1.

A) Assume 1K (1024) entry for Pattern History Table (PHT) with 2-bit Counter per entry (predictor index size of 2 bits), implement and compare the mis-prediction of the following branch predictors:

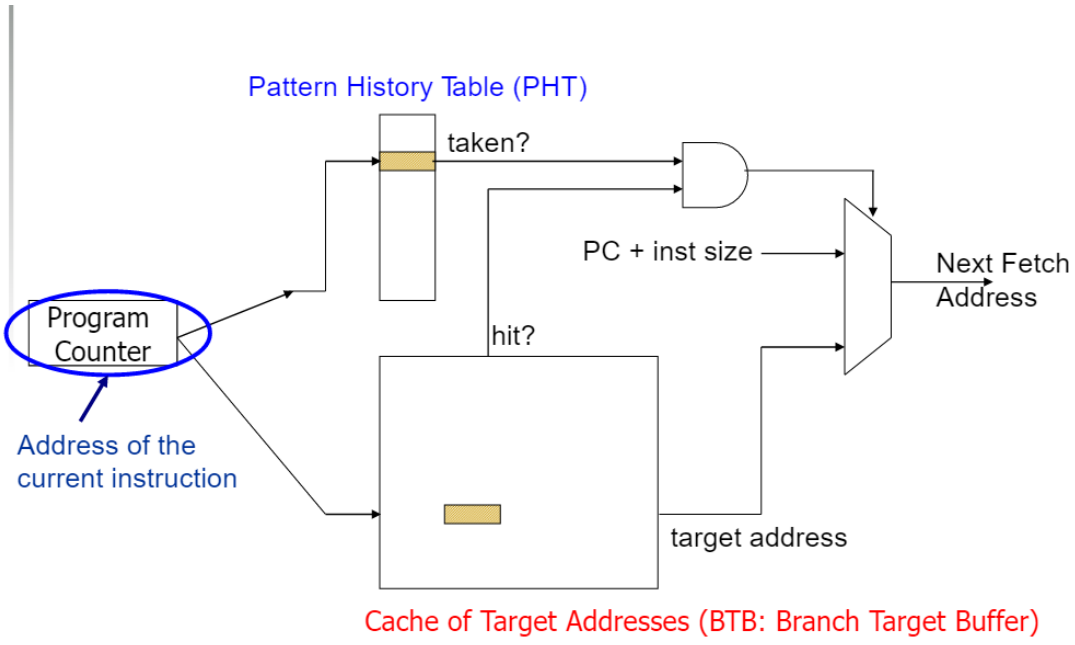
i. One level branch Prediction
ii. Two Level Global Branch Prediction
iii. Two Level Gshare Branch Predictor
iv. Two Level Local Branch Prediction

The code for realizing the different branch predictors can be accessed by the link given below -

The full code is also added at the end of this report as screenshots.

i. One level branch predictor

The schematic of a One level branch predictor is shown below.



The program counter is fed into the pattern history table of 1024 entries which has two bit saturating counters corresponding to the PC. The counters values are

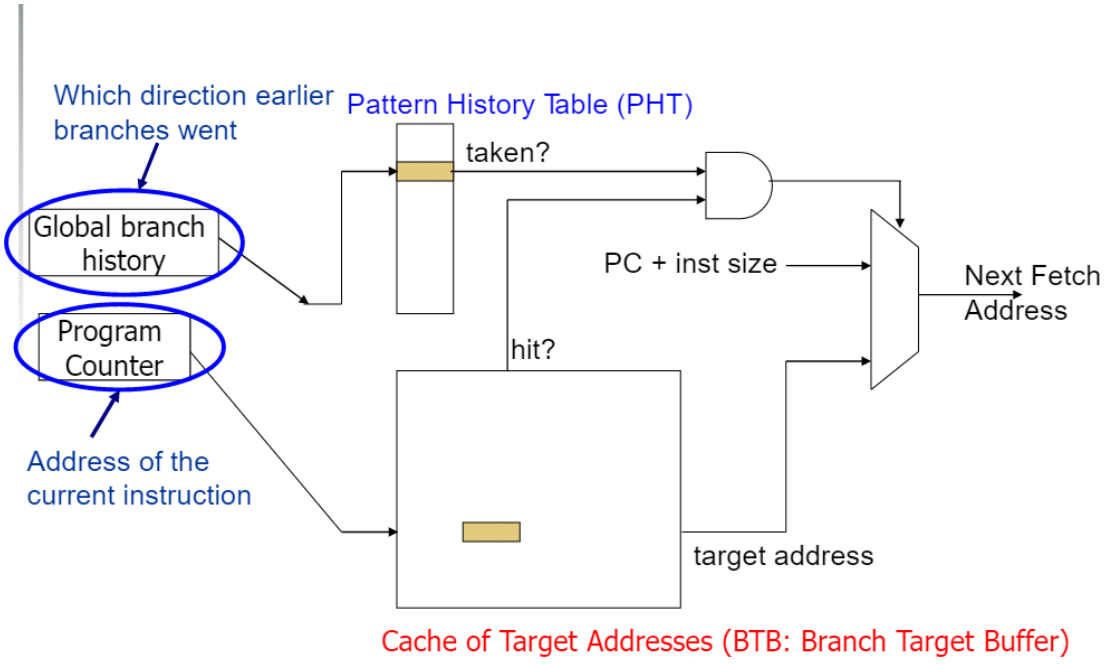
considered as follows-
00 – Strongly not taken
01 – Weakly not taken
10 – Weakly taken
11 – Strongly taken

Based on the value of the counter in the PHT, a branch prediction decision will be taken.

