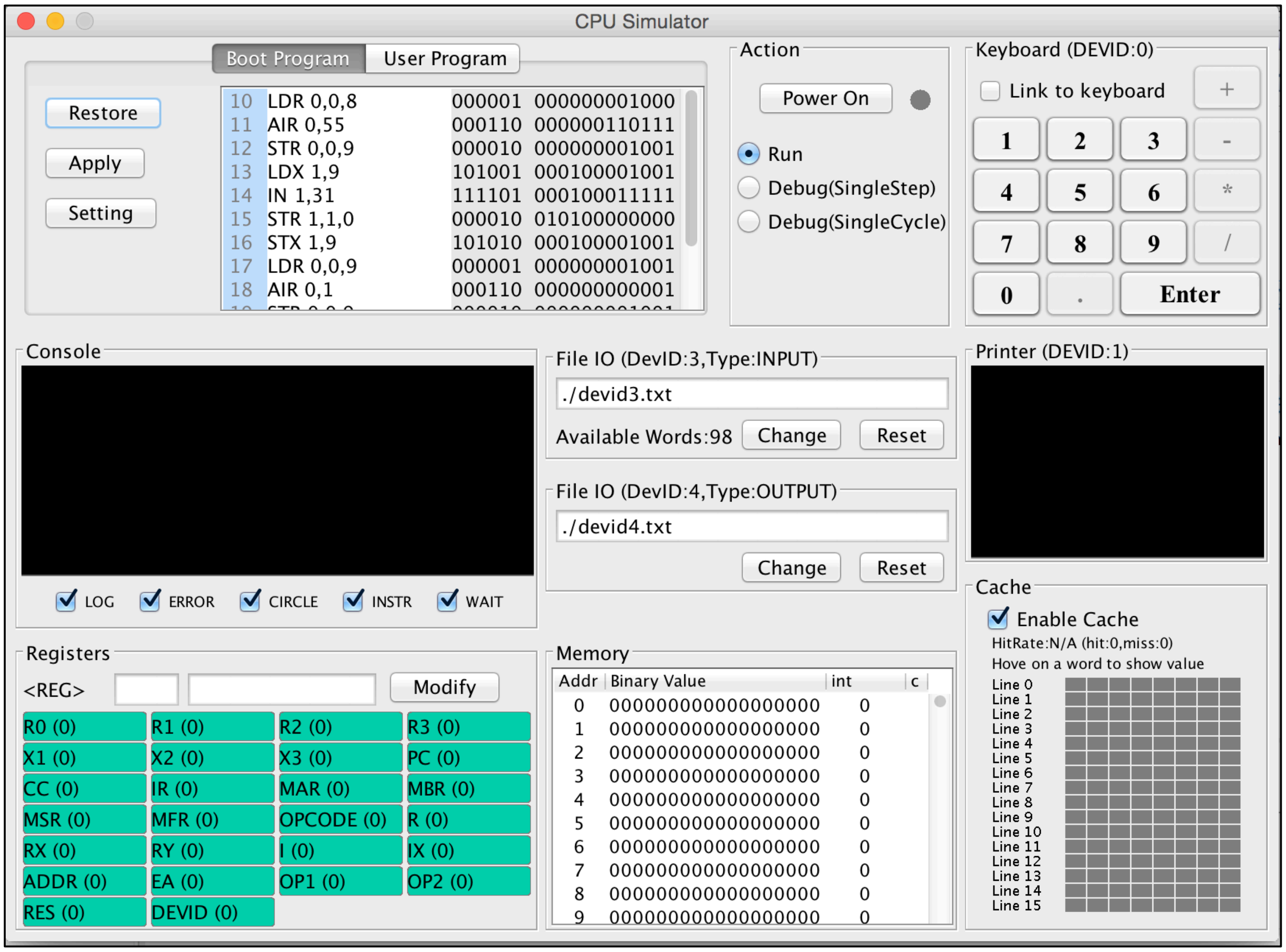
User's Manual

CS6461 CPUSimulator   Part 3

Group 9

1.INTRODUCTION

Console Layout and features



```
LDR 0,0,8        000001  000000001000
AIR 0,55         000110  000000110111
STR 0,0,9        000010  000000001001
LDX 1,9          101001  000100001001
IN 1,31          111101  000100011111
STR 1,1,0        000010  010100000000
STX 1,9          101010  000100001001
LDR 0,0,9        000001  000000001001
AIR 0,1          000110  000000000001
```

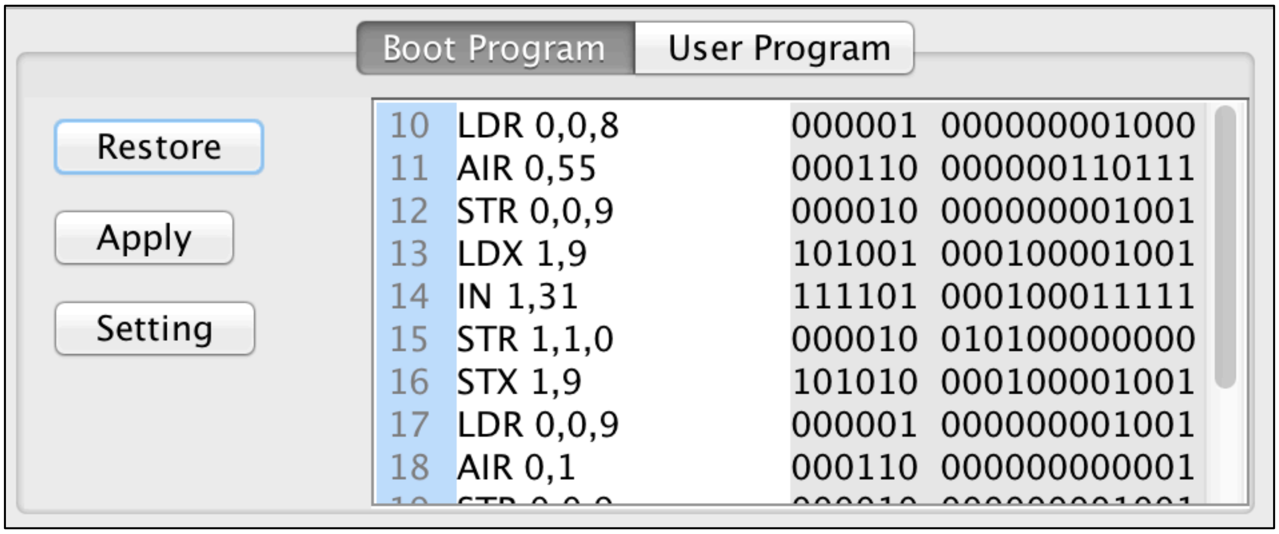
R0 (0)	R1 (0)	R2 (0)	R3 (0)
X1 (0)	X2 (0)	X3 (0)	PC (0)
CC (0)	IR (0)	MAR (0)	MBR (0)
MSR (0)	MFR (0)	OPCODE (0)	R (0)
RX (0)	RY (0)	I (0)	IX (0)
ADDR (0)	EA (0)	OP1 (0)	OP2 (0)
RES (0)	DEVID (0)		

