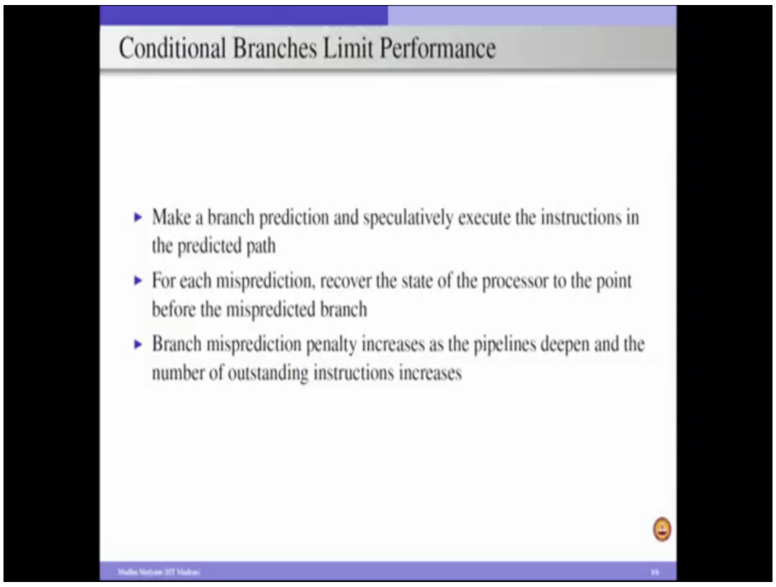
Computer Architechture
Prof. Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 06
Lecture - 21
Advanced Branch Prediction Techniques (Part – 1)

As a part of the fundamentals of pipelining we discussed the basic branch prediction mechanisms, both the static branch prediction mechanisms and 2 bit dynamic branch prediction mechanisms we discussed already. And in this module we are going to discuss correlated branch prediction mechanisms.

(Refer Slide Time: 00:33)



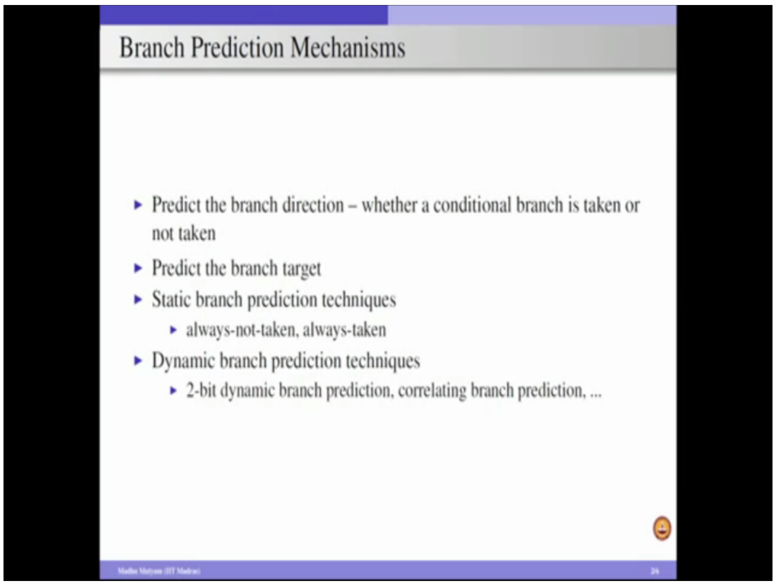
Conditional Branches Limit Performance

- Make a branch prediction and speculatively execute the instructions in the predicted path
- For each misprediction, recover the state of the processor to the point before the mispredicted branch
- Branch misprediction penalty increases as the pipelines deepen and the number of outstanding instructions increases

The conditional branches limit the overall performance of the system. So, we have to come up with efficient branch prediction mechanisms, as well as we have to come up with mechanisms to get the target address. And if our branch prediction mechanism is having high accuracy we can minimize the stalls associated with the control hazards and so that we can improve overall performance. On the other hand, if the prediction accuracy is not so high then we will incur penalty in terms of flushing the pipeline.

And also we have to bring the processor to a state that was there earlier when it is executing the branch instruction. So, as a result like the, it is very important to deal with the conditional branches especially from the performance point of view. And we have to come up with efficient branch prediction mechanisms.

