Introduction to Low Level Programming

The P88 simulator used demonstrated in class allows programmers to write and execute low level program.
Low level programming languages use a syntax that is very close to machine-level languages - that is, it is a format that has very little abstraction from how the machine operates.

A high-level programming language on the other hand, uses a an abstract syntax that is closer to a spoken language, and where a single high level program statement may actually have to be implemented in many separate machine-level instructions. The high-level programming languages are easier to program because the provide an abstract view of how the computer operates.

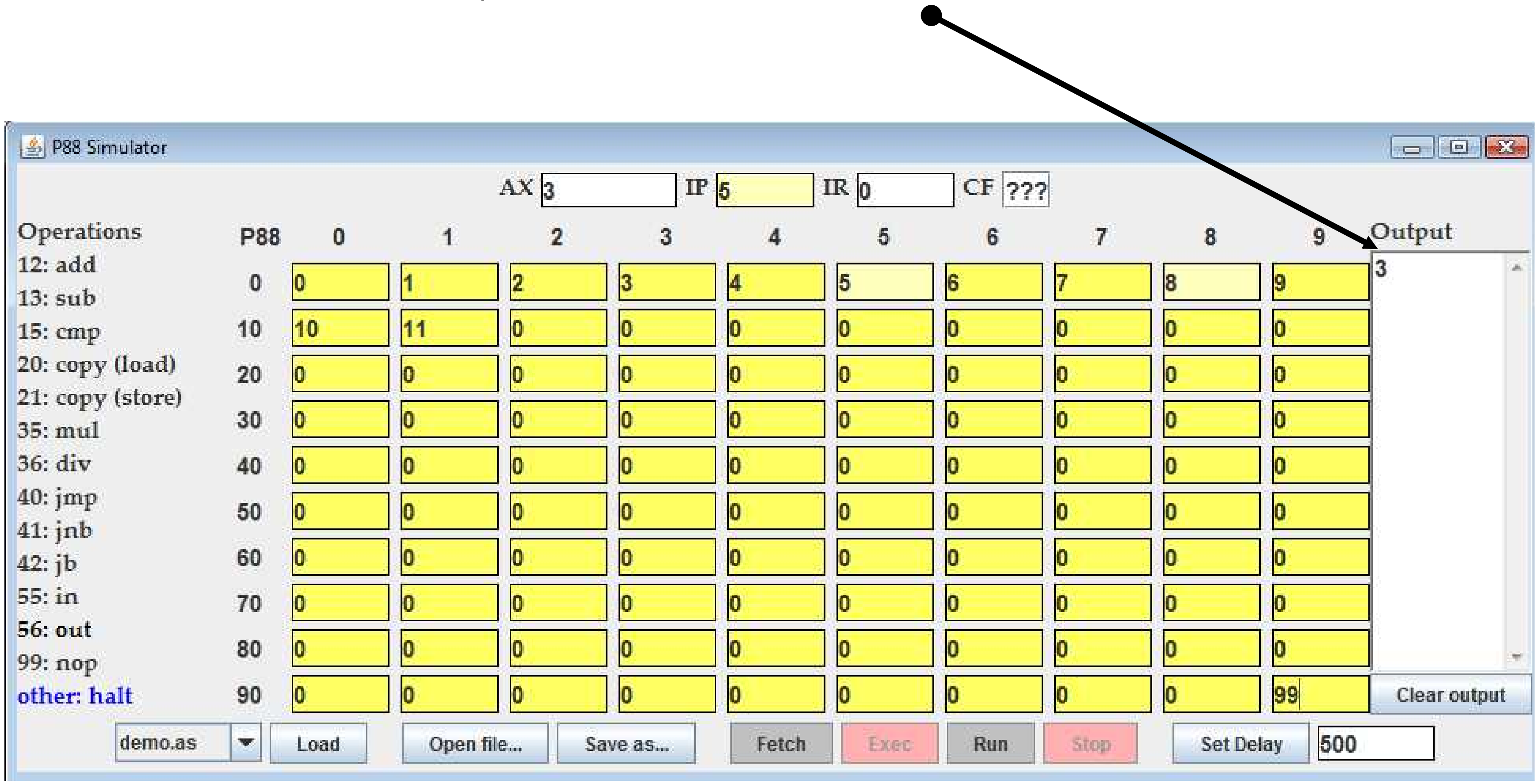
The P88 simulator contains the following components:

There are 100 memory locations labeled along the top x-axis and left side axis as locations 00 - 99. Memory locations 00 through 11 and memory location 99 have the memory address stored as their values and you can see that I've set all other memory locations to 0.

