EG-211 Computer Architecture

Implementation of IAS Machine using Python

Ishan Shanware (IMT2019037)

Introduction

This is a python program, to show the basic working of an IAS Machine. I have implemented 13 Instructions from the IAS Instruction set.

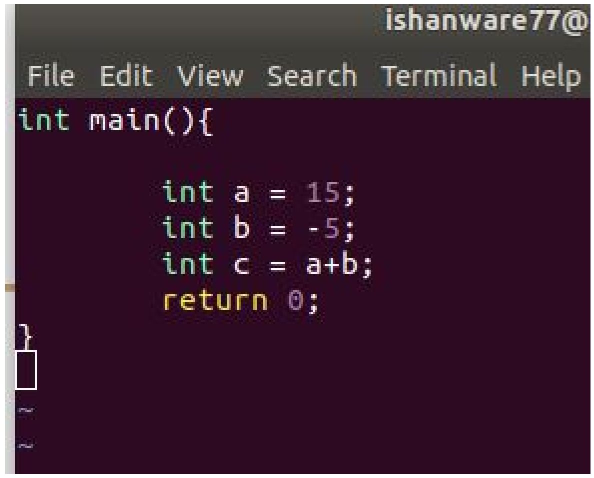
The instruction which my program can decode and execute are:

1. **LOAD M(X)**
2. **LOAD -M(X)**
3. **LOAD M(X), MQ**
4. **LOAD MQ**
5. **STOR M(X)**
6. **ADD M(X)**
7. **SUB M(X)**
8. **MUL M(X)**
9. **DIV M(X)**
10. **JUMP + M(X, 0:19)**
11. **JUMP + M(X, 20:39)**
12. **JUMP M(X, 0:19)**
13. **JUMP M(X, 20:39)**

Sample Programs Implemented

These are the sample programs I have implemented and their corresponding assembly instructions and Binary code.

1. **input1.txt**



Assembly code:

139 Left Null LOAD M(250)

140 Left Null ADD M(251)

141 Left Null STOR M(252)

142 HALT 0000000000

250 15(in Binary)

251 -5 (in Binary)

252 0 (in Binary)

Binary Code:

139 0000000000000000000000000001000011111010

140 0000000000000000000000000101000011111011

141 0000000000000000000000100001000011111100

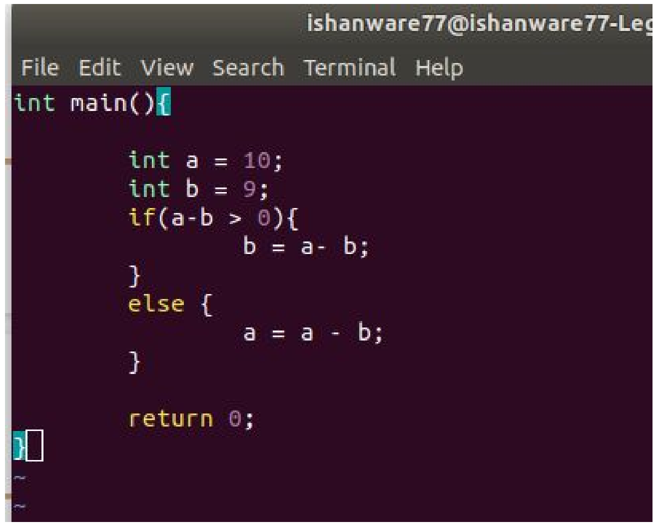
142 1111111100000000000000000000000000000000

250 0000000000000000000000000000000000001111

251 1111111111111111111111111111111111111011

252 0000000000000000000000000000000000000000

2. input2.txt



Assembly Code:

120 LOAD M(200) SUB M(201)

121 LEFT NULL JUMP +M(150, 0:19)

122 STOR M(200) HALT

150 STOR M(201) HALT

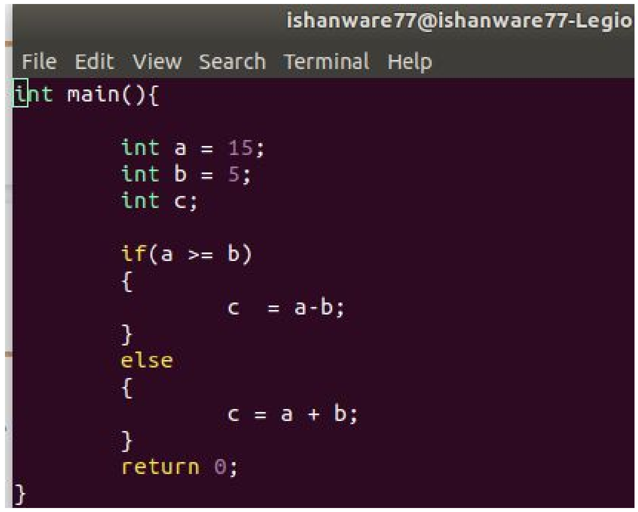
200 10 (in Binary)

201 9 (in Binary)

Binary Code:

120 00000001000011001000000000110000011001001
121 00000000000000000000000001111000010010110
122 0010000100001100100011111111000000000000
150 00100001000011001001111111111000000000000
200 0000000000000000000000000000000000001010
201 0000000000000000000000000000000000001001

3. input3.txt



Assembly code:

50 LOAD M(100) SUB M(101)

51 JUMP +M(70, 0:19) ADD M(101)

52 ADD M(101) STOR M(102)

53 Left Null HALT

70 STOR M(102) HALT

100 15 (in Binary)

101 5 (in Binary)

102 garbage (in Binary)

Binary code:

50 0000000100000110010000000110000001100101

51 0000111100000100011000000101000001100101

52 0000010100000110010100100001000001100110

53 0000000000000000000011111111000000000000

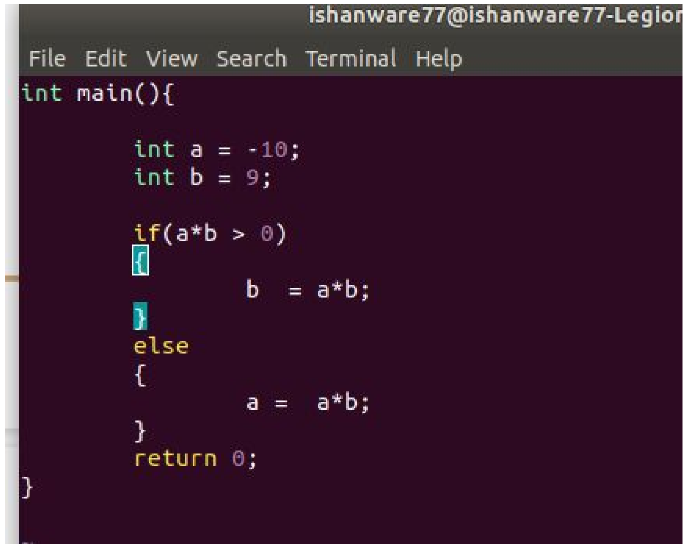
70 0010000100000110011011111111000000000000

100 0000000000000000000000000000000000001111

101 0000000000000000000000000000000000000101

102 0000000000000000000000000000000000000000

4. input4.txt




```
int main(){

        int a = -10;
        int b = 9;

        if(a*b > 0)
        {
                b  = a*b;
        }
        else
        {
                a =  a*b;
        }
        return 0;
}
```

Assembly Code:

120	**LOAD M(200), MQ**	**MUL M(201)**
121	**LOAD MQ**	**JUMP + M(150, 20:39)**
122	**STOR M(200)**	**HALT**
150	**Left Null**	**STOR M(201)**
151	**HALT**	**000000000000000000**
200	**10 (in Binary)**	
201	**9 (in Binary)**	

Binary Code :

120 00001001000011001000000001011000011001001

121 0000101000000000000000001000000000010010110

122 0010000100001100100011111111000000000000

150 00000000000000000000000100001000011001001

151 1111111100000000000000000000000000000000

200 0000000000000000000000000000000000001010

201 0000000000000000000000000000000000001001

Output Format:

Below given is the output printed in the terminal when input3.txt is used as input.

