

**Computer Architecture**  
**Prof. Mainak Chaudhuri**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kanpur**

**Lecture -16**  
**Basic Pipelining, Branch Prediction**

(Refer Slide Time: 00:13)

C program		MIPS-32		PC	ms
		PC	ins		
int main(void){		401060	addiu \$sp, \$sp, -32	401088	sli \$v0, \$v1, 1000
int i, x=0;		401064	move \$a1, 0	40108c	bnez \$v0, 401078
for(i=0; i<1000; i++){		401068	move \$v1, 0	401090	andi \$v0, \$v1, 0x7
if((i/8) != 0){		40106c	sw \$ra, 28(\$sp)		
x++;		401070	sw \$g0, 24(\$sp)		
}		401074	andi \$v0, \$v1, 0x7		
}		401078	bnez \$v0, 401084		
}		40107c	nop		
printf("%d\n", x);		401080	addiu \$a1, \$a1, 1		
return 0;		401084	addiu \$v1, \$v1, 1		

Today, I thought I am working through an example of how the branch predictor mass. So, this is the C code that, we will look at. And this is the MIPS 32 bit MIPS translation. After this is the call printf and all over there, which I exist not very important.

So, just to quickly go over the translation. Dollar a 1 holds X and dollar v 1 holds i. So, here this one is essentially doing i percent 8. So, even and meet 7 and if it is not 0, then you know that it is not divisible by 8 bits. Normal bits by 8, you have last 3 bits here. So, that is essentially this branch. It is just checking if it is not 0, then it jumps to 84, which skips over the increment of X. So, remember that in most cases, if we have if condition.

If the branch is not taken, then only you execute if condition ((Refer Time: 07:39)). If the code, under the condition. Otherwise, you skip it, so it skips over, if it is not 0. And then, here it increments i and compares against 1000. Set less than immediate. Sets v 0 to 1, if t 1 is less than 1000. So, if it is not 0, then it loops that to here. And in the devious slot, it computes i modulo 8. So, main loop is essentially 401078 to 401090. So, that is your main loop.

So, we have two branches that not importance here, this one and this one. So, we wanted to know, what the predictors will tell us? We run this go through our branch predictors. So, we have talked about several. So, we look at some of them today. And run this code through that. And see, how will it predicts? So, any question?

Bimodal

Not taken if count < 2  
Taken if count ≥ 2

Actual: 0 | | | | | | | 0 | | | | ...  
Predicted: 0

Diagram labels:  
BHT  
0  
1024  
← 2 Kbits →

One is 78, other one is 8 c. These are x program counter. So, you can easily see that, if you have a 1024 entry BHT, these two branches are going to fall onto different entries. Will you get it? Because, they could not take a modulo of 1024 hash function to map the BHT. So, essentially we are looking at the last 10 bits, after removing the last two.

Because, you can see that each one is actually a multiple of 4. So, last two bits are going to 0.

So, these two branches are going to fall on different slots of v h d. So, there is no ((Refer Time: 10:35)). That is the first, ((Refer Time: 10:37)) observation. So, now which means we can actually look at these two branches in isolation. You can analyze them separately. You know caring about the other one. So, let us first start with this one. So, this one corresponds to this particular branch. So, what we expect here?

We expect that your, so this branch is going to be taken 7 times out of 8 times and not taken 1's. So, first time for example, it will be actually not taken branch, when i is 0. So, the actual outcome, so let us write it down here. So, first time it is going to be not taken. So, I will probably write 0. So, 7 1's and then 1 0. That is, what the actual outcome is. So, let us see what the predictor tells us.

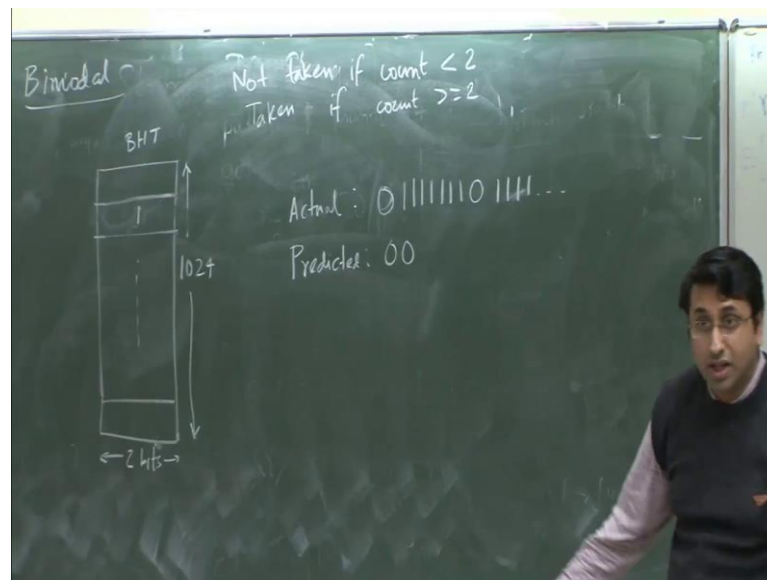
So, I assume that, this table is initialized to all 0's. So, first time I get the branch. So, let us allocate the entry. Let us, suppose this is the entry for this branch. So, initially comes is 0. So, first time the branch comes. I look up the predictor. Predictor says ((Refer Time: 12:19)) by the way, what is the prediction algorithm? So, not taken if count is less than 2 taken, if count greater than or equal to 2. So, this counter can count 0, 1, 2, 3. So, first time the branch counts. I lookup this one, it predicts not taken.

So, predicted. So, how do you update the counter? It has to be decremented. So, nothing to determine, it remains at 0. Next time, the branch comes and we call the predictor it says, what is the prediction?

Student: ((Refer Time: 13:13))

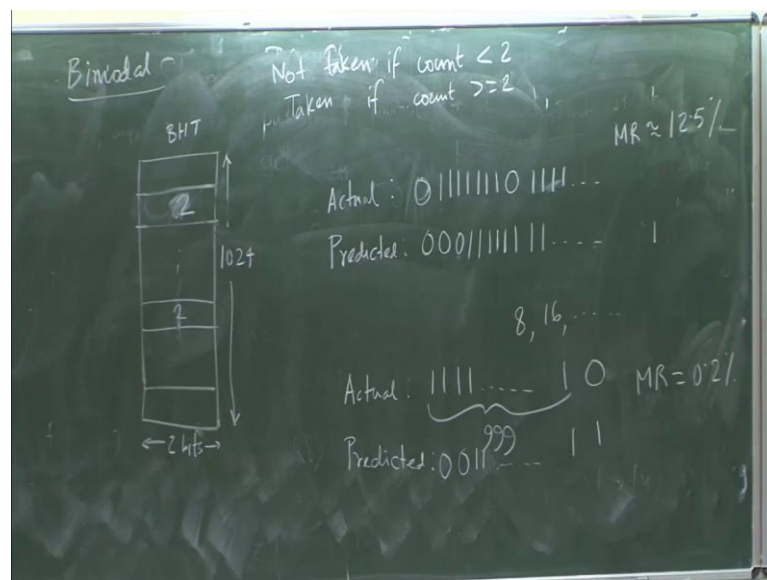
Not taken. But, this time the branch is actually taken.

