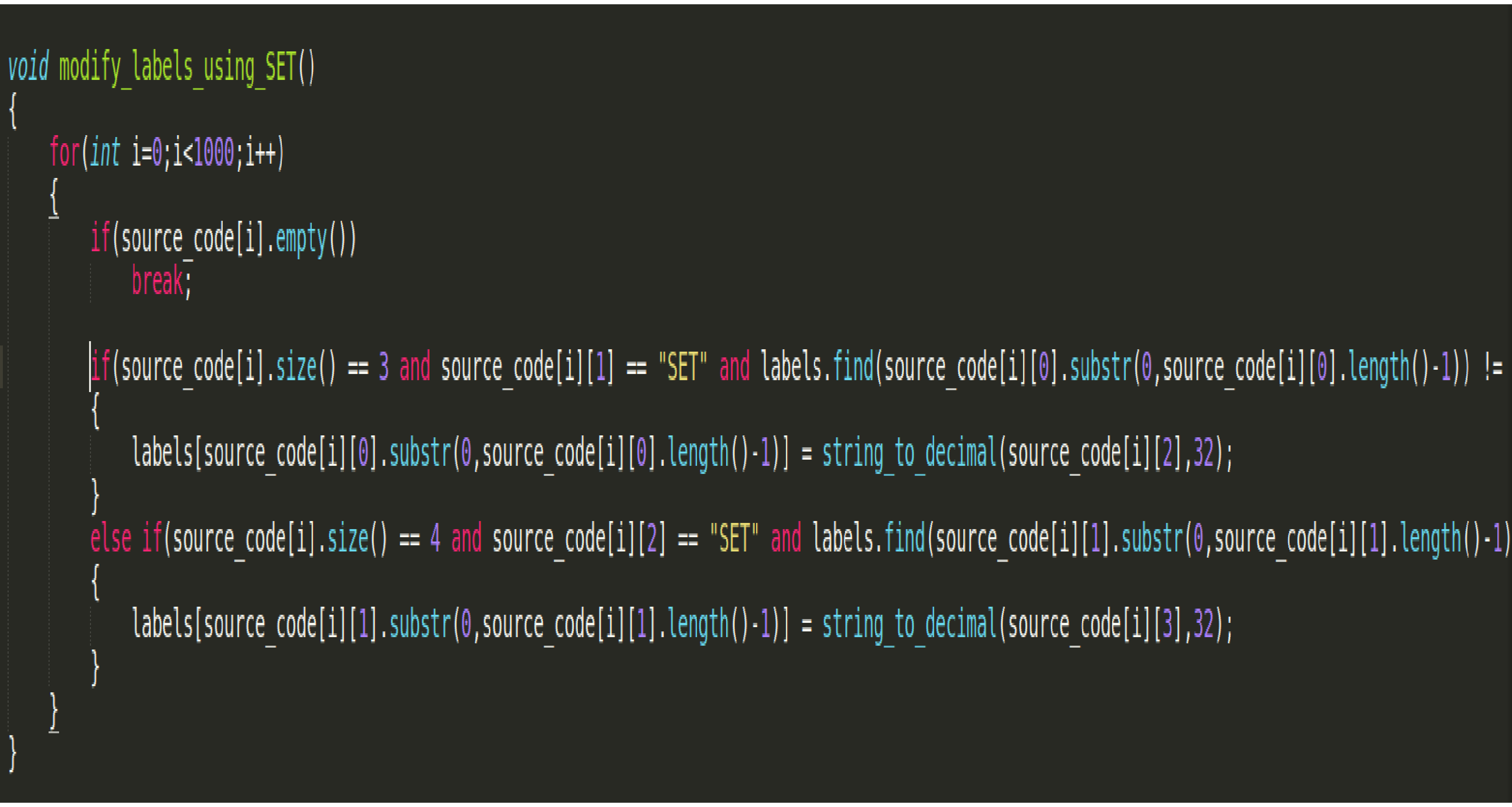
instructions_without_operand
instructions_with_one_operand
instructions_with_pcoffset



```
// Insert instructions along with their opcode and also categorize instructions according to their format
void initialize()
{
    instructions_without_operand.insert({"add",6});
    instructions_without_operand.insert({"sub",7});
    instructions_without_operand.insert({"shl",8});
    instructions_without_operand.insert({"shr",9});
    instructions_without_operand.insert({"a2sp",11});
    instructions_without_operand.insert({"sp2a",12});
    instructions_without_operand.insert({"return",14});
    instructions_without_operand.insert({"HALT",18});

    instructions_with_one_operand.insert({"ldc",0});
    instructions_with_one_operand.insert({"adc",1});
    instructions_with_one_operand.insert({"ldl",2});
    instructions_with_one_operand.insert({"stl",3});
    instructions_with_one_operand.insert({"ldnl",4});
    instructions_with_one_operand.insert({"stnl",5});
    instructions_with_one_operand.insert({"adj",10});
    instructions_with_one_operand.insert({"call",13});
    instructions_with_one_operand.insert({"brz",15});
    instructions_with_one_operand.insert({"brlz",16});
    instructions_with_one_operand.insert({"br",17});

    instructions_with_one_operand.insert({"SET",19});
    instructions_with_one_operand.insert({"data",20});

    instructions_with_pcoffset.insert({"call",13});
    instructions_with_pcoffset.insert({"brz",15});
    instructions_with_pcoffset.insert({"brlz",16});
    instructions_with_pcoffset.insert({"br",17});
```