One level prediction gave a **misprediction rate of 15.46%** for the given trace file for a PHT size of 1024.

ii. Two Level Global Branch Prediction

The schematic of Two level global branch prediction is shown below.

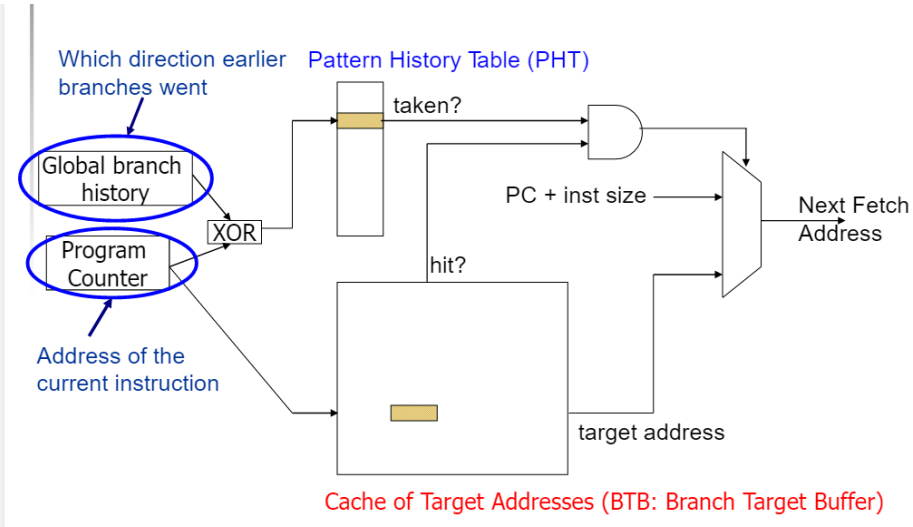


Since we are using a 1024 entry PHT, the global branch history register (GBHR) is taken as 10 bit. The GBHR will hold the history of the path taken for the last 10 branches in this case. The direction this pattern took is maintained as two bit saturating counters, and this is used to predict the branch outcome.

A two level global branch predictor gave a **misprediction rate of 22.56%** for a PHT size of 1024 entries**.**

iii. Two level gshare branch predictor

The diagram of a two level gshare branch predictor is given below-

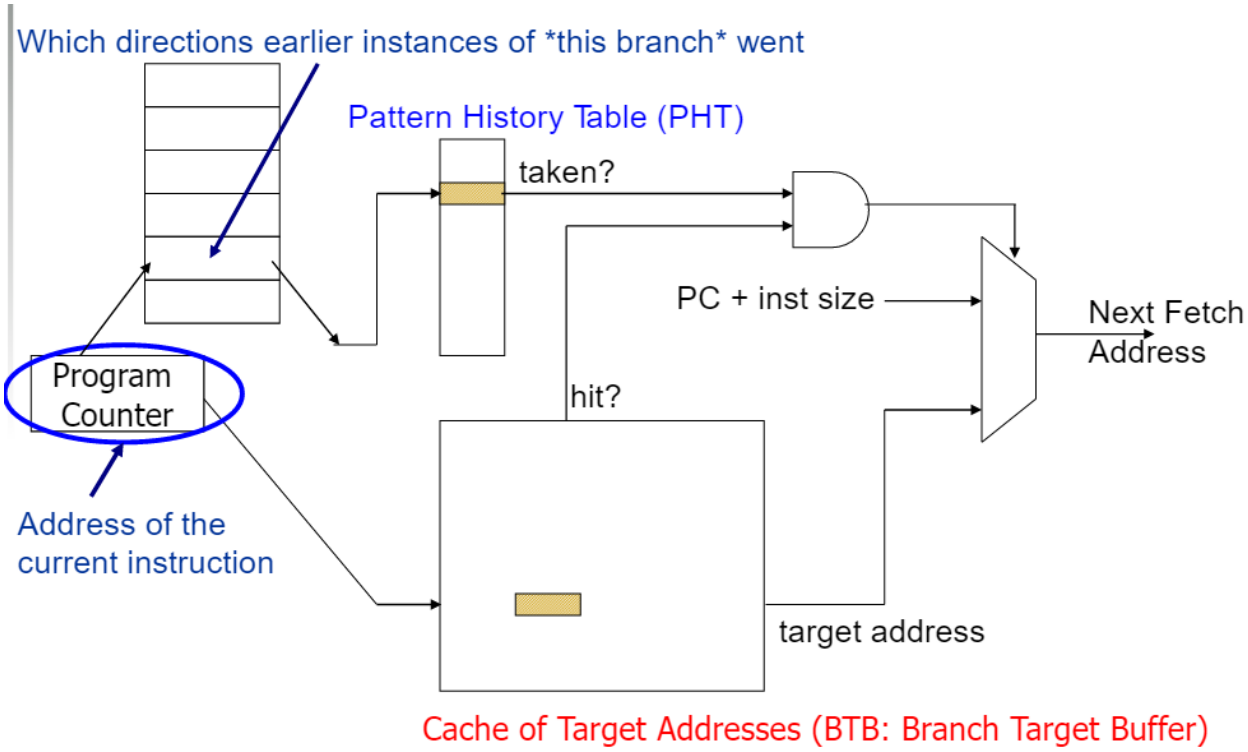


In this predictor, the history of last n branches and the PC are XOR'ed, and the resulting pattern is used to refer to the pattern history table (PHT). Using the PC along with global history will give more context to the global history, and will prevent aliasing. The pattern obtained is used to find the branch path (taken or not taken).