(Refer Slide Time: 01:30)



Branch Prediction Mechanisms

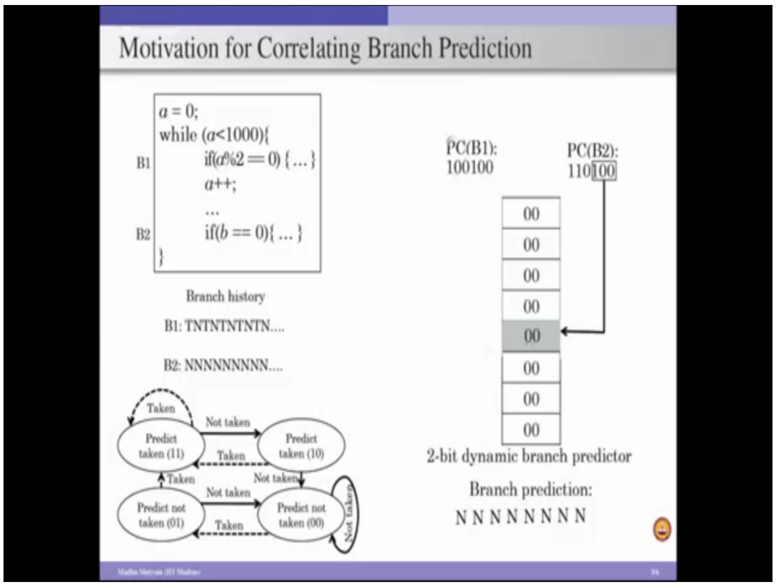
- Predict the branch direction – whether a conditional branch is taken or not taken
- Predict the branch target
- Static branch prediction techniques
  - always-not-taken, always-taken
- Dynamic branch prediction techniques
  - 2-bit dynamic branch prediction, correlating branch prediction, ...

So, the main rule of branch prediction mechanism is to predict the branch direction, whether the branch is going to take place or not. If the branch is not going to take place then we can start fetching this instruction following this branch instruction. And if the branch is going to take place then we have to fetch the instruction from a location specified by the target address. So, effectively, whenever if the branch is going to take place then we have to compute or we have to get the target address.

So, for that we consider the branch target buffer which actually stores all the target addresses for different branch instructions. So, that we are going to discuss in the next module. So, in order to come up with the efficient branch prediction mechanisms, we can take the, we can use the static branch prediction mechanisms as well as the dynamic branch prediction mechanism. In the case of static branch prediction mechanisms, our underlying hardware assumes always either the branch is not take place or always the branch is going to take place.

And we use this static mechanism and we take the help of compiler to reorganize our code, to minimize the stalls associated with this, the control instructions. And already we have discussed this as part of the fundamentals of pipelining. And in the case of dynamic branch prediction mechanisms, we already discussed the 2 bit dynamic branch predictor mechanism. And now we are going to discuss in this module the correlating branch prediction mechanism. So, we start with a motivation, why we require correlating branch prediction mechanism? And then we will explain the idea of correlating branch prediction mechanism and then illustrate the idea with an example.

(Refer Slide Time: 03:23)



Motivation for Correlating Branch Prediction

```
   a = 0;
   while (a<1000){
B1         if(a%2 == 0) {...}
           a++;
           ...
B2         if(b == 0){ ... }
   }
```

Branch history

B1: TNTNTNTNTN....

B2: NNNNNNNNN....

PC(B1): 100100

PC(B2): 110100

2-bit dynamic branch predictor

Branch prediction:

N N N N N N N N

So, consider a simple piece of code, here a=0, there is a while loop, it will iterate for thousand times and the body of the while loop consist of an if condition. If a%2 = 0 then we execute some piece of code and after that we increment ‘a’ and there are set of other instructions. And there is another if condition, if b=0 then there is another set of instructions. So, effectively in this code we have 2 branches B1 and B2. B1 is corresponding to the condition check associated with the a%2 = 0.

And the branch B2 is corresponding to checking whether b is equal to 0 or not. So, let us assume that, so in this while loop there is no instruction which is going to modify the value of b and b is initialized to a non-zero value. So, if that is the scenario then B2 is not going to take place at any point of time. So and also consider the outcomes of these 2 branches B1 and B2 like this, because anyway the B1 is corresponding to checking whether ‘a’ is even number or odd number. So, and also ‘a’ is incremented every iteration. So, as a result, so since ‘a’ value is equal to 0 initially. So, we get the initial branch is taken the next branch is not taken and so on.