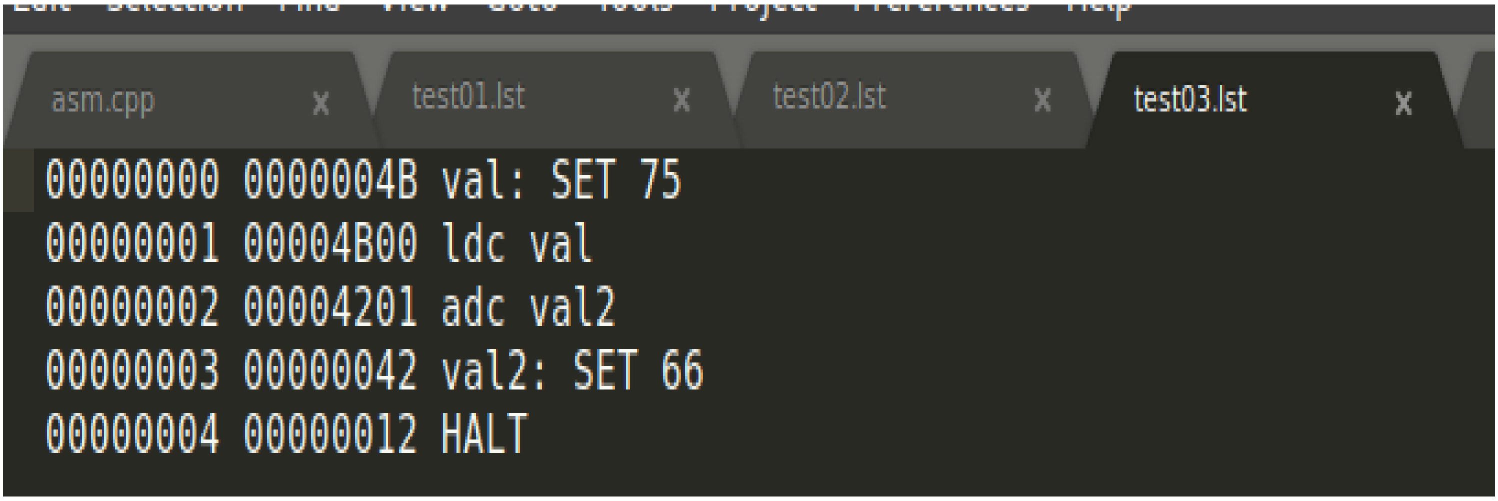
Instruction SET

This instruction sets the value of declared label in the same line to the value provided to it as a operand.



```
void modify_labels_using_SET()
{
    for(int i=0;i<1000;i++)
    {
        if(source_code[i].empty())
            break;

        if(source_code[i].size() == 3 and source_code[i][1] == "SET" and labels.find(source_code[i][0].substr(0,source_code[i][0].length()-1)) !=
        {
            labels[source_code[i][0].substr(0,source_code[i][0].length()-1)] = string_to_decimal(source_code[i][2],32);
        }
        else if(source_code[i].size() == 4 and source_code[i][2] == "SET" and labels.find(source_code[i][1].substr(0,source_code[i][1].length()-1)
        {
            labels[source_code[i][1].substr(0,source_code[i][1].length()-1)] = string_to_decimal(source_code[i][3],32);
        }
    }
}
```

Below is the listing file of test03.asm program

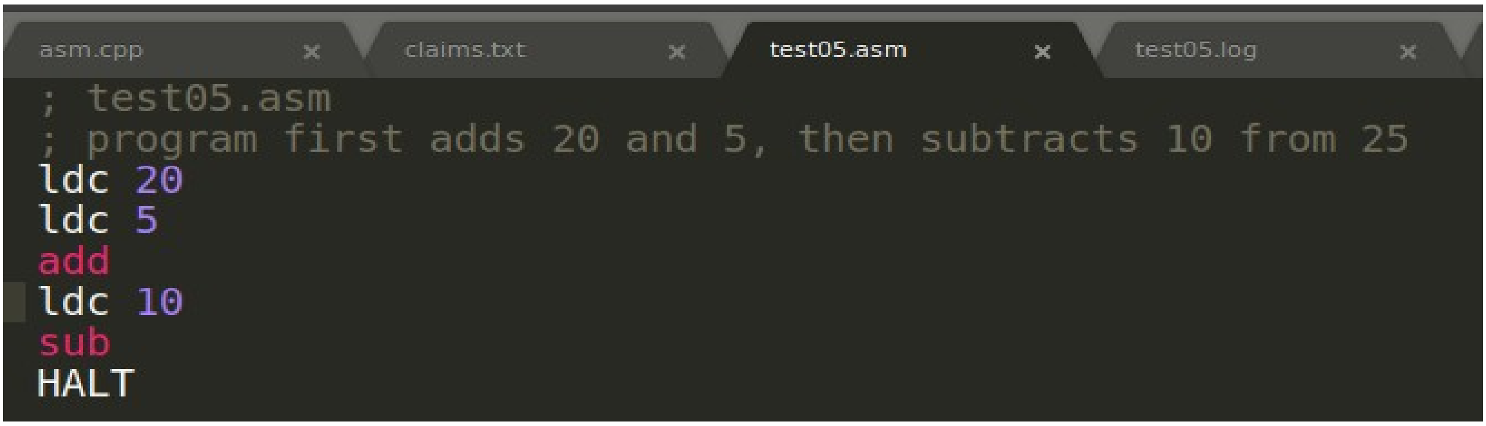


```
00000000 0000004B val: SET 75
00000001 00004B00 ldc val
00000002 00004201 adc val2
00000003 00000042 val2: SET 66
00000004 00000012 HALT
```

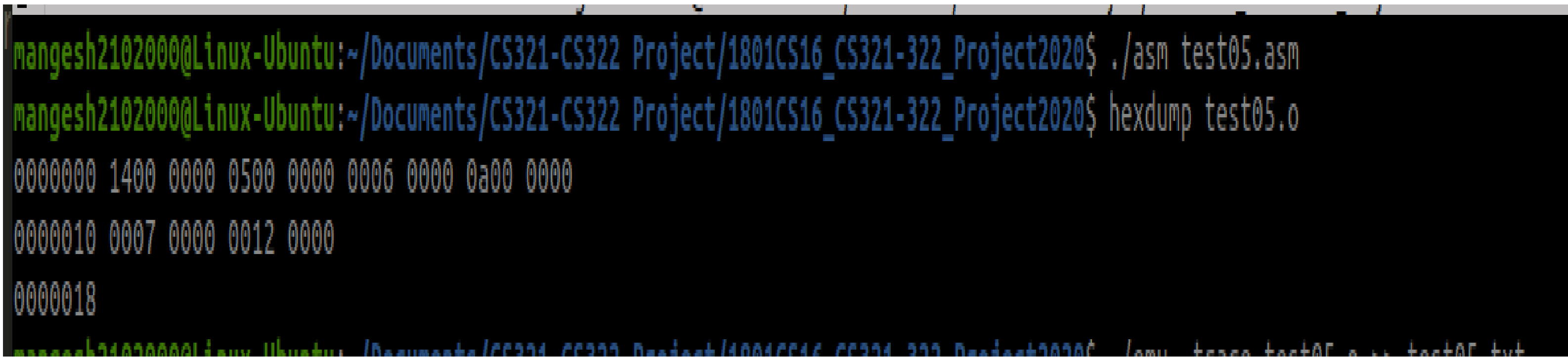
Here we can see that using SET values of labels val and val2 are modified to 75 (4B in hex) and 66 (42 in hex) respectively. The modified values are used in ldc and adc instructions.