(Refer Slide Time: 13:24)



So, counter increases to 1. So, next time the branch comes, what will I say? What is the prediction? Not taken, because the count is 1 which is less than 2.

(Refer Slide Time: 13:38)



But, the actual outcome is actually taken. So, it goes to 2. So I will. Next time, I will say take it. So, counter will go to 3. And then, everything will be fine. So, I will keep on say taken how many times? 1, 2, 3, 4, 5, 6, 7. So, count will saturated at 3, stay there at 3. And then, the branch comes of this one. The value of i is 8. I said, what is the prediction?

Student: 8.

8, which is wrong? So, now the count will, it decremented to 2. So, can somebody tell me what I am to do next? What will be the prediction after this?

Student: ((Refer Time: 14:23))

It is all would be taken. When the counter will essentially remain at c most of the time, occasionally come back to 2, go back to 3 here. So, it will all 1, predictor. So, what the prediction accuracy? All the time, this not will executed.

Student: Approximately 99.9

How many inspirations?

Student: ((Refer Time: 14:49))

So, I will be mispredicting, when the value of i 8, 16, etcetera. And at the beginning, I have two mispredictions here. So, essentially whatever the transfer to be, you can calculate that. So, you can rapidly expect that, it is going to be 12 and half percent, because you know 1 out of 8 times. You make a mispredictions plus 1 is the middle bit, because of this start of mispredictions and all. Any question on this? In this step very basic, because understand this please ask.

So, I will just write, misprediction rate approximately 12.5 percent. Although, if you are asking in the exam, I will expect that you give the right answer exactly. Do not put, this approximation. So, whatever the PC branch now. So let us suppose that, this branch gets separate entry here, initialized to 0. So, what will happen? First time, so what is the actual outcome for this branch? It is going to be what? Actual outcome, what is the string?

Student: ((Refer Time: 16:20))

So, there are how many times taken?

Student: ((Refer Time: 16:30))

999 or 1000?

Student: 999

What yours? What you, because 1 here. How many times the branch will execute?

Student: ((Refer Time: 16:55))

Well, i on the first of all, the code is not correct, so anyway. So, the compiler assume that, it is not 1's. That is why, you need to see that. Let us suppose that, this is 2. How many times the code will execute? If i is? Well, if it is i is 1, then what will happen? Yes, this is correct. So, this will go through 1's, check the compiler will fail. And it will actually not go back and this continues. So, it executes only ones. So, this is 999 and then there is 1.

Student: You know for loop, it only check, before we enter any loop.

No, this is fine yet.

Student: But, why do not we want to do that?

This code is fine.

Student: This is similar to do while loop. You know, for loop is check first and then it will terminate.

Is there any wrong in this one?

Student: Chatting should be before first.

Means, does any wrong in this one? That is I am asking, in this translation. There is a mistake in this translation?

Student: Yes.

Really? Fine.

Student: Sir, if that 1000 was 0, let us say.

No, then this code is wrong. So, compiler may see this actually. It is a constant ((Refer Time: 18:16)). So, compiler would have generated the other code, other piece of code that we are talking about when it turns 0. Asking for this particular value of the loop trip count, is this? And then, the branch will execute exactly 1000 times. Out of which 999

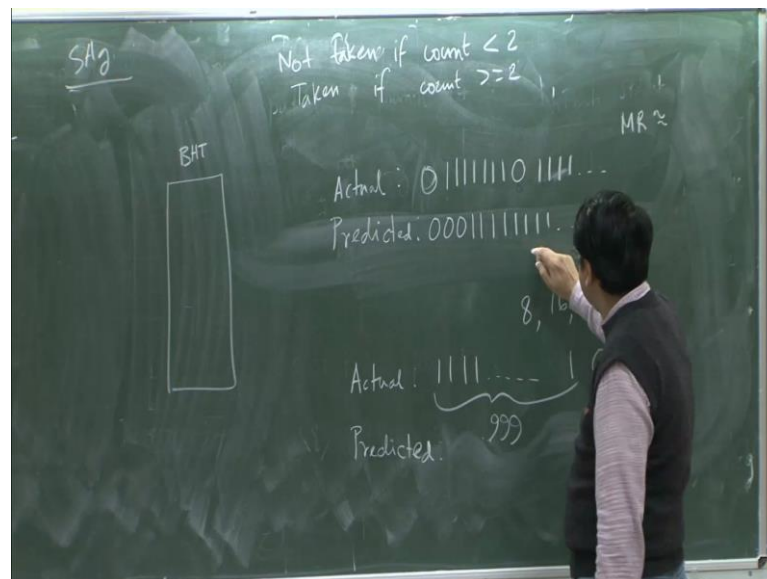
times would be taken. 1's would be, not taken last time, because last time this comparison will actually fail ((Refer Time: 18:40)).

So, now we are given this counter to this branch ((Refer Time: 18:53)). First time, this branch comes. Predicted, first time you say not taken. Counter increases to 1. Next time, you also say not taken. Counter goes to 2. And then, all we have correct. We will all say 1. Last time, we will also say 1, which is the mistake. So, this is highly accurate. This will give you, 998 correct predictions. So, here the mispredictions, write this 99.8 percent, wrong 0.2%.

Student: Ok.

Is this clear? The accuracy of the bimodal predictor. So, it is not very good on branches like this ((Refer Time: 20:04)). But, it is highly accurate for loop translation. So, next one that I want to look at, this is slightly more sophisticated.

(Refer Slide Time: 20:46)



So, we are going to look at a SAG predictor. So, here we have two tables. First one is the pattern history table and the second one is the branch history table. One more thing, I have to report before that. So, in this case with the bimodal ones. Suppose, I make the counters narrower, I get 2 bit counter bits. Suppose, I make it 1 bit counter, so what happens in that case? Does it affect the accuracy of this branch? It does not right.

