

High Performance Computer Architecture
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 16
Branch Prediction

Hello and welcome to today's lecture on branch prediction. In the last couple of lectures, we have been discussing about different types of hazards and in the last lecture we have focused on control hazard. And, we have seen various, how various compiler best approaches can reduce the penalty arising out of control hazards.

(Refer Slide Time: 01:24)



Recap: Importance of Stall Reduction

- Crucial in modern processors, which issue/execute multiple instructions every cycle
 - Need to have a steady stream of instructions to keep the hardware busy
 - Stalls due to control hazards dominate
- So far, we have looked at static schemes for reducing branch penalties
 - Same scheme applies to every branch instruction
 - Potential for increased benefits from dynamic schemes
 - Can choose most appropriate scheme separately for each instruction
- Branches to top of loop have different behavior (Taken) than "if (x == 0) return;" (Not Taken)
 - Can "learn" appropriate scheme based on observed behavior
 - Dynamic (hardware) branch prediction schemes
 - For both direction (T or NT) and target prediction
- Key element of all modern processors

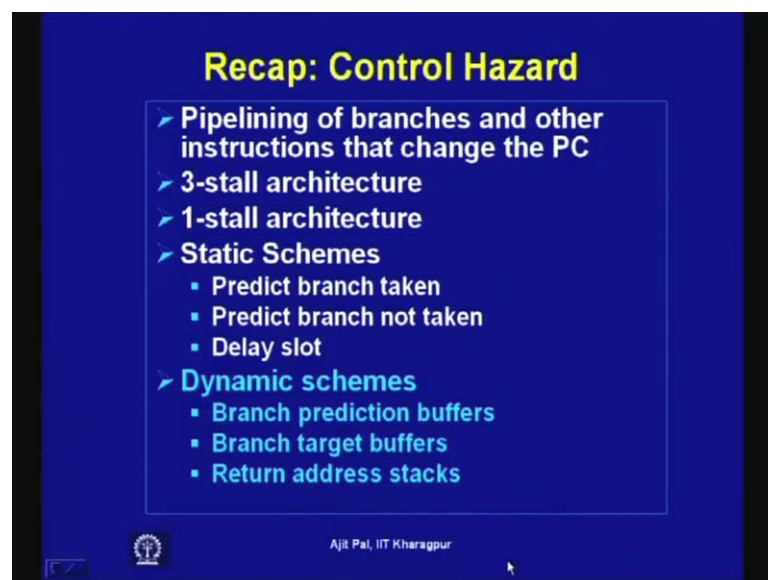
Ajit Pal, IIT Kharagpur

And, today we shall focus on dynamic approaches; so, here is a quick recap on importance of stall reduction. And, this is very crucial in modern processors where multiple instructions are issued. And, to keep the different functional units busy inside the processor, there is a need to have a steady stream of instructions and stalls due to control hazards dominate. It has been found that stalls due to control hazards is dominating; is dominating. The reason for that is as the as more and more number of instructions are issued in a single cycle; as it is done in a super scalar architecture. The gap between two branches reduces in terms of instruction cycle. If you for example, if there is a branch after 4 instructions and if 4 instructions are issued in a particular cycle; then in the next cycle itself the branch will appear. That means, the branch frequency increases.

And, so far we have looked at static schemes for reducing branch penalties and same schema applies to every branch instructions. That means, the techniques that we have discussed in the last lecture, we have discussed some schemes like not taken branch taken or not taken. And, that particular assumption or prediction is applied to every branch instruction. Now, in a real in a whenever you execute a program, the same branch instruction may be taken at a particular instant and another instant it may be un taken. So, dynamically the situation changes which cannot be captured by the compiler best approach; where we assume that always the branch is either taken or not taken.

So, can choose most appropriate schemes separately for each instructions. So, what we can do? We can go for some dynamic technique which can learn appropriate scheme based on observed behaviour. At run time, when the program is in execution; that time it can identify, I mean it can learn from the appropriate scheme based on the observed behaviour which we shall discuss today. And, dynamic branch prediction schemes, there are several branch prediction schemes that we shall discuss for both direction T and N T and target prediction. That means, whether branch will be taken or not taken and also the target at this prediction will be also carried out simultaneously or together. And, these are this feature has become the most important element of all modern processes.

(Refer Slide Time: 04:56)

A presentation slide with a dark blue background and yellow text. The title 'Recap: Control Hazard' is at the top. Below it is a list of topics: 'Pipelining of branches and other instructions that change the PC', '3-stall architecture', '1-stall architecture', 'Static Schemes' (with sub-points: 'Predict branch taken', 'Predict branch not taken', 'Delay slot'), and 'Dynamic schemes' (with sub-points: 'Branch prediction buffers', 'Branch target buffers', 'Return address stacks'). The bottom of the slide features a small logo on the left and the text 'Ajit Pal, IIT Kharagpur' on the right.

Recap: Control Hazard

- Pipelining of branches and other instructions that change the PC
- 3-stall architecture
- 1-stall architecture
- Static Schemes
 - Predict branch taken
 - Predict branch not taken
 - Delay slot
- Dynamic schemes
 - Branch prediction buffers
 - Branch target buffers
 - Return address stacks

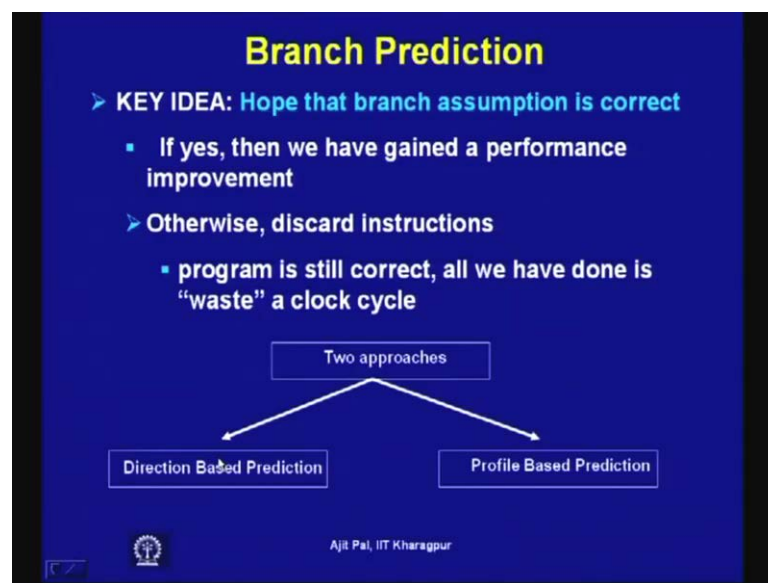
Ajit Pal, IIT Kharagpur

And, we have seen that this is what we discussed in the last lecture; 3 stall architecture and how it can be improved to have one stall architecture. Then, we have discussed

various static schemes, predict branch taken, predict branch not taken and how to use this delay slot in a very effective way; such that the penalty due to branch is reduced, these things we have discussed. So, today we shall discuss about dynamic schemes; where we shall be using branch prediction buffers and also we shall be using branch target buffers. That means, branch prediction buffers will hold information about whether a branch will be predicted taken or not taken.

On the other hand the branch target buffers will hold the information about the target address. Normally, target address is taken after computation in the instruction, but when you are predicting; then, branch target buffer can also be stored in cache memory and from where it can be read and used for generating the target address. And, the instruction fetch can take place from the target address. And, also the return address stacks can be used where there are many return instructions. Those return instruction addresses can be stored in a stack and with the help of all these dynamic schemes, the performance of the processor can be significantly improved over control hazards.