Additional Test Programs

test05.asm

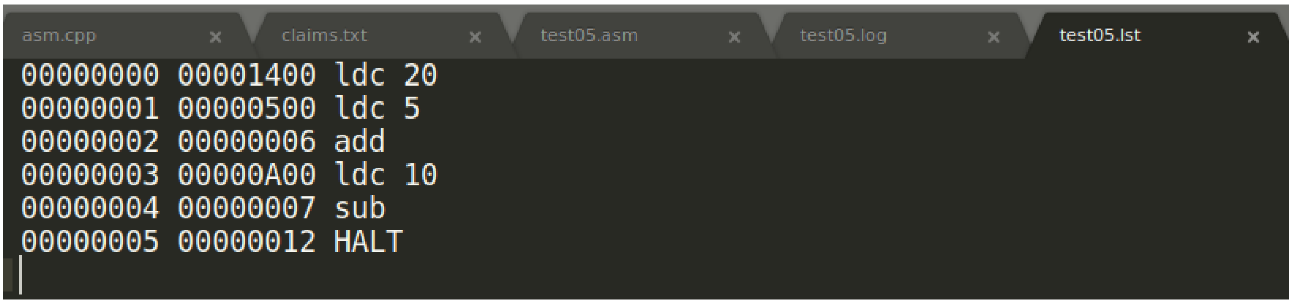


```
; test05.asm
; program first adds 20 and 5, then subtracts 10 from 25
ldc 20
ldc 5
add
ldc 10
sub
HALT
```



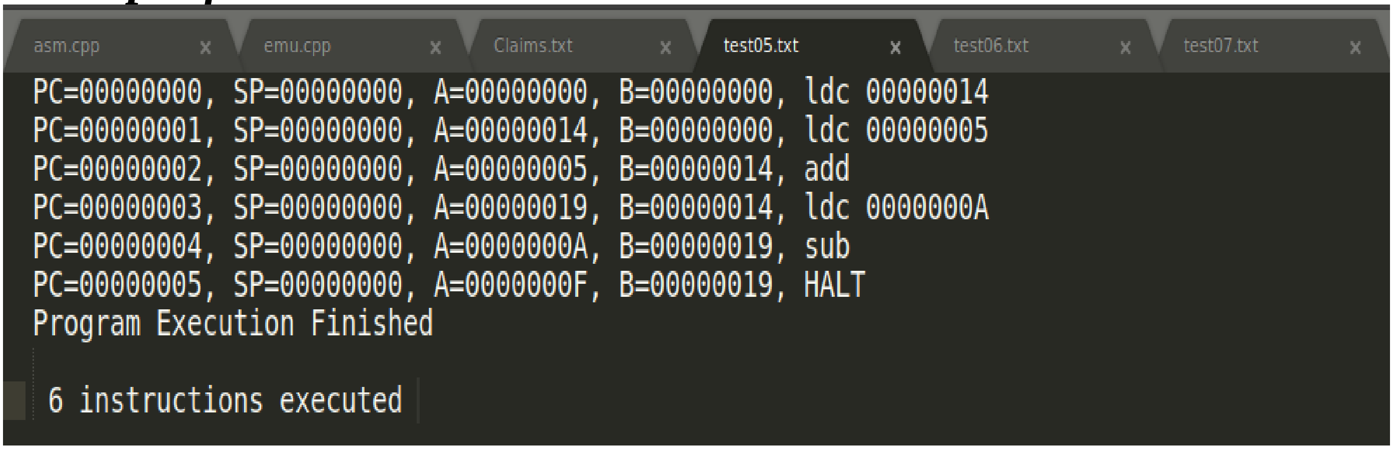
```
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ ./asm test05.asm
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ hexdump test05.o
0000000 1400 0000 0500 0000 0006 0000 0a00 0000
0000010 0007 0000 0012 0000
0000018
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ ./emu -trace test05.o >> test05.txt
```

test05.lst



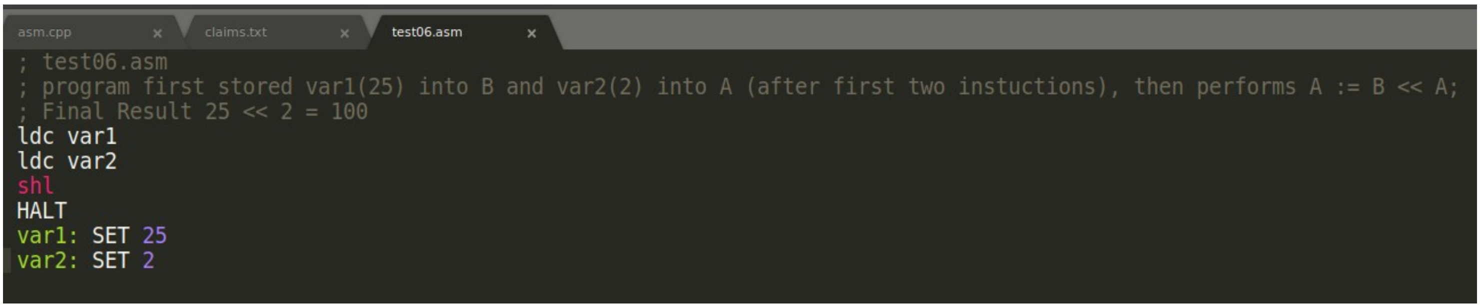
```
00000000 00001400 ldc 20
00000001 00000500 ldc 5
00000002 00000006 add
00000003 00000A00 ldc 10
00000004 00000007 sub
00000005 00000012 HALT
```

Output from emulator:

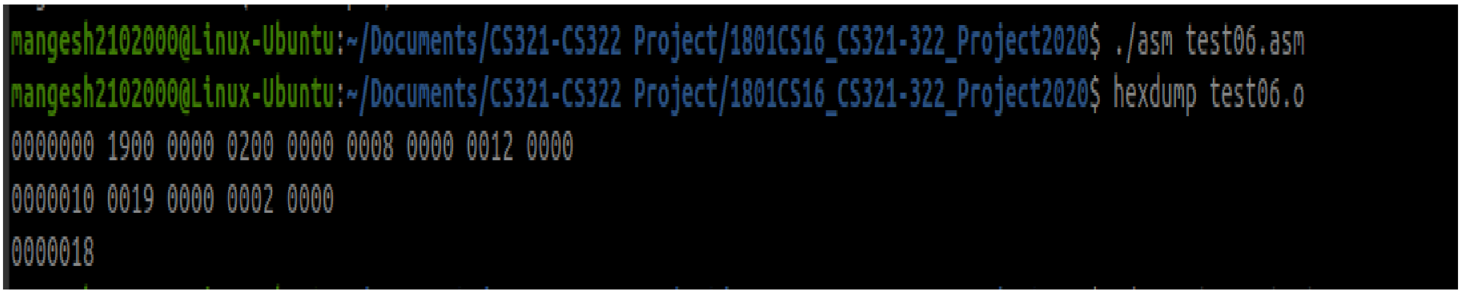


```
PC=00000000, SP=00000000, A=00000000, B=00000000, ldc 00000014
PC=00000001, SP=00000000, A=00000014, B=00000000, ldc 00000005
PC=00000002, SP=00000000, A=00000005, B=00000014, add
PC=00000003, SP=00000000, A=00000019, B=00000014, ldc 0000000A
PC=00000004, SP=00000000, A=0000000A, B=00000019, sub
PC=00000005, SP=00000000, A=0000000F, B=00000019, HALT
Program Execution Finished

 6 instructions executed
```

test06.asm

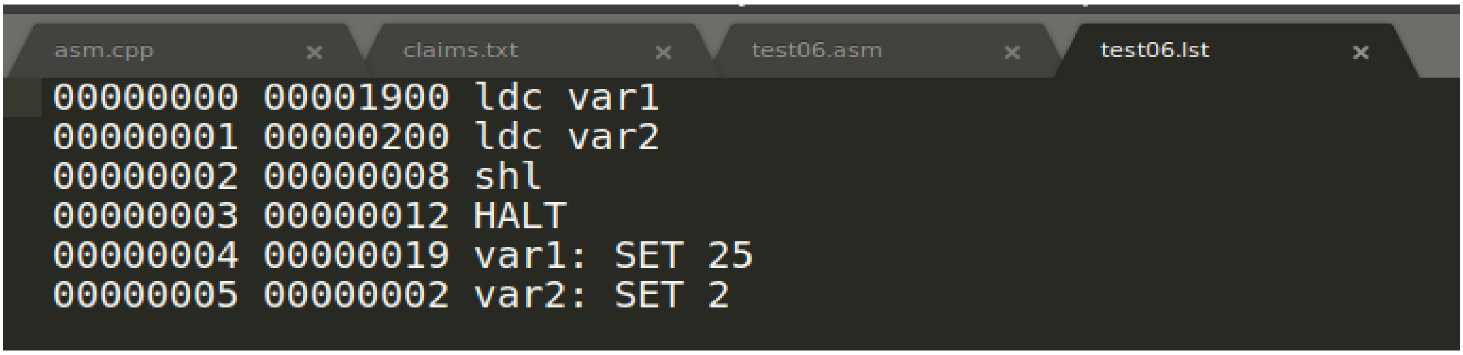


```
; test06.asm
; program first stored var1(25) into B and var2(2) into A (after first two instuctions), then performs A := B << A;
; Final Result 25 << 2 = 100
ldc var1
ldc var2
shl
HALT
var1: SET 25
var2: SET 2
```



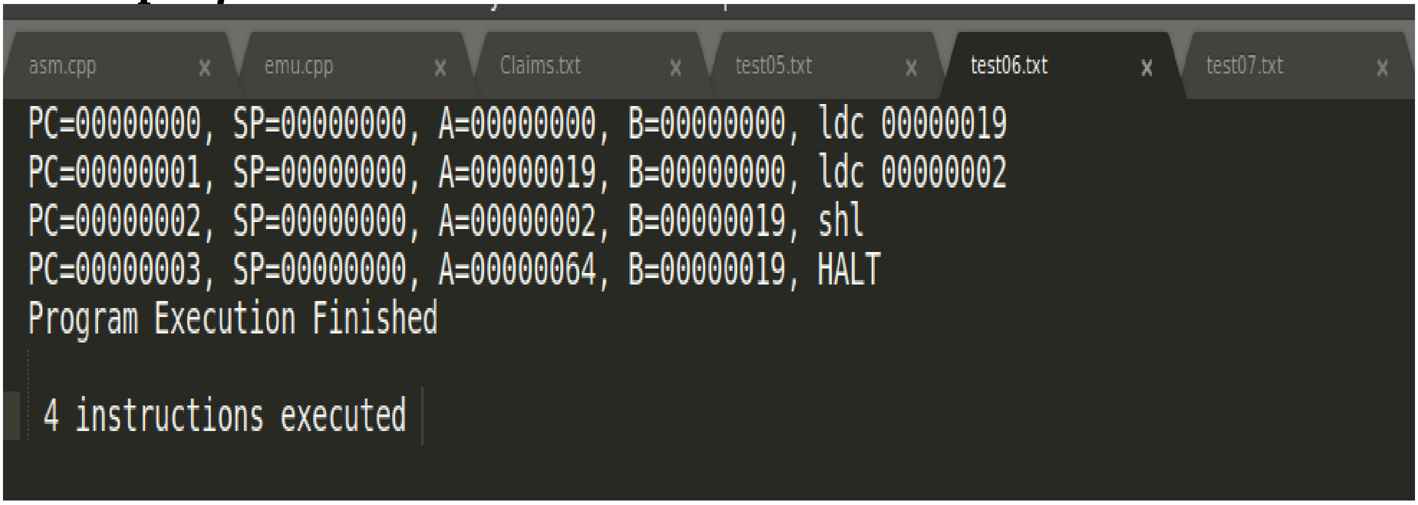
```
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ ./asm test06.asm
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ hexdump test06.o
0000000 1900 0000 0200 0000 0008 0000 0012 0000
0000010 0019 0000 0002 0000
0000018
```