It starts at 0. It actually makes slightly less number of mispredictions. It reach 1. So, in this case of course the not taken, if count is not 1. If count is 0, take if 1. So, in this case the initially, we make two less mispredictions. This one, will actually make just one misprediction and go to 1. And then of course, at the end it will make mispredictions. What about this branch? What you think about that?

So, here what was the predicted pattern? It was 0 0 0. Is there any mistake?

Student: ((Refer Time: 22:15))

No, I will predict 1.

Student: ((Refer Time: 22:26))

So, I am not. I am talking about 2 bit counter. So, for 1 bit, how does it change?

Student: ((Refer Time: 22:33))

So, when this not taken, outcome comes. The next one, I will actually make a misprediction. So, essentially what will now happen is, every time I switch I will make two mispredictions instead of 1, with 1 bit counter. So, mispredictions rate would actually double draffy. Is that clear to everybody? Because, the count will go back to 0, I need to observe once more before I go back to 1.

So, here the 2 bit counter was actually protecting against that, going to the other side. So, in some cases, actually in most cases it having a wider counter helps. It allows you to... So, this called the historicism that is, what you are seeing actually. So, you will not change your outcome. Based on, you know occasional changes in your prediction pattern. So, there are, this furious 0 is coming in. But, you will not change that prediction.

Of course, you are making a mistake. But, you are not making too many mistakes. Here, you make two mistakes ((Refer Time: 23:38)), every time you switch. What happens, if I make it 3 bits? I make wide. How will this change?

Student: ((Refer Time: 23:51))



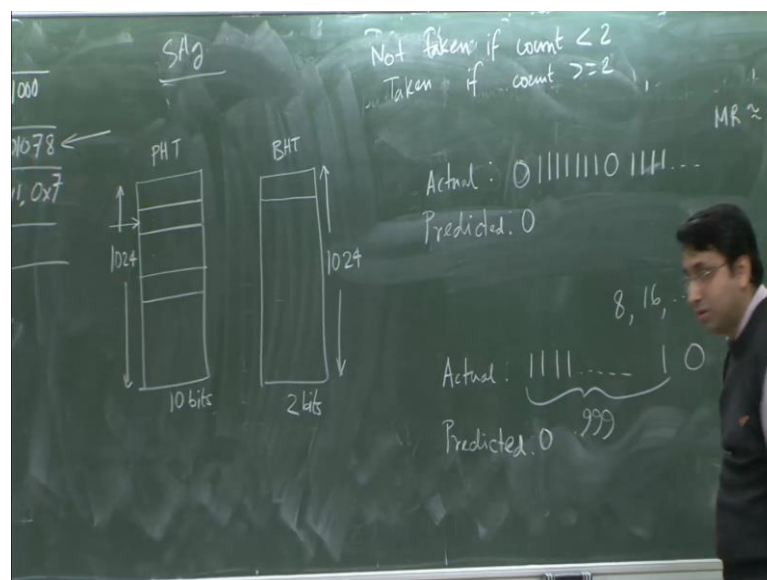
So, initially I will make more mispredictions here. Because, until I reach 4 in that case. Because, 4 5 6 7 are essentially my taken counts, above the midpoint. So, I require 4 observations to get 4 actually. Until, then I will make mispredictions. What about this one? What happens here, any change?

Student: ((Refer Time: 24:16))

Initial mispredictions, but after that I will be exactly like this. So, essentially if you make your counter wider, one tells is that you take longer to large, longer to reach the same state actually. Is not it? So, what may happens is that, here of course we have a branch which has, you know very straight patterns that, you can easily learn. But, branches are not like this. We will have very erratic patterns in between.

So, whenever you change, it will take much longer to change actually. Because, now suppose you have taken a string and then you have a not taken string. With the 3 bit wide count, you have to claim down from 7 to actually below the midpoint. It takes longer to get that. That is your learning time on mispredictions. So, as you make your counters bigger, there is a loss also. It will have much bigger logging time. So, now we can talk about the other one.

(Refer Slide Time: 25:19)



So, we will assume that, this is still 1024 entries and 2 bits. This one will assume also 1024 entries. How wide the histories?

Student: 10 bits.

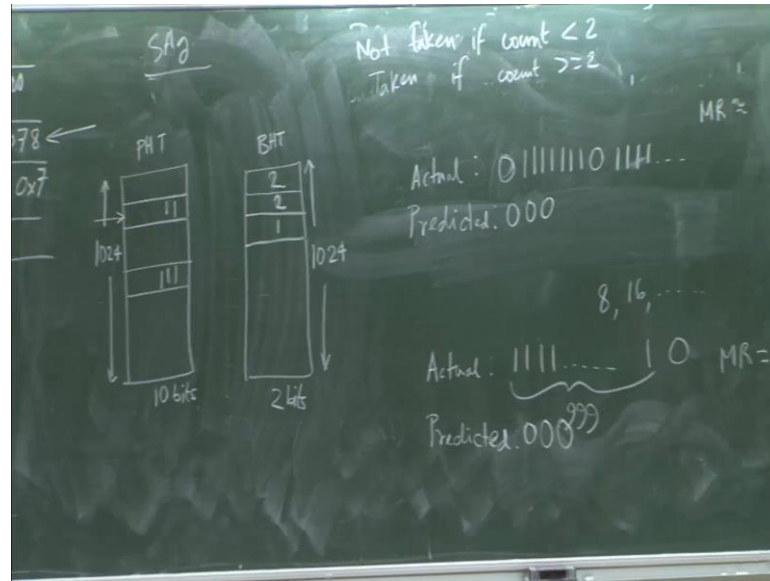
10 bits, log of this. So, we have a 10 bit history. And I will index the BHT with the history. So, now the analysis going to be a little more involved, because you cannot take the branches in isolation. Because, there may be history aligns in BHT. Because these two branches are actually have similar history. But, they may actually try to update the same counter in the singular rotate. So, we have to actually take both the branches together.

So, we have to essentially simulate the code now. So, just see. Just we have to be little bit careful nothing very involves with this. So, let us see what happens? So, first I will execute this branch. This is going to be not taken. So, let us first again. So, the same observation again that, ((Refer Time: 26:35)) these two branches are going to get two different 8 bit in the pattern discrete table, that is for sure in 1024 in the table with the module hash function. There we are going to get two different entries.