So, B1 outcome will oscillate between taken and not taken, but whereas since b is initialized to non-zero value and also b is not updated in the body of the while loop. So, as a result the branch 2 is not taken always. So, it is always the outcome of the second branch is not taken.

We know that the state diagram of 2 bit branch predictor is like this. So, here it consists of 4 states, 00 is the predict not taken, 01 is predict not taken and 11 is predict taken and 10 is predict taken. So, whenever the state of the branch instruction is 00 or 01, we predict that the branch is not going to take place and we proceed further.

And we move from predict not taken to predict taken that is from 00 to 11, only when we have 2 miss predictions. So, one miss prediction will be by taken path and then the other one will be again by another taken path. So, that we will go to predict taken and the value will be changed to 11. And from here again, we will go to predict not taken only if we have 2 miss predictions that is not taken and not taken. When we come to the implementation of these branch predictors in an ideal scenario, what we can do is, we can consider one 2 bit branch predictor for every branch instruction.

So, if we use that, then we can there would not be any interference because of multiple branches, because we are maintaining a separate state diagram for each of the branches. But that is not efficient in terms of the hardware because once we have, let us say, application with significant number of branches for every branch we are going to maintain 4 state, state diagram to implement this dynamic branch prediction. Then it is going to incur significant hardware overhead. So, in order to reduce the overhead associated with maintaining this branch prediction mechanism, we map multiple branches to a single state diagram.

So, in order to do that we take a few lower order bits of our branch address and we index that into a table which maintains a collection of the state diagrams associated with this branch prediction mechanism. And using that value whatever is loaded in that mapped entry then we are going to make a prediction for that particular branch. So, in this particular example, we are considering 2 bit dynamic branch predictors of quantity 8. So, there are 8, 2 bit dynamic branch predictors in our system. And we use LSB 3 bits of our branch address to index into this 8 entry table. And once it points to one particular entry then that particular entry is, the is, going to give the state diagram associated with this particular branch.

So, let us assume that this 2 branches B1 and B2 are having the LSB 3 bits same. So, here the LSB 3 bits are 100 and this also 100. So, effectively these 2 branches are mapped to the 4th entry, this is 0th entry, 1st, 2nd, 3rd and 4th entry. So, these 2 branches are pointing to this 4th entry in this 8 entry table. And the values stored in the 4th entry are going to give the state of the branch predictor associated with these branches and accordingly we will make the prediction for our branches. So, let us start with the branch 1 because the B1 is the one which is happening first in the while loop. So, when B1 is executed first time.

So, we index into this table. So, that our entry is 00. Initially, we reset all the entries in the table. So, all 8 entries are reset to 00 and now when we index this branch 1 into this table with the last 3 bits, then we get 00 and 00 indicates, the predict not taken state of this 4 state branch predictor. And so, this is actually specifying that the branch is not going to take place. So, we predict that B1 is not going to take place and this is what our branch prediction, but in reality the branch is taken here. So, once there is miss prediction then what we have to do is. So, see here from this state diagram. So, this is our, the initial value 00.

And we predicted that it is not taken, but actually it is taken. So, once it is taken so we are moving from 00 state to 01. So, that is what we do here. So, we increment this value from 00 to 01. So, after some time again branch B2 is occurring. So, we index into the table using this address, again the LSB 3 bits are same as the LSB 3 bits of the branch one. So, here it is pointing to the same entry which is now specifying 01, but 01 state also indicates that the branch is not going to take place. So, effectively we predict that the B2 also not going to take place. So, now because this is in 01 state when the branch is not taken place and actual outcome of the branch 2 is also not taken.

So, this effectively both are same. Now, in this 01 state when actual outcome is not taken then we will move back to 00 state. So, that is what we have done here. Again we went back from 01 to 00. And after that again the next iteration of the while loop again B1 comes. So, when B1 comes again it is 00. So, 00 indicates the branch is predicted to be not taken, but in the reality also it is not taken. So, as a result in this 00 state when it is actual outcome is not taken then there would not be any change in the state and it will be a self-loop. So, 00 will not change.