test06.lst



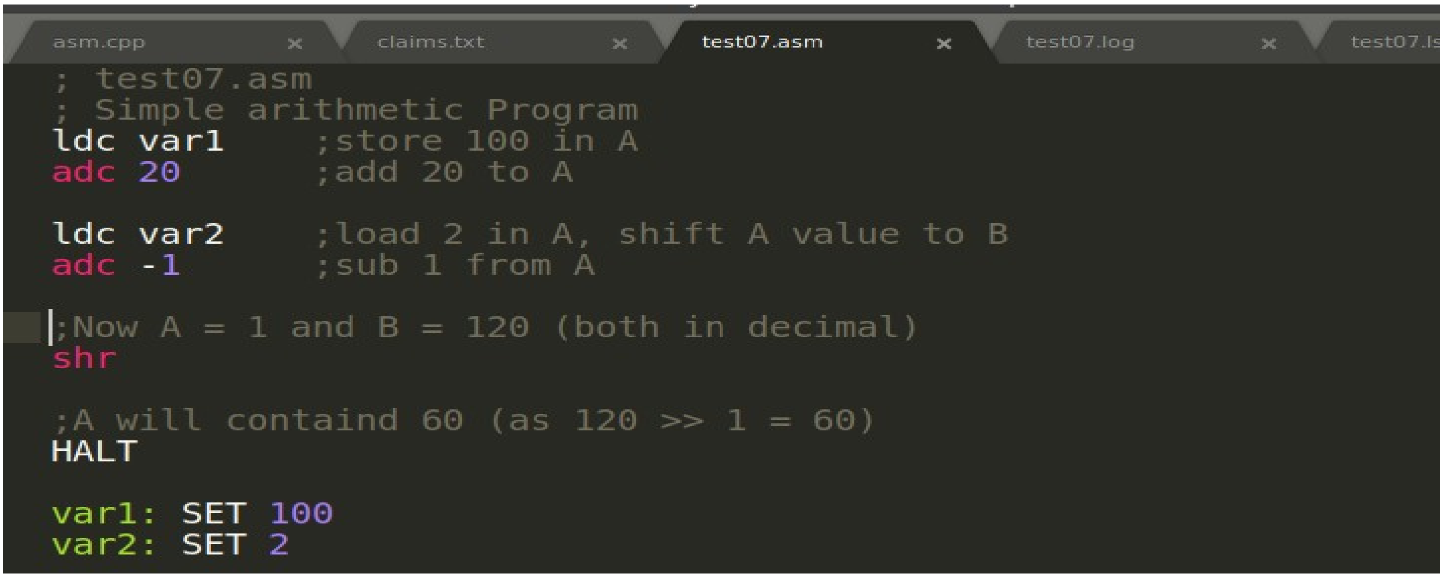
```
00000000 00001900 ldc var1
00000001 00000200 ldc var2
00000002 00000008 shl
00000003 00000012 HALT
00000004 00000019 var1: SET 25
00000005 00000002 var2: SET 2
```

Output from emulator:

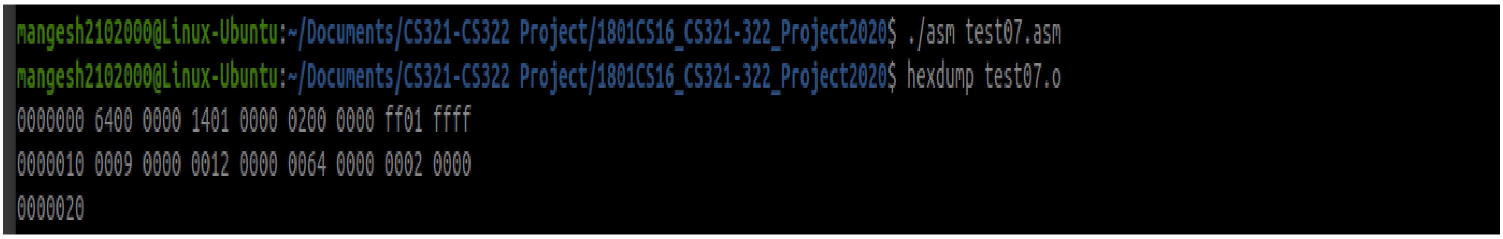


```
PC=00000000, SP=00000000, A=00000000, B=00000000, ldc 00000019
PC=00000001, SP=00000000, A=00000019, B=00000000, ldc 00000002
PC=00000002, SP=00000000, A=00000002, B=00000019, shl
PC=00000003, SP=00000000, A=00000064, B=00000019, HALT
Program Execution Finished

4 instructions executed
```

test07.asm

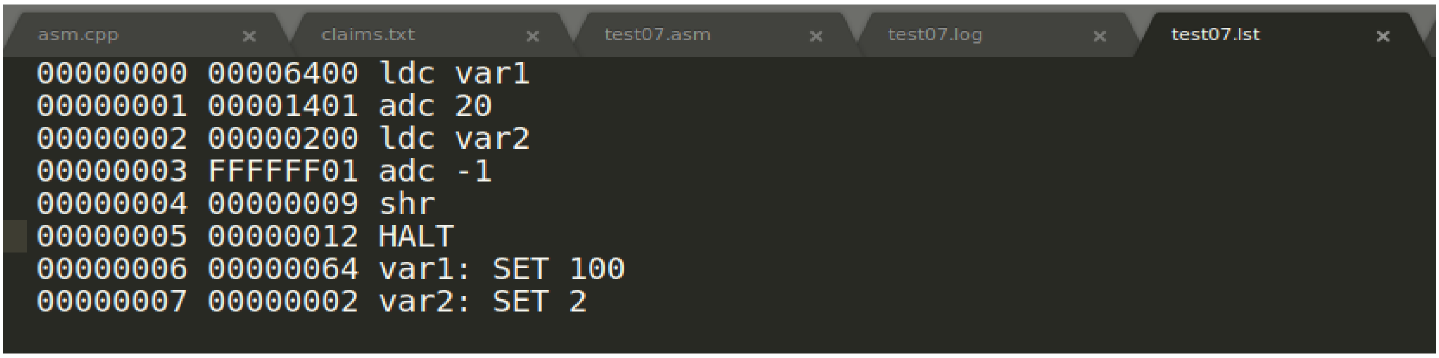


```
; test07.asm
; Simple arithmetic Program
ldc var1     ;store 100 in A
adc 20       ;add 20 to A

ldc var2     ;load 2 in A, shift A value to B
adc -1       ;sub 1 from A

;Now A = 1 and B = 120 (both in decimal)
shr

;A will containd 60 (as 120 >> 1 = 60)
HALT

var1: SET 100
var2: SET 2
```