So, let us allocate two different entries to them. So, this is for this branch ((Refer Time: 26:51)) and this is for this branch. So, this branch executes indexes in to this entry. Then, this initializes them to 0, everything initialized to 0. This all 0 indexes into the first counter, because this is essentially my index to the second level table. This is 0, so I say not taken. And I shift in 1 bit of history, which is of course not taken. So, nothing can be change this, still remain 0.

And here it still remains 0, because this actually not taken branch. I will go to 20. Then, shown up this branch, this goes here. This is 0, this index to this table. This entry is 0, I say not taken ((Refer Time: 27:40)).

(Refer Slide Time: 27:46)



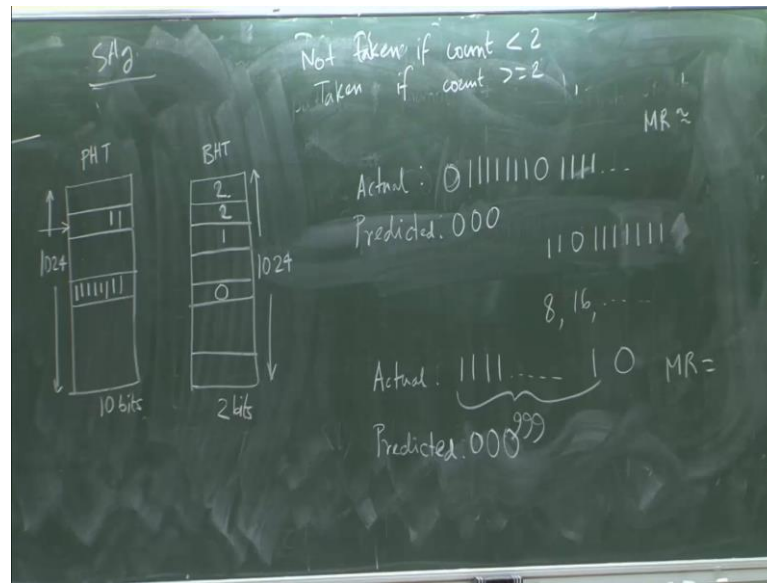
And then, I shift it to a bit of history which is actually taken ((Refer Time: 27:46)). So, it becomes 1 and it gets incremented to 1. Is this clear to everybody, what is happening? Next iteration, this branch shows the, this is second. So, I go here, this is 0. I look up this entry, pretty not taken. And the branch is taken. So, I shift in 1 bit of history here and these increases to 2, because you know that we increment the counter with the old BHT. You get it? So, BHT is updated later.

Now, this branch comes, the value is 1. So, index is the second entry in the table. This is 0. So, I say not taken. So, I shift in another bit of history, this goes to 1. Then, this branch comes. This is 1. This is 1. I say not taken. This becomes 1 1. This increases to 2. Then, this branch comes, I look up a new entry the third entry. I say not taken. Increases to 1, another bit of history shift. So, if anybody following for certain? So, however to take and here the action is start making predictions.

Student: First table formula a 1. First table

So, even before this, so what will happen is that once. This branch is easier to analyze. So, let us first look at this entry.

(Refer Slide Time: 29:46)



So, when this is filled up completely with all 1's, because I stored it only v actually, for this branch the history long even, if you move the side units all 1's. So, this particular entry the last one, this index is in last entry. So, this one will gradually made up after this point. Once we have filled up this is to, because after this point, this branch will only access this entry. There will be nothing else that access to BHT.

So, this will gradually buildup and get to 3. In fact as soon as process 2, I will start making correct prediction. So, how many mispredictions will I make, before I get to this point? Can mispredictions will fill after history and too more to cross the section. So, 12 mispredictions, before I start making correct prediction and of course, there will be another misprediction at the end of this particular loop, this particular branch. Is that clear to everybody?

What about this branch? This is a slightly more difficult actually. So, here what will happen is, if you move a slightly viewer, how many different histories can I get slightly moved of 10 bits here? There are quite of few. So, essentially what I need is, so this branch will gradually accumulate history. And it will actually index into 10 different entries in BHT and if you take some time for all of them to give the correct predictions.

Essentially, what will happen is that, all except this particular history. Let me see, if I come up with the 1. So, this particular history the 10 bit history, this will learn to say not taken. The next one, this is the 7. 7 1's, next one is not taken. So, whichever counter it

will index into, some counters some whatever the value that is. It will essentially remain at 0. Learn to remain at 0, because every time when comes that history here, it essentially gets into that entry and pass.

All other histories that this branch can have will be, will learn to say taken, which is learn to say taken. And it will never index into this counter, because it will never get this history. So, there is no aliasing the same state in BHT, between these two branches. This branch will modify ten different entries and this branch will modify some other entries. Let me activate this entry in BHT. Is this clear to everybody?

So, can anybody guess what will be the accuracy for this one, for these branches? This code improves wrong, already 1's. It will improve. So, can you guess how long it would take, before I start making all correct prediction.

Student: 25 mispredictions are there.

25, how do you get 25? How do you get this number?

Student: For 7, it is like. For 7 1's, it will go to different entries of BHT, which only use in statistics. Then, there would be 10.

Why for 7? You get more. It is 10 bit history.

Student: No. Those why just 7, you can more than that initial 7 entry. We will to BHT only ones.

So, initial 7 why not 8?

Student: Because, the fraction will do not get repeated and that is that.

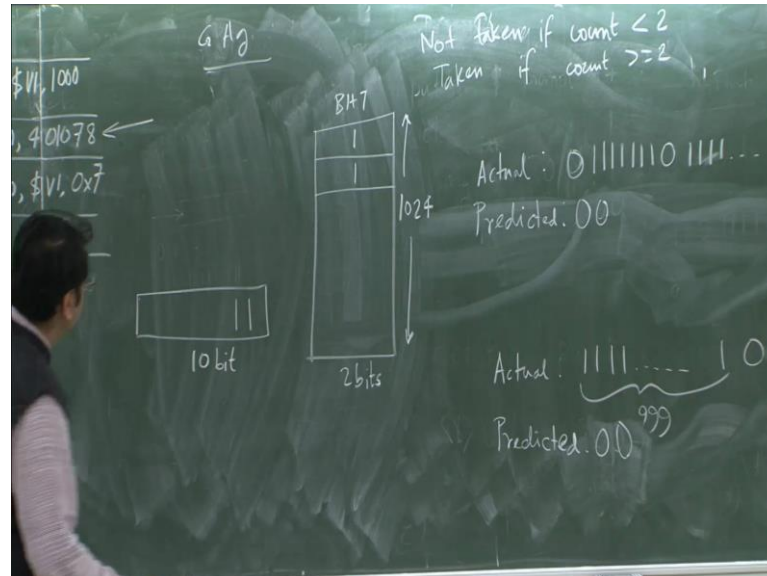
I am saying that, even we get more than that, even first 8 9 9, 9 outcomes 8. So, we will. So, then

Student: Then, we have 10 mispredictions. But, only one of them will be correct predictions. It is like two times each of them. So, it is 29 20 turns to its 80 plus 7 Plus 9, 9 actually more than that.

