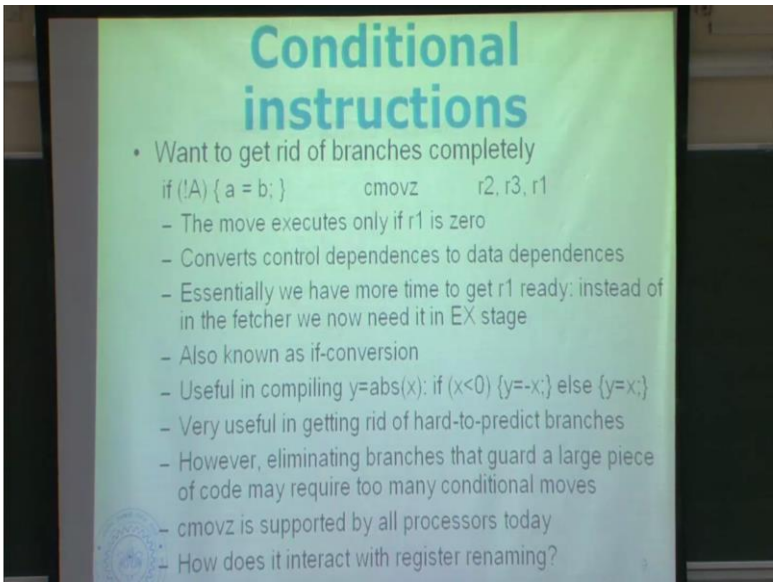
Computer Architecture

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

Lecture - 31
Case Study: MIPS R10000

We were discussing MIPS and ((Refer Time: 00:19)), I just remind you; it is a 64 MIPS processor. So, it is tools and renames. Of course, ((Refer Time: 00:31)) and we look at the fetch and decode stage last time, also looked at the ((Refer Time: 00:41)).

(Refer Slide Time: 00:55)



Conditional instructions

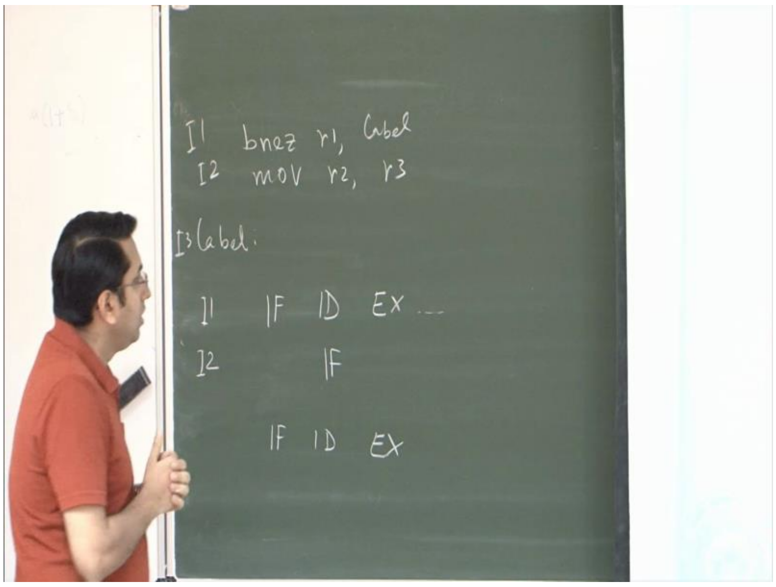
- Want to get rid of branches completely
  if (!A) { a = b; } cmovz r2, r3, r1
  - The move executes only if r1 is zero
  - Converts control dependences to data dependences
  - Essentially we have more time to get r1 ready: instead of in the fetcher we now need it in EX stage
  - Also known as if-conversion
  - Useful in compiling y=abs(x): if (x<0) {y=-x;} else {y=x;}
  - Very useful in getting rid of hard-to-predict branches
  - However, eliminating branches that guard a large piece of code may require too many conditional moves
  - cmovz is supported by all processors today
  - How does it interact with register renaming?

So, this causes a uses, class of distractions called traditional distractions. We have discussed it earlier. So far that, I spend some time explaining, what these are? Since, actually the idea recognizes that, if possible nearly get rid of branches completely to program. Because, we are seen that, branches cause problem in nature simply it predict it. So, take this particular a code says that, if not A than small a equal to small b.

