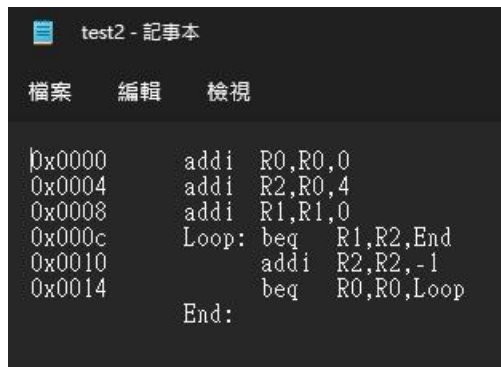


★ 我的 input:



```
test2 - 記事本
檔案 編輯 檢視

0x0000      addi  R0,R0,0
0x0004      addi  R2,R0,4
0x0008      addi  R1,R1,0
0x000c      Loop: beq   R1,R2,End
0x0010              addi  R2,R2,-1
0x0014              beq   R0,R0,Loop
                        End:
```

★ C++ code 說明:

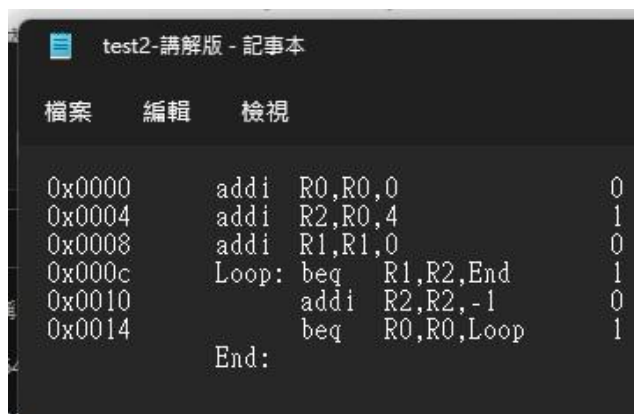
1. 宣告:



```
8   int his[17][3] = { 0 }; //紀錄history //000
9   int state[17][8] = { 0 }; //紀錄狀態// 0->SN // 1->WN // 2->WT // 3->ST
10  int which_address = 0; //紀錄當前output_address為第幾個
11  int a; //a & b是處理entry用的// a 表 entry數量
12  vector<string> output_entry[17]; //要輸出的entries
13  vector<string> output_address; //按照模擬的順序紀錄address
14  vector<string> address; //存讀到的address
15  vector<string> data_inst; //存讀到的instruction//addi or beq
16  vector<string> data_reg; //存讀到的register//R1 R2....
17  vector<int> data_entry; //從0開始數 存第幾行所對應到第幾個entry
18
```

特別講一下假如 entry 輸入為 2，則 a 為 2，data_entry 所記錄的是那行 instruction 會對應到第幾個 entry。

Ex: entry=2 時，a=2，每行所對到的 entry:



```
test2-講解版 - 記事本
檔案 編輯 檢視

0x0000      addi  R0,R0,0      0
0x0004      addi  R2,R0,4      1
0x0008      addi  R1,R1,0      0
0x000c      Loop: beq   R1,R2,End 1
0x0010              addi  R2,R2,-1 0
0x0014              beq   R0,R0,Loop 1
                        End:
```

2. 輸入 entry 數量: Entry 是看 PC 的，且因為是 bit，所以不會是奇數個。
輸入完 entry 數量後，根據 entry 數量做對應數量的 entry 初始化

```
130
131 int main()
132 {
133     string str;
134
135     int b = 0;
136     cout << "entry?" << endl;
137     cin >> a; // a 表 entry 數量
138     while (a % 2 == 1) cin >> a; // entry 因為是看 pc 裡的 bit 所以不會有奇數個
139     cout << endl;
140     begin_output(a); // 初始化所有 entry
141
142 }
```

3. 初始化: 呼叫 bbegin_output，用到幾個 entry 就初始化幾個，初始狀態皆為: (000, SN SN SN SN SN SN SN SN)

```
19 void begin_output(int a) // 初始化 output_entry
20 {
21     string buffer = { " ( 000, SN SN SN SN SN SN SN SN )" };
22     for (int i = 0; i < a; i++)
23     {
24         output_entry[i].push_back(buffer);
25     }
26 }
27
```

4. 讀檔: 讀檔我分成三個部分在讀: address/instruction/register，例如: 0x0000 addi R0,R0,0，我會把 0x0000 存到 address，addi 存到 data_inst，R0,R0,0 存到 data_reg 裡，並同時按行的順序紀錄對到第幾個 entry 到 data_entry 中。

```
142
143 int n = 0, m = -1, loop_num, back_num; // 每行讀進來的分三個部分存: address/inst/reg 用 n 做紀錄
144 // m 用來算 Loop 從第幾行開始以及到第幾行結束
145
146 ifstream infile("test2.txt", ios::in);
147 if (!infile.is_open())
148 {
149     cout << "file can not be open" << endl;
150     return 0;
151 }
152 else
153 {
154     while (infile >> str)
155     {
156         if (str == "R0,R0,Loop") back_num = m; // Loop 結束的位置
157         if (str == "Loop:") loop_num = m; // Loop 開始的位置
158         else if (str == "End:") break; // End 不理她
159         else
160         {
161             if (n == 0) // n=0 時為 address
162             {
163                 address.push_back(str), m++, n++;
164
165                 // 給予對應的第幾個 entry
166                 if (b != a - 1) data_entry.push_back(b), b++;
167                 else data_entry.push_back(b), b = 0;
168             }
169             else if (n == 1) data_inst.push_back(str), n++; // n=1 instruction
170             else data_reg.push_back(str), n = 0; // n=2 register
171         }
172     }
173 }
```