Current PC = 50

FETCHING INSTRUCTION

AC =

MQ =

MBR = 0000000100000110010000000110000001100101

IBR = 00000110000001100101

IR = 00000001 Decimal value of IR = 1

MAR = 000001100100 Decimal value of MAR = 100

PC = 50

DECODE INSTRUCTION

LOAD M(100)

EXECUTE CHANGES

AC = 0000000000000000000000000000000000001111 Decimal value of AC = 15

MQ =

MBR = 0000000000000000000000000000000000001111

IBR = 00000110000001100101

IR = 00000001 Decimal value of IR = 1

MAR = 000001100100 Decimal value of MAR = 100

PC = 50

End of cycle --

Current PC = 50

FETCHING INSTRUCTION

AC = 0000000000000000000000000000000000001111 Decimal value of AC = 15

MQ =

MBR = 0000000000000000000000000000000000001111

IBR =

IR = 00000110 Decimal value of IR = 6

MAR = 000001100101 Decimal value of MAR = 101

PC = 51

DECODE INSTRUCTION

SUB M(101)

EXECUTE CHANGES

AC = 0000000000000000000000000000000000001010 Decimal value of AC = 10

MQ =

MBR = 0000000000000000000000000000000000000101

IBR =

IR = 00000110 Decimal value of IR = 6

MAR = 000001100101 Decimal value of MAR = 101

PC = 51

End of cycle --

Current PC = 51

FETCHING INSTRUCTION

AC = 0000000000000000000000000000000000001010 Decimal value of AC = 10

MQ =

MBR = 0000111100000100011000000101000001100101

IBR = 00000101000001100101

IR = 00001111 Decimal value of IR = 15

MAR = 000001000110 Decimal value of MAR = 70

PC = 51

DECODE INSTRUCTION
JUMP +M(70, 0:19)

EXECUTE CHANGES
AC = 0000000000000000000000000000000000001010 Decimal value of AC = 10
MQ =
MBR = 0000111100000100011000000101000001100101
IBR =
IR = 00001111 Decimal value of IR = 15
MAR = 000001000110 Decimal value of MAR = 70
PC = 70

End of cycle --

Current PC = 70

FETCHING INSTRUCTION
AC = 0000000000000000000000000000000000001010 Decimal value of AC = 10
MQ =
MBR = 0010000100000110011011111111000000000000
IBR = 11111111000000000000
IR = 00100001 Decimal value of IR = 33
MAR = 000001100110 Decimal value of MAR = 102
PC = 70

DECODE INSTRUCTION
STOR M(102)

Memory at 102 changed to 0000000000000000000000000000000000001010

EXECUTE CHANGES
AC =

MQ =

MBR = 0010000100000110011011111111000000000000

IBR = 11111111000000000000

IR = 00100001 Decimal value of IR = 33

MAR = 000001100110 Decimal value of MAR = 102

PC = 70

End of cycle ---

Current PC = 70

FETCHING INSTRUCTION

AC =

MQ =

MBR = 0010000100000110011011111111000000000000

IBR =

IR = 11111111 Decimal value of IR = 255

MAR = 000000000000 Decimal value of MAR = 0

PC = 71

DECODE INSTRUCTION

HALT

End of cycle ---

For each set of instructions, The output prints the changes after each Fetch and execution cycle.

The content AC, MQ, MBR, IBR, IR, MAR, PC is printed for each fetch and execution cycle along with decoding the opcode.

Whenever any of AC, MQ, MBR, IBR, IR, MAR, PC is empty , the other side of “=” is shown blank, indicating it is empty.

Eg: AC =

Means that AC is empty.

PC is taken as decimal for easier implementation, AC, MQ, MBR, IBR, IR, MAR are strings.