So, the point is that you will make some constant number of mispredictions at the beginning and then you lockup. So, you make all correct predictions after that. So, it is

going to be much higher than much smaller than this actually. So, one more thing I will look at that is the G A g predictor.

(Refer Slide Time: 34:56)



So, by the way did you notice that actually, I make more mispredictions for the, for this branch seeing the S A g predictor compared to the bimodal. So, that is one drawback of these predictors, because they take more time to learn. So, basically when you are accumulating the history, until you get the full history. This says which is the learning time? So, that is the outside of having a bigger history. If you take a more time to reach the steady state of the predictor.

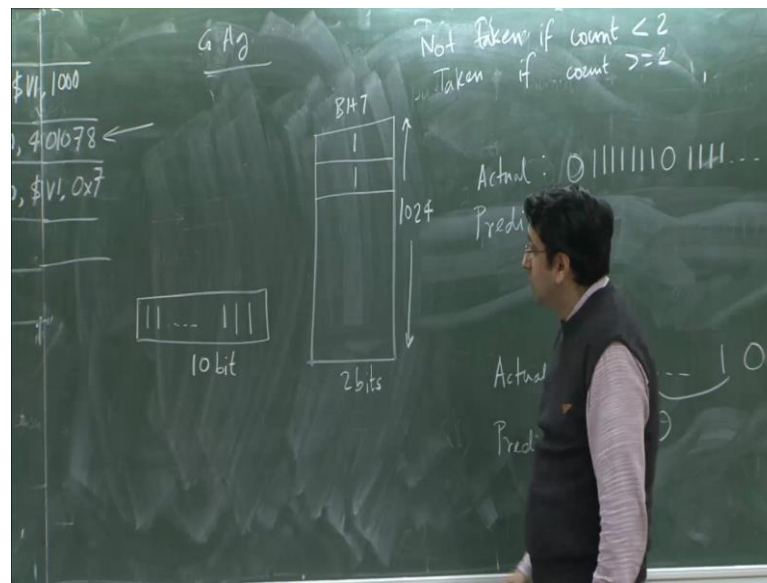
So, Gag. So, here I have a 10 bit history, which is global shared by all branches. And that indexes takes us into a BHT, which has 1024 entries 2 bits. So, here also we have to carry both the branches together. We cannot really isolate in between, because we have a global shared history here. So, let us see what happens? So, initially everything is 0. So, this branch first shows the first instant of this branch. It takes this history, which is 0 uses that to index into one entry on the table, usually it will index the first entry which is 0.

So, I say not taken. Then, this branch, this is not taken. So, the history is not updated, nothing to be done. Then, this branch comes, it is actually taken. History is 0, I look up this entry I say not taken. Update the history, make this 1, because this branch is actually taken. Then, I come to this branch. This is also taken. So, I look up the history goes to

entry 1. I say not taken, I shifting 1 bit. So, although I am showing that, I am shifting it on this.

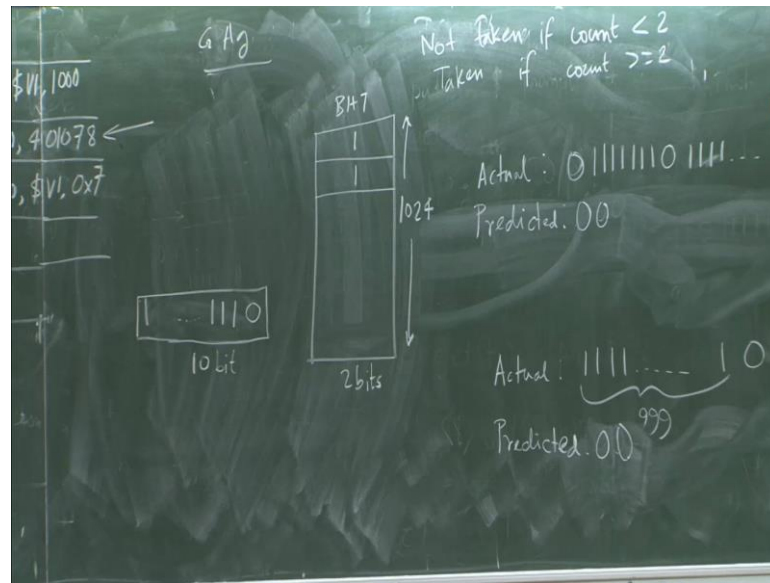
These actually you know. This is shifting, when you shifting on this side and the new bit comes in the s l b, actually. So, then this guy shows i. Index of third entry, which is 0. I say not taken. So, that continues what will happen soon? So, here what will happen is you will fill up this 10 bit much faster now.

(Refer Slide Time: 37:54)



So, after five executions, it will be all 1's here after five iterations of the loop. And you will say not taken but, this is still not the steady state history actually. So, why you get there is, you index the last counter. You say not taken. So, still now I have said all not taken. I have not yet said anything taken. Now, what will happen? So, it will again shifting. So, this thing will continue for couple iterations after, which this branch will be not taken.

(Refer Slide Time: 38:33)



So, then you will get a new history here, 0 here and all 1's. So, now tell me how long it will take, before I make correct predictions? Or do you think, it is going to be very bad in this case compared to SAg predictor? What kind of history, some are going to get it here?

Student: There will be 11 patterns, we will get that.

There will be how many patterns?

Student: In 11 pattern.

So, how do you get that?

Student: From 0 to n mispredictions.

So, how do you systematically derive that? So, you have to essentially consider t, these two branches in sequence ((Refer Time: 39:14)). So, if you do that, let me see. So, this branch first. So, essentially what you have to do is that, you take this one, this one, this one, this one, this one. That will give you the full history and then you slightly send this 10 bit over that.