When a program is loaded into memory, or you write a program, the instructions to be executed as well as the data used and stored by your program will be in these memory locations.

It is good programming practice to keep the instructions in one area, and the data in another.

You will also see that there is an output area which will be used when a out command is executed.



AX 3 IP 5 IR 0 CF ???

Operations
12: add
13: sub
15: cmp
20: copy (load)
21: copy (store)
35: mul
36: div
40: jmp
41: jnb
42: jb
55: in
56: out
99: nop
other: halt

P88	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
10	10	11	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0	0	99

Output
3

500

The P88 simulator contains the following components:

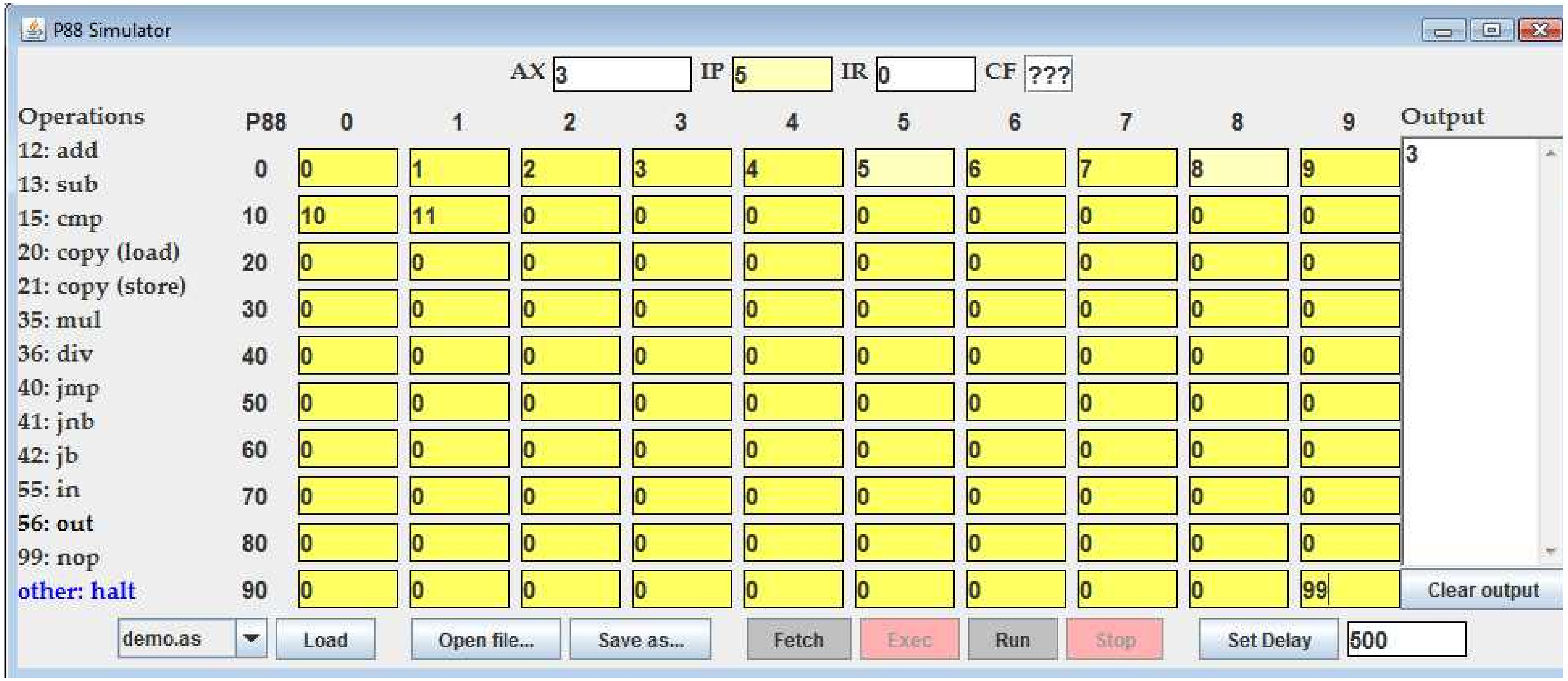
The AX register, which was referred to as the "accumulator" is where computations take place. This is also called the Computation Register, and for the purpose of this computer, can also be thought of as the Arithmetic Logic Unit which will be discussed in greater detail in future classes.