5. 針對讀到的 **input** 做模擬執行，執行在 **simulation** 這個 **array** 中，**simulation[2]** 就會是 **R2**，以此類推，並且我將 **input** 分成兩個部分，**Loop** 以前與 **Loop**，以我的 **input** 來說就是分成下面這樣

```
0x0000    addi  R0,R0,0
0x0004    addi  R2,R0,4
0x0008    addi  R1,R1,0
```

Figure 1

```
0x000c    Loop: beq  R1,R2,End
0x0010          addi  R2,R2,-1
0x0014          beq  R0,R0,Loop
          End:
```

Figure 1

```
175     int simulation[10] = { 0 }; //模擬執行的主角 存的為對應的reg的值//ex: R2=simulation[2]
176     for (int i = 0; i < loop_num; i++) //處理Loop以前
177     {
178         int negtive = 1; //處理負值
179         int fir, sec, value; // addi 的三個主角 // ex:addi R2,R0,4 // fir=R2 sec=R0 value=4
180         int times = 1; //第一次讀到數字就是fir依序為sec value
181
182         output_address.push_back(address[i]); //對應的地址 & entry
183         callfuc(i, 'N'); // 這部分沒有任何branch//均為 N
184
185         for (auto& k : data_reg[i])
186         {
187             if (k == '-') negtive = -1;
188             if (isdigit(k))
189             {
190                 if (times == 1) fir = atoi(&k), times++;
191                 else if (times == 2) sec = atoi(&k), times++;
192                 else
193                 {
194                     times = 0, value = atoi(&k) * negtive;
195                     if (negtive == -1) negtive = 1;
196                 }
197             }
198         }
199         simulation[fir] = simulation[sec] + value; //因為只會有addi
200     }
201 }
```

這邊是先處理 **Figure 1** 的部分，將有用到的 **register** 根據 **addi** 給值，並在每行執行完後就呼叫 **function** 做預測，且因為這個部分都是 **addi**，所以 **outcome** 一定為 **N**。

6. 呼叫 **function** 做預測: 做預測時會先呼叫 `callfuc()` 先將我這行的 code 輸出再呼叫 `solution()` 做預測。

```
121
122 void callfuc(int num, char pre)
123 {
124     for (auto& n : data_inst[num])cout << n;
125     cout << " ";
126     for (auto& n : data_reg[num])cout << n;
127     cout << " ";
128     solution(data_entry[num], pre);
129 }
```

```
28
29 void solution(int ent, char u) // ent 表第幾個entry // u 表outcome
30 {
31     vector<string> temp; //目前所要的entry
32
33     //output address
34     cout << output_address[which_address] << " ";
35     cout << "entry: " << ent << endl;
36
37     which_address++;
38
39     int num = his[ent][2] * 1 + his[ent][1] * 2 + his[ent][0] * 4; //2進位轉10進位
40     char pre; //prediction
41
42     if (state[ent][num] == 0 || state[ent][num] == 1) pre = 'N'; //do predict
43     else pre = 'T';
44
45
46     for (int i = 0; i < a; i++) //output entries
47     {
48         for (int j = 0; j < output_entry[i].size(); j++)
49         {
50             cout << output_entry[i][j];
51         }
52         cout << endl;
53     }
54
55     //output outcome & prediction
56     cout << "Outcome: " << u << " " << " Prediction: " << pre << " >> ";
57
58     if (u == pre) cout << "correct"; //check predict susses, or not
59     else cout << "miss";
60
61     //update state & history
62     his[ent][0] = his[ent][1];
63     his[ent][1] = his[ent][2];
64
65     if (u == 'T')
66     {
```

```

64
65     if (u == 'T')
66     {
67         if (state[ent][num] == 0 || state[ent][num] == 1 || state[ent][num] == 2)state[ent][num]++;//ST狀態遇到T不變
68         his[ent][2] = 1;//update history
69     }
70     else //u==N
71     {
72         if (state[ent][num] == 3 || state[ent][num] == 1 || state[ent][num] == 2)state[ent][num]--;//SN狀態遇到N不變
73         his[ent][2] = 0;
74     }
75
76     //update history in answer vector
77     temp.push_back(" ( ");
78     for (auto& n : his[ent])
79     {
80         string m(to_string(n));
81         temp.push_back(m);
82     }
83     temp.push_back(", ");
84
85     for (auto& m : state[ent])//update states
86     {
87         if (m == 0) temp.push_back("SN ");
88         else if (m == 1)temp.push_back("WN ");
89         else if (m == 2)temp.push_back("WT ");
90         else temp.push_back("ST ");
91     }
92
93     temp.push_back(") ");
94     output_entry[ent] = temp;//update ent'th row of output_entry
95
96     cout << endl << endl << endl;
97
98 }
99