A two level gshare branch predictor gave a **misprediction rate of 21.52%** for PHT size of 1024 entries.

iv. Two level local branch prediction

The below figure shows the working of a two level local branch predictor.



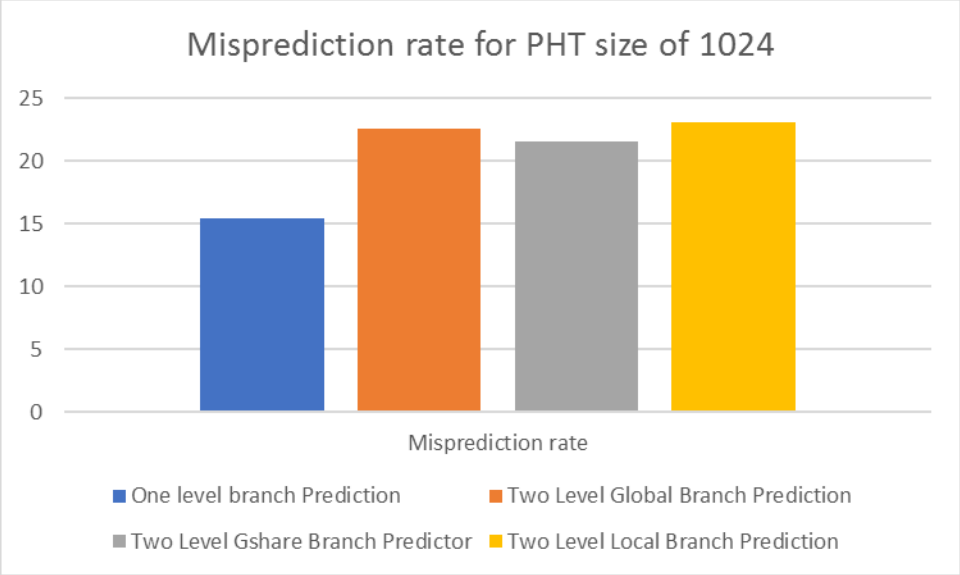
The two level local branch predictor uses a table of local branch histories. The local branch histories are 10-bit wide as the PHT is of 1024 entries. The local branch history table contains 128 such 10-bit local history entries (as given in the question). The local history pattern is used to refer to the PHT, and find the branch direction.

A two level local branch predictor gave a **misprediction rate of 23.11%** for PHT size of 1024 entries.

Results

The below table and graph shows the misprediction rate for the address trace.

Policy	Misprediction rate
One level branch Prediction	15.461743
Two Level Global Branch Prediction	22.56724
Two Level Gshare Branch Predictor	21.520515
Two Level Local Branch Prediction	23.113783



As we can see, a single level branch predictor gives the best results in the case of PHT size of 1024.

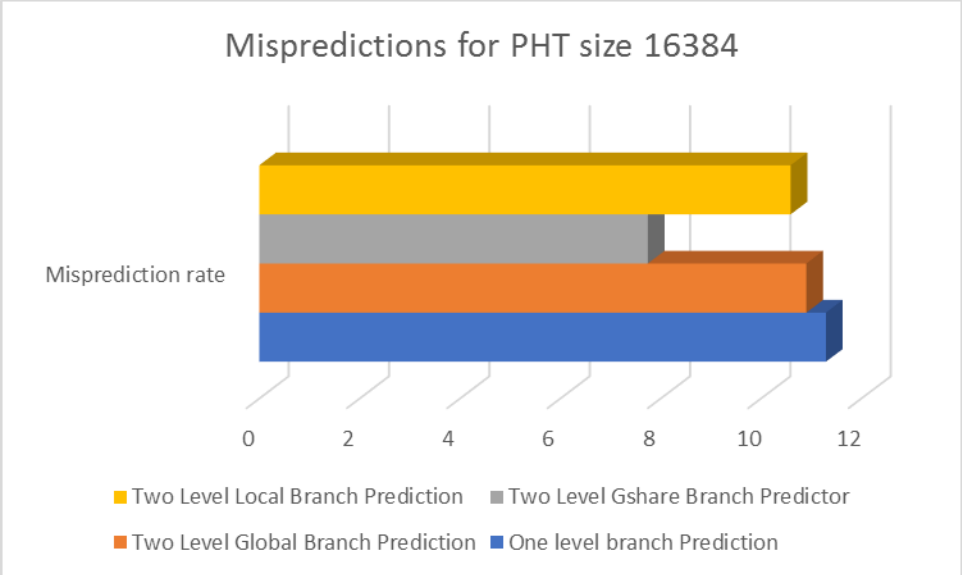
The two level global predictor gives bad performance because the PHT size of 1024 entries means that we can use only 10 bits for global branch history. This may result in a lot of conflict in the patterns, and would have resulted in an increased misprediction rate.

The two level gshare predictor gives a slightly better performance compared to the global branch predictor as expected, but still the 10 bits of history is the bottleneck of this policy too.

The two level local branch predictor also gives very inferior performance. The root cause of this is that we are only using a 128 entry local history table. This will result in a lot of conflicts in the table as we go through a wide range of PCs. Therefore, local branch predictor is inferior in this case.