The IP register is the Instruction Pointer - it will contain the address (00 to 99) of the next instruction that will be fetched and executed.

The IR register is the Instruction Register - this contains the current instruction to execute.

When the Fetch button is pressed, the instruction identified by the IP will be loaded into the IR.

The CF register is the Condition Flag register - it will contain the results of a comparison operation.



AX 3 IP 5 IR 0 CF ???

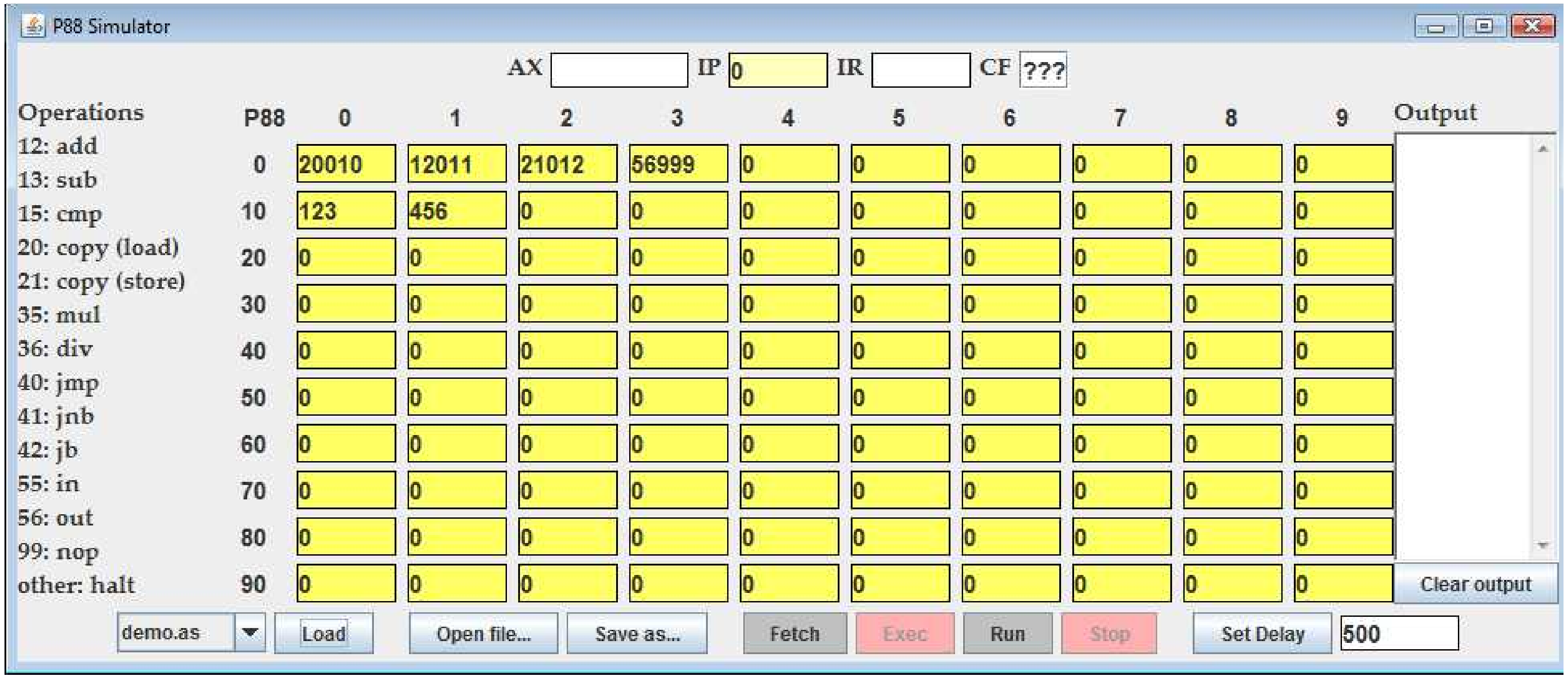
Operations
12: add
13: sub
15: cmp
20: copy (load)
21: copy (store)
35: mul
36: div
40: jmp
41: jnb
42: jb
55: in
56: out
99: nop
other: halt

P88	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
10	10	11	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0	0	99

Output
3

demo.as

Set Delay 500



AX | IP 0 | IR | CF ???