```

Solution()的參數 `ent` 表示我現在這行 code 是對到第幾個 `entry`，而 `u` 是 outcome，一開始先輸出這行 code 的 address 以及為第幾個 `entry`，然後根據 history 所轉成的 10 進位數字與對到的 `entry` 做預測，假如是第 1 個 `entry` 且 history 為 000，000 轉成 10 進位是 0，所以會檢查 `state[1][0]` 的狀態，如果裡面數字為 0 則為 SN，1 為 WN，2 為 WT，3 為 ST，然後根據狀態給 `pre` (我的預測結果)T 或 N，之後輸出所有 `entry(output_entry)`、`outcome(u)`、我的預測 (`pre`)，以及預測是否正確。接下來要更新 `entry` 裡的資訊，根據 outcome 的結果更新 history 與 state(狀態)，outcome 為 T 時，history 每項往前一項移並把最後一項更新成 1，然後 state 裡面的數字會加 1 表示轉為下一個狀態，如果原本已經是 ST 則不用改變；outcome 為 N 時，就相反，history 同理但最後一項變為 0，state 數字減 1 退一個狀態，若 SN 則不用，之後根據更新的結果，將新的 `entry` 字串藉由 `temp` 來做更新。

7. 模擬 Loop: 這裡我分成兩部分做，一開始先根據 code 找出 Loop 的 beq 所要用到的兩個 register，即 beq_1 與 beq_2，以及找出 Loop 裡面的 addi 會用到的 register 與值是多少，存在 middle_addi 與 value 中。

```
179
180 int beq_1, beq_2; //LOOP的branch條件 //當 beq_1==beq_2 -> 跳
181 int middle_addi[10][2] = { 0 }; //LOOP裡的addi
182 int value[10] = { 0 }; //LOOP裡的addi的value
183 for (int i = loop_num; i < back_num; i++) //處理Loop
184 {
185     int times = 1; //作用跟上面的差不多
186     int negative = 1;
187
188     for (auto& k : data_reg[i])
189     {
190         if (k == '-') negative = -1;
191         if (data_inst[i] == "beq" && i != back_num) //紀錄LOOP所要用到的兩個register
192         {
193             if (isdigit(k))
194             {
195                 if (times == 1) beq_1 = atoi(&k), times++;
196                 else beq_2 = atoi(&k), times = 1;
197             }
198         }
199         else
200         {
201             if (isdigit(k)) //LOOP裡執行的addi們
202             {
203                 if (times == 1) middle_addi[i][0] = atoi(&k), times++;
204                 else if (times == 2) middle_addi[i][1] = atoi(&k), times++;
205                 else
206                 {
207                     times = 1;
208                     value[i] = negative * atoi(&k);
209                     if (negative == -1) negative = 1;
210                 }
211             }
212         }
213     }
214 }
```

8. **模擬 Loop 執行:** 模擬 Loop 的第二部分，藉由剛剛找到的 `beq_1` 與 `beq_2` 可以知道所要的 `registor` 是哪兩個，假如 `beq_1 = 1`，`beq_2 = 2`，則表

`beq R1,R2,End (simulation[1], simulation[2])`。只要進迴圈就代表這個 `beq` 沒發生，所以 `outcome` 為 `N`，然後呼叫預測，之後跑 Loop 裡面的 `addi`，其 `outcome` 必為 `N`，一樣每跑一行 `code` 就做一次預測，直到最後一行的 `beq R0,R0,Loop`，因為他一定會發生，所以 `outcome` 為 `T`，然後一樣呼叫預測。一直做直到跳出迴圈就代表 Loop 裡的第一個 `beq` taken 了，所以 `outcome` 為 `T`，呼叫預測，然後 `End` 結束。

```
216
217 //模擬LOOP
218 for (int i = loop_num; simulation[beq_1] != simulation[beq_2]; i = loop_num)//Loop: beq R1,R2,End
219 {
220     output_address.push_back(address[i]);
221     callfuc(i, 'N');
222
223     i++;
224     for (int j = i; j <= back_num; j++)
225     {
226         if (data_inst[j] == "addi")
227         {
228             output_address.push_back(address[j]);
229
230             simulation[middle_addi[j][0]] = simulation[middle_addi[j][1]] + value[j];
231             callfuc(j, 'N');
232         }
233         else
234         {
235             output_address.push_back(address[j]);
236             callfuc(j, 'T');
237         }
238     }
239 }
240
241 output_address.push_back(address[loop_num]);
242 callfuc(loop_num, 'T');
243
244 infile.close();
245 return 0;
246
```

9. **注意事項:**

- `entry` 能為 2、4、8 或 16，要再上去的話改一下宣告那邊就行了。
- `instruction` 只能出現 `addi` & `beq`。
- `input` 須維持這個上下兩部分的模式，且 `Loop` 要在下面的部份並只能有一個，但兩部分的 `code` 中間要再加個幾行 `addi` 是沒問題的。