So, this case translates that particular instruction CMOVZ stands for Conditional Moved to 0. And essentially, what is done is that, whose value r 3, r 2 provided r 1 essentially. So, essentially converts the control dependences to data dependences. Now, what were essentials see that? We have two sources for this instructions, r 1 and r 2. And r 1 in the sense, it quotes, the branches completely, it is predicated essentially.

So essentially, we have to now more time to get r 1 ready, instead of that. In the lecture, we now need exchange, what does mean is that, if you instead later move instructions, of course, you MIPS assemble will translate CMOVZ possibly add instruction with additional 0. So, suppose, instead of this instruction, we have a single CMOVZ instruction which moves r 3 to r 2, but before that branches ((Refer Time: 02:36)).

(Refer Slide Time: 02:42)

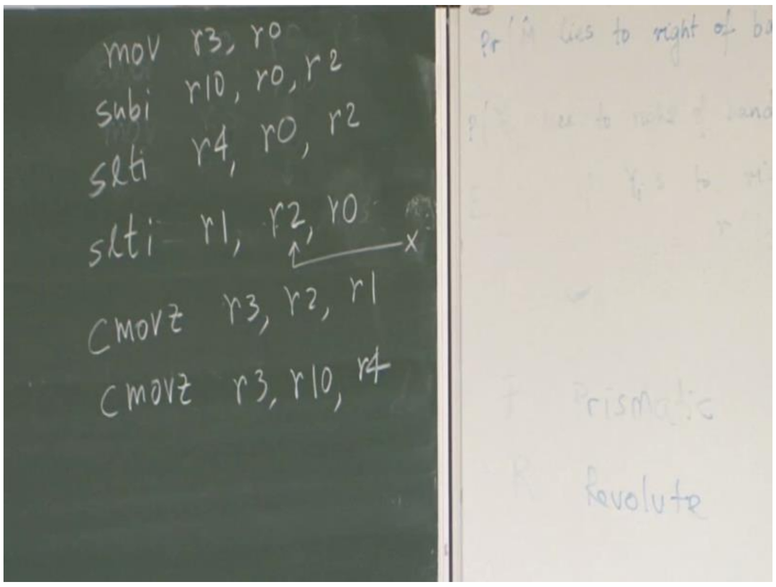


So, ((Refer Time: 02:38)) roughly use like this, if you do a translation of this. Branches not equal to 0, r 1 relative, achieve that, r 1 is a capital A. And relative is something and here, you have a MOV r 2 column. ((Refer Time: 03:08)) there is no branch. That is correct, that is the translation of CMOVZ.

So essentially, what would have happened is that, this instruction get is fetched. So, let us call this I 1, let us call this I 2, I 1, I F, I D, E X, etcetera. So, when we face I 2, suppose I 3, we need the orthodontics this instruction here. Who know that, whereas, if we look at that instruction, we look at the execution stage for r 1 and r 3 to get ready? Only the execution stage of this instruction will be stall, only if at least one of this is not ready r 1 and r 3.

So, that is the meeting of this purpose statements we have more time to get r 1 ready, instead of the fetcher, we have in the EX stage. This also known as, if conversions, that is a technical term used in the compiler community. And they useful in combining this things of course. For example, if you want to say, y equal to execute of x, essentially this rightly, x is less than o, then y minus x.

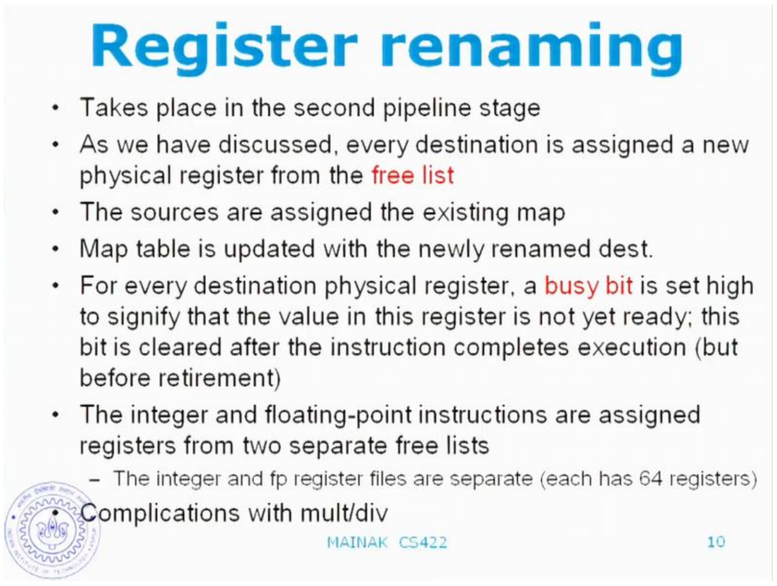
(Refer Slide Time: 04:48)



```
mov   r3, r0
subi  r10, r0, r2
slti  r4, r0, r2
slti  r1, r2, r0
cmovz r3, r2, r1
cmovz r3, r10, r4
```