Addr	Binary Value	int	c
0	000000000000000000	0	
1	000000000000000000	0	
2	000000000000000000	0	
3	000000000000000000	0	
4	000000000000000000	0	
5	000000000000000000	0	
6	000000000000000000	0	
7	000000000000000000	0	
8	000000000000000000	0	
9	000000000000000000	0	

Our CPU Simulator consists of nine parts: Code Editor, Action, Console, Registers, Keyboard, Printer, Cache, Memory and File IO (INPUT & OUTPUT). 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.

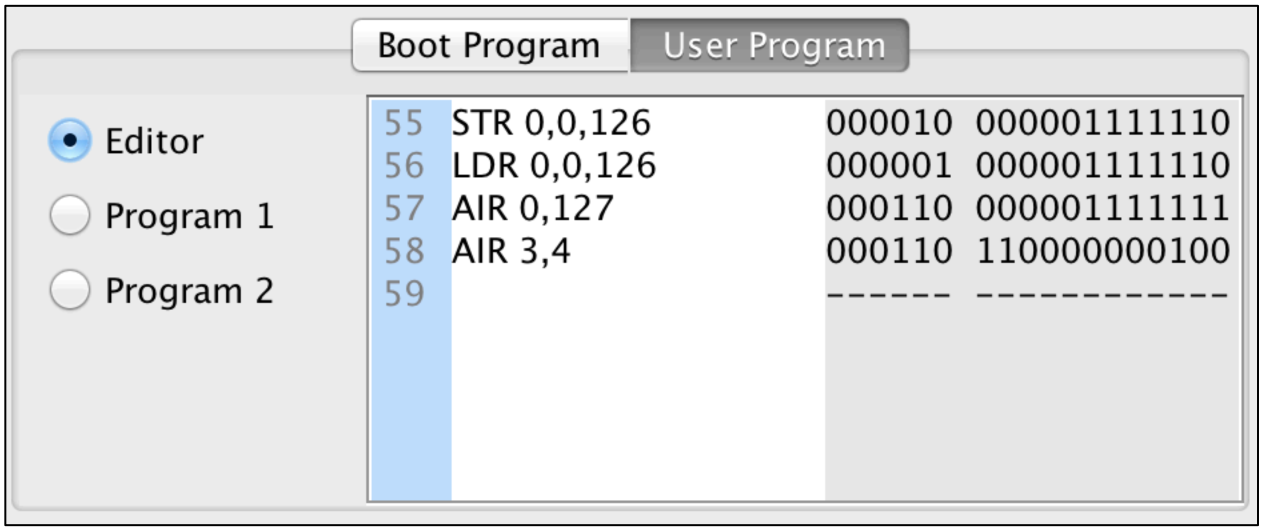
Code Editor: In this part, there are two different types of program: the boot program and the user program.

For the boot program, there are three actions: Restore, Apply and Setting. Users can use Restore to reset the boot program, use Apply to apply the modifications of the boot program and use Settings to set the values in the first five addresses of the main memory.



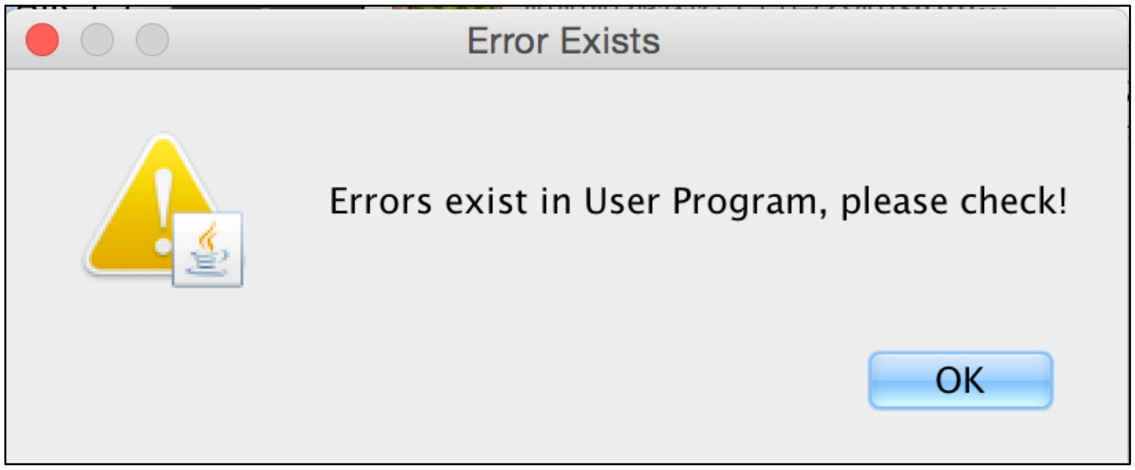
```
10  LDR 0,0,8      000001  000000001000
11  AIR 0,55       000110  000000110111
12  STR 0,0,9      000010  000000001001
13  LDX 1,9        101001  000100001001
14  IN 1,31        111101  000100011111
15  STR 1,1,0      000010  010100000000
16  STX 1,9        101010  000100001001
17  LDR 0,0,9      000001  000000001001
18  AIR 0,1        000110  000000000001
```