(Refer Slide Time: 39:35)

	PC	ins	
sp, \$sp, -32	401088	sli \$v0, \$v1, 1000	
a1, 0	40108c	bne \$v0, 401078	←
\$v1, 0	401090	andi \$v0, \$v1, 0x7	
28(\$sp)	:		
24(\$sp)			
0x7			
0x4			
1			

1 ... 1110

  
 10 bit

011' 0

8H7  

1

  
 1024  
 2 bits

Not  
 Take

So, 01111 etcetera then again, at some point you get a 0. This length, this distance between these two zeros will increase, now essentially. We will get doubled instead of 7, which become much larger. So, the point is that the learning will happen eventually as you can guess. So, how many histories you said, 11?

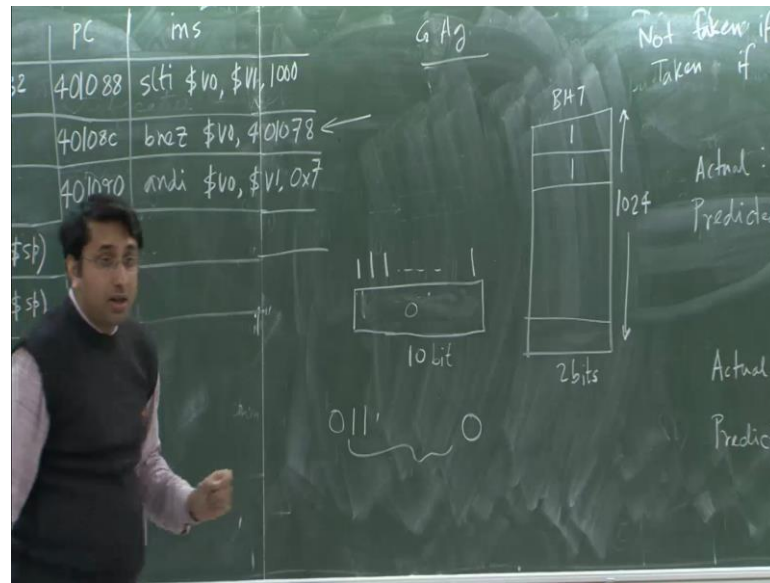
Student: ((Refer Time: 39:58)) because in 10th position remain 10 position and all 1's.

0's can be in all the 10 positions, exactly.

Student: all the 1's.

And all 1's, because there can be how many 14 you write? So, all 1's.

(Refer Slide Time: 40:23)



So, what you saying is that, you can get 11 different histories, because you can have 0 0 on any of this positions and all 1's, 10 need 10 point histories. And one history will be all 1's. And they all run correctly, because all these histories 0's in any of this positions will learn to say.

Student: ((Refer Time: 40:40)) There are only 6 histories.

Why?

Student: Because, 0's will conjugate even places.

0 can only be even places. I see that is all 1. But, the point is that there is... Saying that, figure that out. What I am trying to emphasize is there is something about this all 1's pattern. The problem with this pattern is that, what would be the rest outcome? Suppose, I have all 1's, I have seen last ten ones can you tell me, what is going to come here? It can be 0 or 1, both are possible actually.

So, this last entry will have real term, real correct predictions. Whenever you will get a 0, it is going to be give you a wrong prediction. Once we will be predicted correctly, because this patterns will actually saturated three come down two sometimes and then go back to here. Other counters will have a unique next outcome other history patterns. So, they will all give you correct outcome. But, this history pattern will actually have mispredictions.


So, you get all this out, you find that actually this predictor is going to be worse than the SAg predictor. So, it says that these two branches are not really globally correlated. So, when you put that together, it actually leads to distract in the BHT, which is happening in the last entry, which you merging. You are merging the history of two branches. After these, the next outcome may be 0, may be 1, because of this problem.

So, how do you solve this problem? Well, so you use the SAg predictor. So, here really there is nothing to come from GAg predictor. So, what I suggest is that, you go back home, please work out this example, if you do not follow. Any questions? So, last time you see.

(Refer Slide Time: 43:59)

## 21264 Tournament

- Essentially a hybrid of SAg and GAg
  - Combines a local and a global predictor
  - SAg has 1024 entries (p) with 10 bits each (m) in PHT and the second level has 1024-entry BHT each being 3 bits (q)
  - GAg has 12-bit GHR (m) and 4096-entry BHT each being 2 bits (q)
  - Meta predictor or selector is BIM with 4096 entries (indexed by GHR)
- Need to carry two PHT entries along the pipeline
  - One from SAg and one from Gag
- Update protocol for the components?



MAINAK CS422

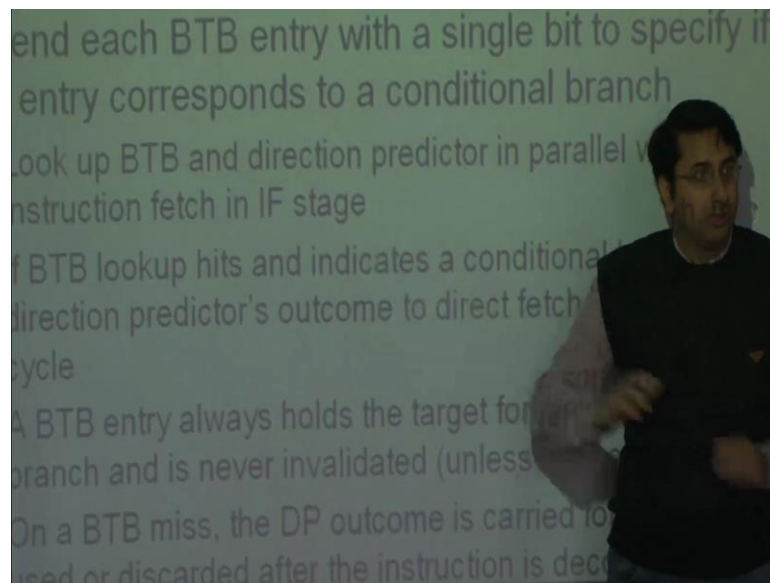
70

So, last time we discussed this hybrid predictor, where we can combine SAg and GAg predictors. So, we talked about why we want to do that also? Because, GAg is essentially here giving you global correlation. And SAg is telling you about local correlation. You know and of course, a large program we have probably mixed of both of these. So, what you correct on both. So, we have selector which selects which one is going to be good for, which branch? And that is all we combined. Ehen all talked about, how to update the components here?

Student: ((Refer Time: 47:24))

I see I thought I mentioned that. So, this predictor was implemented in the microprocessor from alpha to alpha. alpha processor available in market, used to be designable compact. And it was called a two domain predictor, because of name. And this is just here. One example today in your every ten processor, it finds similar things. The parameters may be different. And the architecture would be very different, very similar with a multiple components, then the selector which select between these components and all.