So, a very useful getting rid of hard to predict branches. For example, this could actually here very hard to predict branches, predict on, how the value of x predicts. However, eliminating branches that guard a large piece of code, may require too many conditional moves. So, that may not be a feasible actually. So, this is normally tried on small pieces of a conditional structures and CMOVZ stage is supported by all processors today. Now, the question is that, how does it interact with register renaming. To understand, why,this is at all problem.

(Refer Slide Time: 05:29)



Register renaming

- Takes place in the second pipeline stage
- As we have discussed, every destination is assigned a new physical register from the free list
- The sources are assigned the existing map
- Map table is updated with the newly renamed dest.
- For every destination physical register, a busy bit is set high to signify that the value in this register is not yet ready; this bit is cleared after the instruction completes execution (but before retirement)
- The integer and floating-point instructions are assigned registers from two separate free lists
  - The integer and fp register files are separate (each has 64 registers)
- Complications with mult/div

MAINAK CS422 10

So, have not it look at list of register renaming. So, let us first start see, what renaming task are MIPS ((Refer Time: 05:33)). So, any question of conditional instructions? So this takes place in the second pipeline stage. So, is essentially has decode rename use the single pipe stage. As we have discussed, every destination is assigned a new physical register from the free list.

So, this all already discussed in while discussing register renaming and also, whenever you made the instruction; that destination of instruction get it used physical register. For example here, actually r 3, r 10, r 4, r 1, ((Refer Time: 06:07)); these are all destinations that give physical register from free list. The sources are assigned, the existing map, so for that, we have table to which the table maintains the correct map. Of course, logical register to physical register is it.

The map table is updated with the newly renamed destination. For every destination in physical register, is it busy bits set high to signify that. The value in this register is not yet ready. This bit is cleared, after the instruction completes execution, but possibly before retirement. So these are also we have discussed this particular or exactly, ((Refer Time: 06:46)) interacts with your register value bit.

So, integer and floating point instructions are assigned registers from two separate free lists, because these are nearly distinct register sets integer. The integer and f p register files are separate each has 64 registers. So notice that, bits has 32 logical registers for each of bits and have double number of register bits. There is a slight complication involving multi-div. So, remember this two instructions, they have two bits instructions.

Because, if you multiplying two values to 32 bit values, produces 64 bit results and that is normally split into the high and low registers. Similarly, if I do a division, they quotient were reminder go to different registers high and low. The question is, how to relate them? Because, I do not have an option of relating two destinations here in this particular program.

So, what we does is that, first of all puts restriction on, so as he said renames four instructions in the cycle. Now, there are restrictions on the position of multi instructions participated, it cannot come to anywhere. Whenever, it encounters of multi div distraction is very important. It actions dominates the important there. It cannot rename any part of distraction. It breakdowns this each of these interaction in two smaller instructions that and rename them separately.

So, multi is broken down in two instructions, div will broken down in two instructions. And they will go separately from the instructions. However, their compile will be maintained throughout the practical. These two instructions are actually atomically combined, thus the position of this, and the position of the basic photograph.