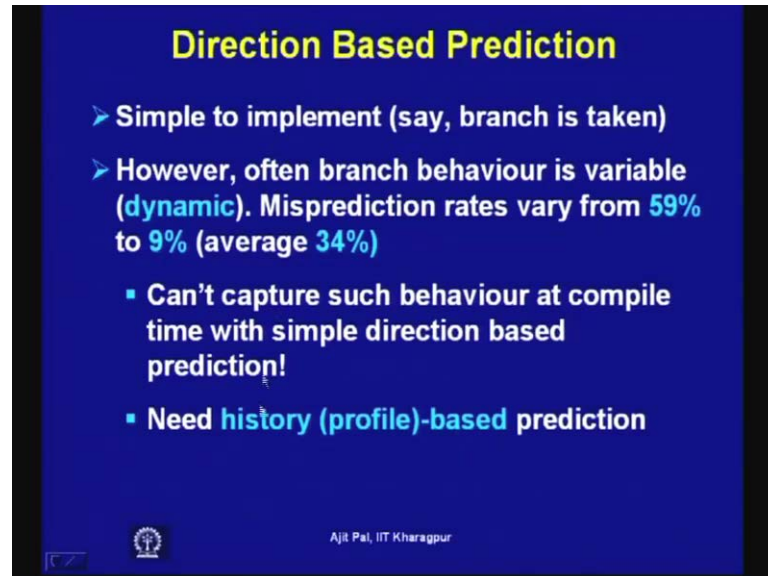
(Refer Slide Time: 06:35)



And, as I have already mentioned in the last lecture, basic idea of branch prediction is to it is assumption that branch assumption is correct. If yes, then we have gained a performance improvement. Otherwise, we discard instruction and in such a case few cycles are wasted. And, there are two basic approaches; first one is direction based

approach, which are used in compiler which are used by compilers. And, another is profile based approach which we shall discuss in today.

(Refer Slide Time: 07:16)



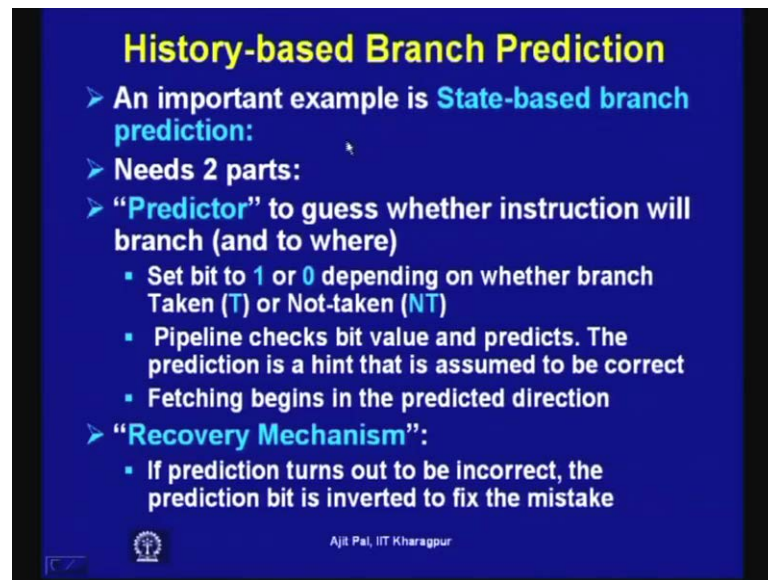
Direction Based Prediction

- Simple to implement (say, branch is taken)
- However, often branch behaviour is variable (**dynamic**). Misprediction rates vary from **59%** to **9%** (average **34%**)
 - Can't capture such behaviour at compile time with simple direction based prediction!
 - Need **history (profile)-based** prediction

Ajit Pal, IIT Kharagpur

So, we have already seen the direction based approach. The it is these are very simple to implement and say branch is taken or not taken. However, often branch behaviour is variable. So, it is dynamic as I mentioned and misprediction rates can vary from 59 percent to 9 percent and on the average it is 34 percent. So, this cannot capture these static based static approach used by compilers; based on compilers cannot capture such behaviour at compile time with simple direction based prediction. So, it is necessary to have history based approach; that means, as the program is executed the history based on the history or profile, the prediction is done. So, you have to maintain some information in hardware which will do this.

(Refer Slide Time: 08:12)



History-based Branch Prediction

- An important example is **State-based branch prediction**:
- Needs 2 parts:
- **“Predictor”** to guess whether instruction will branch (and to where)
 - Set bit to **1** or **0** depending on whether branch Taken (T) or Not-taken (NT)
 - Pipeline checks bit value and predicts. The prediction is a hint that is assumed to be correct
 - Fetching begins in the predicted direction
- **“Recovery Mechanism”**:
 - If prediction turns out to be incorrect, the prediction bit is inverted to fix the mistake

Ajit Pal, IIT Kharagpur

So, this history based branch prediction, an important example is state based branch prediction. So, here you will require 2 parts; number 1 is predictor, predictor will try to guess whether instruction will branch or not and also where it will branch. So, this is the job of the predictor and it will set a bit to 1 or 0 depending on whether branch is taken or not taken. So, when a branch is taken, it will store some information a particular flag bit which will be 1, if it is taken and if it is not taken, it will be 0. So, pipeline checks bit values, bit value and predicts. That means, whatever is stored in a based on a previous history, that will be used to predict whether the branch will be taken or not taken; the prediction is a hint that is assumed to be correct.

So, you must remember one thing; this prediction that we are doing is essentially a hint and prediction may be correct, may be incorrect. So, if it is correct, then we gain; if it is not correct, then the gain is not there. So, that you should keep in your mind. And, whenever you do the prediction, fetching begins in the predicted direction; whether it is taken or not taken. And, obviously, this type of state-based branch prediction should have a recovery mechanism. By recovery mechanism I mean whenever the prediction turns out to be wrong, incorrect or wrong; the prediction bit is inverted to fix the mistake. That means, see earlier a branch was taken; now it is untaken or not taken. So, the bit has to be inverted and that is done in the recovery mechanism.

(Refer Slide Time: 10:04)

History-based Branch Prediction

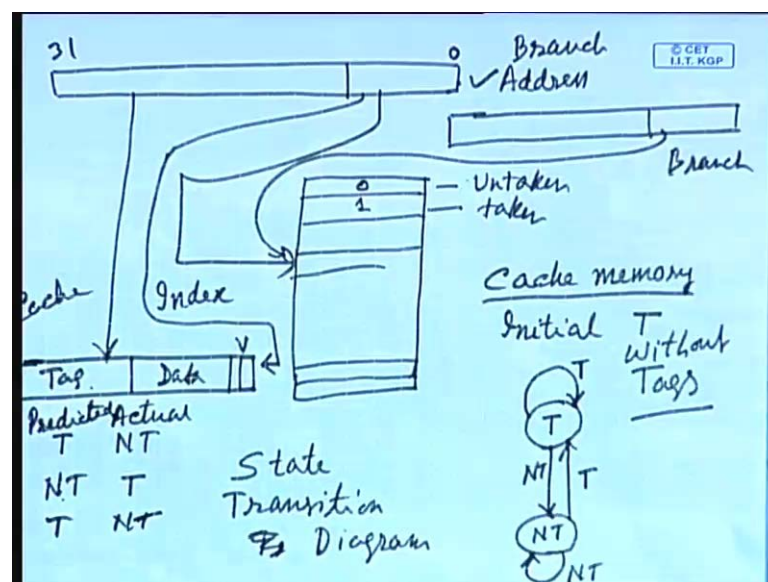
➤ One bit predictor:

- Use result from last time this instruction executed
- Small memory indexed by the **low-order bits** of the branch instruction
- Stores a **single bit** of information: T or NT
- Starts off as **T**, flips whenever a branch behaves opposite to prediction
- Benefits for larger pipelines, more complex branches

Ajit Pal, IIT Kharagpur

So, let us first focus on the simplest history based branch prediction; that is using one bit predictor. So, you will be using a one bit and use results from last time; this instruction executed. And, small memory indexed by the low order bits of the branch instruction.