(Refer Slide Time: 49:24)



So, just to go back to the basic scheme of things, that we discussed about prediction. So, we said that you look up the branch target buffer in the fetch stage, in the instruction program buffer. And you look up the direction predictor, after the instruction is decoded. So, that was the timeline for BTB and the direction predictor look up. So, essentially why it until, the instruction is decoded to look up, the direction predictor and the rational for that moves up, it only make sense.

Look up the direction predictor for conditional branches rather it sends more point in viewer. So, that is why we wanted to wait until, we know that this instruction skip condition branch or not. The problem is that, this may be low performance, if you are bleatingly has no accuracy. Because, what you are, what I am doing is that, in the very next cycle, I am relying on the BTB outcome until to get my fetch essentially.

That is what I am waiting. Now, you may say that well it just a cycle, because in the next cycle itself I am going to get the direction predictors outcome, because instruction will decode. But, even losing one cycle may be a lot of trouble. For example, if you have an alternating conditional branch, the BTB always wrong. Because, the BTB will tells what happened last time? And what is going to happen next time exactly, opposite of that?

So, the question is how do you really fix this problem? How can I avoid losing this cycle, because it seems very straight that I cannot do anything, because instruction itself has the target for the branch. And I have a branch predictor sitting in the next pipeline stage. Why can I just access in this cycle? So, there are in fact this is what is used in most commercial processors today. You have a puce BTB direction predictor architecture.

So, what you do is, you extend each BTB entry with a single bit. You specify, if the entry corresponds to conditional branch or not. And remember that, this is not going to change. It just one time, an instruction is either a conditional branch or not. That is it. You cannot change in future. So first time, you encounter this instruction. You can store this particular information, once the branch is decoded in the BTB.

So, next time onward whenever you get the instruction become the BTB tells, you quickly run, this is the conditional branch. So, what you do is you look up the BTB and the direction predictor in parallel with instruction fetch in the prostate itself. If BTB lookup, hits and indicates the conditional branch, use the direction predictors outcome direct fetch the next cycle. A BTB entry, always hold the target of a conditional branch and it is never invalidated unless replaced.

So, first time encounter the branch. You put the target of the branch in the BTB entry, because here for conditional branch is a BTB entry has the slightly different meaning. It is not telling you, what happened last time? It tells you, that if you really take the branch where should it go? That is what it telling actually. And on a BTB miss of course, you have nothing to do. But, what it do is, the direction predictor outcome is carried forward through the pipeline.

It is a single bit. So, it is taken or not taken. And, used for discarded after the instruction decoded, because at that time you know, whether it is a conditional branch or not. And then, you can make use of this outcome at that point. Is this scheme clear to everybody? What is happening itself? I will actually nullified many of those single cycle bubbles,

where BTB can be very, very poor, because I can use the direction predictors outcome. What is the down cycles? What am I using? And is this, I guess hope create the time is nothing.

Student: In case of multiple wrong predictions, we have to maintain this type of BTB?

No, Why? Each conditional branch will get hopefully different entry in the BTB. Well, they will actually BTB is the tads structure. So, each entry will tell me, whether it is a conditional branch or not. Is this clear to everybody? That is the first question really what is happening? So, I look up the BTB and the direction predictor in parallel, the BTB has a special bit, which tells me if this instruction is conditional branch or not.

So, if I hit in the BTB and that indication comes from me, I can use the direction predictors outcome of it. If I hit the BTB and the BTB tells that, it is not a directional it is not a conditional branch. Then, I will use whatever the BTB tells me to do, that is the last time what happened actually? If I miss the BTB, then what I do is, I really do not know what its instruction is? It could be a conditional branch.

So, in that case I carry the direction predictor outcome between. It is a single bit and was the branch is decoded, one instruction is decoded. I either use it, if it is a conditional branch or put it away, if it is a normal instruction.

Student: sit, the BTB and the direction predictor in parallel?

Yes, they are lookup in parallel. So, whichever is slower we will. So, there are three things which are happening in parallel. BTB look up, direction predictor lookup, instruction fetch, whichever is slower will determine side walls. So, may have or may not happen to side walls? But, there is a major down side walls, what is that?

Student: What is the target for branch? We are going for the target of BTB.

The BTB entry if it is no. Then, I show it will be wrong, in case my direction predictor is wrong. That is clear to everybody? So, how many lookups do I make in the direction predictor, in this scheme? How many number of lookups?

Student: Number of instructions.

Number of instructions, for every instruction I make lookup for direction predictor. Why is it bad? I said it is very bad. Why, what I am using in that?

Student: ((Refer Time: 54:43))

Why not?

Student: When you need to store it, I mean you wanted to be store in register or somewhere, even if you can...

No no. It is a separate structure.

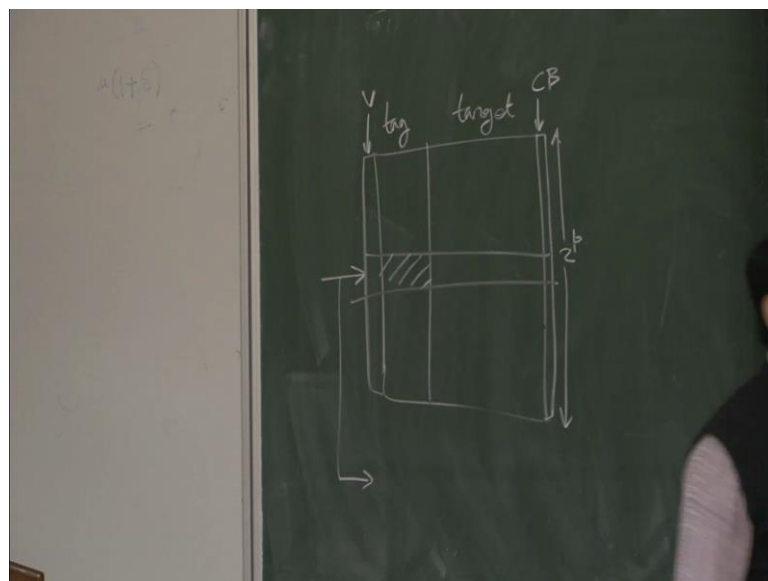
Student: You have some any operator. That is should be powerful actually

That is not we mentioned. There are three things happening in parallel. BTB lookup, direction predictor lookup and instruction fetch, whichever is slower we will determine my cycle. That is very small size. It accordingly, I can accommodate. These are simple tables very small actually, what do I do?

