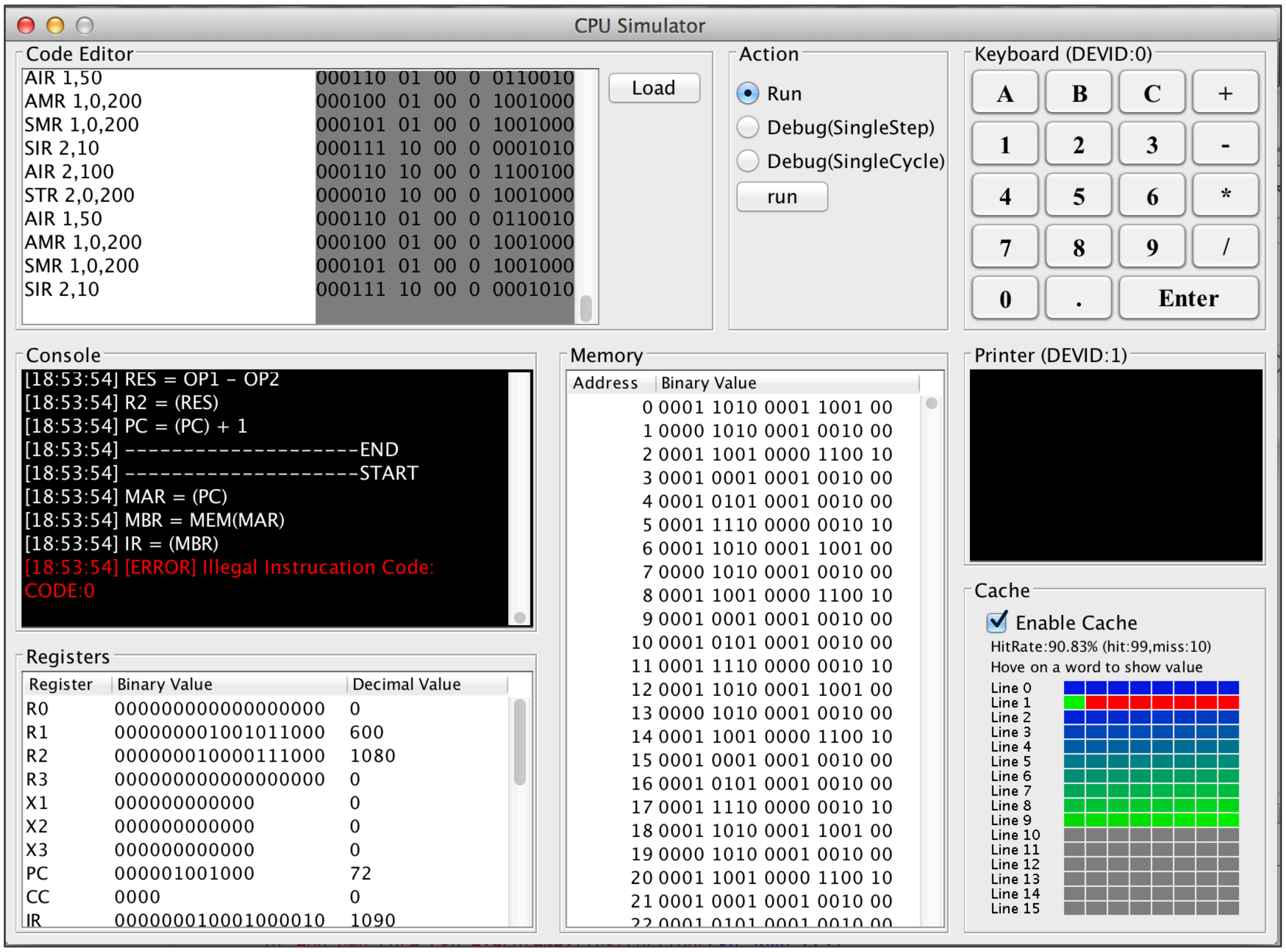
User’s Manual

CS6461 CPUSimulator Part 2

Group 9

1.INTRODUCTION

Console Layout and features



```
AIR 1,50        000110 01 00 0 0110010
AMR 1,0,200     000100 01 00 0 1001000
SMR 1,0,200     000101 01 00 0 1001000
SIR 2,10        000111 10 00 0 0001010
AIR 2,100       000110 10 00 0 1100100
STR 2,0,200     000010 10 00 0 1001000
AIR 1,50        000110 01 00 0 0110010
AMR 1,0,200     000100 01 00 0 1001000
SMR 1,0,200     000101 01 00 0 1001000
SIR 2,10        000111 10 00 0 0001010
```