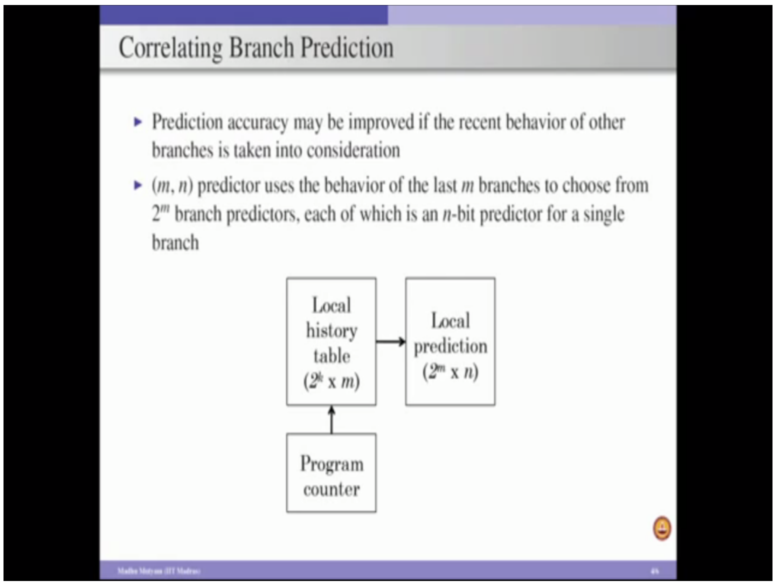
And now again B2 comes because of the second iteration and here it is pointing to 00 and again it predicts that the branch is not taken place, and the actual outcome of B2 also not taken. So, effectively there is no problem and there is no change in 00 value. And again, when B1 comes, so 00 we predict that the branch is not going to take place, but in reality the branch is taken here. So, because of this miss prediction, so now we are actually incrementing the value of this from 00 to 01. And again, the next branch comes it will be like 01. So, it is not, it also predicts that the branch is not taken place and reality also B2 is not taken.

So, again it is decrementing the value because of there is a state transition from 01 to 00 or not taken scenario and we just continue like this. So, effectively, here if you see because of this the 2 bit dynamic branch predictor and also because of the interference between B1 and B2. So, B1 is actually not predicting the outcomes properly. So that in other words we have to somehow learn the information from the other branch. So, that B1 can be improving its, the chances of predicting correctly.

So, if we do that, then we can improve the overall accuracy, but the 2 bit dynamic branch predictor if you are using as it is then we may not get significant the accuracy because of the interference with the other branches and so on. Because here B1, B2 are mapped to the same the dynamic branch predictor and as a result because of this interference so our overall accuracy is degraded. So, this actually shows that we need to come up with a better branch predictor than just considering a 2 bit dynamic branch predictor.

So, we will consider a 2 level branch predictors, are also called as correlating branch predictors, and here the prediction accuracy can be improved, if the recent behavior of the other branches is taken into consideration. So, in the previous example, if we see, here because these 2 branches are pointing to the same entry and they are modifying this value. As a result, the branch 1 outcomes are not correctly predicted. So, if we can somehow map these 2 branches to different locations, then there is a chance that we can improve the overall accuracy of the branches. So, for that what we can do is,

(Refer Slide Time: 14:17)



Correlating Branch Prediction

- Prediction accuracy may be improved if the recent behavior of other branches is taken into consideration
- (m, n) predictor uses the behavior of the last m branches to choose from 2^m branch predictors, each of which is an n-bit predictor for a single branch

Local history table (2^k x m)
Local prediction (2^m x n)
Program counter

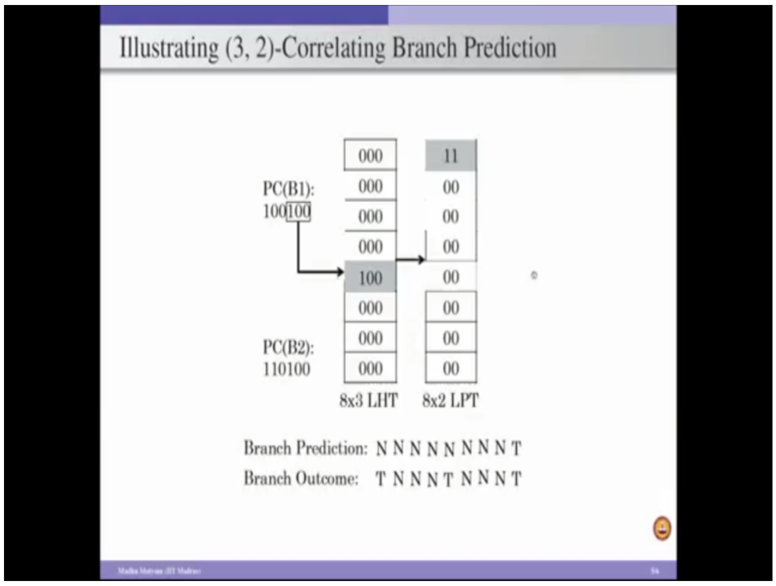
We can come up with a 2 level predictor, which is represented as (m, n) predictor and which uses the behavior of the last m branches to choose from 2 power m branch predictors. And each of these branch predictors is an n bit predictor for a single branch. Effectively, when we consider the outcome of last m branches, so it is m bit information we can get, because each branch outcome is either taken or not taken which is like a 0 or 1.