For the user program, there are also three actions: Editor, Program1 and Program2. The Editor is designed for users to write their own programs. 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 is not in the right format, an error message will be showed. Program 1 is the assembler for the required program in Part II: A program that reads 20 numbers (integers) from the keyboard, prints the numbers to the console printer, requests a number from the user, and searches the 20 numbers read in for the number closest to the number entered by the user. Print the number entered by the user and the number closest to that number. Your numbers should not be 1…10, but distributed over the range of 1 … 65,536. Therefore, as you read a character in, you need to check it is a digit, convert it to a number, and assemble the integer. Program2 is the assembler for the required program in Part III: A program that reads a set of a paragraph of 6 sentences from a file into memory. It prints the sentences on the console printer. It then asks the user for a word. It searches the paragraph to see if it contains the word. If so, it prints out the word, the sentence number, and the word number in the sentence.



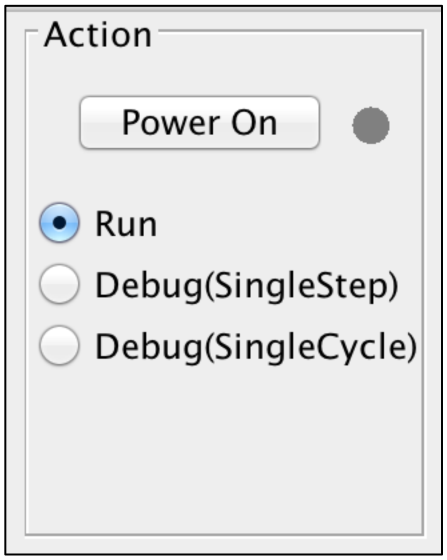
```
55 STR 0,0,126    000010  000001111110
56 LDR 0,0,126    000001  000001111110
57 AIR 0,127      000110  000001111111
58 AIR 3,4        000110  110000000100
59                ------  ------------
```

Error message will showed if the instructions contain some errors.



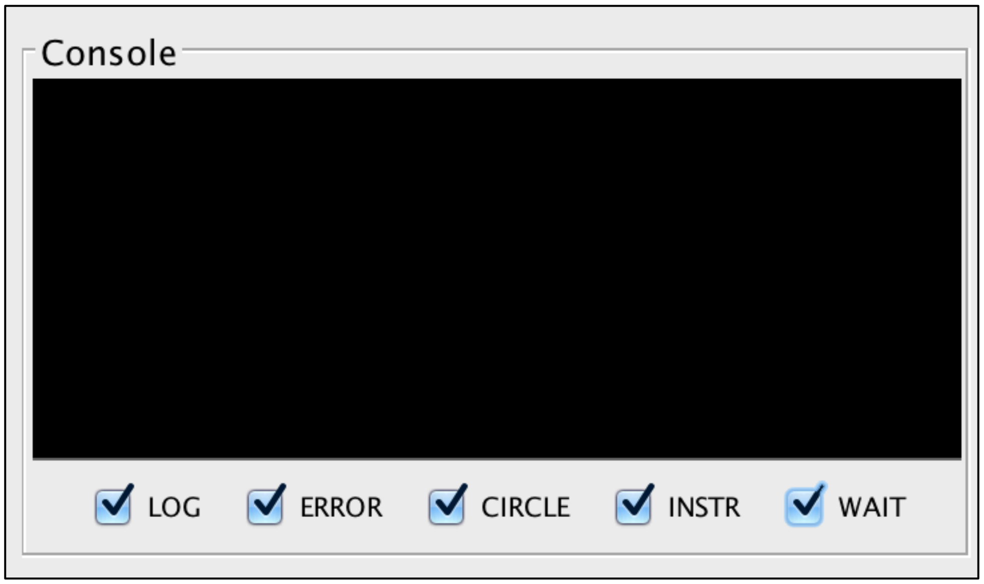
Errors exist in User Program, please check!

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. There is also a Power On/Off button. When our CPU simulator is loaded, it's supposed to be at the Power Off status. Users can click the Power On button to turn on the machine. When the programs finish running, the simulator will turn off automatically.

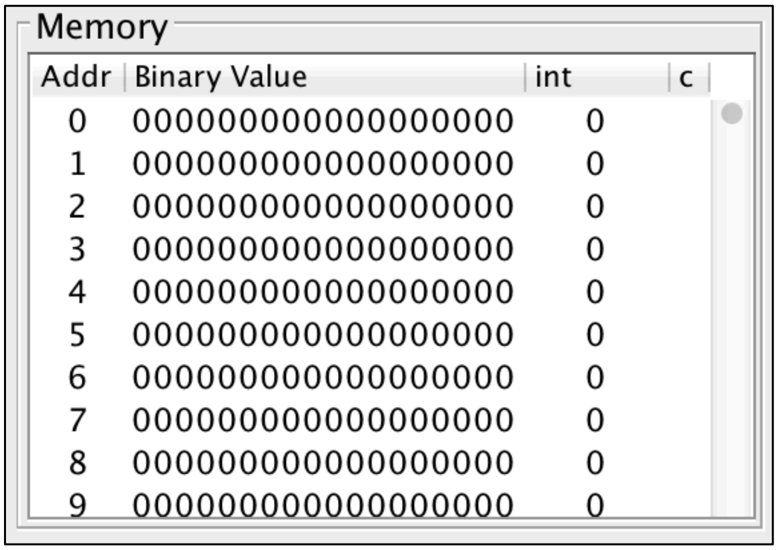


Selected: [✓ Run] Debug(SingleStep) Debug(SingleCycle)

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. Users can choose which information they want to display in the monitor. There are 6 types of information: LOG, ERROR, CIRCLE, INSTR and WAIT.