[✓ Run] Debug(SingleStep) Debug(SingleCycle)

```
[18:53:54] RES = OP1 - OP2
[18:53:54] R2 = (RES)
[18:53:54] PC = (PC) + 1
[18:53:54] --------------------END
[18:53:54] --------------------START
[18:53:54] MAR = (PC)
[18:53:54] MBR = MEM(MAR)
[18:53:54] IR = (MBR)
[18:53:54] [ERROR] Illegal Instrucation Code:
CODE:0
```

Register	Binary Value	Decimal Value
R0	000000000000000000	0
R1	000000001001011000	600
R2	000000010000111000	1080
R3	000000000000000000	0
X1	000000000000	0
X2	000000000000	0
X3	000000000000	0
PC	000001001000	72
CC	0000	0
IR	000000010001000010	1090

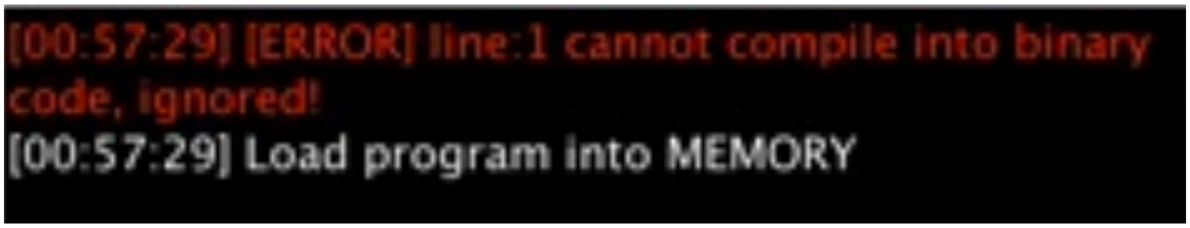
Address	Binary Value
0	0001 1010 0001 1001 00
1	0000 1010 0001 0010 00
2	0001 1001 0000 1100 10
3	0001 0001 0001 0010 00
4	0001 0101 0001 0010 00
5	0001 1110 0000 0010 10
6	0001 1010 0001 1001 00
7	0000 1010 0001 0010 00
8	0001 1001 0000 1100 10
9	0001 0001 0001 0010 00
10	0001 0101 0001 0010 00
11	0001 1110 0000 0010 10
12	0001 1010 0001 1001 00
13	0000 1010 0001 0010 00
14	0001 1001 0000 1100 10
15	0001 0001 0001 0010 00
16	0001 0101 0001 0010 00
17	0001 1110 0000 0010 10
18	0001 1010 0001 1001 00
19	0000 1010 0001 0010 00
20	0001 1001 0000 1100 10
21	0001 0001 0001 0010 00
22	0001 0101 0001 0010 00

[✓] Enable Cache
HitRate:90.83% (hit:99,miss:10)
Hove on a word to show value

Line 0
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
Line 11
Line 12
Line 13
Line 14
Line 15