Effectively, when we consider the history of last m branches, we have m bit string. Using m bit string, we can index into a 2 power m entry table. So, we will point to one entry in the 2 power m entry table and that is going to give us an n bit value. And using that n bit value we predict whether the branch is going to take place or not take place. So, pictorially this 2 level branch predictor can be represented something like this. So, our program counter is going to give the branch address and we index some bits of this branch address into this local history table, which takes some k bits of our branch.

So that, we can index into this 2 power k entry local history table, using this k bits of the branch. And each entry of this local history table is having m bits. And using this m bits we are going to index into 2 power m entry local prediction table. And each entry in this table consists of n bits and this n bits are going to tell us whether the branch is going to take place or not for this particular branch. Our prediction mechanism involves 2 levels, one is we accumulate the last m outcomes of this particular branch or all the branches which are having the same k bits are indexed into this history table and that is going to give us the history of last m outcomes of all the branches which are having the same k bits.

And once you have that m bit information, we index into this local predictor table and this is going to give in turn n bit information and using that n bit information, we are going to make a prediction for our branches. So, as a result we can minimize the interference because of multiple branches mapping to the same entry whatever we have seen the previous example. So, now we will illustrate this correlated branch prediction mechanism with an example.

(Refer Slide Time: 16:56)


Illustrating (3, 2)-Correlating Branch Prediction

Branch Prediction: N N N N N N N N T

Branch Outcome: T N N N T N N N T

So, consider here, we have 2 tables one is a local history table and this local history table has 8 entries and each entry has 3 bits in it. And we have another table which is local prediction table which also has 8 entries, but each entry is now 2 bit information. Effectively, if we index into this local history table using some 3 bit value from our branch address and we will get another 3 bit information and using this 3 bit information we can index into this 8 entry local prediction table. And that mapped entry is going to give you a 2 bit information and using that 2 bit information, based on our 2 bit dynamic branch predictor we will make a prediction, whether the branch is going to take place or not.

And for this illustration, we are going to use the same code whatever we considered earlier we have 2 branches B1 and B2. And whose LSB 3 bits are same that is 100. Effectively, these 2 branches are pointing to the 4th entry in our LHT. So, we start with branch 1 because the branch 1 is the first branch occurred in our while loop. So, initially these 2 tables are reset to zero. So, the contents are cleared. So, everything is 0. Now, when we index, the, using the LSB 3 bits of branch 1 into this LHT, so it is pointing to this entry which is 000.

Now, this 000 indicates that we have to go to the 0th entry in our LPT. So, that is this one. So, once we go to this entry it is 00, according to our 2 bit dynamic branch predictor 00 indicates that, the branch is not going to take place. So, it is effectively the prediction is the branch is not going to take place. So, accordingly we make the prediction. So, our branch prediction is not taken, but actual outcome of B1 we know that it is taken. So, we have this taken as the actual outcome. So, that means we have made miss prediction here a wrong prediction.

And so, as a result we have to change the values accordingly. So, because of this t now we are actually changing the value from 00 to 01 in the LPT. And also, now we have to push this history on to this local history table because this is the first branch outcome we got. So, we have to push this value into this entry. So, previously it was 000 now we are pushing this 1. So, effectively we have 100 we can implement this as a shift register and we are shifting the value.

So, that the previous LSB value is moved out and the new value is entered here at the MSB position. So, now the next branch which is B2, when B2 comes this is 100 again. It is also pointing to the 4th entry, but now the 4th entry is having a value 100. So, effectively we have to go to the 4th entry in our LPT. And when we go to the 4th entry it is 00. So, we predict that branch B2 also will not going to take place. So, this is n, but the actual outcome also n. So, as a result there would not be any change in the value of this, but now because this is a second prediction we made. And so, we have to insert 0 on to this we have to push 0 on to this.