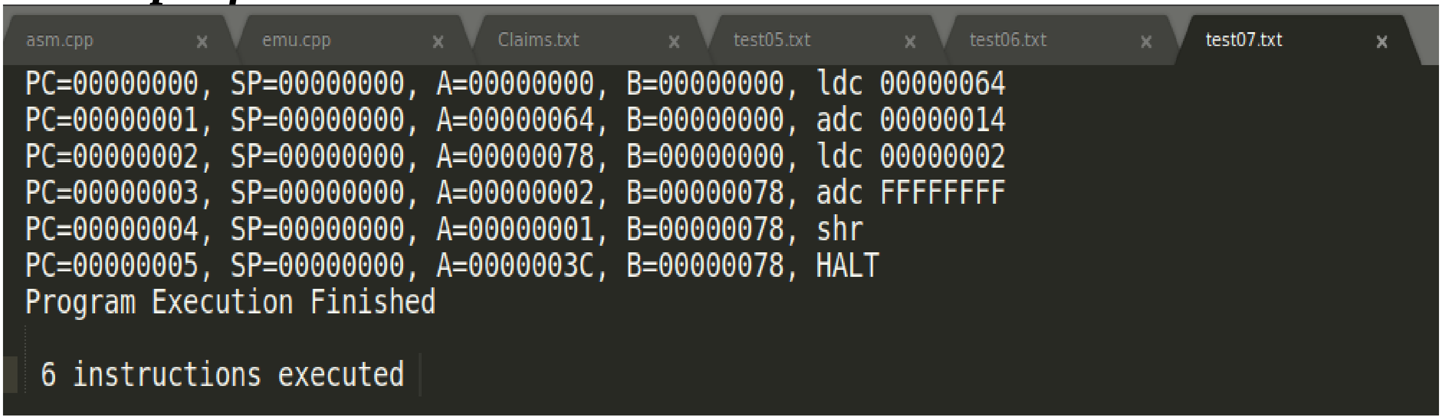
```
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ ./asm test07.asm
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ hexdump test07.o
0000000 6400 0000 1401 0000 0200 0000 ff01 ffff
0000010 0009 0000 0012 0000 0064 0000 0002 0000
0000020
```

test07.lst



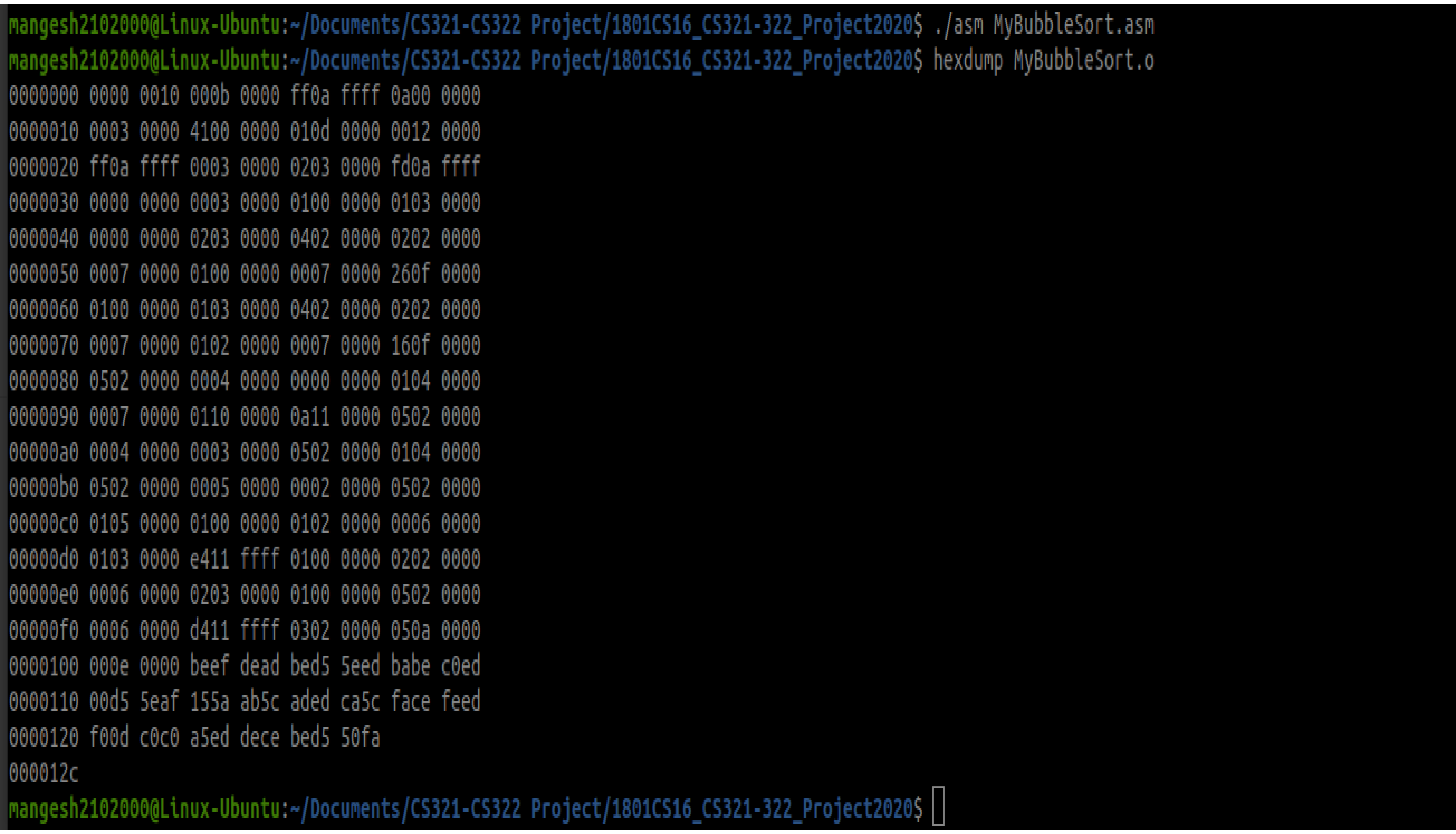
```
00000000 00006400 ldc var1
00000001 00001401 adc 20
00000002 00000200 ldc var2
00000003 FFFFFF01 adc -1
00000004 00000009 shr
00000005 00000012 HALT
00000006 00000064 var1: SET 100
00000007 00000002 var2: SET 2
```