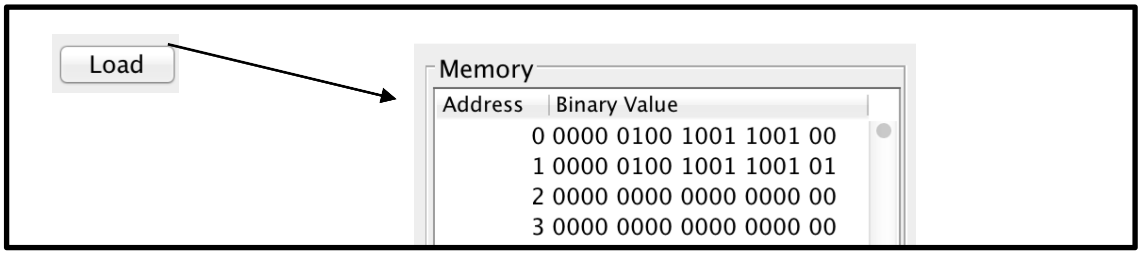
Our CPU Simulator consists of eight parts: Code Editor, Action, Console, Registers, Keyboard, Printer, Cache and Memory. Each part has specific functions and they work together as a completed CPU Simulator. The above graph is the screenshot of our CPU Simulator's interface.

Code Editor: In this part, users can enter their instructions into the editable text area, and the binary digit of the instruction will be displayed instantly besides the original one. The instruction should be in a format as Instruction plus parameters. If the instruction entered does not match this format, there will be an error reminder showed in the Console part. The notation will be displayed as the follows in the graph.



```
[00:57:29] [ERROR] line:1 cannot compile into binary
code, ignored!
[00:57:29] Load program into MEMORY
```

After entering all the instructions, the user can click the Load button on the right side of this part to load the instructions into memory. We set AMR, SMR, AIR and STR as the default instructions.



Address	Binary Value
0	0000 0100 1001 1001 00
1	0000 0100 1001 1001 01
2	0000 0000 0000 0000 00
3	0000 0000 0000 0000 00

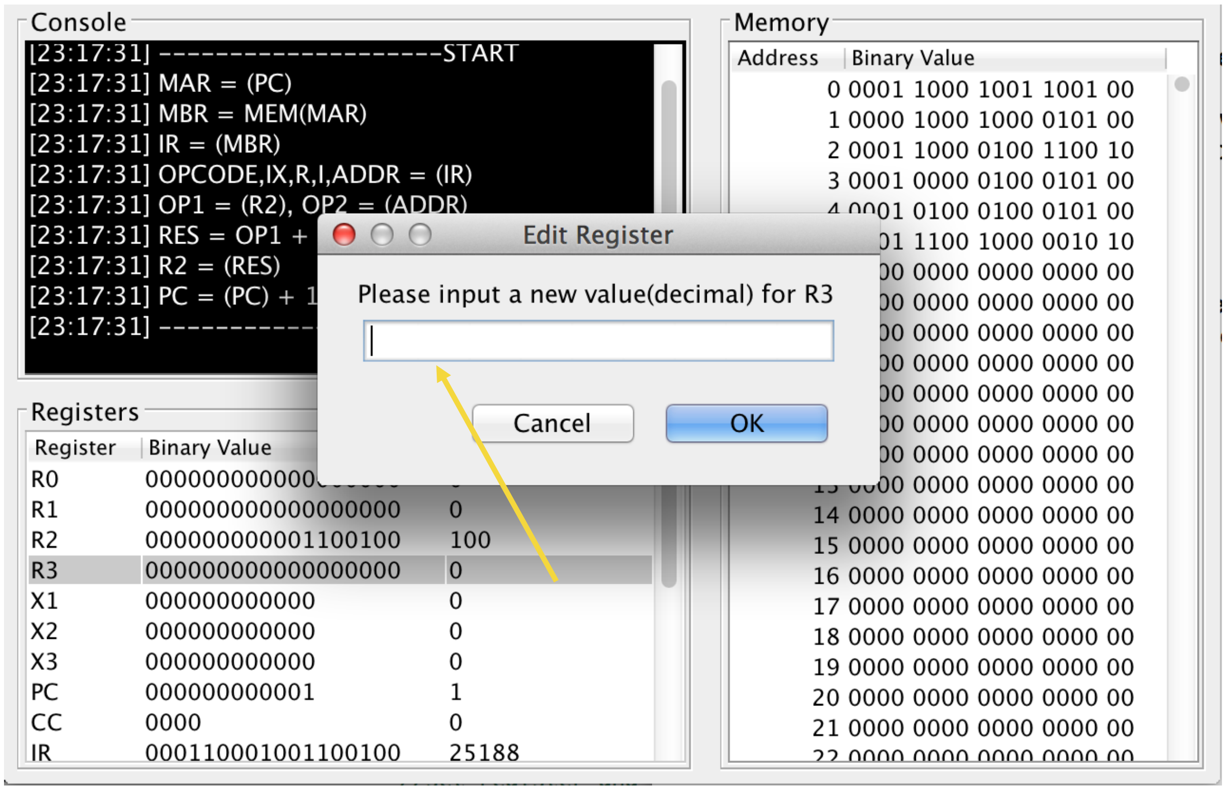
(in this version, we set PC=0 in the beginning)