So, previously it was 100, now when we push 0 the effectively it becomes 010. Now, you can clearly see when the branch B1 comes it is pointing to this entry same entry, but it is actually going to a different entry in the LPT and when B2 comes it is going to the 4th entry. So, when B1 comes it went to 0th entry in LPT whereas, when B2 comes it went to the 4th entry in the LPT. Though both are pointing to the same location in the LHT, but because of the history whatever we accumulated using that we are actually pointing to different locations in our LPT. And next again when B1 comes now the history is 010. So, as a result we have to pointing to the, we have to point to the second entry in the LPT. So, this is 0th entry, this is the 1st entry and the 2nd entry, which is 00.

So, again we predict the branch is not going to take place. In reality also, B1 when it occurs second time the branch is not taken place. So, effectively there is no change. But now there is no change in the LPT value for this entry, but there is a change in the LHT because now we are inserting this 0 to this entry. So, effectively previously it was 010 now it will become 001.

So, in other words you can clearly see here. So, far we made we executed 3 branches whose outcomes are not taken, not taken and taken starting from the latest from here to the oldest. So, as a result we can clearly say this not taken was 0, this is not taken 0 and this is taken so it is 1. In other words out LHT always the MSB bit always indicates the latest outcome of the branch that was executed. And this one is the LSB bit is actually indicating the oldest among these three branches the oldest outcome of the branches among these 3.

So, now again B2 is executed. So, now it is pointing to 001 which is effectively the first location in the LPT which is 00. So, we predict that the branch is not taken place. And for all the B2 the outcomes of all B2 the branches are not taken. So, there is no problem with that. So, there is no change in this value, but now these also this is not taken. So, effectively last 3 branch outcomes are not taken. So, effectively now we initialize the value to 000 or when we push this 0 on to this. So, previously it was 001 and now it becomes 000 and now again when branch 1 comes it is now pointing to 000.

So, the value is 01. So, we know that according to the 2 bit dynamic branch predictor if the state is 00 or 01 we predict that the branch is not going to take place. So, accordingly we use n here and, but the reality the branch 1 is taken. So, there is a miss prediction. So, as a result because of the miss prediction we increment the value. So, previously it was 01 now we incremented to 10. And we push this value to this entry. So, it is now 100 and we repeat this for branch 2 for some more time and for B1 also.

So, as a result after this, a sequence of steps, our LHT is having 000. And now we are about to work with the branch 1, so now when branch 1 comes it is pointing to this entry which is 000 and 000 is pointing to the 0^{th} entry in LPT which has the value 10. This indicates that the branch is predicted to be taken. Because we know that from the 2 state the dynamic branch prediction, if the state is 0 or if the state is 10 or 11, we predict the branch to be taken. So, effectively now it is taken. So, once it is predicted to be taken, we fetch the predict instruction from the predicted path, but in reality also this branch is taken.

So, effectively we can clearly see here, after initial training period is over, even when B2 is mapped to the same location in the LHT as that of B1, because of this 2 level branch prediction, after some time we are able to predict correctly the outcome for branch 1. But this was not the case when we consider a single level branch prediction using this 2 bit dynamic branch prediction. So, if we see the 2 bit, outcome of these branches when we consider a 2 bit dynamic branch prediction, see even here in this case also, we predicted the branch is not taken place, but now when we consider this 2 level branch prediction now our B1 is actually predicted correctly using this 2 level branch prediction.

So, because of that, so it is always better to go for correlated branch prediction to improve the overall the accuracy of our branch prediction mechanism. Of course compared to the 2 bit dynamic branch prediction we are going to incur extra overhead in terms of maintaining these tables. So, previously when we consider the 2 bit dynamic branch prediction we were using just this 16 bits because 8 entries and each entry is having 2 bits, but now when we consider this particular example 3 2 correlating branch predictor.

So, we are actually having total 16 entries and 8 entries having 3 bits in each and the other 8 entries have 2 bits in each. So, effectively we are now going to have the 40 bits compared to the 16 bits of 2 bit dynamic branch prediction mechanism and we consider 8 entry table, but this overhead is not significant, but as long as the branch prediction mechanism is going to improve the overall performance with a marginal increase in the hardware overhead, we can go ahead with such type of branch predictor mechanisms in our system.

So, with that I am concluding this module and in the next module, I am going to discuss, the another, the advanced branch predictor, which is called as our tournament branch predictor. And also we discuss the branch target buffer implementation.

Thank you.