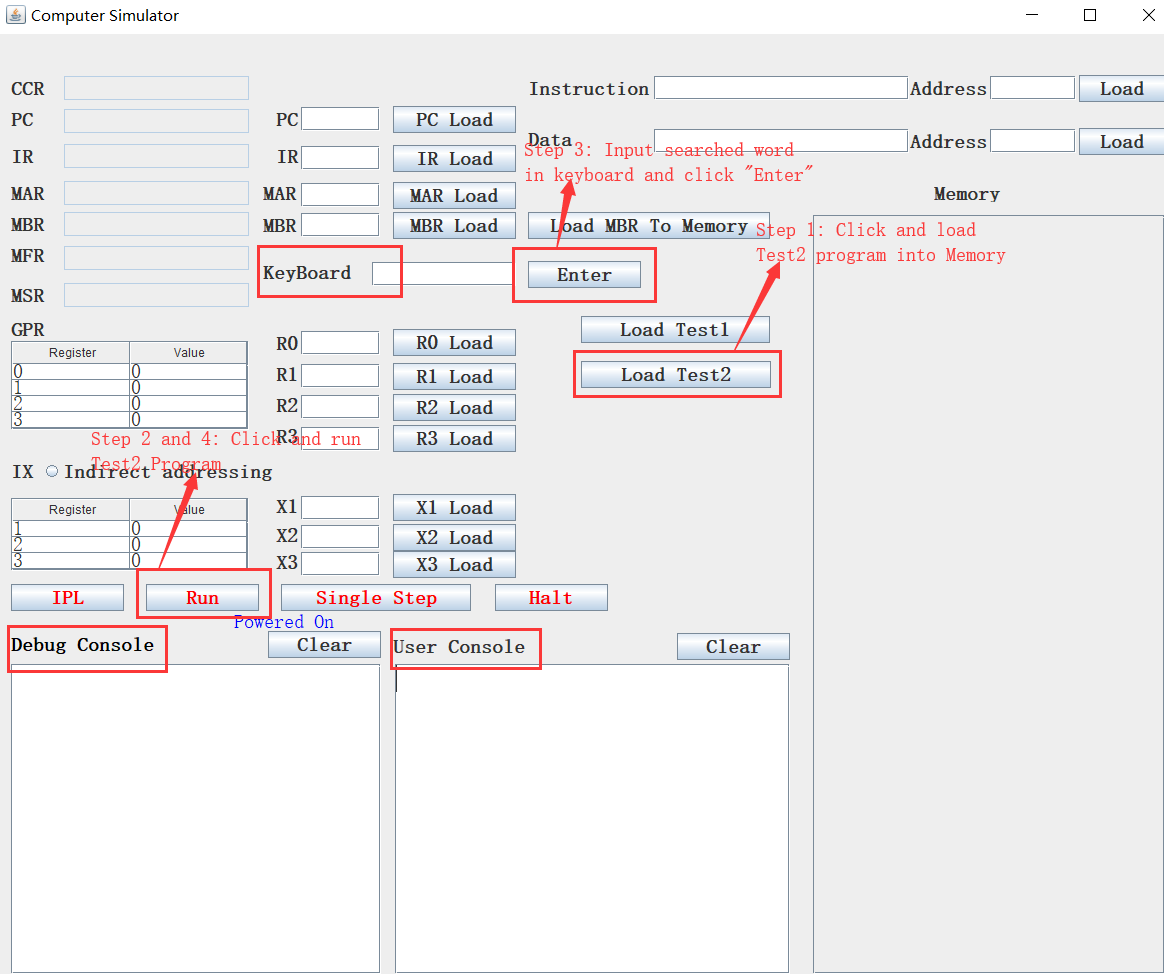
1. **Demonstration**
   1. layout

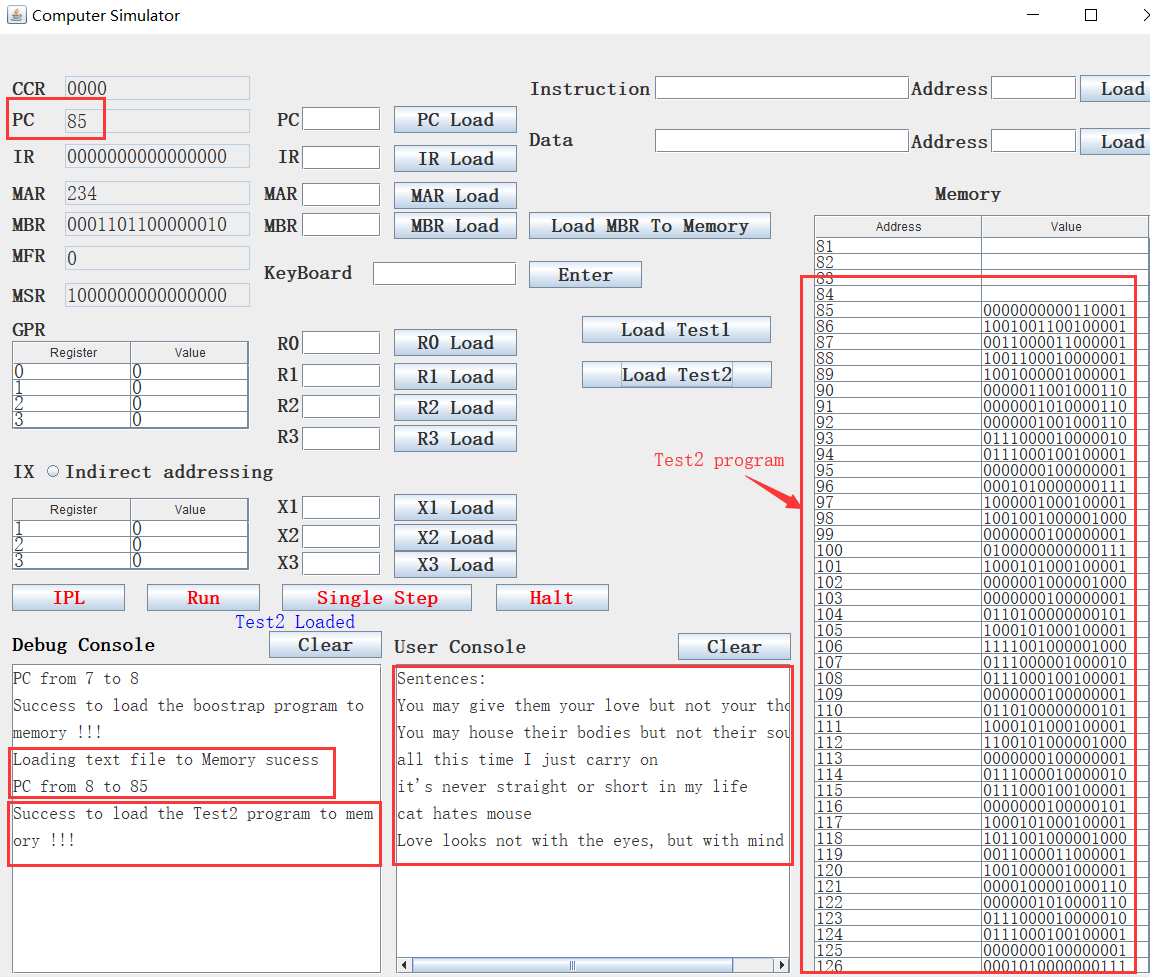
we add “Load Test2” button to load test2 program in our simulator, and we split the console into “Debug Console” and “User console”. Debug console will display more detailed information for engineers to view; User Console will only display the final result for users. See the following screen short:

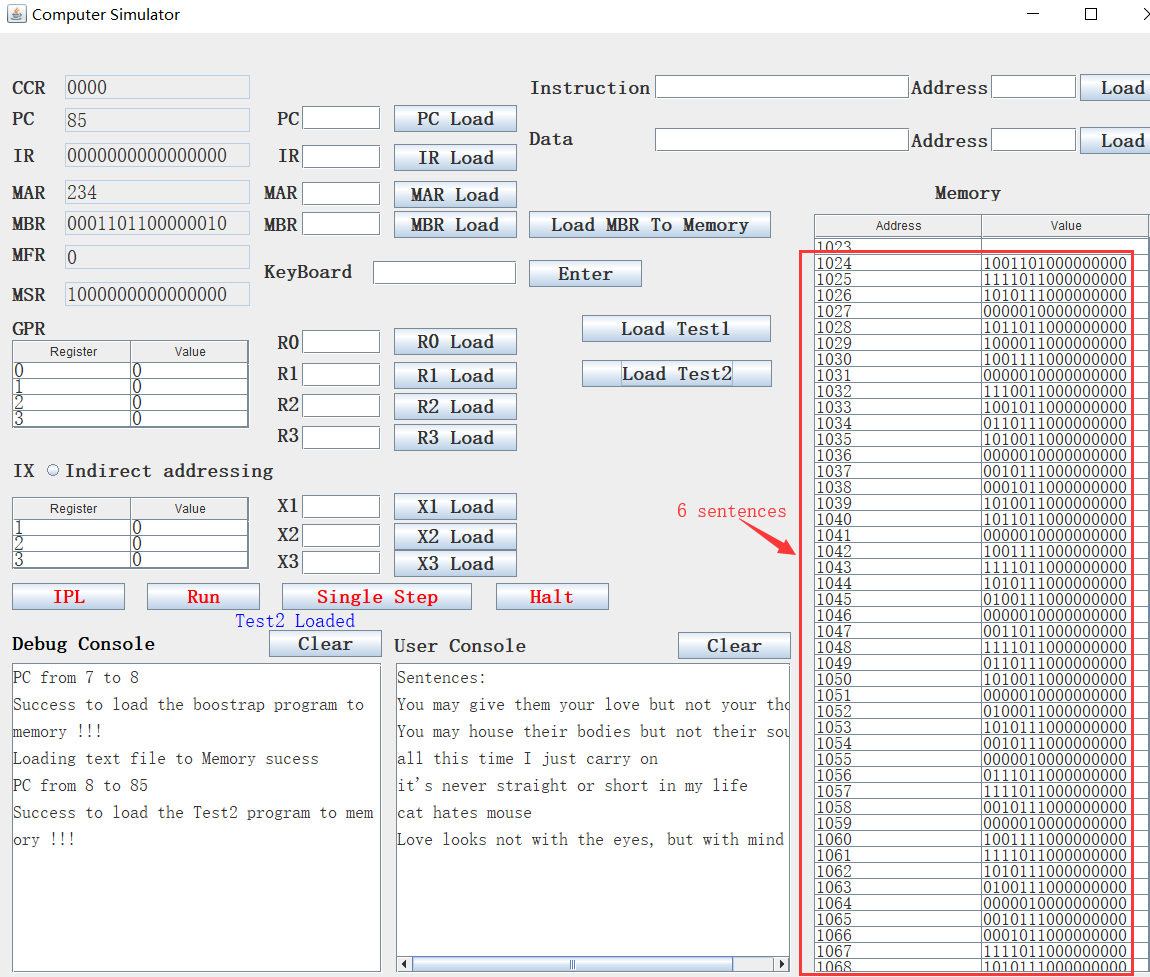


* 1. Operation and Result

1.2.1 Load sentences and program to memory

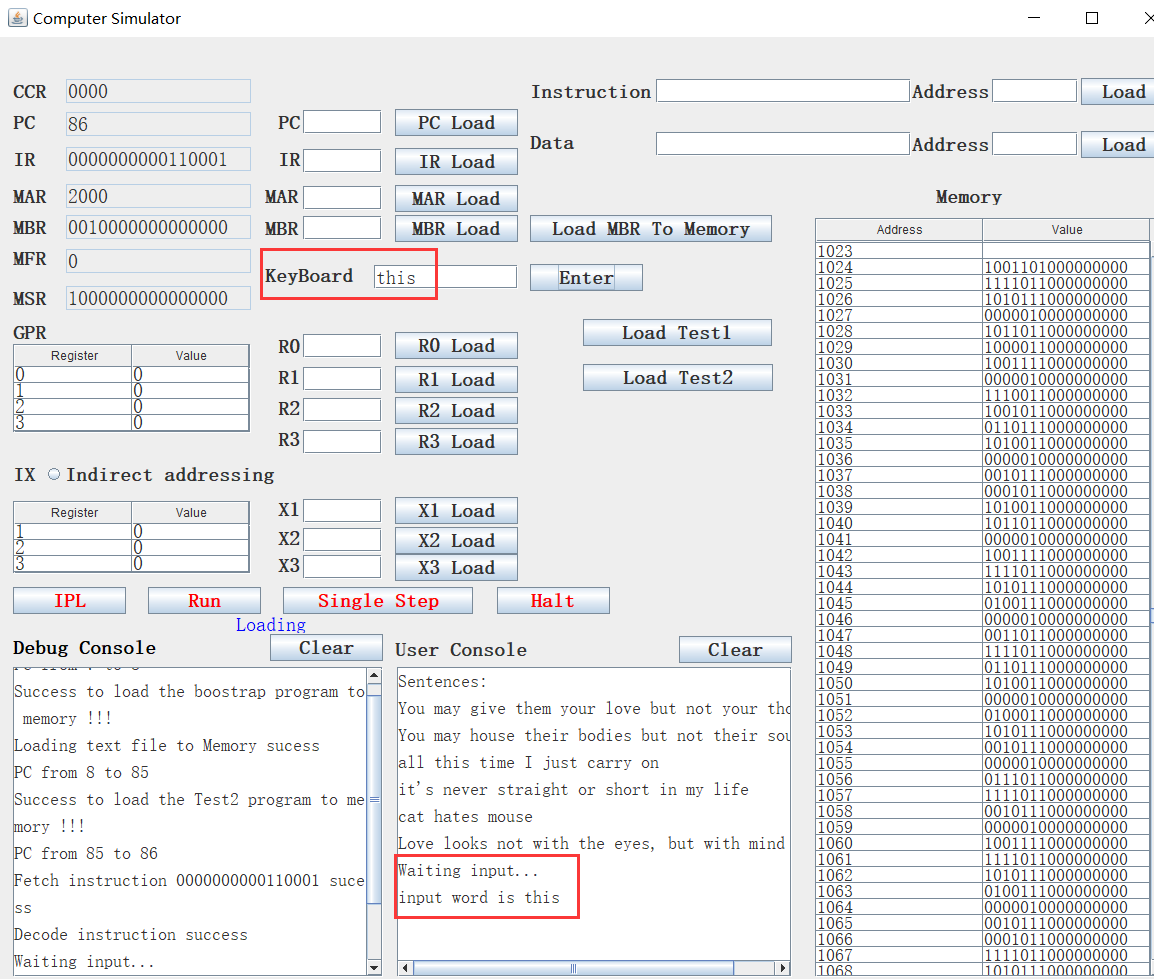
The Test2 program is in the current folder named “Test2.txt” and the 6 sentences are in the file “sentences.txt”. Users first should click “IPL” to start the simulator, then click “Load Test2” button which will load the sentences.txt and the Test2 program into memory. The 6 sentences will be loaded to memory begin with memory location 1024, and the Test2 program will be loaded to memory begin with memory location 85, meanwhile, the PC will be set to 85. See the following screen short:





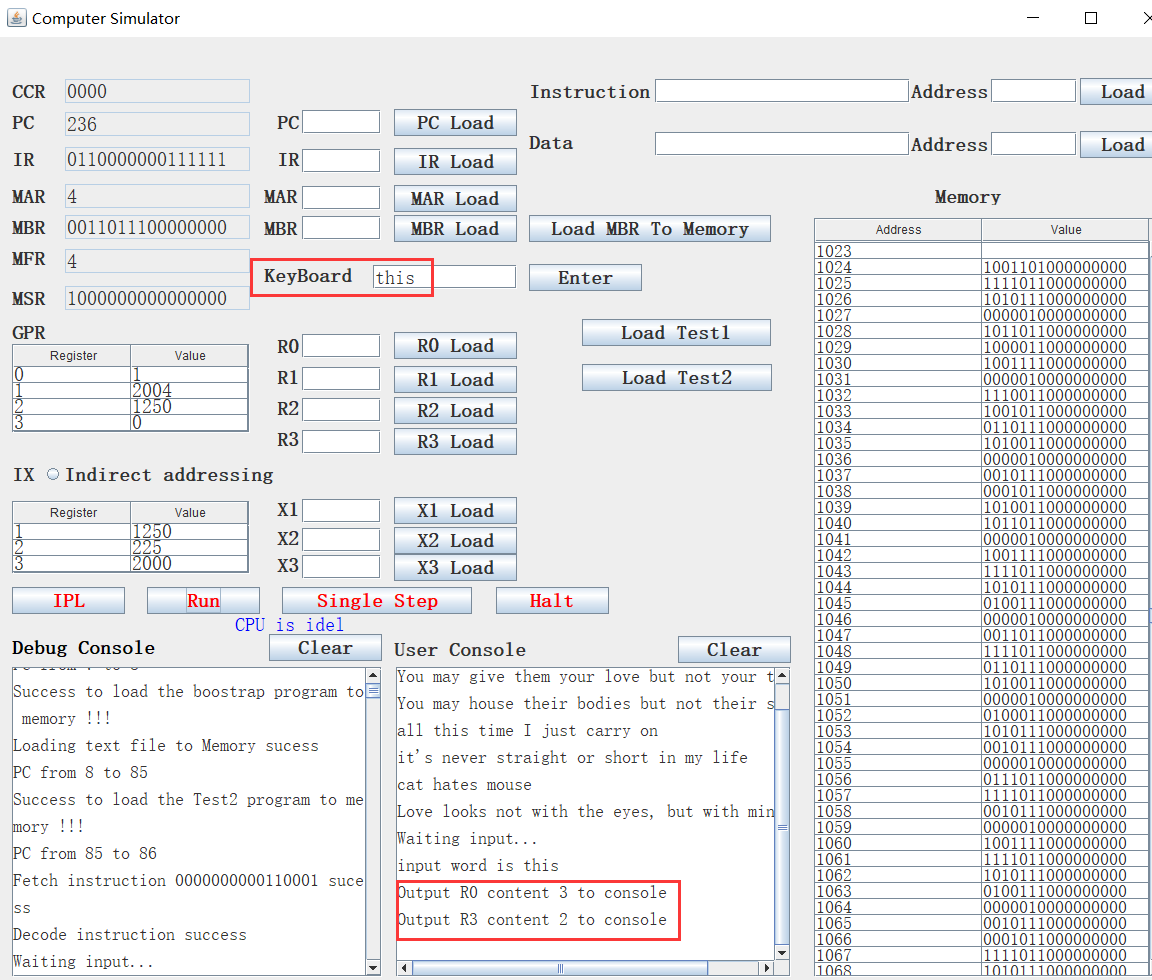
1.2.2 Input searched word:

After loading sentences file and program to memory, users can click “**Run**” to execute the program. During the execution, the program will stop and ask users to input the searched word in the keyboard, users should input the searched word in the keyboard and then click “**Enter**”, then the input word will be displayed on the user console. See the following screen short:



1.2.3 Run and get the result

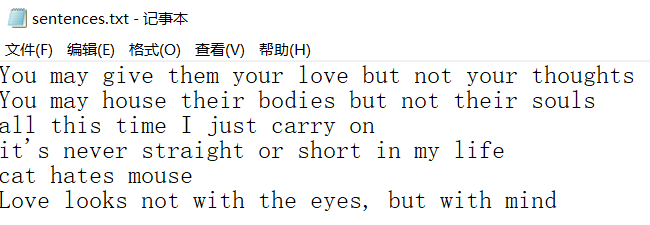
After inputting the searched word, the user should click “**Run**” button to continue run the program, and the result will be displayed on the user console. See the following screen short:



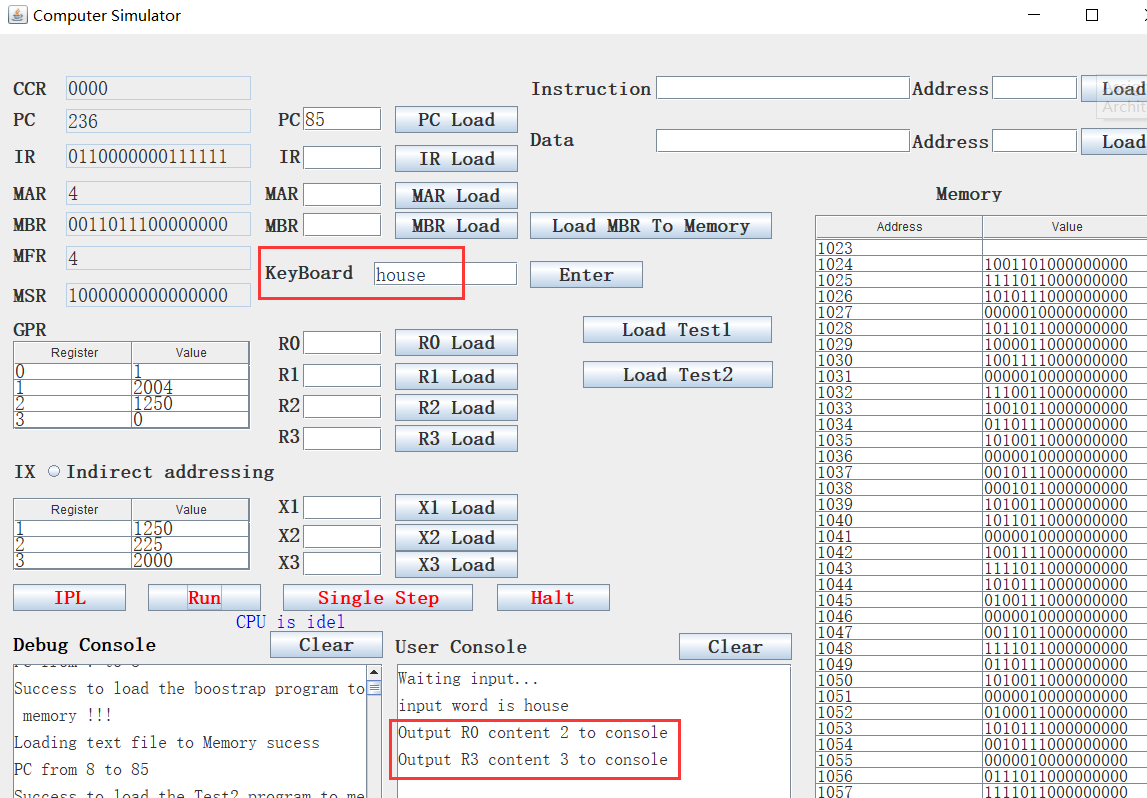
We put the line number in R0, word number in R3. The above result means the searched word is in the 3th line and the 2th word.

* 1. Test cases

We show several test cases on our simulator. The 6 sentences are the following:

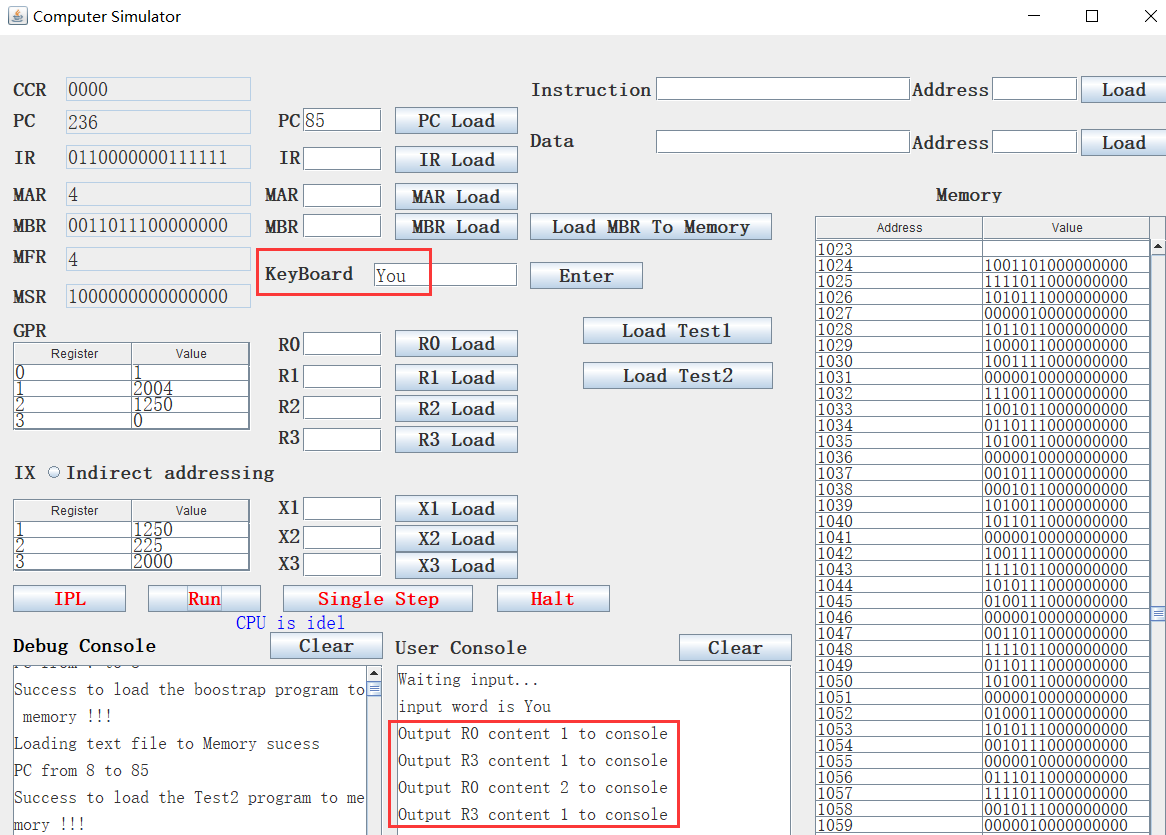


1.3.1 Find word “house”



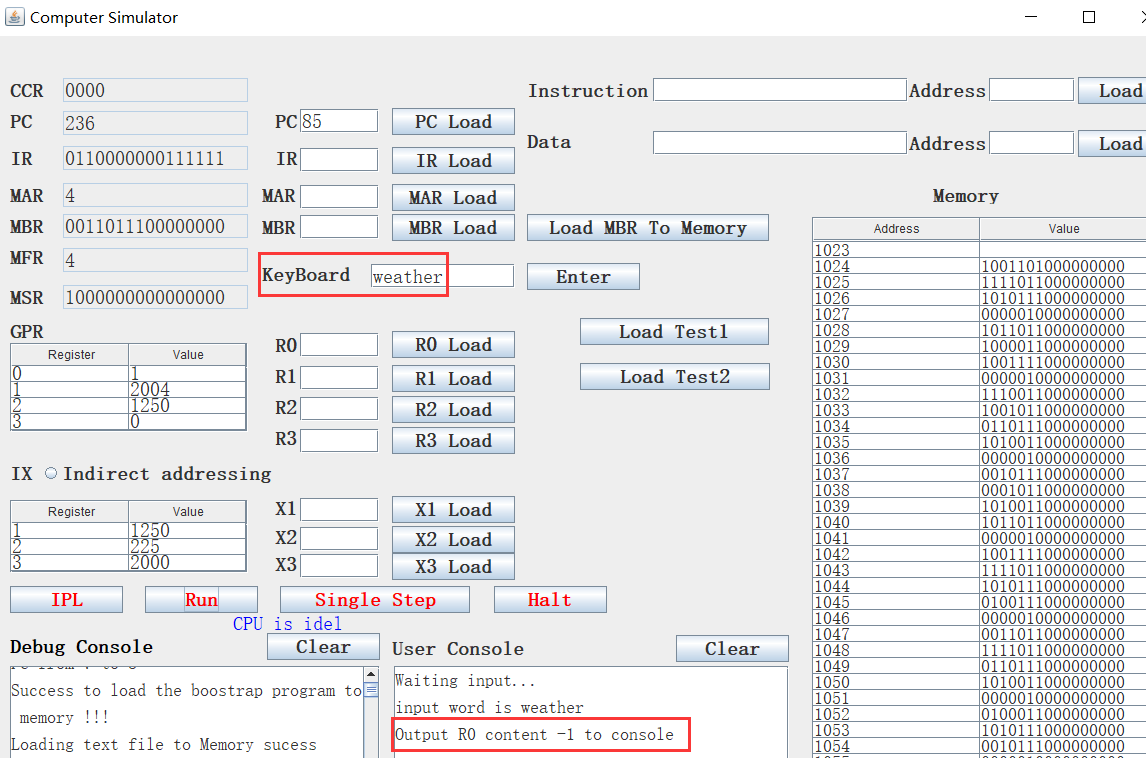
The result shows that the word “house” is in 2th line, 3th word.

1.3.2 Find word “You”



The result shows that the work “You” appears twice. One is in 1th line, 1th word; another is in 2th line, 1th word.

1.3.3 Find word “weather”



We will put -1 to R0 which represents no such word in the sentences. The above result shows that there is no word “weather” in these sentences.

1. **Design notes**
   1. Case sensitive

Our simulator is case sensitive.

* 1. The maximum text length supported

Since we keep the data and instructions in memory separately, and the memory has a maximum storage capacity limit, our simulator has a maximum text length supported, which is 976. We keep the data in memory beginning with 1024, and instructions in memory beginning with 85. Memory after 2000 are used for program execution. Each memory slot stores one character. We put number 257 at the end of the text in the memory.

* 1. The maximum searched word supported

We store the searched word in memory beginning with 2004, and the maximum memory length is 2048, so the maximum searched word length supported is 43, the last memory slot stores the ending tag which is the number 257.

* 1. The output results

If find the searched word, the line number is put in R0, and the word number is put in R3. If not find the searched word, -1 will be put in R0.