(Refer Slide Time: 10:29)



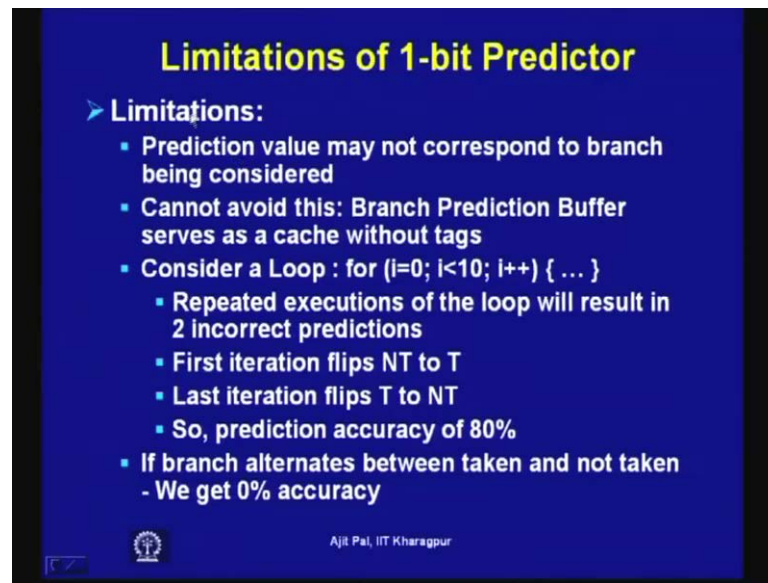
So, what is being done; so, suppose you have got a, this is the address branch. I mean branch that adjust for a corresponding to the branch instructor to be 32-bit. So, what can be done? May be 4 bit or 8 bit will be used to index a small memory where this will index a memory where you will store that single bit. That means, it will be 0 or 1; this is

the 0 correspondence to untaken are not taken and this correspondence to taken. So, you can see, you are storing 1 bit corresponding to each branch address that has been encountered and this is used for indexing purpose. So, this is a kind of cache memory.

Later on I shall discuss about the cache memory in details. There will see there also the lower order address bits are used to as a, as an index and to point two memory locations. So, here also the same thing is being done and however you are storing only a single bit of information T or NT and starts off as T. So, that means, initially initial value is usually T and it flips whenever a branch behaves opposite to prediction and benefits for large pipelines; particularly whenever you are using a large pipelines, then it is benefited. Let us see how it really works. So, you have got using a single bit, you have if you look at the state transition diagram of this predictor, it has got two states corresponding to taken that the value is 1 or not taken or untaken.

So, if the outcome is taken then it comes back to this; it remains here. But if the prediction turns out to be wrong, then if it is not taken; then it will go to this state and next time if it is not taken, then it will remain here. So, and if it will remain in the not taken state as long as the prediction turns out to be; I mean if the branch is not taken and here it will remain in this as long as the it is taken. So, whenever it is in taken state, if the branch not taken; then, it will go to this. On the other hand, whenever it is in the not taken state, if a branch is taken; then, it will go to this state. So, this is how the, this particular things work and this the State Transition diagram. And, as I mentioned for simple pipelines the benefit may not be much. However, whenever you have got very large pipelines; say 12 stages, 10 stages or more, then you will be benefited more.

(Refer Slide Time: 14:12)



Limitations of 1-bit Predictor

- **Limitations:**
 - Prediction value may not correspond to branch being considered
 - Cannot avoid this: Branch Prediction Buffer serves as a cache without tags
 - Consider a Loop : for (i=0; i<10; i++) { ... }
 - Repeated executions of the loop will result in 2 incorrect predictions
 - First iteration flips NT to T
 - Last iteration flips T to NT
 - So, prediction accuracy of 80%
 - If branch alternates between taken and not taken
 - We get 0% accuracy

Ajit Pal, IIT Kharagpur

Now, let us have a look at the limitation of one bit predictor. Now, prediction value may not correspond to the branch being considered. So, this is a very interesting observation. See, here it is been said that prediction value may not correspond to the branch being considered. Say, what we are doing? We are indexing by using the 8 bit address. So, this is a branch address corresponding to a branch instruction. Now, there may be another branch instruction having the same bits; that means, same lower order 8 bit. So, this will also point to the same location.

Now, what can happen? This particular we are trying to predict when the branch address is this; I mean this instruction address corresponding, corresponds to this. But the value correspond which is being stored here correspondence to this branch instruction. So, what is happening; we are trying to predict from the lower order address which may have come from some other branch. The main flaw here is, you see normally in a cache memory we have tag bits. Later on I shall discuss in detail. In that case you know what is being done? In case of conventional cache memory, you have this is the different fields of the cache memory. So, here it has got 2 parts; one is your tag and another is your data and of course, there are few flag bits which I am not showing valid and another thing.

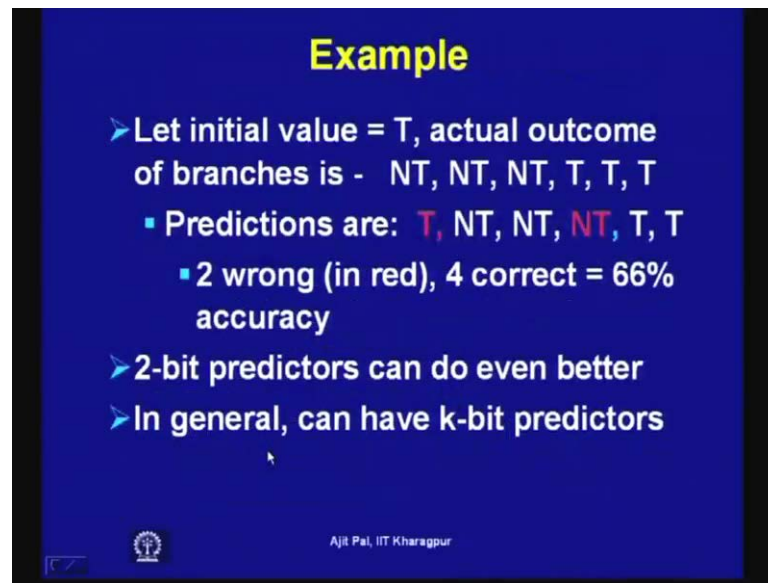
Now, these tag bits are missing in this particular case. So, although we are using cache memory, but tag without tags. That means, without tag bits what is the, what is done by the tag bits? The higher order address is stored in the tag bit. So, whenever I mean the

pointing is done with the help of this lower order address, then higher order address is compared with the tag value that is being stored. So, in such a situation the problem that is arising here, will not occur. That means, if you since we our cache is without tag; so, it will be always hit. And, in case of conventional cache memory, you know it is not always hit. That means, the only when the higher order address is same as the tag bit; so, these two has to be same only then there is a hit.