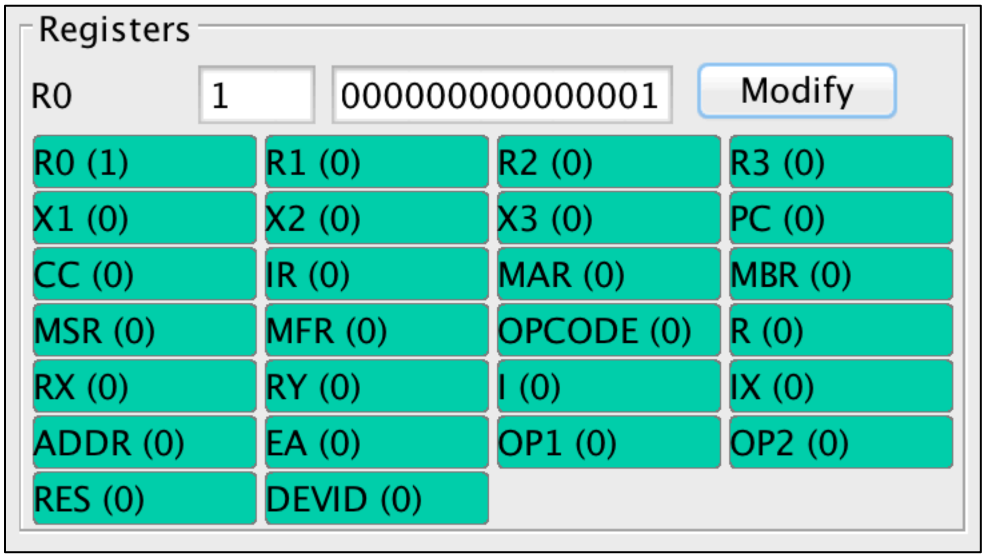
Memory: In the Memory part, binary values will be displayed. They are related to the instructions, addresses and data that have been stored into the memory. Users can figure out what's exactly stored in the memory.



Memory

Addr	Binary Value	int	c
0	000000000000000000	0	
1	000000000000000000	0	
2	000000000000000000	0	
3	000000000000000000	0	
4	000000000000000000	0	
5	000000000000000000	0	
6	000000000000000000	0	
7	000000000000000000	0	
8	000000000000000000	0	
9	000000000000000000	0	

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 clicking the corresponding display area of the register, which means the green rectangle area in the following graph. Change the binary or decimal value in the editable text area, and then click the Modify button to save the modification.

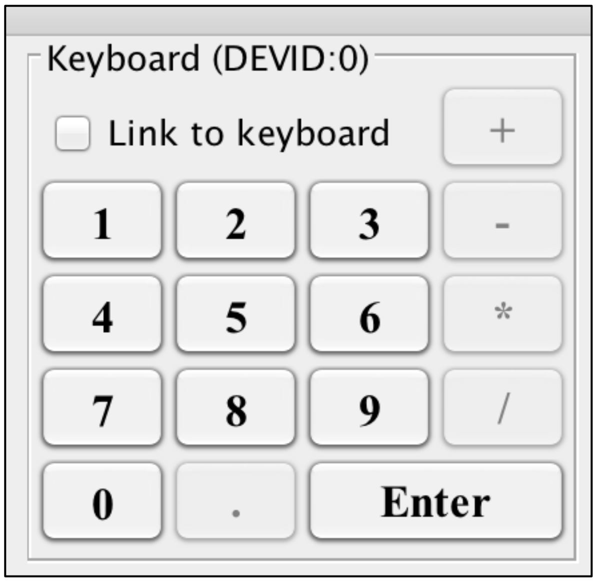


Registers

R0 | 1 | 0000000000000001

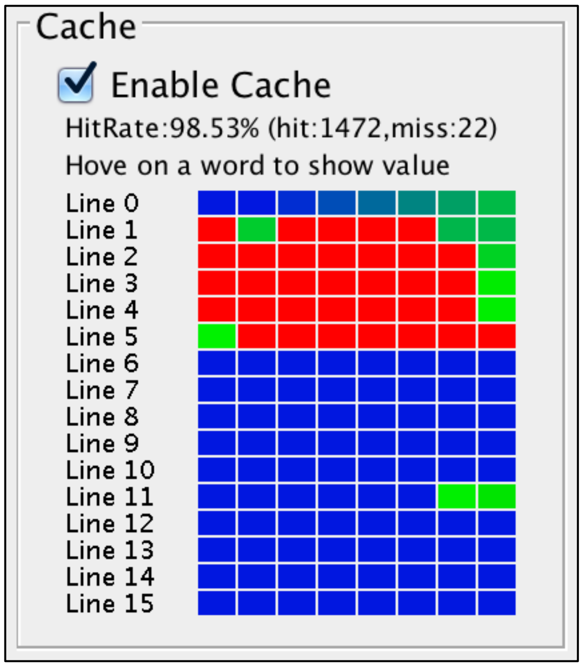
R0 (1)	R1 (0)	R2 (0)	R3 (0)
X1 (0)	X2 (0)	X3 (0)	PC (0)
CC (0)	IR (0)	MAR (0)	MBR (0)
MSR (0)	MFR (0)	OPCODE (0)	R (0)
RX (0)	RY (0)	I (0)	IX (0)
ADDR (0)	EA (0)	OP1 (0)	OP2 (0)
RES (0)	DEVID (0)		

Keyboard: The users can use this keyboard simulator to enter numbers and characters. When users choose the Link to keyboard option, they will be able to use their real keyboard to enter the numbers and characters. The DEVID for the keyboard simulator is 0.