Action: This part is designed for users to run instructions. There are three modes the users can choose. One is to run the whole program; this means all the instructions in the memory. Clicking the *Run* button will trigger this. Another one is to run a single instruction in a specific address. This address is determined by the current value of PC. The users can edit the value of PC through the Register part. We will talk about this in detail in the description of Register part. Clicking *Debug (SingleStep)* button can trigger this action. The third one is to run a single circle of a specific instruction. Clicking *Debug (SingleCycle)* button can trigger this action.

Console: This part will show what the CPU does after the users run the whole program or a single instruction step by step. In another word, it's the trace of executions of instructions.

Memory: In the Memory part, binary values will be displayed. They are related to the instructions, addresses and data that have been stored into the memory. Users can figure out what's exactly stored in the memory.

Registers: The users can find out the current binary values of different registers here. The users also can modify the value of a specific register by double click the editable text area besides that register. If the users are familiar with binary value, they can double click the text area in the Binary Value column. Then they can enter a binary value to change the current value of that register. The decimal value of that register will change automatically as the users modify the binary value. When it comes to the decimal value, the operation is similar with dealing with binary value.



```
[23:17:31] --------------------START
[23:17:31] MAR = (PC)
[23:17:31] MBR = MEM(MAR)
[23:17:31] IR = (MBR)
[23:17:31] OPCODE,IX,R,I,ADDR = (IR)
[23:17:31] OP1 = (R2), OP2 = (ADDR)
[23:17:31] RES = OP1 +
[23:17:31] R2 = (RES)
[23:17:31] PC = (PC) + 1
[23:17:31] ----------
```

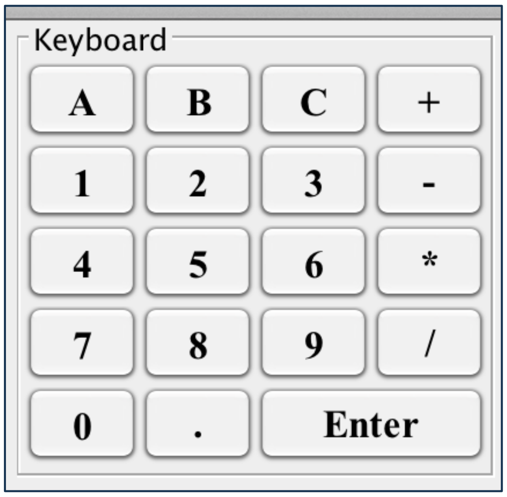
Please input a new value(decimal) for R3

Register	Binary Value	
R0	000000000000	
R1	0000000000000000000	0
R2	0000000000001100100	100
R3	0000000000000000000	0
X1	000000000000	0
X2	000000000000	0
X3	000000000000	0
PC	000000000001	1
CC	0000	0
IR	0001100010011001000	25188

Address	Binary Value
0	0001 1000 1001 1001 00
1	0000 1000 1000 0101 00
2	0001 1000 0100 1100 10
3	0001 0000 0100 0101 00
4	0001 0100 0100 0101 00
	01 1100 1000 0010 10
	00 0000 0000 0000 00
	00 0000 0000 0000 00
	00 0000 0000 0000 00
	00 0000 0000 0000 00
	00 0000 0000 0000 00
	00 0000 0000 0000 00
	00 0000 0000 0000 00
13	0000 0000 0000 0000 00
14	0000 0000 0000 0000 00
15	0000 0000 0000 0000 00
16	0000 0000 0000 0000 00
17	0000 0000 0000 0000 00
18	0000 0000 0000 0000 00
19	0000 0000 0000 0000 00
20	0000 0000 0000 0000 00
21	0000 0000 0000 0000 00
22	0000 0000 0000 0000 00