In order to prove the above mentioned reasons, we can try using a 16384 entry PHT, and a local history table of 1024 entries.

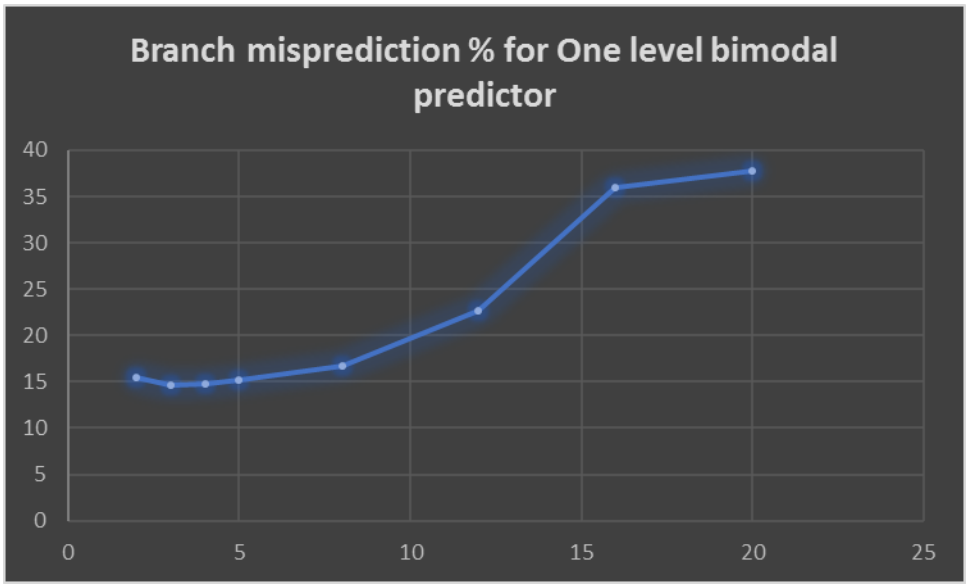
Policy	Misprediction rate
One level branch Prediction	11.292711
Two Level Global Branch Prediction	10.902562
Two Level Gshare Branch Predictor	7.744678
Two Level Local Branch Prediction	10.590464

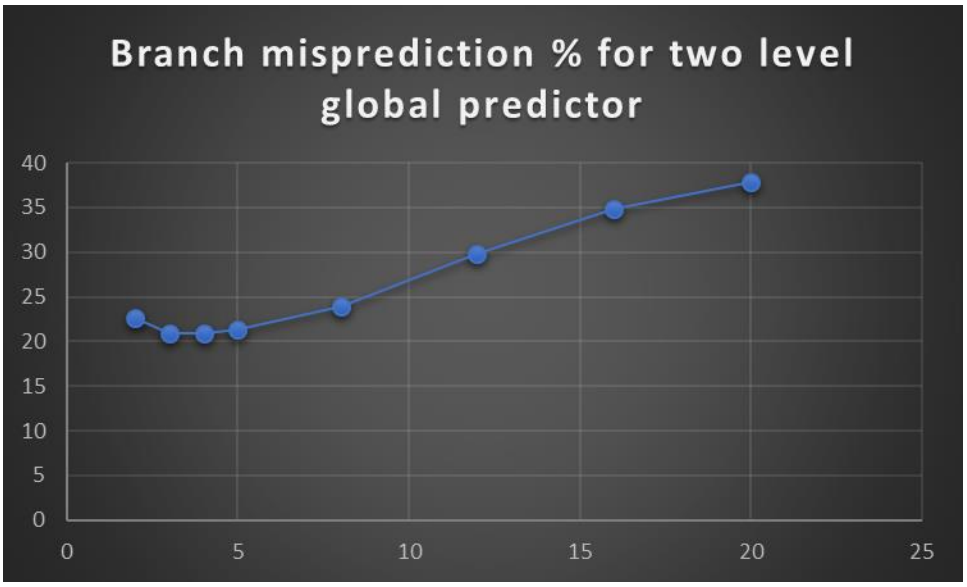


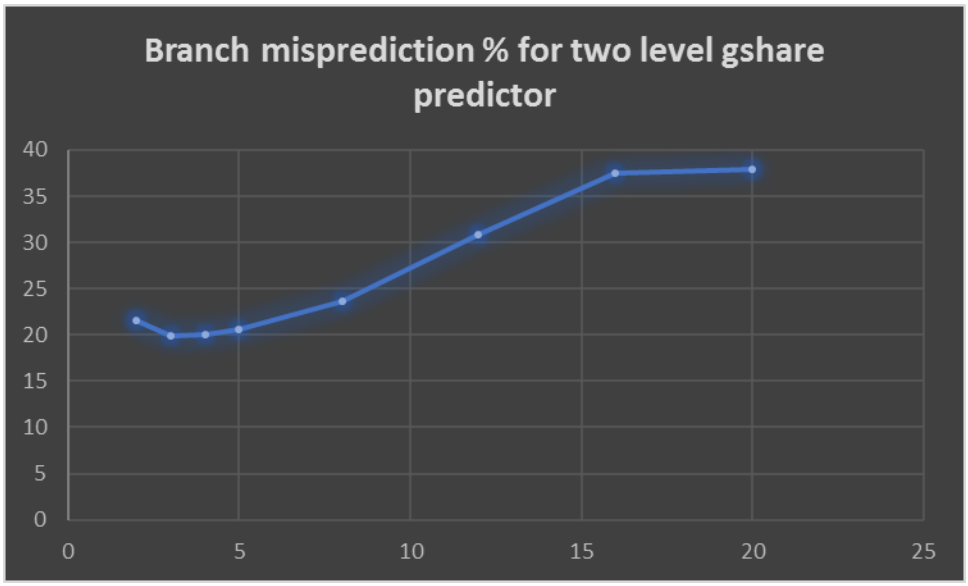
As expected, the two level predictors have much better performance since the amount of history used has increased. Also, gshare gives the best performance for PHT size of 16384, because of increased context due to the use of PCs.