So, this higher order address is compared with this tag and only when they are same there is a hit. But in this particular simple situation there is no tag bit. So, it is always hit and this is, this problem is arising out of this; prediction value may not correspond to branch being considered. So, as I have already mentioned this cannot be avoided because branch prediction buffer, serves as a cache without tags. So, you are holding the information in branch prediction buffer, which is acting as a cache memory without tags. Now, let us consider some examples, consider a loop which is looping 10 times and repeated executions of the loop will result in 2 incorrect predictions. So, first iteration flips not taken to taken and last iteration flips from taken to not taken. So, at least I mean there will be 2 mispredictions. So, 2 mispredictions and out of 10.

So, the prediction accuracy is 80 percent in such a case. Now, if the branch alternates between taken and not taken; what can happen? A particular program branch is for a particular branch address, it is alternately changing taken not taken, taken not taken. So, whenever we use one bit, then what will happen; the prediction accuracy will be 0. Because you will predict taken, but it is not taken. So, you will modify to not taken, next time this is actual and this is predicted. So, predicted is taken actual is not taken. So, you have modified it to not taken and again it will be taken; that means, if it alternates it will continue and taken, it will be not taken. So, what will happen; if the misprediction rate is very high 100 percent. So, you get 0 percent accuracy.

(Refer Slide Time: 19:32)



Example

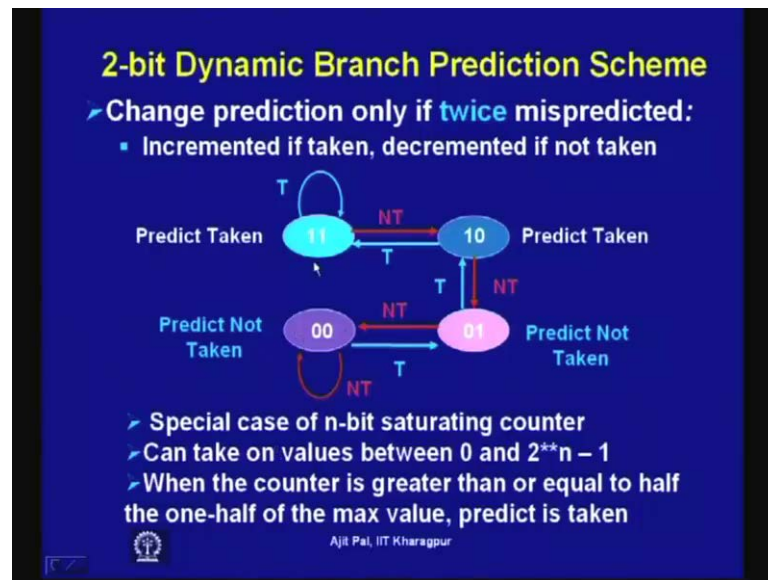
- Let initial value = T, actual outcome of branches is - NT, NT, NT, T, T, T
 - Predictions are: T, NT, NT, NT, T, T
 - 2 wrong (in red), 4 correct = 66% accuracy
- 2-bit predictors can do even better
- In general, can have k-bit predictors

Ajit Pal, IIT Kharagpur

So, this is the limitations of one bit predictor; and this is another example. So, actual outcome of branch is not taken, not taken, not taken, taken, taken, taken and so this is how I mean for a particular applications happens. So, initially it was taken. So, next because previous branch was not taken; so, it was modified to not taken. So, it matching here. So, if you guess one was not taken. So, it is not taken here; it is matching here, but here if you guess that was not taken, but now it is taken. So, there is wrong; this is incorrect, this is wrong prediction. So, previous was taken; so, here it is taken; so, here is again matching.

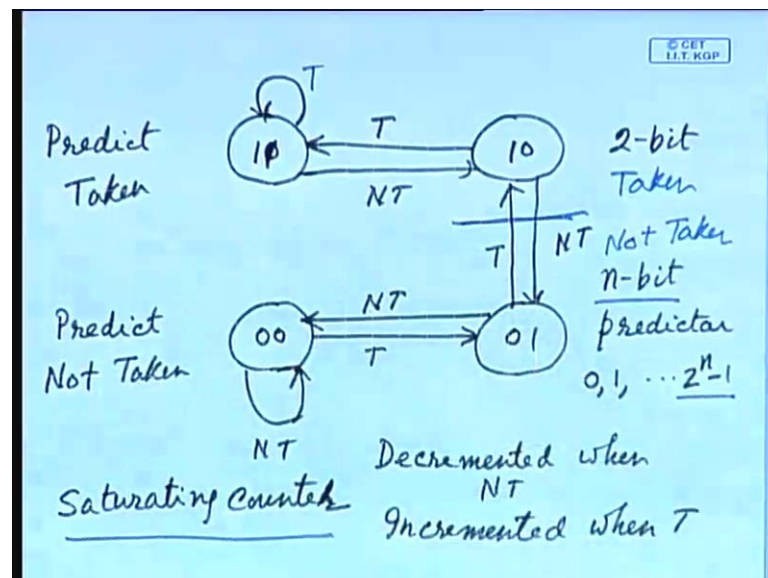
So, we find that in this case there are two wrong which is found by red color and 4 correct. So, there is 60 percent of accuracy. Now, how can we improve this prediction? So, in the state of a single beat, we can go for 2 bit predictor which can do even better. So, of course, this is a special case of k bit predictor, we can have k bit in a prediction predictor, but for the sake of simplicity we shall consider 2 bit predictor. And, we shall see how this 2 bit predictor performs and whether, later on we shall also see by increasing the number of bit; whether there is any benefit or not.

(Refer Slide Time: 21:04)



So, change prediction only twice mispredicted; so, incremented if taken, decremented if not taken. So, this is the basic idea of 2 bit dynamic branch prediction scheme.

(Refer Slide Time: 21:24)



So, it is done in this way. Say, suppose since you are using two bits, it will have 4 states. Let us assume the lower two states corresponds to predict not taken and the top one corresponds to predict taken. And, whenever let us start with this state 0, 0; we shall be having two bits. Now, whenever predict here the prediction is predict not taken and if it is not taken, it will keep on looping; not taken. And, if it taken, it is incremented by 1; so,

it goes to this state; so, here it is 0 1. Now, if it is again not taken it comes back to this state; that means, it is decremented when not taken and incremented when taken.

So, when taken it is incremented; now, if it is taken it will go to this, taken and not taken we have already taken care of. Now, it will go to taken; means it will increment by 1. So, it will become 1 0. In this particular case, if it not taken again it will come back here; it will be decremented. And, if it is taken it will go to this state, taken; if it is not taken I mean go to this state we will with value 1 1. And, if it not taken it will come back here. So, we find that, the for a; this is the case for 2 bit predictor. In general we can have n bit predictor. In a n bit predictor the number of states can be starting from 0 1 up to 2 to the power n minus 1. So, in the case of 2 bit, it is 0, 1, 2 and 3; so, these are the states.

Then, we see that it is incremented whenever it is taken and it is decremented whenever it is not taken. So, we can start with this or we can start with this; that can be our initial point. So, there if it is taken, it will remain here. Now, this we are realizing with the help of a saturating counter. What is the difference between a saturating ordinary and a saturating counter? The difference lies in this; as we know in case of ordinary counter, it start incrementing from 0, then 1, then 2. In this way it go up till 2 to the power n minus 1; that means, that will correspond to all 1. Then, again it will come back to 0. So, that is the conventional counter, but here it is not so. As, you can see whenever it is incrementing, after if you whenever it reaches the values 2 to the power n minus 1 it remains here; that means, if it is taken continuously it will remain here. It will not, it will saturate at that point.