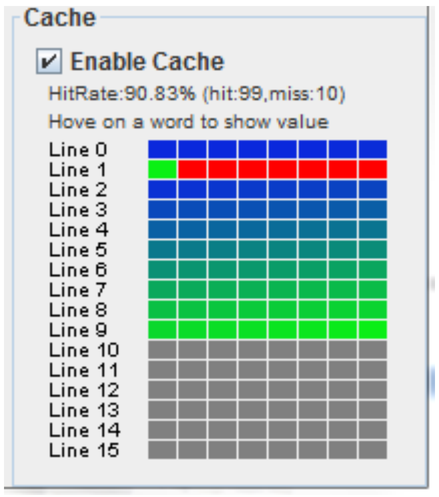
(double click in Binary Value area can make POPUP window into binary input mode, Decimal Value area for decimal input mode)

Keyboard: The users can use this keyboard simulator to enter numbers and some characters.



Printer: This field will display the results of users' operations.

Cache: This field will show the status of the cache line by using different colors during the execution process. Red color means cache-miss, and green color means cache-hit. If a word in cache has not been visited recently, it will become blue. The Cache is designed with 16 lines and each line contains 8 words. The Cache has a selective Enable Cache button. When it is enabled, the CPU will run with Cache. Otherwise, the CPU only accesses Memory.



HitRate:90.83% (hit:99,miss:10)

Hove on a word to show value

2. HOW TO RUN DEMO

Step 1: Enter some instructions into the Code Editor part

Step 2: Click Load button to load instructions into memory

Step 3: Three chocies:

a. Click Run button to run all the instructions. See the execution trace in the Console part

b. Click Debug(SingleStep) button to run a single instruction based on the current value of PC

c. Click Debug(SingleCycle) button to run a single circle of an instruction based on current value of PC

Above are the basic steps to load, store and run instructions. You can also change the values of some specific registers and click Debug (SingleStep) or Debug(SingleCycle) button to see how the CPU works under this specific situation.

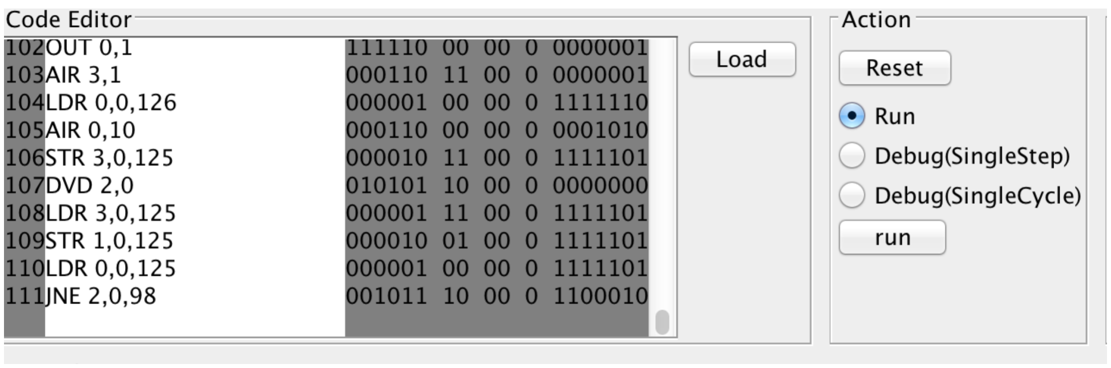
For running instructions, users can enable or disable the cache. If users enable the cache, the cache console will show the status of the cache line by using different colors during the execution process. Red color means cache-miss, and green color means cache-hit. If a word in cache has not been visited recently, it will become blue.

3. HOW TO RUN PROGRAM1 AND SCREENSHOTS

Step 1: Run .jar file

Step 2: Click 'LOAD' button and then 'RUN' button

Program 1 is preloaded in code editor and automatically transformed into binary code. After step 2, the program should be waiting for the event from KEYBOARD.