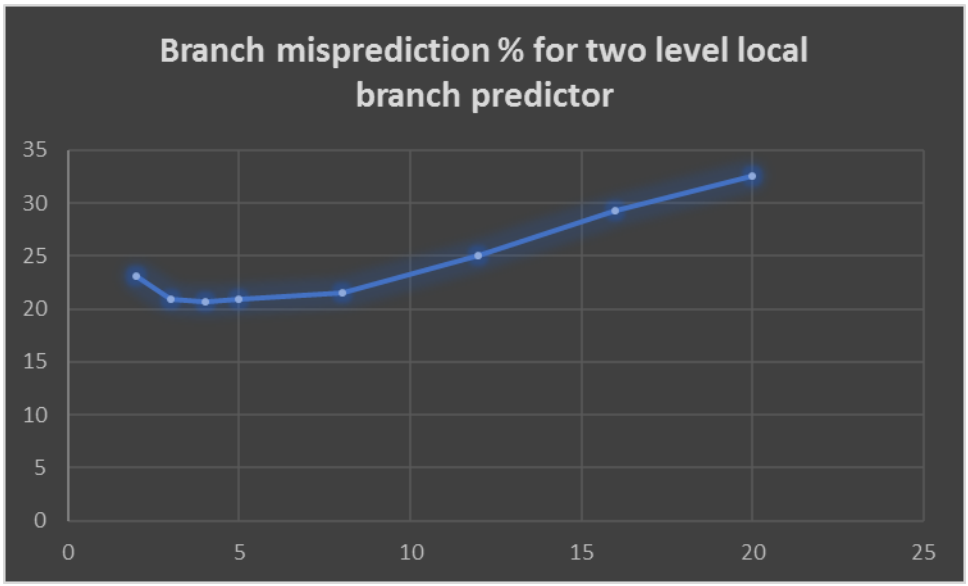
1 B. Repeat the Problem 1.A, this time with variable entry size (n-bit Counter) with size of 2 bits, 3 bits, 4 bits, 5 bits, ... 20 bits. Generate a dedicated line plot of the data using MS Excel or some other graphing program for each branch predictor. On the y-axis, plot "percentage of branches mis-predicted". On the x-axis plot the the predictor size (basically, the width of n-bit counter). Draw a separate plot for each branch predictor. Analyze the data and argue the effect of increasing counter bits on branch predictor performance. In your view, how many bits seems sufficiently enough for branch predictor?

The graphs of the misprediction percent for different prediction schemes corresponding to various counter size is given below.









From the graphs, it is easy to notice that on increasing the counter size from two to three there is an immediate decrease in the misprediction percentage. But, from then on the performance decreases gradually on increasing the counter size. Counters till 5-bit width has almost the same or better performance as a two-bit counter. But after that, increasing the counter width decreases performance quite severely.

As we increase the number of bits in the saturating counter, the number of mispredictions it takes to change the current prediction also increases. Take for example, a two-bit, 3-bit and 12-bit counter. For a two-bit counter, there are four states, therefore, at most two mispredictions will result in a change in state from taken to not taken or vice versa. But, in the case of a 3-bit counter with 8 states, it will take a maximum of 4 mispredictions to

change the prediction. But, in the case of a 12-bit one, it will take 2048 mispredictions for a state change. Therefore, higher bit counters will result in higher latency in changing the prediction, and will result in a lot of unwanted mispredictions. Therefore, it is better not to use many bits for the saturating counters.