Similarly, if you keep on decrementing whenever it is not taken, it will reach the point 0 and it will remain in that as long as it is not taken. So, this is called saturating counter and when the values is half, half of 2 to the power of n minus 1 or half or more; then it is taken and it is less than half, then it is not taken. So, there is a boundary as you can see, the upper half corresponds to predict taken and lower half corresponds to predict not taken. So, whenever it switches between these two, it changes between taken and not taken. So, this is how it works for general for a n bit counter and for a 2 bit predictor, this is how it works. So, this is a 2 bit predictor as I have mentioned, this is a special case of n bit saturating counter and it can take on values between 0 to 2 to the power n minus 1. And, the counter is greater than or equal to half of the 1 half of the next value predict is taken otherwise predict not taken; as I have already mentioned.

(Refer Slide Time: 26:30)

2-bit Predictor

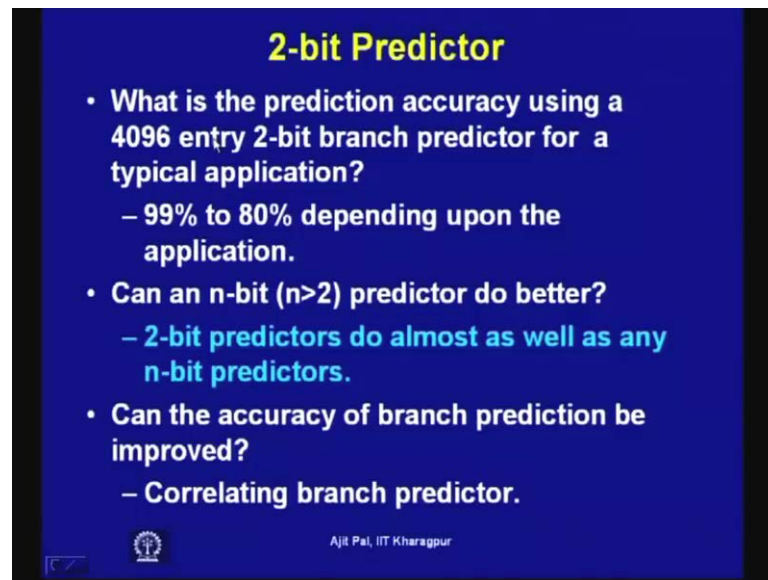
- The branch prediction buffer (BPB) is implemented as a special cache:
 - Accessed during the IF stage

Lower Address Bits	Prediction Bits
01FF	11
05CD	10
---	---

Aji Pal, IIT Kharagpur

So, this is your 2 bit predictor and here also the branch prediction buffer is implemented as a special case. As I have already mentioned this we are storing the bit values in a cache memory; the branch prediction buffer and this is accessed during instruction fetch stage. We can see this cache memory is accessed by at the instruction fetch stage and again lower address values are used for as a pointer. So, lower order address bits, these are the lower order address bits which are used as pointer and which is these are the predicted values 1 1, 1 0 depending on those stages that I have already mentioned; that is 1 0, 0 1. So, 1 1, 1 0, 0 1, 1 0 these values are being stored that predicted values are stored here corresponding to different; I mean instructions corresponding to branches.

(Refer Slide Time: 27:40)



2-bit Predictor

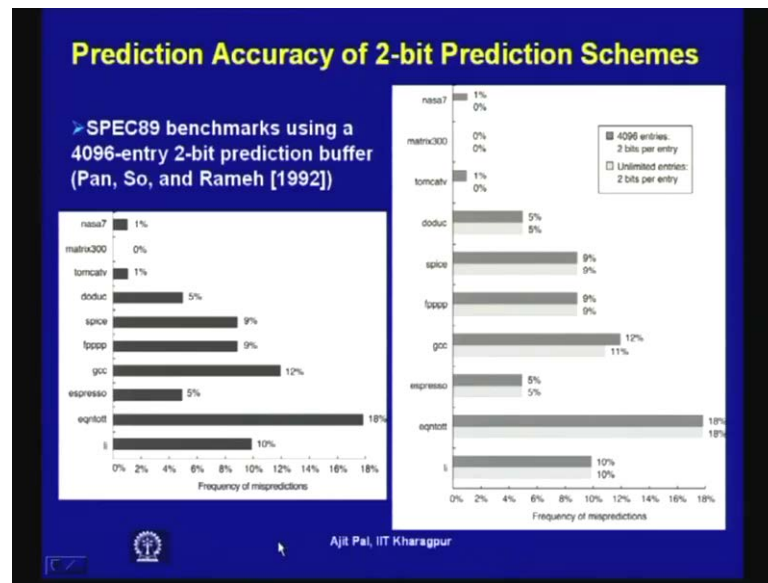
- What is the prediction accuracy using a 4096 entry 2-bit branch predictor for a typical application?
 - 99% to 80% depending upon the application.
- Can an n-bit ($n > 2$) predictor do better?
 - 2-bit predictors do almost as well as any n-bit predictors.
- Can the accuracy of branch prediction be improved?
 - Correlating branch predictor.

Ajit Pal, IIT Kharagpur

Now, let us look at the performance of this 2 bit predictor. What is the prediction accuracy using a 4096 entry 2 bit branch predictor for a typical application? And, it has been found that 99 percent to 80 percent depending upon the application. So, it will depend on the application program. So, from one application to another application it will vary, but prediction accuracy appears to be quite good; it is 99 to 80 percent. Now, as I mentioned, there is a scope for increasing the number of bits in a predictor. Now, a question naturally arises, what is the optimum number of bits. That should be used in a predictor; where 2 bit is sufficient or 3 bit better than that significantly better than that or 4 bit is significantly better than that.

So, what is the optimum number of bits that should be used for predicting and it has been found that 2 bit predictors do almost as well as n bit predictors. So, I shall show you this statistics for application programs. Another scope is can the accuracy of the branch prediction be improved? So, there are two ways by which you can improve the performance; one is by increasing the size of the branch target buffer, another is by using better predictions scheme. So, later on we shall see how the accuracy of the branch prediction can be improved. So, first let us see what is the improvement as we increase the number of bits.

(Refer Slide Time: 29:29)



So, it has been considered for SPEC 89 benchmarks using 4096 entry 2 bit predictor buffer. So, study was performed long back, back in 1992 by Pan, So, and Rameh. So, here you can see this is the frequency of Mispredictions for different programs. So, mispredictions as you can see rise from 18 percent to 0 percent. So, for this particular problem or application eqntott; I do not know exactly what is this application, but for this application the misprediction rate is very high. So, the you can see the frequency of misprediction is dependent on the application.

Now, let us compare this with that of unlimited number of bits; that is having a unlimited number of entry that you have got. So, here you have got only 4096 entry. Now, if you increase the number of entries, unlimited entries two bits entry, unlimited number of 2 bit entry. What is the frequency of this prediction? As you can find in most of the cases there is some improvement, but very small. For example, for unlimited entries nasa 7 example gives 0 percent misprediction; on the other hand with the 4096 entries, 2 bit entries gives 1 percent misprediction. And, if you consider the another example gcc, here there is a decrease of only 1 percent of misprediction. So, we find that there is marginal or no increase in performance as we increase the size of the buffer. So, from this part what conclusion we can make? The conclusion that we can make is that 4096 bit is quiet sufficient or there is no need to increase the size of the branch prediction in buffer to achieve better performance.