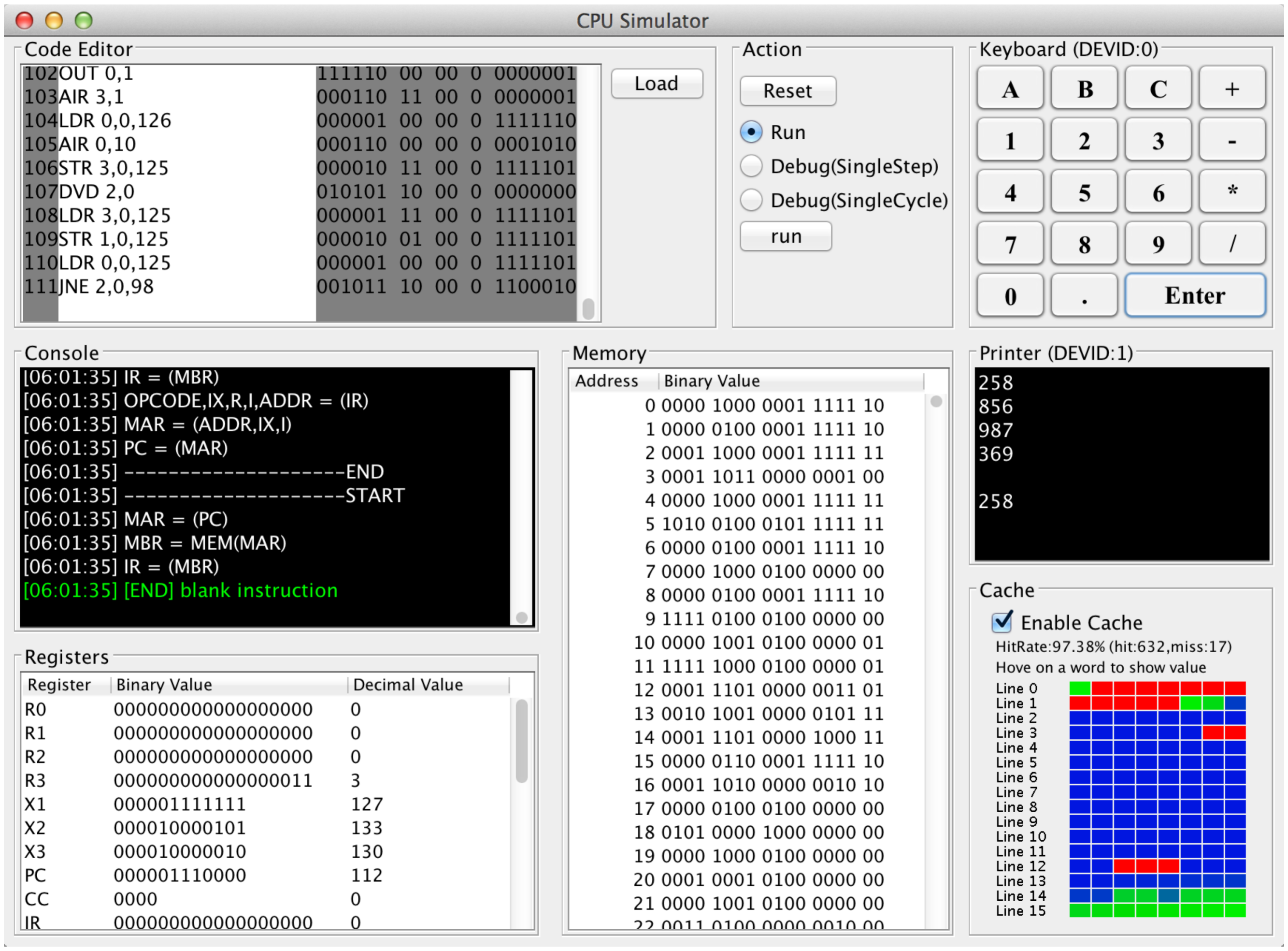


```
102OUT 0,1         111110 00 00 0 0000001
103AIR 3,1         000110 11 00 0 0000001
104LDR 0,0,126     000001 00 00 0 1111110
105AIR 0,10        000110 00 00 0 0001010
106STR 3,0,125     000010 11 00 0 1111101
107DVD 2,0         010101 10 00 0 0000000
108LDR 3,0,125     000001 11 00 0 1111101
109STR 1,0,125     000010 01 00 0 1111101
110LDR 0,0,125     000001 00 00 0 1111101
111JNE 2,0,98      001011 10 00 0 1100010
```