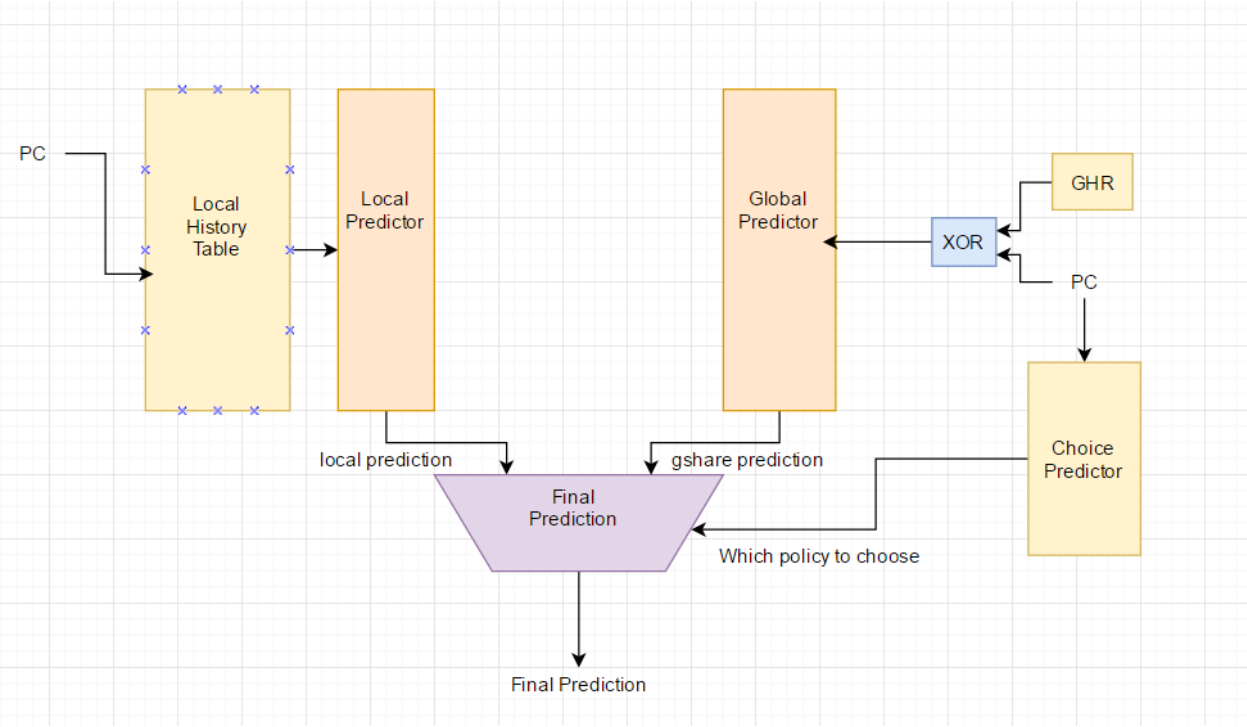
Further, three-bit counters give better performance that the other alternatives for all the schemes except the two-level local branch predictor. Even in the two-level local branch predictor, the performance of the three-bit counter is almost at par with the best performance observed. **Therefore, it can be concluded that three-bit counters give the best performance.**

So, three-bit counters are sufficiently enough for a branch predictor.

Extra credit question

A tournament predictor similar to (but uses gshare) an alpha 21264 tournament predictor was designed. The design essentially consists of a **two-level local predictor** and a **two-level gshare predictor** working in tandem was done. A 4096 entry choice predictor with 3-bit saturating counters is used to choose whether to use local predictor or gshare predictor.

A schematic of the predictor is given below-

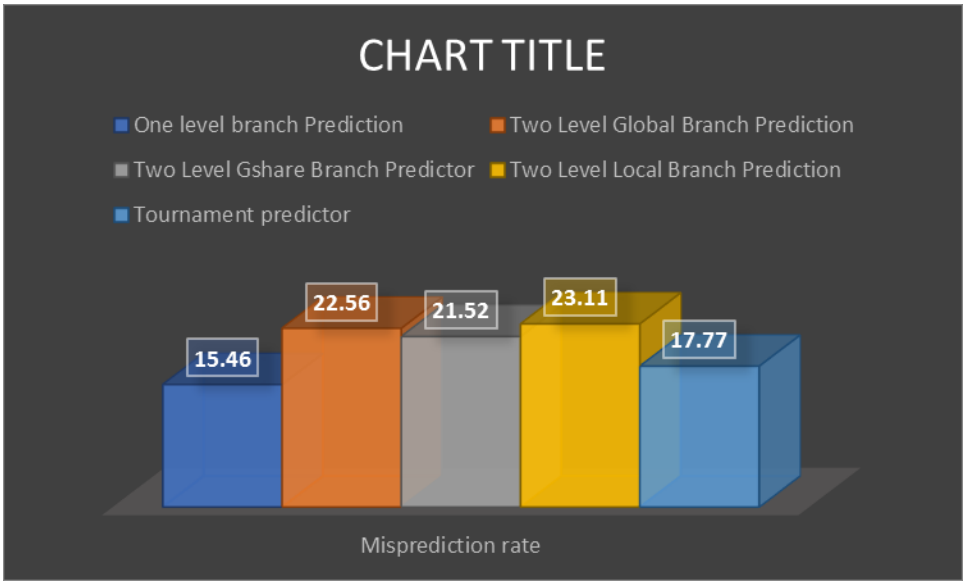


The design that I have implemented is different from the alpha predictor, but it uses a similar concept. The local predictor is initially selected, and based upon the misprediction in the local predictor it may shift to a gshare predictor. The saturating counters in the choice predictor table decides which selector to use. The choice predictor is accessed using the program counter.

Results

Tournament predictor –

Specifications 1- 1024 entry PHT (2-bit counter), 128 entry local history table, 4096 entry choice predictor (3-bit counter)