Operations
12: add
13: sub
15: cmp
20: copy (load)
21: copy (store)
35: mul
36: div
40: jmp
41: jnb
42: jb
55: in
56: out
99: nop
other: halt

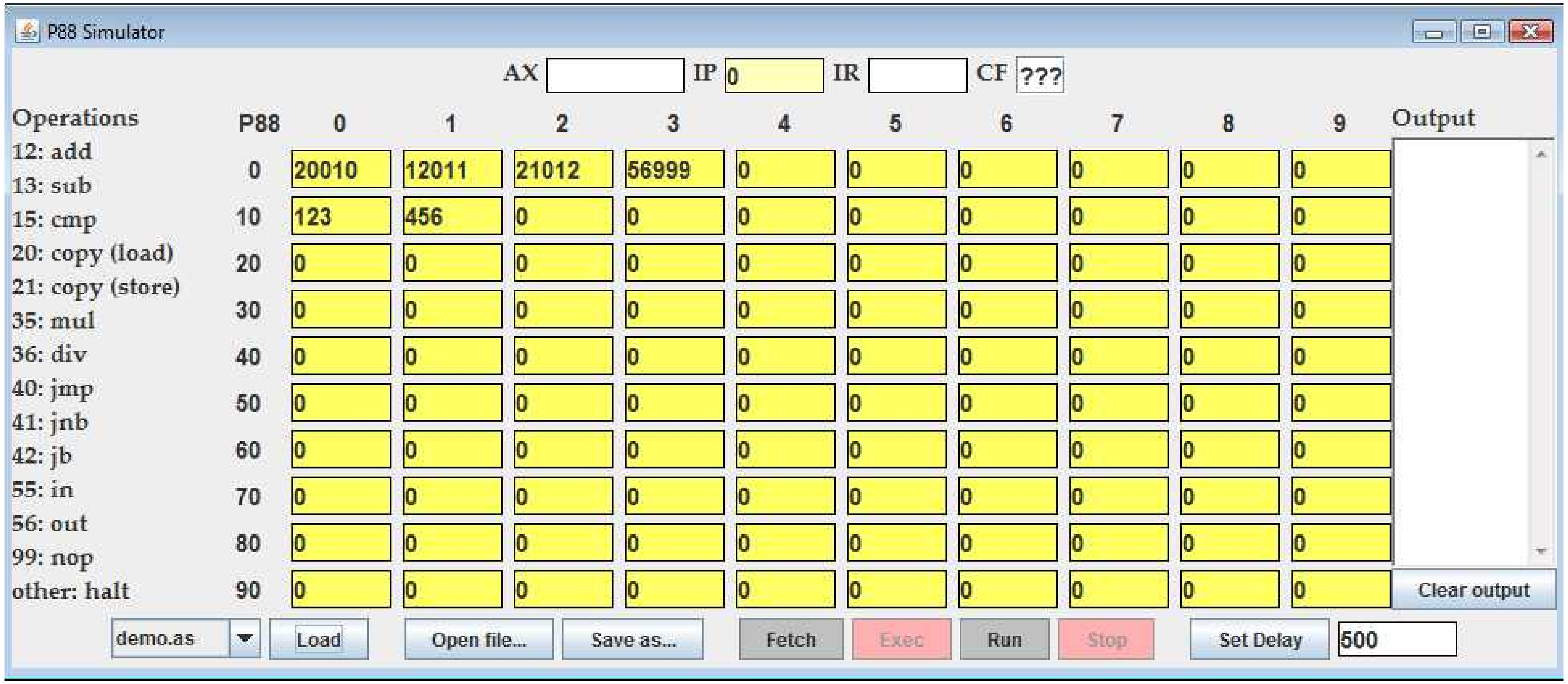
P88	0	1	2	3	4	5	6	7	8	9
0	20010	12011	21012	56999	0	0	0	0	0	0
10	123	456	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0	0	0

Set Delay: 500

This is the demo.as program, it uses memory locations 00 to 04 to hold the program statements to be executed, and it uses memory locations 10 to 12 to hold data (variables).