[✓ Run] Debug(SingleStep) Debug(SingleCycle)

Step 3: Click several number keys in Keyboard panel then an ‘ENTER’ as end;

This step should be repeated for 4 times in order to input 4 numbers. For the convenience of testing, we narrow down 20 numbers into 3. Our program 1 will find the nearest number of the 4th out from the first 3 numbers. By simply modifying some instructions (some AIRs before SOBs) in code editor, the program can be adjusted to handle 20 numbers.



```
102OUT 0,1        111110 00 00 0 0000001
103AIR 3,1        000110 11 00 0 0000001
104LDR 0,0,126    000001 00 00 0 1111110
105AIR 0,10       000110 00 00 0 0001010
106STR 3,0,125    000010 11 00 0 1111101
107DVD 2,0        010101 10 00 0 0000000
108LDR 3,0,125    000001 11 00 0 1111101
109STR 1,0,125    000010 01 00 0 1111101
110LDR 0,0,125    000001 00 00 0 1111101
111JNE 2,0,98     001011 10 00 0 1100010
```

```
[06:01:35] IR = (MBR)
[06:01:35] OPCODE,IX,R,I,ADDR = (IR)
[06:01:35] MAR = (ADDR,IX,I)
[06:01:35] PC = (MAR)
[06:01:35] --------------------END
[06:01:35] --------------------START
[06:01:35] MAR = (PC)
[06:01:35] MBR = MEM(MAR)
[06:01:35] IR = (MBR)
[06:01:35] [END] blank instruction
```

Register	Binary Value	Decimal Value
R0	0000000000000000000	0
R1	0000000000000000000	0
R2	0000000000000000000	0
R3	0000000000000000011	3
X1	000001111111	127
X2	000010000101	133
X3	000010000010	130
PC	000001110000	112
CC	0000	0
IR	0000000000000000000	0

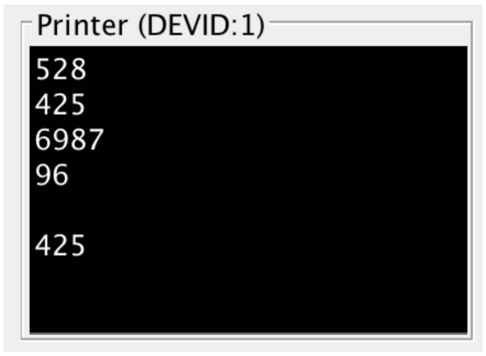
```
Address   Binary Value
       0 0000 1000 0001 1111 10
       1 0000 0100 0001 1111 10
       2 0001 1000 0001 1111 11
       3 0001 1011 0000 0001 00
       4 0000 1000 0001 1111 11
       5 1010 0100 0101 1111 11
       6 0000 0100 0001 1111 10
       7 0000 1000 0100 0000 00
       8 0000 0100 0001 1111 10
       9 1111 0100 0100 0000 00
      10 0000 1001 0100 0000 01
      11 1111 1000 0100 0000 01
      12 0001 1101 0000 0011 01
      13 0010 1001 0000 0101 11
      14 0001 1101 0000 1000 11
      15 0000 0110 0001 1111 10
      16 0001 1010 0000 0010 10
      17 0000 0100 0100 0000 00
      18 0101 0000 1000 0000 00
      19 0000 1000 0100 0000 00
      20 0001 0001 0100 0000 00
      21 0000 1001 0100 0000 00
      22 0011 0100 0000 0010 00
```

```
258
856
987
369

258
```

HitRate:97.38% (hit:632,miss:17)

Step 4: Wait for the result.



```
528
425
6987
96

425
```