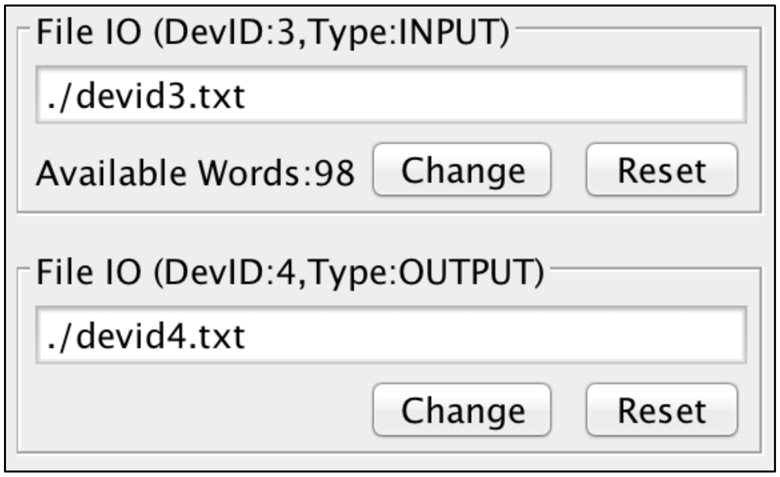
Printer: This field will display the results of users' operations. The DEVID for the printer simulator is 1.

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. Users can also check the hit rate and the value of each word in this field.



HitRate:98.53% (hit:1472,miss:22)
Hove on a word to show value

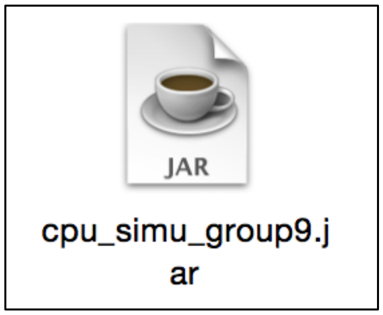
File IO (INPUT and OUTPUT): Users can use this field to read the contents of a specific file and write contents to a specific target file. The DEVID for the file input is 3. The DEVID for the file output is 4. Users can click the Change button to select a specific file from the computer. Users also can click the Reset button to reread the target file.


File IO (DevID:3,Type:INPUT)
./devid3.txt
Available Words:98

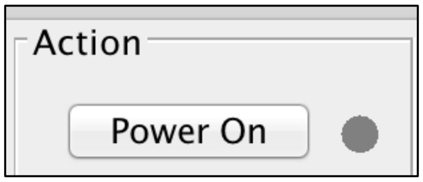
File IO (DevID:4,Type:OUTPUT)
./devid4.txt

2. HOW TO RUN DEMO

Step 1: Run .jar file



Step 2: Click Power On button in the Action field.

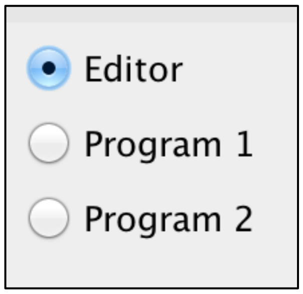


Step 3: Choose the program you want to run, you have three choices for now:

a. Enter your own assemblers under the Editor mode.

b. Choose program1 to run the required testing program in Part II

c. Choose program2 to run the required testing program in Part III

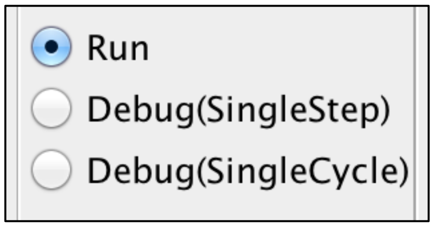


Step 4: Three choices:

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



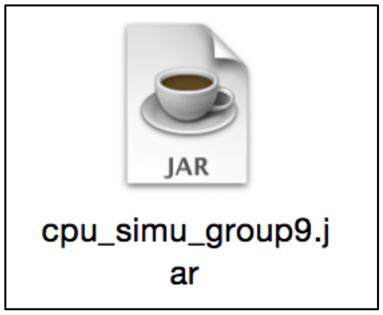
Step 4: Clicke “Confirm Program Source ” to start executing the program you have chosen.

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.

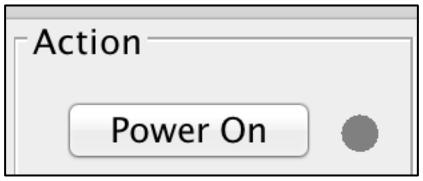
For running instructions, users can enable or disable the cache. If users enable the cache, the cache console will show the status of the cache line by using different colors during the execution process. Red color means cache-miss, and green color means cache-hit. If a word in cache has not been visited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