Along with this output in the terminal, an out.txt file is generated which prints the content of 1000*40 memory, to show the changes made after the set of instructions were executed.

memory.txt contains the content of 1000*40 memory before the instructions were executed by the IAS Machine.

For example: If the set of instructions passed contains a store instruction, you can check if the IAS Machine made the changes at the location specified in out.txt

Instructions to Run project.

1. To pass in a set of Instructions, type the instructions in a text file .

   The format is exactly the same in which the binary code is mentioned above.

   Ensure that you specify the position of the instruction in memory.

   For eg:

50   0000000100000011001000000001100000001100101
51   0000111100000010001100000001010000001100101
52   00000101000000110010100100001000001100110
53   0000000000000000000011111111000000000000
70   00100001000000110011011111111000000000000
100 0000000000000000000000000000000000001111
101 0000000000000000000000000000000000000101
102 0000000000000000000000000000000000000000

Type the position in memory between [1, 1000] (I have followed 1 based indexing) then the corresponding instruction in binary.

Eg: 122 0010000100001100100011111111000000000000

The first 4 digits are for position of instruction in memory in decimal, then after a space type in instruction in binary.

Follow this for each line.

(if position is 1 digit or 2 digit or 3 digit add space to fill up space until 4 then add a space to type in the instruction.)

(basically first 4 bits reserved for position and then a space after which comes 40 bit instruction or data.)

eg:

12   0000000000000000000000000000000000000000 (notice 3 spaces after 12)

1000 0000000000000000000000000000000000000000 (notice 1 space after 1000)

999 0000000000000000000000000000000000000000 (notice that 2 spaces after 999)

You need to input a signed binary for data , specifying the position in memory first.

Ensure that the input given in the text file always begins with Instructions and not data.

2. Next specify the name of the file in the “fillmemory.py” python file, by making changes to the top most line, i.e,

```
file1 = open("./input3.txt",'r')
```

3. You are all done, to run the project:
    a) Run the “main.py” file with Python3. This would show the output on the terminal as shown before, and generate “memory.txt”

(containing the content of memory before the instructions are executed) and “out.txt” (which contains the content of memory after the set of instructions were executed by the IAS Machine.

b) Run the “fillmemory.py” file with Python3 first, to generate the memory, this would also make the memory.txt file (containing content of memory before execution of instructions begin). Then run “main.py” file to execute the instructions, this would print output in the terminal, and also generate the “out.txt” file containing the content of memory after instructions were executed by IAS.

Purpose of Files:

1. **fillmemory.py** : reads instructions from input txt file and creates a 1000*40 bit memory. I am using an array of strings as the memory. Length of each bit is 40.

2. **bintodec.py**: This file contains functions that convert Binary to Decimal, one function converts unsigned binary to decimal, and the other converts signed binary to decimal.

3. **dectobin.py** : This file contains functions that convert Decimal to binary, one function converts Decimal to unsigned binary , while the other converts Decimal to a 40 bit signed binary number(this is mostly used to manipulate the strings for ADD, SUB, MUL and DIV instructions)

4. **main.py**: contains the main loop for fetch, decode and execute cycle. Instructions are executed using a while loop, until a HALT instruction is encountered.

Created after running python files:

5. **memory.txt**: made by fillmemory script, shows content of memory before instructions are executed.

6. **out.txt**: shows content of memory after all the instructions given as input are executed, in the way they are intended to be executed.

The zip file contains the output images as well in Output_Images Directory(for terminal output of input1.txt, input2.txt, input3.txt)

extra_sample_inputs directory contains a bunch of other input txt files made while testing the code.

Assumptions

1. **Opcode for HALT Instruction is 11111111.**
2. **After JUMP M(X, 0:19) or JUMP + M(X, 0:19) , PC is incremented to specified location(X), after executing the left instruction first, the right instruction is executed.**
3. **Memory following 1 based indexing, as in PC can have the smallest value as 1. Memory has indexes from 1 to 1000.**

Ishan Shanware

(IMT2019037)