(Refer Slide Time: 31:41)

Correlating Branch Predictor

- 2-bit predictor uses only the recent behavior of a single branch to predict its future behavior
- It may be possible to improve the accuracy of branch prediction:
 - Branch predictors that use the behaviour of other branches to make the prediction are called **correlating predictors** or **two-level predictors**

Example:

```
if (a==2){
    b=2;
}
if (b==2){
    b=0;
}
if (a != b) {
    ...
}
```

Assuming a and b are in R1 and R2

```
DSUBUI R3, R1, #2
BNEZ R3, L1      ; branch b1
DADD R1, R0, R0
L1: DSUBUI R3, R2, #2
    BNEZ R3, L2      ; branch b2
    DADD R2, R0, R0
L2: DSUBU R3, R1, R2
    BEQZ R3, L3      ; branch b3
```

• Behavior of b3 is correlated with that of b1 and b2.
– if both b1 and b2 are NT, b3 will be T

Ajit Pal, IIT Kharagpur

Now, we shall consider about better schemes. So, you have seen by increasing the size of the buffer, we are not gaining much. So, what is the other alternative? Other alternative is to use a better scheme; better scheme than the 2 bit predictor. So, the better scheme that has been proposed, one of them is known as correlating branch predictor. So, this 2 bit predictor uses only the recent behaviour of a single branch to predict its future behaviour.

So, what we are doing here; what happens in the recent past, that has that information is being used to predict the future. Obviously, it is not performing well. So, what can be done, it may be possible to improve the accuracy of branch prediction by using branch predictors that use the behaviour of other branches to make predictions. These are known as correlating predictors or two level predictors. So, there may be more than one branches in a program; it is quite obvious. So, we are so far we have restricted to the, what happen to the present branch instruction. Now, what we are trying to do we are trying to look at other branches and let us see whether other branches effects the current branch. That means, we are looking at the behaviour of other branches to make the prediction and this is known as correlating predictors or two level predictor. So, let us look at this example on the left side. If a is equal to 2 then b is equal to 2. So, these are variables if a is equal to 2, b is equal to 2 and if b is equal to 2 then b is equal to 0 and if a is not is equal to b then something.

So, this is the example and whenever we go for this misinstruction assembly language instructions and assembly language program corresponding to this high level language program, we get this. and, here this a and b are considered this variables, a and b are being stored in the register R 1 and R 2. So, then here what we are doing DSUBU. So, we are subtracting 2 from R 1 then storing in R 3 then we are comparing b not equal to z. R 3 this is the branch instruction and this R 3 is compared whether branch, if it is not equal to 0 then it branches to L 1; if it is equal then it executes the next instruction. And, similarly that b is also stored in R 2. So, here the we are subtracting 2 from R 2 then storing in R 3 again if we are comparing if it is equal to 0 and that R 3 whether it is 0 or not.

If the branch not equal to 0, if it is not equal to 0 it branches to L 2 and for this is the target address L 2 otherwise it executes DADD R 2, R 0, R 0 and this is the L 2 branch. And, the third branch this b 3; that this is this one. If it is not equal to b equal to 0; so, that means, if a not equal to b then it branches to L 3. So, this is the assembly language program. Here the important observation from this example is that behaviour of b 3; so, whether that branch will be taken or not taken is dependent on is b 3 can be correlated with that of b 1 and b 2. That means, what happen in b 1 and b 2 is affecting b 3; that is see both b 1 and b 2 are not taken, then b 3 will be taken. So, this is the important observation of this correlating branch predictor; that means, if both b 1 and b 2 are not taken, then b 3 will be taken. So, this important observation can be used in the prediction in a correlating branch predictor.

So, previous two branches information about previous two branches can be used to predict the behaviour of b 3; that is the basic idea of the correlating branch predictor. And, this is a generalization of correlating branch predictor.

(Refer Slide Time: 36:25)

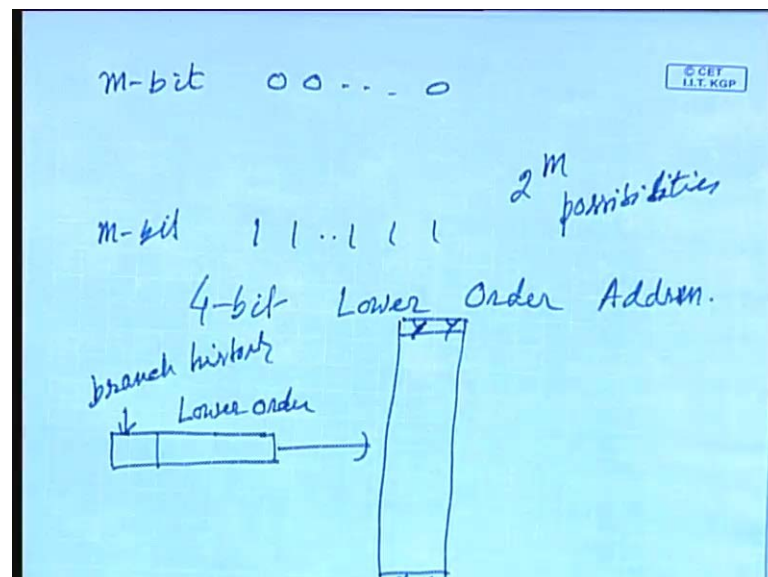
Correlating Branch Predictor

- An (m,n) predictor:
 - Makes use of the outcomes observed for the last m branches:
 - Uses m number of n -bit predictors
 - Behavior of a branch can be predicted by choosing from 2^m branch predictors
 - Yields improved prediction accuracy for small hardware cost
 - History of last m branches can be kept as a shift register
 - Each bit records whether corresponding branch was T/NT
 - Branch prediction buffer can then be indexed by concatenating the lower-order bits of address with the m -bit history

Ajit Pal, IIT Kharagpur

So, here an (m, n) predictor; that makes use of outcomes observed from the last m branches. So, there may be m branches and those m branches can be taken or not taken; that will lead to 2 to the power m alternatives. So, previous m branches, each of them may be taken or not taken. So, they this bit can be 0 or 1.

(Refer Slide Time: 37:05)

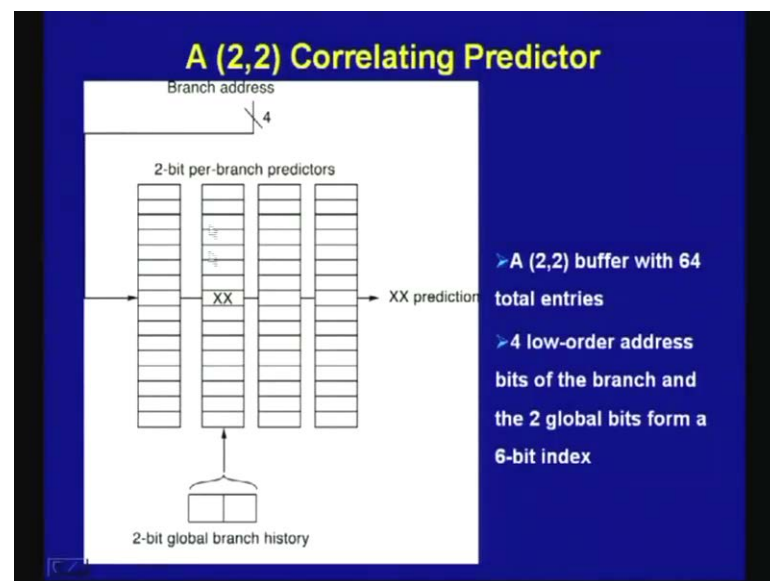


So, if there are; that means, if it has got m bits and each bit can be, if it is all are not taken then it can be 0 and corresponding m bits if all are taken, then it can be all 1. So, you can see we have got 2 to the power m possibilities. Whenever you consider m

outcomes of the previous m branches. And, behaviour of a branch can be predicted by choosing from 2 to the power m branch predictors, yields improved prediction accuracy for small hardware cost. So, we shall see what is the hardware cost involved in it and history of last m branches can be kept in a as a shift register.