3. HOW TO RUN PROGRAM1 AND SCREENSHOTS

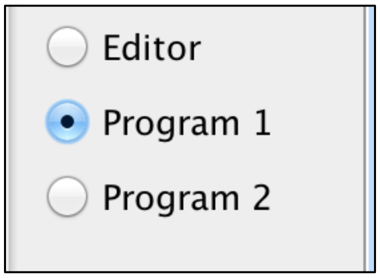
Step 1: Run .jar file



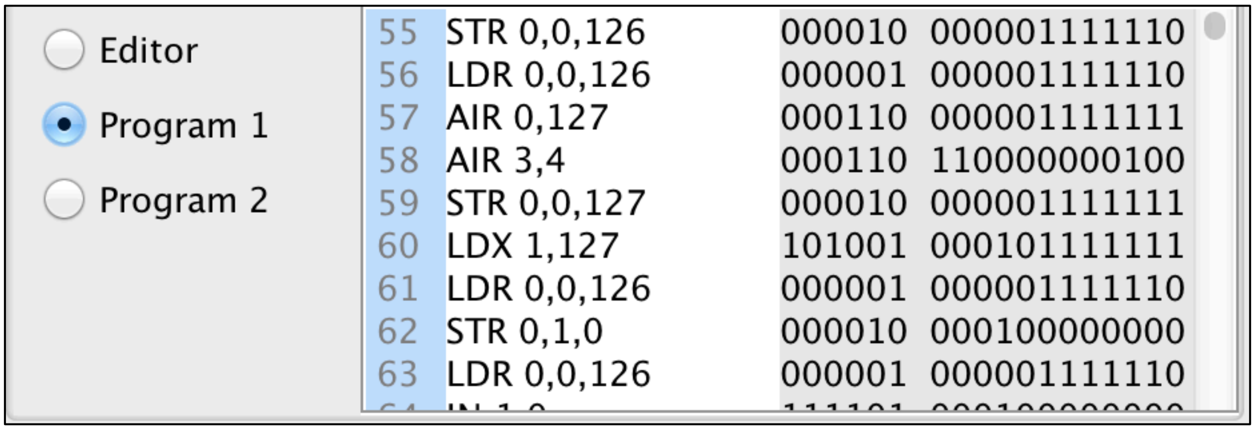
Step 2: Click Power On button in the Action field.



Step 3: Choose Program 1.

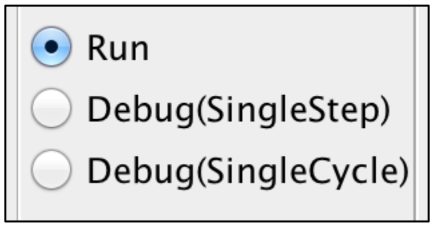

Editor [✓ Program 1] Program 2

Then the assembler of Program 1 will be loaded in the code editor area and be transformed into binary code.

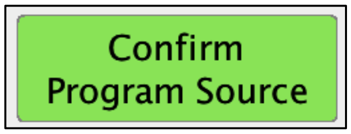

Editor [✓ Program 1] Program 2

```
STR 0,0,126        000010  000001111110
LDR 0,0,126        000001  000001111110
AIR 0,127          000110  000001111111
AIR 3,4            000110  110000000100
STR 0,0,127        000010  000001111111
LDX 1,127          101001  000101111111
LDR 0,0,126        000001  000001111110
STR 0,1,0          000010  000100000000
LDR 0,0,126        000001  000001111110
```

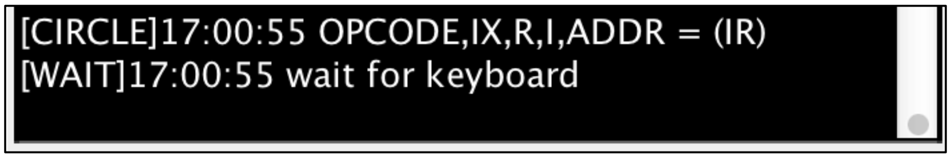
Step 4: Choose Run mode.



Step 5: Click "Confirm Program Source".



After step 5, the program should be waiting for the event from KEYBOARD.

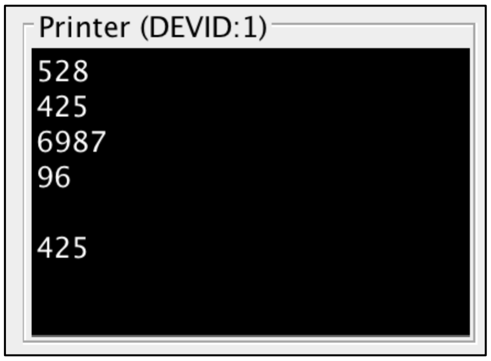


```
[CIRCLE]17:00:55 OPCODE,IX,R,I,ADDR = (IR)
[WAIT]17:00:55 wait for keyboard
```

Step 6: 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.

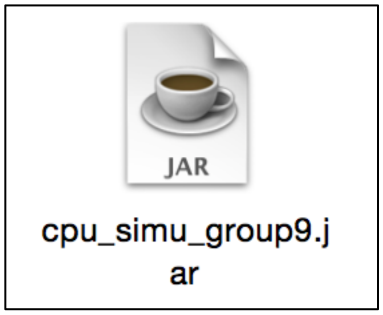
Step 7: Wait for the result.



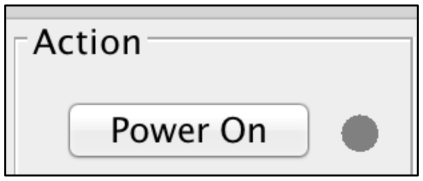
```
528
425
6987
96

425
```

4. HOW TO RUN PROGRAM2 AND SCREENSHOTS

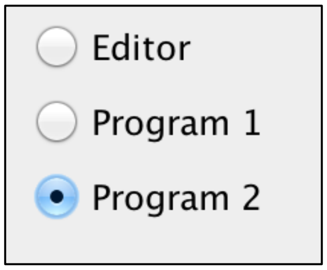
Step 1: Run .jar file



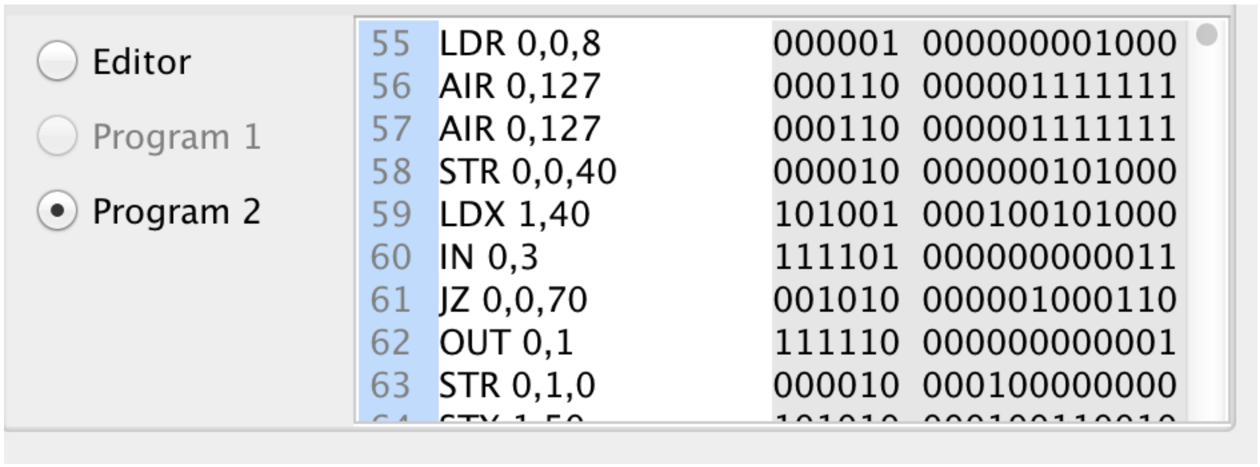
Step 2: Click Power On button in the Action field.