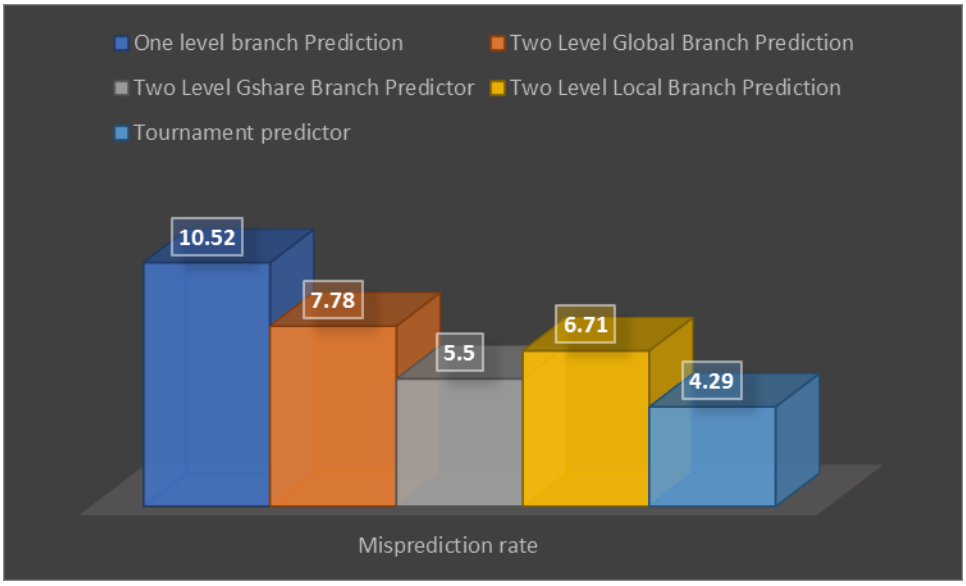
As we can see, for a PHT size of 1024 with 2-bit saturating counters, a tournament predictor has much better performance than its constituent gshare and local predictor. This is because this tournament predictor will choose the best out of gshare and local. When a set of addresses that suits gshare predictor comes, the choice predictor will choose the gshare one as soon as the local predictor starts mispredicting. The same phenomenon occurs in the reverse case too. That is the reason why tournament predictor gives better performance.

Next, I tried to maximize the accuracy of the tournament predictor by changing the metrics, while at the same time making the changes realistic in a real system.

The **specifications** of the system are as follows-

Local PHT = Global PHT = 65536 entries
No of bits in the Global history register = No of bits in Local history register = 16
Saturating counters used = 3 bits
Local history table entries = 65536

Using a 16-bit branch history (as in most branch predictors used in industry), the following results were obtained.



Thus, the tournament predictor gives a misprediction rate of 4.29 %.

That is, the **prediction accuracy is 95.71%.**

Code

The code given below is for the following specifications. The values of variables must be changed at different junctures of the assignment to get specific results.