Output from emulator:



```
PC=00000000, SP=00000000, A=00000000, B=00000000, ldc 00000064
PC=00000001, SP=00000000, A=00000064, B=00000000, adc 00000014
PC=00000002, SP=00000000, A=00000078, B=00000000, ldc 00000002
PC=00000003, SP=00000000, A=00000002, B=00000078, adc FFFFFFFF
PC=00000004, SP=00000000, A=00000001, B=00000078, shr
PC=00000005, SP=00000000, A=0000003C, B=00000078, HALT
Program Execution Finished

 6 instructions executed
```

MyBubbleSort.asm



```
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ ./asm MyBubbleSort.asm
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ hexdump MyBubbleSort.o
0000000 0000 0010 000b 0000 ff0a ffff 0a00 0000
0000010 0003 0000 4100 0000 010d 0000 0012 0000
0000020 ff0a ffff 0003 0000 0203 0000 fd0a ffff
0000030 0000 0000 0003 0000 0100 0000 0103 0000
0000040 0000 0000 0203 0000 0402 0000 0202 0000
0000050 0007 0000 0100 0000 0007 0000 260f 0000
0000060 0100 0000 0103 0000 0402 0000 0202 0000
0000070 0007 0000 0102 0000 0007 0000 160f 0000
0000080 0502 0000 0004 0000 0000 0000 0104 0000
0000090 0007 0000 0110 0000 0a11 0000 0502 0000
00000a0 0004 0000 0003 0000 0502 0000 0104 0000
00000b0 0502 0000 0005 0000 0002 0000 0502 0000
00000c0 0105 0000 0100 0000 0102 0000 0006 0000
00000d0 0103 0000 e411 ffff 0100 0000 0202 0000
00000e0 0006 0000 0203 0000 0100 0000 0502 0000
00000f0 0006 0000 d411 ffff 0302 0000 050a 0000
0000100 000e 0000 beef dead bed5 5eed babe c0ed
0000110 00d5 5eaf 155a ab5c aded ca5c face feed
0000120 f00d c0c0 a5ed dece bed5 50fa
000012c
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/1801CS16_CS321-322_Project2020$ 
```

MyBubbleSort.lst, MyBubbleSort.log and MyBubbleSort.o are present in the zip.

Emulator :

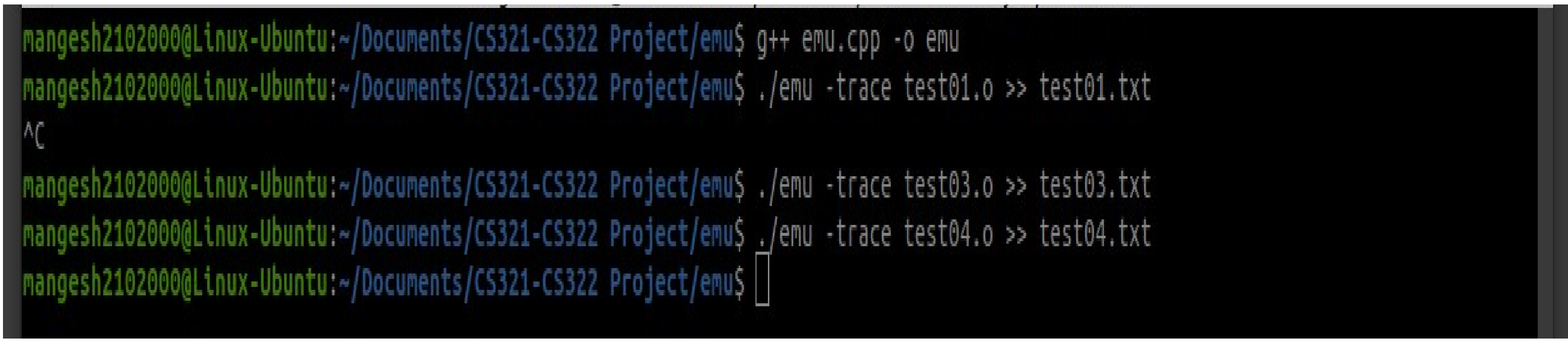
Instuctions to complie and run test files (Please keep all files in the same folder)

First Complie the Emulator using following instruction:

To Compile : **g++ emu.cpp -o emu**

To Run : **./emu [options] filename.o**
-trace show instruction trace
-read   show memory reads
-write show memory writes
-before show memory dump before execution
-after show memory dump after execution
-wipe   wipe written flags before execution
-isa   display ISA

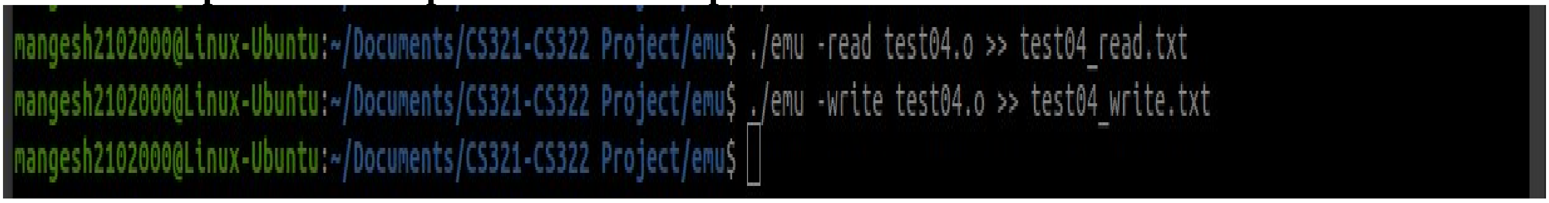
Sample For trace option (Files for test05,test06,test07 are also shown above)