So, this is how the information of m branches can be kept and each bit records whether corresponding branch was taken or not taken as I have already told. If it is taken it will be 1, if it is not taken then it sorry, if it is taken then it will be 1, if it is not taken it will be 0. And, branch prediction buffer can then be indexed by the concatenating by the lower order bits of address with the m bit history. So, this m bit history and the lower order bit address of the instruction can be used as a pointer; can be indexed by the concatenating index by this. So, let me show it how it can be done with the help of a (2, 2) correlating predictor. So, in this particular case we are using 4 bit 4 lower order bit of the address.

(Refer Slide Time: 39:08)



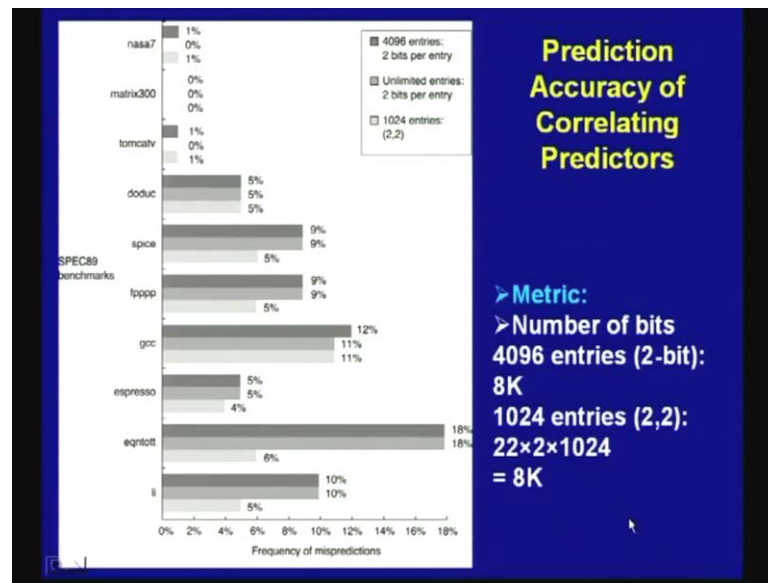
Then, we are considering m is equal to 2, m is equal to 2; means it will be having there are possibility of 4 alternatives previous branches can have 4 alternatives; 0 0, 0 1, 1 0, 1 1. And, the 2 bits that is being stored are taken, not taken which are stored here; that 2 bit that we have seen whenever we are using 2 bit, then it will require 4 sets 0 0, 0 1 and 0 2 and 1 1. And, this is being stored in this particular memory. So, for this purpose of visualization these are shown in a separately; but they can be stored in a linear manner in

the memory. So, here you can see the branch address is pointing to a particular location and this with the help of this index, it is you can have 1 of the 4. That means, in this case m is equal to 2; there are 4 possible alternatives and the 2 bit global branch history is pointing to one of these 4.

So, that means, that global branch history means that previous the two branches whether both of them were taken or both of them were not taken, one was taken another was not taken that history is being used. And, that is being that is pointing to one of the 4 possible columns. So, we have got 60 to 64 total number of entries and out of which you can see one out of 16 is pointed out by the branch address and again one out of this 4 is pointed by the 2 bit global branch history. And, this is how the prediction XX; that is your 2 bit prediction is available from here and which is being used for the purpose of branch prediction. So, 2 to 2 comma 2 buffer with 64 total entries is one here; 4 low order address bits of the branch and two global order bits from the index.

So, for the purpose of visualization, it is found here but it may be considered as a single entry. That means, you can have 6 bit, 4 bit; this is the lower order bit address and the higher order bit can be address can be coming from branch history. So, this will be acting as a pointer in such a case, your it can be a linear memory. So, there will be 64 entries starting from all 0 to all 1 and each having 2 bits. So, this is how the branch target buffer can be organized and the information about the correlating predictor can be stored in this branch target buffer.

(Refer Slide Time: 42:15)



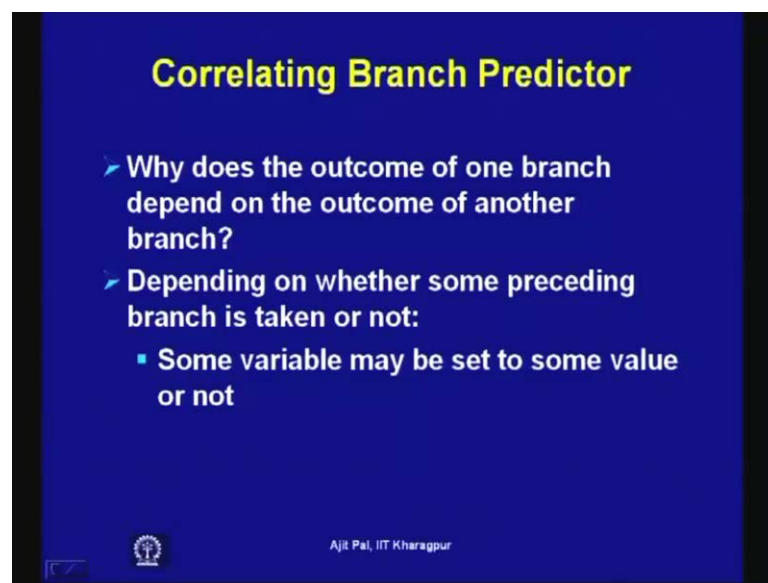
And, let us see what kind of prediction accuracy is obtained with the help of this particular type of entry; I mean whenever we use this correlating predictor. The this was done for the sake of comparison, the comparison should be done on equal footing. That means, you see we are comparing the, this correlating predictor with the 2 bit predictor. In case of correlating predictor, you have get the number of memory requirement; that is in the branch target buffer has to be same as that of 2 bit predictor. Only then the comparison will be on equal footing; that is what has been tried here. For example, the number of bits in 4096 entry is in a 2 bit predictor is 8k. So, with the same number of memory for 1024; here there is a mistake 2 into 2 into 1024 2 into 2 into 2; that means, for 1024 entries and 2 bit predictor and here it will be 2 bit for, you know that the number of that m and n that is lining here that is hurry.

So, it will be 1024 into 8. So, that will give you 8 k. So, with the same number of bits the comparison is being done and this is the 4096 entry and 2 bit entry, unlimited entry and 1024 entries; that is your correlating predictor (2, 2). So, here as you can see the there is significant improvement in performance for the this the last cord this corresponds to the correlating predictor. So, here you can see the decrease is, there is some decrease for some application. So, 5 percent from 9 percent this is coming down to 5 percent for the same size of branch target buffer, branch prediction buffer. Here also coming down from 9 to 5 for application f p p p from 9 percent to 5 percent. And, for g c c it is coming down

from 11 I mean 12 to 11 percent; it is same as the case where you have got unlimited buffers.