To begin the program, a 0 is loaded into the IP register, this is where the next instruction (or first instruction in this case) is located. Take a look at the program, and decode it using the operations table listed above on the left.

Address	Contents	Command	
00	20 010	copy (load) 10	means take the value that is in memory location 10 and load it into the AX register. The value at location 10 is 123.
01	12 011	add 011	means take the value that is in memory location 11 and add it to what is in the AX register. The value at location 11 is 456, so the new value in the AX register is 123 + 456 = 579
02	21 012	copy (store) 12	means take the value in the AX register and store in location 12
03	56 999	out	means take the value in the AX register and copy to the output
04	0	not defined	means terminate the program, the operation is not defined.



AX IP 0 IR CF ???

Operations
12: add
13: sub
15: cmp
20: copy (load)
21: copy (store)
35: mul
36: div
40: jmp
41: jnb
42: jb
55: in
56: out
99: nop
other: halt

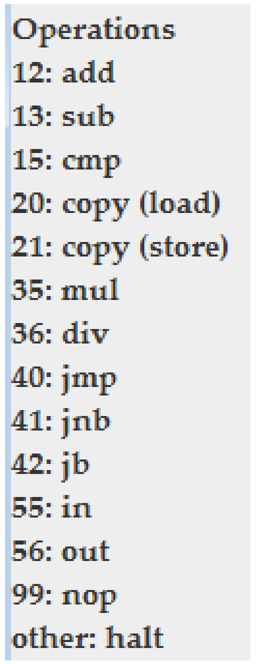
P88	0	1	2	3	4	5	6	7	8	9
0	20010	12011	21012	56999	0	0	0	0	0	0
10	123	456	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0	0	0

demo.as

Set Delay 500

You can use the pull-down menu to select different programs that are available, and the load them into memory. You can then decode them the way the demo.as program was decoded, or you can step through the program one instruction at a time using "Fetch" and then "Execute". You can also select Run which will step through the program automatically. Use the "Set Delay" to set the number of milliseconds between instructions.

Ready to write a program?



Operations
12: add
13: sub
15: cmp
20: copy (load)
21: copy (store)
35: mul
36: div
40: jmp
41: jnb
42: jb
55: in
56: out
99: nop
other: halt

The first step should be to write a high-level abstraction of what you would like the program to do, and how it should do it. Pseudo-code

Next, write the instructions (in English) using the operations that are available in the P88 language (see left). We'll refer to this as assembly language.

Next, translate the instructions into the machine code that can be loaded into the memory sells. I good guideline is to start the IP with value 0, and put the first instruction in address 0.

Then save your file, test it, make any necessary changes, and submit it for grading, along with the Pseudocode and assembly code.

Write a program:

If you have an idea for a program then feel free to implement it; be sure to submit the Pseudocode, assembly code, and P88.as file; otherwise try to complete the "absolute value" program started below:

The Absolute value Program
Description: prompt the user for an integer. Output the absolute value of the integer entered by the user.

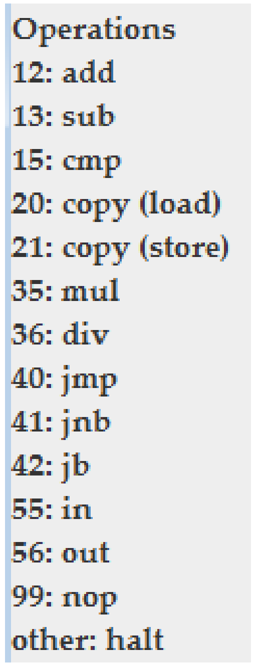
Pseudo-code

```
Read x

if x < 0 then
   x = x * -1
   output x

else
   output x

end program
```



Operations
12: add
13: sub
15: cmp
20: copy (load)
21: copy (store)
35: mul
36: div
40: jmp
41: jnb
42: jb
55: in
56: out
99: nop
other: halt

Assembly

```
00   in

01   cmp 50
02   jnb 004

03   mult 51

04   56

05   0


50   0
51   -1
```

store 0 in memory 50 and store -1 in location 51 when you are entering the program code

Machine Code

Your task is to translate the assembly code into machine code and put it into the P88 simulator.