1. **Test 2 program**

The following is the machine code of Test2

//ask users to input a searched word

|  |
| --- |
| 85 IN 0, 0  //load the content of M[18] to X3. M[18] holds value 2000 |
| 86 LDX 3, 18  // load the content of M[6] to R3. M[6] holds value 0 |
| 87 LDR 3, 0, 6  //load the content of M[19] to R2. M[19] holds value 1023 |
| 88 LDR 2, 0, 19  //load the content of M[18] to R1. M[18] holds value 2000 |
| 89 LDR 1, 0, 18  //Increase R1 by 3. Now R[1]=2003 |
| 90 AIR 1, 3  //Increase R2 by 1, now R[2]= 1024 |
| 91 AIR 2, 1  //Increase R1 by 1. R[1]= 2004 |
| 92 AIR 1, 1  //store R2 to M[14] |
| 93 STR 2,0,14  //load c(M[14]) to X1, X1=1024 |
| 94 LDX 1,14 |
| //load M[c(X1)] to R0. M[c(X1)]=M[1024]. It’s the ascii value of the first character of the 6 sentences  95 LDR 0, 1, 0  //R0 = c(R0) – 10. 10 is the decimal of LF – new line |
| 96 SIR 0, 10  // load M[16] to R2. M[16] = 141 |
| 97 LDX 2,16  // Jump to (X2+18) if R0 is 0. X2+18=159 |
| 98 JZ 0, 2, 18  // load M[c(X1)] to R0. M[c(X1)]=M[1024]. It’s the ascii value of the first character of the 6 sentences |
| 99 LDR 0, 1, 0  // R0 = c(R0) – 32. 32 is the ascii value of the white space |
| 100 SIR 0, 32  // load M[17] to X2. M[17]=183 |
| 101 LDX 2, 17  //jump to 183 if R0 is 0 |
| 102 JZ 0, 2, 0  //load M[c(X1)] to R0. M[c(X1)]=M[1024]. It’s the ascii value of //the first character of the 6 sentences |
| 103 LDR 0, 1, 0  // R0 = c(R0) – 257. M[13]=257 |
| 104 SMR 0, 0, 13  // load M[17] to X2. M[17]=183 |
| 105 LDX 2, 17  //Jump to 213 if R0 is equal to 0 |
| 106 JZ 0, 2, 30  // move R1 to M[14] |
| 107 STR 1, 0, 14 |
| // load M[14] to X1  108 LDX 1, 14  // load M[2004] to R0 |
| 109 LDR 0, 1, 0  // R0 = R0 – M[13], M[13] is 257 |
| 110 SMR 0, 0, 13 是否是257  // load M[17] to X2 |
| 111 LDX 2, 17  // jump to 208 if R0 is equal to 0 |
| 112 JZ 0, 2, 25  // load M[2004] to R0 |
| 113 LDR 0, 1, 0  // store R2 to M[14] |
| 114 STR 2, 0, 14  // load M[14] to R1 |
| 115 LDX 1,14  //R0 = R0 – M[2004] |
| 116 SMR 0, 1, 0  // load M[17] to X2 |
| 117 LDX 2, 17  // jump to 205 if R0 is equal to 0 |
| 118 JZ 0, 2, 22  // load M[6] to R3. M[6]=0 |
| 119 LDR 3, 0, 6  // load M[18] to R1, R1 now is 2000 |
| 120 LDR 1, 0, 18  // increase R1 by 4. R1 now is 2004 |
| 121 AIR 1, 4  // increase R2 by 1. Now R2 is 1025 |
| 122 AIR 2,1  // store R2 to M[14] |
| 123 STR 2, 0, 14  // load M[14] to X1 |
| 124 LDX 1,14  // load M[c(R1)] to R0 |
| 125 LDR 0, 1, 0  // R0 = R0 – 10 |
| 126 SIR 0, 10  // load M[16] to X2 |
| 127 LDX 2, 16  // Jump to 141 if R0 is equal to 0 |
| 128 JZ 0, 2, 0  // load M[2004] to R0 |
| 129 LDR 0, 1, 0  // R0 = R0 - 32 |
| 130 SIR 0, 32  // load M[16] to X2 |
| 131 LDX 2, 16  // jump to 150 if R0 is equal to 0 |
| 132 JZ 0, 2, 9  // load M[2004] to R0 |
| 133 LDR 0, 1, 0  // R0 = R0 – M[13]. M[13] is 257 |
| 134 SMR 0, 0, 13  // load M[17] to X2 |
| 135 LDX 2, 17  //jump to 213 if R0 is equal to 0 |
| 136 JZ 0, 2, 30  // load M[15] to X2 |
| 137 LDX 2, 15  //jump to 212 |
| 138 JMA 2, 31  //load M[18] to R1 |
| 139 LDR 1, 0, 18  //Increase R1 by 4 |
| 140 AIR 1, 4  // load M[6] to R3 |
| 141 LDR 3, 0, 6  // load M[2001] to R0 |
| 142 LDR 0, 3, 1  // Increase R0 by 1 |
| 143 AIR 0, 1  // store R0 to M[2001] |
| 144 STR 0, 3, 1  // load M[6] to R0 |
| 145 LDR 0, 0, 6  //store R0 to M[2002] |
| 146 STR 0, 3, 2  //increase R2 by 1 |
| 147 AIR 2, 1  // load M[15] to X2 |
| 148 LDX 2, 15  // jump to 93 |
| 149 JMA 2, 2  // load M[6] to R3 |