So, comparison between unlimited whenever we go for unlimited buffer with that is performing same with limited buffer. That means, with 8k entries that is 8 kilo bits. So, again here we are getting a espresso, we are getting a decrees of 5 to 4 to 4 5 to 4, but here there is a dramatic improvement of performance, we have seen for this example eqntott for limited buffer of 8k and 8 kilo bit. And, for unlimited buffer it was 18 percent. But as you can see here for co relating buffers, it is coming down to 6 percent. So, this is there is a significant improvement in performance of this correlating predictor. And, similarly for l I application it is coming down from 10 percent to 5 percent. So, what we can say from this, our conclusion is this co relating predictors perform better compared to big predictor, with limited branch target, branch prediction buffer for and also with unlimited advanced branch prediction buffer. So, that is the reason for by this particular approach has been found to be attractive.

(Refer Slide Time: 46:33)



Correlating Branch Predictor

- Why does the outcome of one branch depend on the outcome of another branch?
- Depending on whether some preceding branch is taken or not:
 - Some variable may be set to some value or not

Ajit Pal, IIT Kharagpur

Now, the question arises, we have already seen how this correlating branch predictor works. And, what we are trying to do; we are noting down the predictions of other branches or other the predictions of the other branches been used for. I mean the outcome of other branches are being used to predict the behaviour of the present branch.

In what way, I mean why it is possible and how it is possible; that you should understand and why does the outcome of one branch depend on the outcome of another branch.

So, we are considering other branches to predict the behaviour of the present branches. Why this is so? The reason for that is depending on whether some preceding branch is taken or not taken; some variable may be set to some value or not. So, what is happening you know some variables are propagating from one branch to another branch. And, values are modified by the previous branches and which are being used by the present branch. And, this is how one branch is effecting on another branch and that is the reason why this correlating branch prediction is working better compared to the best on local branch.

(Refer Slide Time: 48:04)

Correlating Branch Predictor Example

```
d=2;
While(TRUE){
  B1: if(d==0)
    d=1;
  B2: if(d==1)
    d=0;
    else d=2;
}
```

Assembly code:

```
BNEZ R1, L1 ; branch B1 (d!=0)
DADDUI R1, R0, #1
L1: DADDUI R3, R1, #-1
    BNEZ R3, L2 ; branch B2 (d!=1)
    ...
L2:
```

> If B1 is not taken, then B2 will be not taken

Ajit Pal, IIT Kharagpur

So, here there is another example of correlating branch prediction. Example, this is d is equal to 2; if this d is equal to 0 then d is equal to 1. If d is equal to 1, then d is equal to 0 else d is equal to 2 and so on. So, this is the simple example and the corresponding assemble language program is shown here. Branch not equal to 0 R 1, L 1; that means, branch is taking place if d is not equal to 0 and here branch is taking place b is not equal to 1. So, b 1 is not taken; then b 2 will not be taken. So, you see because of the variable that is passing with that d, that variable value is passing from one branch to another branch and from if b 1 is not taken, then b 2 will not be taken. Because d 1 the this that the variable d is being modified by previous is branch and that is being used by

subsequent branch decision. And, that is the reason why this correlating branch predictor is performing well. And, from this in this particular example; for example, if b 1 is not taken and b 2 will not be taken.

(Refer Slide Time: 49:28)

1-bit Predictor for the Example

d=?	B1 Prediction	B1 Actual	New B1 Prediction	B2 prediction	B2 Actual	New B2 Prediction
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT
2	NT	T	T	NT	T	T
0	T	NT	NT	T	NT	NT

- Initialized to NT
- Prediction Accuracy = 0%

Ajit Pal, IIT Kharagpur

So, let us consider situation where we shall be considering one bit predictor for the example; this is the example that we are considering. How this behaves; how this example behaves for one bit predictor? So, the value of d is changing alternately it is becoming 2 and 0; 2 and 0. And, the that b 1 prediction is not taken, taken, not taken, taken. I mean if you substitute here you will get this outcome; this is the prediction initially not taken. So, initially not taken then actually it was taken; so, that was initially change to taken, but unfortunately next time it is not taken. So, this is the new b 1 prediction and that is not matching and again it was not taken; so, it was changed to not taken. So, prediction was not taken, but it was actually taken. So, this is, this was the case for branch b 1. And, in case of branch b 2 again initially it was not taken, but actually it was taken. So, it was modified to taken, but unfortunately next time again it was not taken and again not taken. So, not taken was the prediction, but actually it was taken.

So, this prediction was changed to taken and unfortunately next time again it was not taken. So, in this particular case we find this that for one bit predictor, predictor is 0 percent. Because if you consider the first table, first I mean second column not taken,

taken, not taken and taken corresponding the d value is equal to 2 0 2 0. And, with the initializing, I mean initial value of N T; this is the prediction and actually just the opposite for all the 4 cases. Similarly, here for the branch b two, the prediction was not taken and taken and not taken and actually it was taken, not taken, taken, not taken. So, again it was the prediction accuracy was 0 percent; I mean always wrong. So, we find that one bit predictor for this example; for this particular for example, and for these two values of d alternate values of 2 and 0; prediction accuracy is 0.

(Refer Slide Time: 52:15)

(1,1) Correlating Predictor for the Example

➤ Two bits: Prediction if last branch
not taken / Prediction if last branch taken

➤ Initialized to NT/NT

d=?	B1 Prediction	B1 Actual	New B1 Prediction	B2 prediction	B2 Actual	New B2 Prediction
2	NT/NT	T	T/NT	NT/NT	T	NT/T
0	T/NT	NT	T/NT	NT/T	NT	NT/T
2	T/NT	T	T/NT	NT/T	T	NT/T
0	T/NT	NT	T/NT	NT/T	NT	NT/T

Prediction accuracy nearly 100%

Ajit Pal, IIT Kharagpur

Now, let us see how this (1, n) that is m is equal to 1 n is equal to 1. So, this is a correlating predictor and we shall apply this to for a simple correlating product predictor having 1 and m is equal to 1 and n is equal to 1. And, here it has got two bits prediction if last branch not taken and prediction if last branch is taken. So, we are using two bits and it is initialized to not taken, not taken. So, in this particular case as the values of d changes from 2 to 0 2 to 0; the b 1 prediction was not taken not taken and but actually it was taken. So, it was modified and the new branch prediction was taken slash not taken and it was next time. It was taken not taken, but it was not taken. So, this is the misprediction; this is the prediction.

And, in this way you can show all the corresponding to this branch, we find that only for the first row not taken, not taken and here it is taken. And, on the except for this first row here also I find that for b 2 prediction not taken, not taken it was taken. So, it changed to

taken by taken. So, this correspondence to the first one correspondence to the b 1 second one correspond to b 2. So, we find here that, except the first row for all cases prediction is correct. That means, here it is taken, not taken here also taken, not taken, taken, not taken, taken, slash not taken. Same, taken, slash not taken, taken, slash not taken. Similarly, the second row here, not taken by taken not taken by taken. So, here except the first row for all other cases we are finding that prediction accuracy is correct. That means, prediction is correct. So, we can say that prediction accuracy is nearly 100 percent for subsequent cases.

Oaky, let us stop here today with this correlating branch predictor. In the next lecture, we shall discuss about another important predictor; that is known as tournament predictor. And, you know when some games are proceeding; for example, a game of cricket, prediction is some normally prediction is running two ways. Say out of so many matches 100 matches how many match a particular team own? Another comparison is done on this particular ground. So, many matches took place and in those so many matches, how matches the particular team own? So, you can see here one parameter is global another parameter is local. So, this local and global parameters are used in tournament prediction. Somewhat similar concept is being used in the tournament predictor; that I shall be discussing in the next lecture.

Thank you.