Step 3: Choose Program 2.

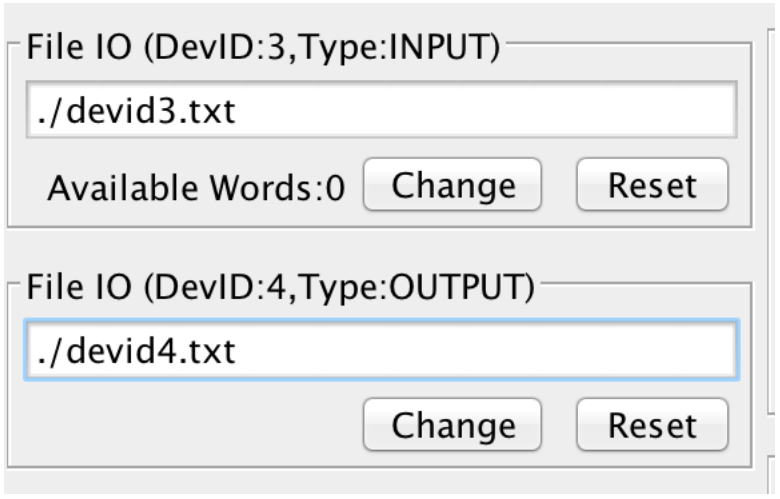


Then the assembler of Program 2 will be loaded in the code editor area and be transformed into binary code.



```
LDR 0,0,8       000001 000000001000
AIR 0,127       000110 000001111111
AIR 0,127       000110 000001111111
STR 0,0,40      000010 000000101000
LDX 1,40        101001 000100101000
IN 0,3          111101 000000000011
JZ 0,0,70       001010 000001000110
OUT 0,1         111110 000000000001
STR 0,1,0       000010 000100000000
```

Step 4: Choose the paragraph file


File IO (DevID:3,Type:INPUT)

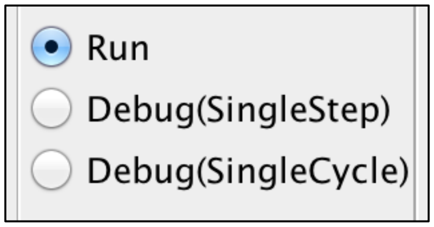
./devid3.txt

Available Words:0

File IO (DevID:4,Type:OUTPUT)

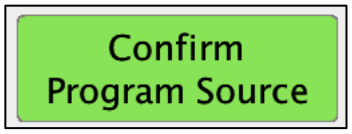
./devid4.txt

Step 5: Choose Run mode.

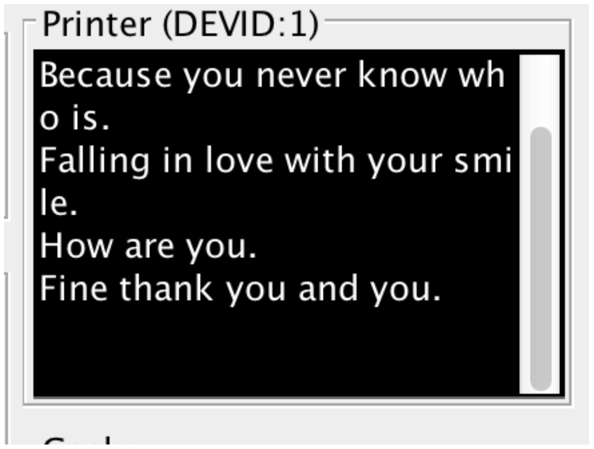


Run [✓]

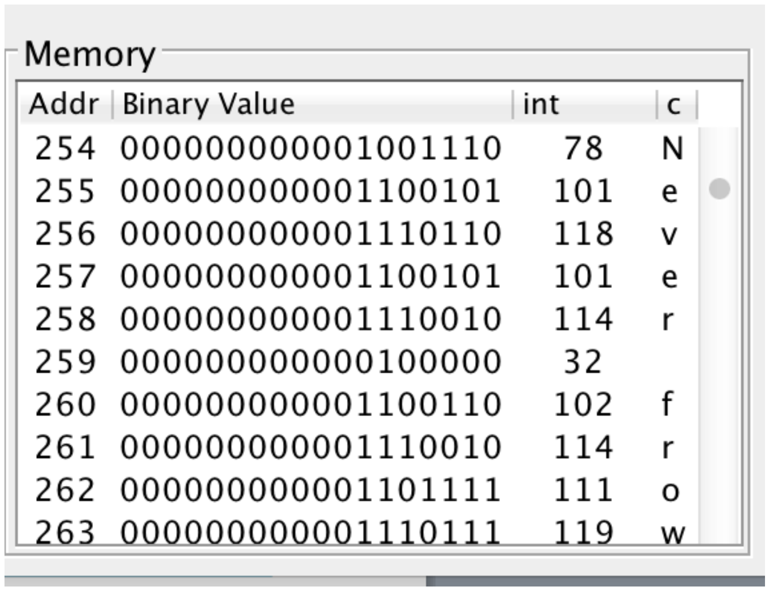
Step 6: Click "Confirm Program Source".