| 150 LDR 3, 0, 6  // load M[18] to R1 |
| 151 LDR 1, 0, 18  //Increase R1 by 4 |
| 152 AIR 1, 4  // load M[2002] to R0 |
| 153 LDR 0, 3, 2  //increase R0 by 1 |
| 154 AIR 0, 1  // store R0 to M[2002] |
| 155 STR 0, 3, 2  //increase R2 by 1 |
| 156 AIR 2, 1  // load M[15] to X2 |
| 157 LDX 2, 15  //Jump to 93 |
| 158 JMA 2, 2  //store R3 to M[14] |
| 159 STR 3, 0, 14  // load M[14] to R0 |
| 160 LDR 0, 0, 14  //R0 = R0 – M[2000] |
| 161 SMR 0,3,0  //load M[16] to X2 |
| 162 LDX 2, 16  //jump to 172 if R0 is not equal to 0 |
| 163 JNE 0, 2, 31  // load M[6] to R0 |
| 164 LDR 0, 0, 6  //store R0 to M[2003] |
| 165 STR 0, 3, 3  //load M[2001] to R0 |
| 166 LDR 0,3,1  //increase R0 by 1 |
| 167 AIR 0, 1  //output R0 to console printer |
| 168 OUT 0, 1  //load M[2002] to R3 |
| 169 LDR 3,3,2  //Increase R3 by 1 |
| 170 AIR 3, 1  //output R3 to console printer |
| 171 OUT 3, 1  //load M[2001] to R0 |
| 172 LDR 0, 3, 1  //increase R0 by 1 |
| 173 AIR 0, 1  //store R0 to M[2001] |
| 174 STR 0, 3, 1  //load M[6] to R0 |
| 175 LDR 0, 0, 6  //store R0 to M[2000] |
| 176 STR 0, 3, 2  // load M[6] to R3 |
| 177 LDR 3, 0, 6  //load M[18] to R1 |
| 178 LDR 1, 0, 18  //increase R1 by 4 |
| 179 AIR 1, 4  //increase R2 by 1 |
| 180 AIR 2, 1  // load M[15] to X2 |
| 181 LDX 2, 15  //jump to 93 |
| 182 JMA 2, 2  //store R3 to M[14] |
| 183 STR 3, 0, 14  //load M[14] to R0 |
| 184 LDR 0, 0, 14  //R0 = R0 – M[2000] |
| 185 SMR 0,3,0  //load M[17] to X2 |
| 186 LDX 2, 17  //Jump to 196 if R0 is equal to 0 |
| 187 JNE 0, 2, 13  //load M[6] to R0 |
| 188 LDR 0, 0, 6  //store R0 to M[2003] |
| 189 STR 0, 3, 3  //load M[2001] to R0 |
| 190 LDR 0,3,1  //increase R0 by 1 |
| 191 AIR 0, 1  //output R0 to console printer |
| 192 OUT 0, 1  //load M[2002] to R3 |
| 193 LDR 3,3,2  //Increase R3 by 1 |
| 194 AIR 3, 1  //output R3 to console printer |
| 195 OUT 3, 1  //load M[2002] to R0 |
| 196 LDR 0, 3, 2  //increase R0 by 1 |
| 197 AIR 0, 1  //store R0 to M[2002] |
| 198 STR 0, 3, 2  //load M[6] to R3 |
| 199 LDR 3, 0, 6  //load M[18] to R1 |
| 200 LDR 1, 0, 18  //increase R1 by 4 |
| 201 AIR 1, 4  //increase R2 by 1 |
| 202 AIR 2, 1  //load M[15] to X2 |
| 203 LDX 2, 15  //Jump to 93 |
| 204 JMA 2, 2  //increase R3 by 1 |
| 205 AIR 3, 1  //load M[15] to X2 |
| 206 LDX 2, 15  //Jump to 91 |
| 207 JMA 2, 0  //load M[18] to R1 |
| 208 LDR 1, 0, 18  //increase R1 by 4 |
| 209 AIR 1, 4  //load M[6] to R3 |
| 210 LDR 3, 0, 6  //load M[15] to X2 |
| 211 LDX 2, 15  //Jump to 122 |
| 212 JMA 2, 31  //store R3 to M[14] |
| 213 STR 3, 0, 14  //load M[14] to R0 |
| 214 LDR 0, 0, 14  //R0 = R0 – M[2000] |
| 215 SMR 0,3,0  //load M[20] to X2 |
| 216 LDX 2, 20  //jump to 225 if R0 is equal to 0 |
| 217 JNE 0, 2, 0  //load M[2001] to R0 |
| 218 LDR 0, 3, 1  //increase R0 by 1 |
| 219 AIR 0, 1  //output R0 to console printer |
| 220 OUT 0, 1  //load M[2002] to R3 |
| 221 LDR 3, 3, 2  //increase R3 by 1 |
| 222 AIR 3, 1  //output R3 to console printer |
| 223 OUT 3, 1  //jump to 224 |
| 224 JMA 2, 5  //load M[2003] to R0 |
| 225 LDR 0, 3, 3  //jump to 235 if R0 is equal to 0 |
| 226 JZ 0, 2, 5  //load M[6] to R0 |
| 227 LDR 0, 0, 6  //increase R0 by -1 |
| 228 AIR 0, -1  //output R0 to console printer |
| 229 OUT 0, 1  //load M[6] to R0 |
| 230 LDR 0, 0, 6  //store R0 to M[2001] |
| 231 STR 0, 3, 1  //store R0 to M[2002] |
| 232 STR 0, 3, 2  //increase R0 by 1 |
| 233 AIR 0, 1  //store R0 to M[2003] |
| 234 STR 0, 3, 3 |