Student: If we use 1 bit for BTB that is one extra number of instructions level that something will happen.

No, I think you are not following, what I am talking about actually. May be you have forgotten the BTB organization.

(Refer Slide Time: 55:44)



So, the BTB has what? It has a value bit. It has a tag. It has a target. And then, I am attaching one more bit here, which is conditional branch bit. So, every instruction comes, looks up the BTB with a program counter. It may be heater or misdepending on the tag, if this entry holds a conditional branch this bit will be set. So, I know it is a conditional branch or not. And if it is, then I am also looking up the direction predictor in parallel like PHT, BHT whatever it is there.

So, I use that outcome, if hits and it is a conditional branch. Otherwise, if you hits somewhere where this is 0 this bit. I will have to discover this complete lookup, whatever I call, because it does not matter. And if I miss in the BTB, I really do not know what its instruction is? So, I will carry forward this particular outcome will be. So, in the next step when instruction will decoded, then I will be using that for discovering. So, this single bit is that is one, very small one.

Student: And we scope of target is rather than the number of bits

Scope of target, what you mean by that?

Student: Rather than number of bits per target

Why, whatever was there before, is there here.

Student: We are putting one extra here

This one extra bit, that is over itself. That is the only over it.

Student: We are mapping here from PC.

Yes, from PC.

Student: ((Refer Time: 57:18))

No, that is here. This one, the number of entries. If you use P bits or PC, it would be two to the P. Nothing to do at the width, what do I use?

Student: ((Refer Time: 57:34))

That is what, I was waiting to hear. You lose battery life in the very beginning, if you are running on the battery or you are essentially. You are expending a lot of power. You are

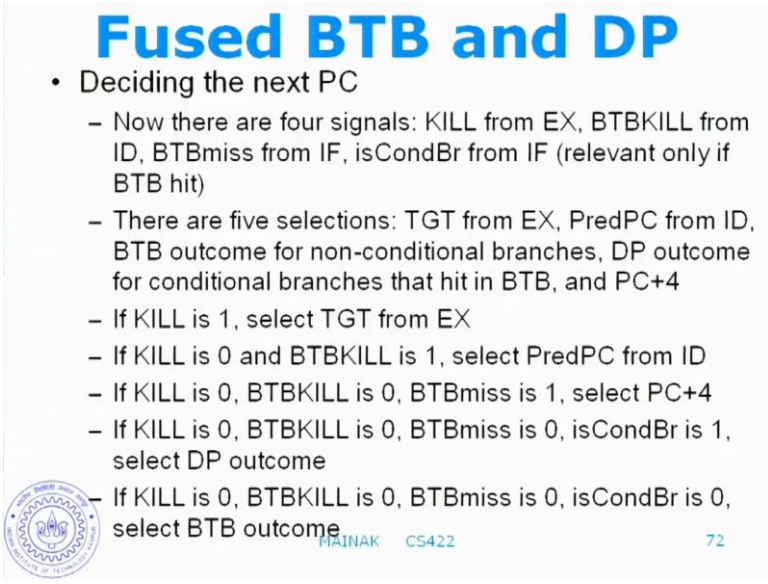


looking up a branch predictor for every instruction is a huge wastage. So here, so there is a tradeoff. What is the tradeoff? If I do not have this, puce BTB and DB I will probably have lower performance, which means I run longer. But, each cycle may have lower power consumption.

So, overall energy may be lower or weaker. That depends on, how good this is actually. If you can bring back those cycles, so you can actually expend more power for cycle. But, you have less number of cycles. So, that actually total energy is less. Then, this is the winning design, get it. Otherwise, it is not a very good design, because today you cannot lose energy. That is a very important thing.

So, you have to keep that in mind that, here you are spending a lot more extra energy when looking up the branch predictor for every instruction. So, that has to be tradeoff. So, often what processor do is, they put a very simple branch predictor with BTB to get a some outcome, which may be better than BTB. And then, you will have a laborite branch predictor in the decoder actually, which we lookup. So, there is a tradeoff here. With this, we may not be that good all the time. Just keep that in mind.

(Refer Slide Time: 59:21)



## Fused BTB and DP

- Deciding the next PC
  - Now there are four signals: KILL from EX, BTBKILL from ID, BTBmiss from IF, isCondBr from IF (relevant only if BTB hit)
  - There are five selections: TGT from EX, PredPC from ID, BTB outcome for non-conditional branches, DP outcome for conditional branches that hit in BTB, and PC+4
  - If KILL is 1, select TGT from EX
  - If KILL is 0 and BTBKILL is 1, select PredPC from ID
  - If KILL is 0, BTBKILL is 0, BTBmiss is 1, select PC+4
  - If KILL is 0, BTBKILL is 0, BTBmiss is 0, isCondBr is 1, select DP outcome
  - If KILL is 0, BTBKILL is 0, BTBmiss is 0, isCondBr is 0, select BTB outcome

MAINAK CS422 72

So, just quickly I want to decide the next PC in this scheme. So, there are now four signals coming in a kill from execute by stage, BTB kill from decoder, BTB miss from the fetch. It is a conditional branch from the fetch relevant. Only if the BTB hits this bit, coming out from the BTB, that is relevant only if there is a BTB bit. So, there are five

selections now, target from the execute stage, predicted PC from the decoder, BTB outcome for non-conditional branches, direction predictor outcome, for conditional branches that hit in the BTB and PC corresponds.

So, I have to select one of these, based on these signals, these 4 signals. So, what is the selection logic? If kill is 1, select target from execution. We have to say that, this is golden rule. It is always correct. So, if kill is 1, it overwrites everything. If kill is 0 and BTB kill is 1. Then, you select predictor PC from the decoder. If kill is 0, BTB kill is 0 and BTB miss is 1. That means, you miss the BTB there is no interaction.

We have nothing to do really. You just select PC plus 4. But, also carry within the direction predictor outcome, because you really do not know what this instructions. If kill is 0, BTB kill is 0, BTB miss is 0 and conditional branch is 1. Then, you will select the direction predictor outcome. So, remember that this outcome is not really a target. It is not a PC, it is a bit taken or not taken. So, if you combine that with the BTB entry, whatever is here to get the final topic?

And finally, if kill is 0, BTB kill is 0, BTB miss is 0, conditional branch is 0, you select the BTB outcome, because essentially these are the branches. These are non-conditional branches, like procedural calls and conditional jumps and all these things. So, that is your selection logic. Any question? I think I will stop here. Next time we will look up some of the research materials from this.