Step 7: The paragraph will be loaded into memory and shown in console.


Printer (DEVID:1)

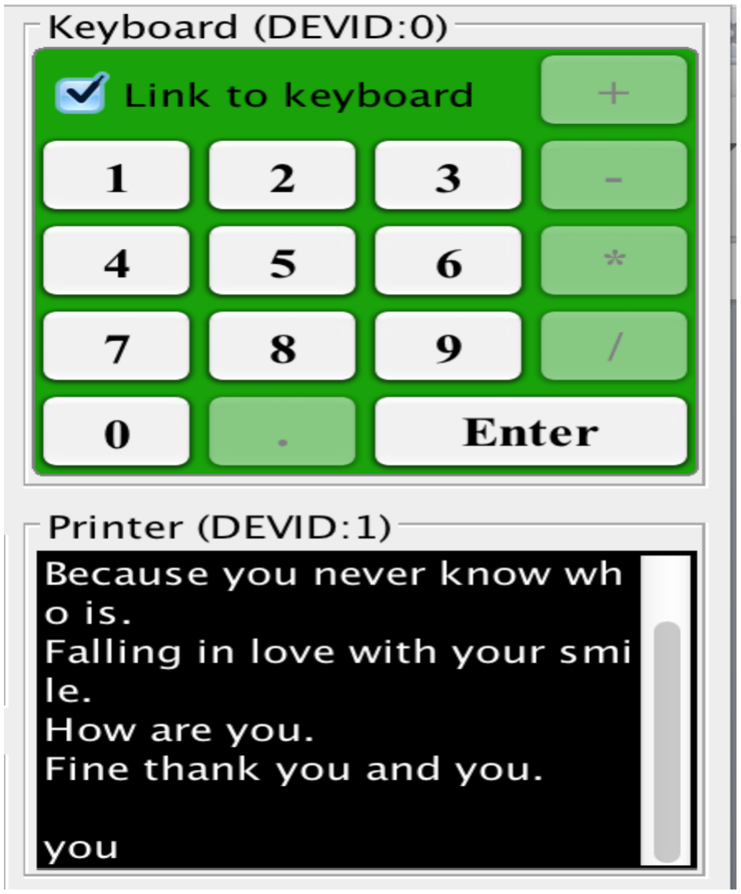
```
Because you never know wh
o is.
Falling in love with your smi
le.
How are you.
Fine thank you and you.
```



Memory

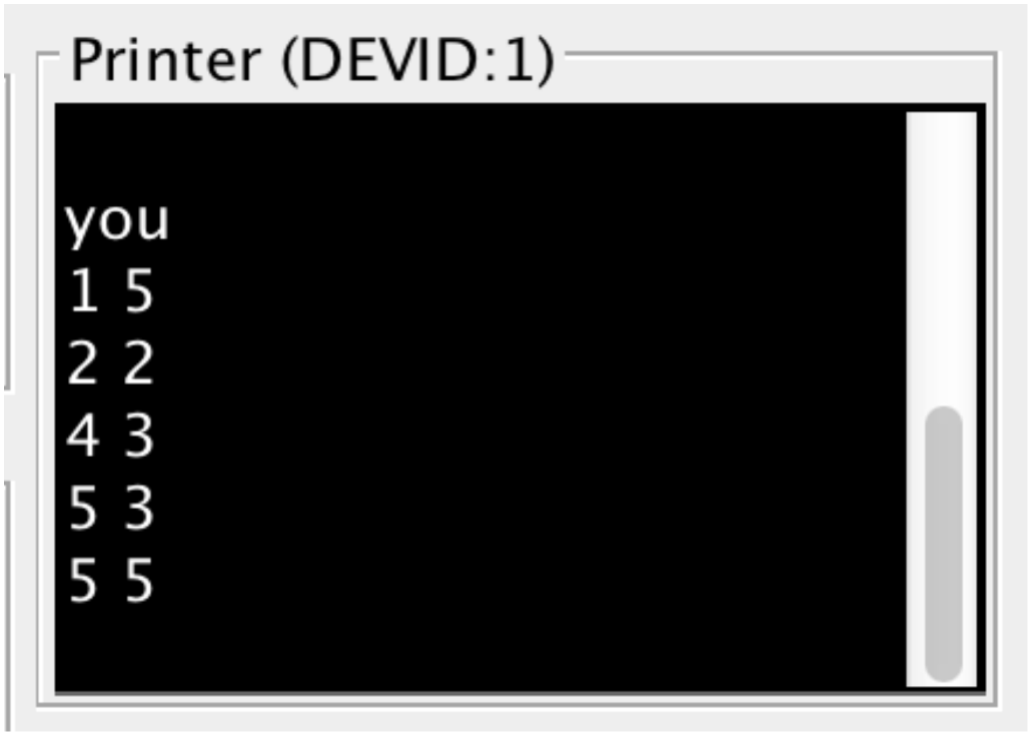
Addr	Binary Value	int	c
254	0000000001001110	78	N
255	0000000001100101	101	e
256	0000000001110110	118	v
257	0000000001100101	101	e
258	0000000001110010	114	r
259	0000000000100000	32	
260	0000000001100110	102	f
261	0000000001110010	114	r
262	0000000001101111	111	o
263	0000000001110111	119	w

Step 8 : Click the checkbox so that you can enter the word directly from your keyboard. In this example, we simply enter the word "you" in console in order to find the word "you" appears in which sentence number and word number in that sentence.



```
Because you never know wh
o is.
Falling in love with your smi
le.
How are you.
Fine thank you and you.

you
```

Step 9: After entering the word "you", you should press the enter button on your keyboard. Then it begins to match the word.



```
you
1 5
2 2
4 3
5 3
5 5
```

The first number in one line represents the sentence number.

The second number in one line represents the word number.

In our example, the word "you" appears in the fifth word in the first sentence. It also appears in the second word in second sentence. The rest can be understood in the same matter.