Local PHT = Global PHT = 65536 entries
No of bits in the Global history register = No of bits in Local history register = 16
Saturating counters used = 3 bits
Local history table entries = 65536

```
#include <stdio.h>
#include <stdbool.h>

unsigned int pht1level [65536];
unsigned int pht2level [65536];
unsigned int gshare [65536];
unsigned int phtlocal [65536];
unsigned int choosertable [65536];

struct globalpattern {
  unsigned int ghr:16;
}pattern, pattern1, localpattern [65536];


void phtinit(void);
bool onelevelpred(unsigned long int addr, char a);
bool l2globalmispred (char a);
bool gshared (unsigned long int address, char a);
bool localhistory (unsigned long int address, char a);
bool tournamentchooser (unsigned long int address, char a, bool localpred, bool gsharepred);


void main()
{
  FILE *fp, *fp1;
  int i, type;
  unsigned long int address, instr;
  unsigned int masked_addr;
  char a;

  phtinit();

  // open trace file for reading

  fp = fopen("branch-trace-gcc-file.trace", "r");
  int l1mispred = 0;
  int l2mispred = 0;
  int gsharemispred = 0;
  int localmispred = 0;
  int tourmispred = 0;


  // read first 100 lines of trace file;
```

```
48    while(!feof (fp))
49    {
50
51      // read 1 line of trace file
52
53      fscanf(fp, "%ld %c", &address, &a);
54
55      // output virtual address in hex
56
57      //printf("%ld %c\n", address,a);
58      if (!onelevelpred(address, a))
59        l1mispred++;
60      if (!l2globalmispred(a))
61        l2mispred++;
62      if (!gshared(address, a))
63        gsharemispred++;
64      if (!localhistory(address, a))
65        localmispred++;
66    }
67    fclose(fp);
68
69
70    fp1 = fopen("branch-trace-gcc-file.trace", "r");
71    phtinit();
72    while(!feof (fp1))
73    {
74      fscanf(fp1, "%ld %c", &address, &a);
75
76      if (!tournamentchooser(address, a, localhistory(address, a), gshared(address,a)))
77        tourmispred++;
78
79    }
80
81    fclose(fp1);
82    float l1, l2, gshr, loc, tour;
83    l1 = l1mispred*100.0/16416279;
84    l2 = l2mispred*100.0/16416279;
85    gshr = gsharemispred*100.0/16416279;
86    loc = localmispred*100.0/16416279;
87    tour = tourmispred*100.0/16416279;
88
89    printf("%d %f\n%d %f\n%d %f\n%d %f\n%d %f\n", l1mispred, l1, l2mispred, l2,
90      gsharemispred, gshr, localmispred, loc, tourmispred, tour);
```

```
91
92  }
93
94  bool onelevelpred(unsigned long int address, char a)
95  {
96    int index = (address >> 2) % 65536;
97    bool prediction;
98
99    if (pht1level[index] <= 3)
100   {
101     if (a == 'T')
102     {
103       pht1level[index]++;
104       prediction = false;
105
106     }
107     else
108       if (a == 'N')
109       {
110         if (pht1level[index] > 0)
111             pht1level[index]--;
112
113         prediction = true;
114       }
115
116   }
117   else if (pht1level[index] > 3)
118   {
119     if (a == 'N')
120     {
121       pht1level[index]--;
122       prediction = false;
123     }
124     else
125       if (a == 'T')
126       {
127         if(pht1level[index] < 7)
128           pht1level[index]++;
129         prediction = true;
130       }
131   }
132   return prediction;
133
134 }
```

```
136  void phtinit()
137  {
138
139    int i;
140
141    for (i=0;i<65536;i++)
142    {
143      pht1level[i] = 0;
144    }
145    for (i=0;i<65536;i++)
146    {
147      pht2level[i] = 0;
148    }
149    for (i=0;i<65536;i++)
150    {
151      gshare[i] = 0;
152    }
153    for (i=0;i<65536;i++)
154    {
155      phtlocal[i] = 0;
156    }
157    for (i=0;i<65536;i++)
158    {
159      choosertable[i] = 0;
160    }
161
162
163
164    pattern.ghr = 0;
165
166
167  }
168
169  bool l2globalmispred (char a)
170  {
171
172    int index = pattern.ghr;
173    //index%=1024;
174    char status;
175    if (a=='T')
176      status = 1;
177    if (a=='N')
178      status = 0;
179
```

```
180     pattern.ghr = pattern.ghr << 1 | status;
181
182     //printf ("%d\n", status);
183
184     bool prediction;
185
186     if (pht2level[index] <= 3)
187     {
188       if (a == 'T')
189       {
190         pht2level[index]++;
191         prediction = false;
192
193       }
194       else
195         if (a == 'N')
196         {
197           if (pht2level[index]>0)
198               pht2level[index]--;
199
200           prediction = true;
201         }
202
203     }
204     else if (pht2level[index] > 3)
205     {
206       if (a == 'N')
207       {
208         pht2level[index]--;
209         prediction = false;
210       }
211       else
212         if (a == 'T')
213         {
214           if(pht2level[index] < 7)
215             pht2level[index]++;
216           prediction = true;
217         }
218     }
219     return prediction;
220
221
222
223   }
```

```
225  bool gshared (unsigned long int address, char a)
226  {
227
228    int index = pattern1.ghr;
229    //index%=1024;
230    char status;
231    if (a=='T')
232      status = 1;
233    if (a=='N')
234      status = 0;
235
236    pattern1.ghr = pattern1.ghr << 1 | status;
237
238    index = 0xFFFF & (index ^ ((address) & 0xFFFF));
239
240
241    bool prediction;
242
243    if (gshare[index] <= 3)
244    {
245      if (a == 'T')
246      {
247        gshare[index]++;
248        prediction = false;
249
250      }
251      else
252        if (a == 'N')
253        {
254          if (gshare[index]>0)
255              gshare[index]--;
256
257          prediction = true;
258        }
259
260    }
261    else if (gshare[index] >3)
262    {
263      if (a == 'N')
264      {
265        gshare[index]--;
266        prediction = false;
267      }
268      else
```

```
269         if (a == 'T')
270         {
271           if(gshare[index] < 7)
272             gshare[index]++;
273           prediction = true;
274         }
275     }
276     return prediction;
277 
278 
279 
280   }
281 
282   bool localhistory (unsigned long int address, char a)
283   {
284 
285     int localindex = (address) % 65536;
286 
287     int index = localpattern[localindex].ghr;
288 
289     char status;
290     if (a=='T')
291       status = 1;
292     if (a=='N')
293       status = 0;
294 
295     localpattern[localindex].ghr = localpattern[localindex].ghr << 1 | status;
296 
297     bool prediction;
298 
299     if (phtlocal[index] <=3)
300     {
301       if (a == 'T')
302       {
303         phtlocal[index]++;
304         prediction = false;
305 
306       }
307       else
308         if (a == 'N')
309         {
310           if (phtlocal[index]>0)
311               phtlocal[index]--;
312 
313           prediction = true;
```

```
314         }
315
316     }
317     else if (phtlocal[index] >3)
318     {
319       if (a == 'N')
320       {
321         phtlocal[index]--;
322         prediction = false;
323       }
324       else
325         if (a == 'T')
326         {
327           if(phtlocal[index] < 7)
328             phtlocal[index]++;
329           prediction = true;
330         }
331     }
332     return prediction;
333
334   }
335
336
337   bool tournamentchooser (unsigned long int address, char a, bool localpred, bool gsharepred)
338   {
339     int tour_index = address % 65536;
340
341     if (choosertable [tour_index] <= 3)
342     {
343       if (!localpred)
344       {
345         choosertable [tour_index]++;
346         return false;
347       }
348       else
349       {
350         if (choosertable[tour_index]>0)
351           choosertable [tour_index]--;
352         return true;
353       }
354     }
355     else
356       if (choosertable [tour_index] > 3)
357     {
358       if (!gsharepred)
```

```
337  bool tournamentchooser (unsigned long int address, char a, bool localpred, bool gsharepred)
338  {
339    int tour_index = address % 65536;
340
341    if (choosertable [tour_index] <= 3)
342    {
343      if (!localpred)
344      {
345        choosertable [tour_index]++;
346        return false;
347      }
348      else
349      {
350        if (choosertable[tour_index]>0)
351          choosertable [tour_index]--;
352        return true;
353      }
354    }
355    else
356      if (choosertable [tour_index] > 3)
357    {
358      if (!gsharepred)
359      {
360        choosertable [tour_index]--;
361        return false;
362      }
363      else
364      {
365        if (choosertable[tour_index] < 7)
366          choosertable [tour_index]++;
367        return true;
368      }
369    }
370  }
371
372
373
374
```