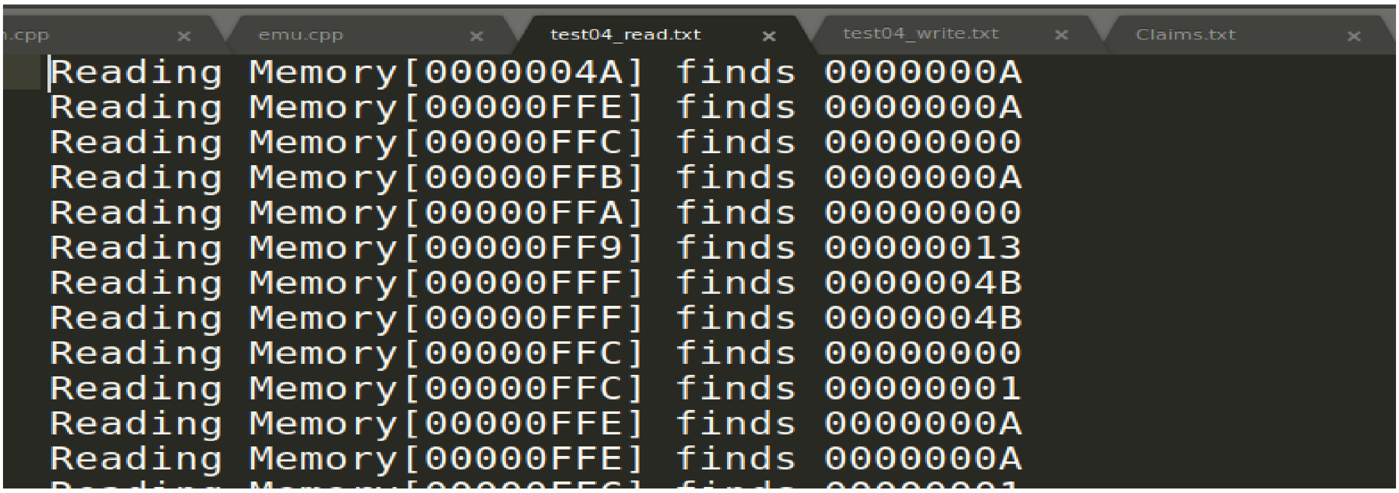
```
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/emu$ g++ emu.cpp -o emu
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/emu$ ./emu -trace test01.o >> test01.txt
^C
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/emu$ ./emu -trace test03.o >> test03.txt
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/emu$ ./emu -trace test04.o >> test04.txt
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/emu$ 
```

Files for all examples are present in the zip

Sample For read option and write option

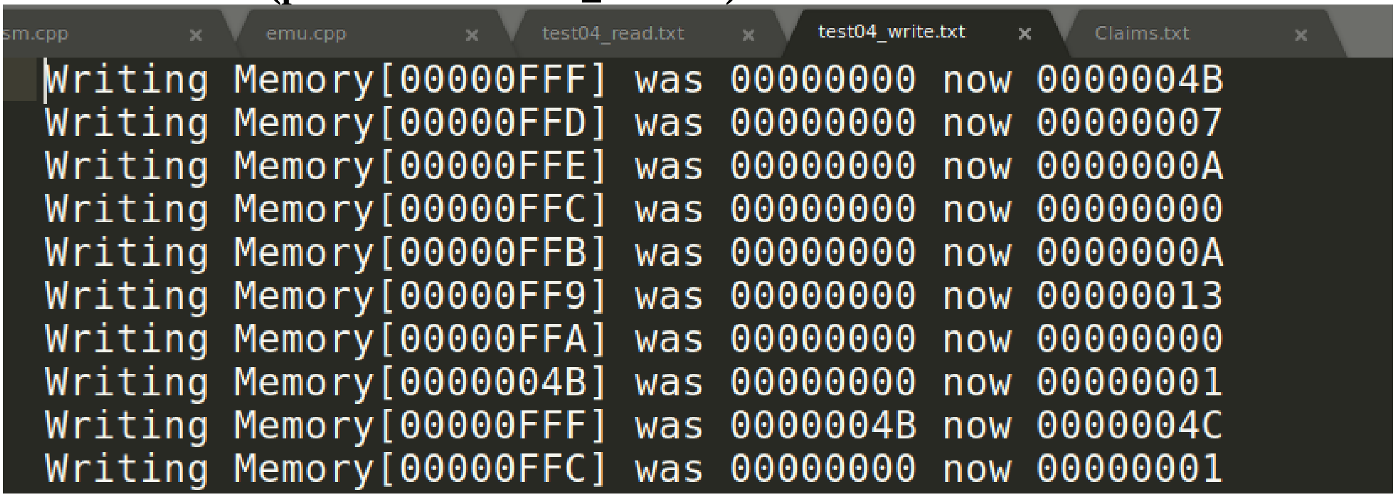


```
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/emu$ ./emu -read test04.o >> test04_read.txt
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/emu$ ./emu -write test04.o >> test04_write.txt
mangesh2102000@Linux-Ubuntu:~/Documents/CS321-CS322 Project/emu$ 
```



```
Reading Memory[0000004A] finds 0000000A
Reading Memory[00000FFE] finds 0000000A
Reading Memory[00000FFC] finds 00000000
Reading Memory[00000FFB] finds 0000000A
Reading Memory[00000FFA] finds 00000000
Reading Memory[00000FF9] finds 00000013
Reading Memory[00000FFF] finds 0000004B
Reading Memory[00000FFF] finds 0000004B
Reading Memory[00000FFC] finds 00000000
Reading Memory[00000FFC] finds 00000001
Reading Memory[00000FFE] finds 0000000A
Reading Memory[00000FFE] finds 0000000A
Reading Memory[00000FFC] finds 00000001
```

.... continued (please check test04_read.txt)



```
Writing Memory[00000FFF] was 00000000 now 0000004B
Writing Memory[00000FFD] was 00000000 now 00000007
Writing Memory[00000FFE] was 00000000 now 0000000A
Writing Memory[00000FFC] was 00000000 now 00000000
Writing Memory[00000FFB] was 00000000 now 0000000A
Writing Memory[00000FF9] was 00000000 now 00000013
Writing Memory[00000FFA] was 00000000 now 00000000
Writing Memory[0000004B] was 00000000 now 00000001
Writing Memory[00000FFF] was 0000004B now 0000004C
Writing Memory[00000FFC] was 00000000 now 00000001
```

..... continued (please check test04_write.txt)

All other options can be checked similarly.