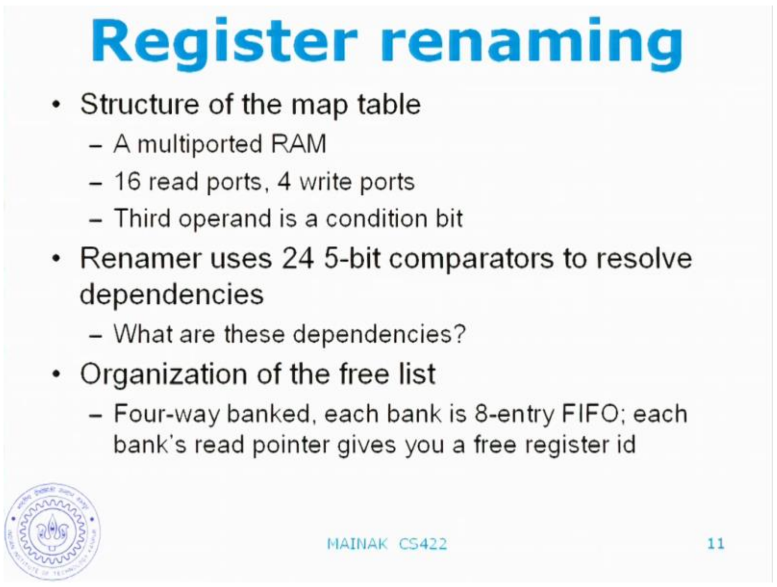
Student: ((Refer Time: 08:48))

Because, you have renamed two destinations and here, you do not have optional here. The hardware allows you to enable one destination it resides. So, from the free list, so the vacant physical register is look at it can provide you, only four registered every side, the four new register at most not ((Refer Slide Time: 09:13)).

Student: ((Refer Time: 09:17))

So, while distracted, so the point is that, what would be the other alternative? You make arbitrary large, the each instruction classified any number of registers. So, then, your hardware will become ((Refer Time: 09:38)) complicated. That is the problem, think about it.

(Refer Slide Time: 09:43)



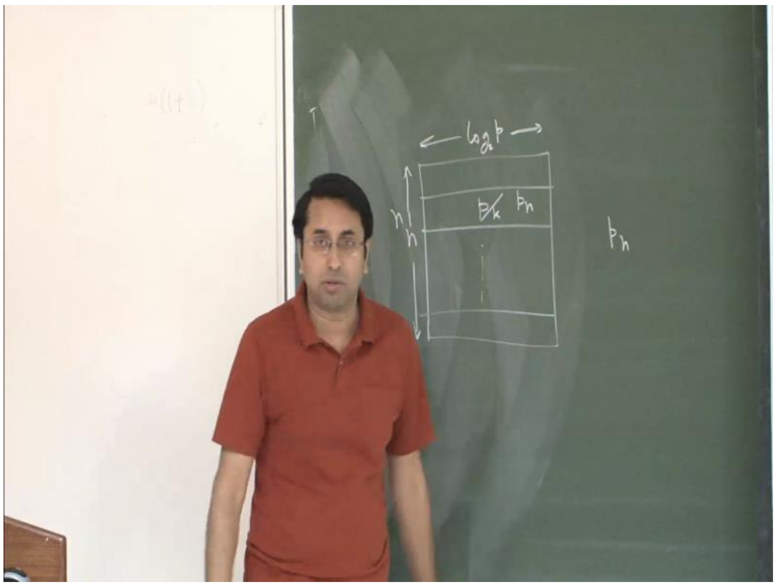
Register renaming

- Structure of the map table
  - A multiported RAM
  - 16 read ports, 4 write ports
  - Third operand is a condition bit
- Renamer uses 24 5-bit comparators to resolve dependencies
  - What are these dependencies?
- Organization of the free list
  - Four-way banked, each bank is 8-entry FIFO; each bank's read pointer gives you a free register id

MAINAK CS422 11

So, we have already talked about the C d organization. So, will soon talk about that, we will see, why these topic. So, what is the map table looks like, it is a multi ported ram. So, it talking about the table, which maintains logical to physical register mapping. So, if I have forgotten about the structure in this table, it looks like this.

(Refer Slide Time: 10:15)



If I take logical registers will have entries and if I had keep physical registers, ((Refer Time: 10:14)) assume belong g p. So, particular register x, you will have the corresponding physical register map story. So, it has 16 read ports and 4 write ports. So, this is probably understandable 4 write ports, why is that?

Student: ((Refer Time: 10:43))

In a cycle, you will at most update 4 entries, this table. So, become 4 writes. So, 4 write ports. Why do have 16 read ports?

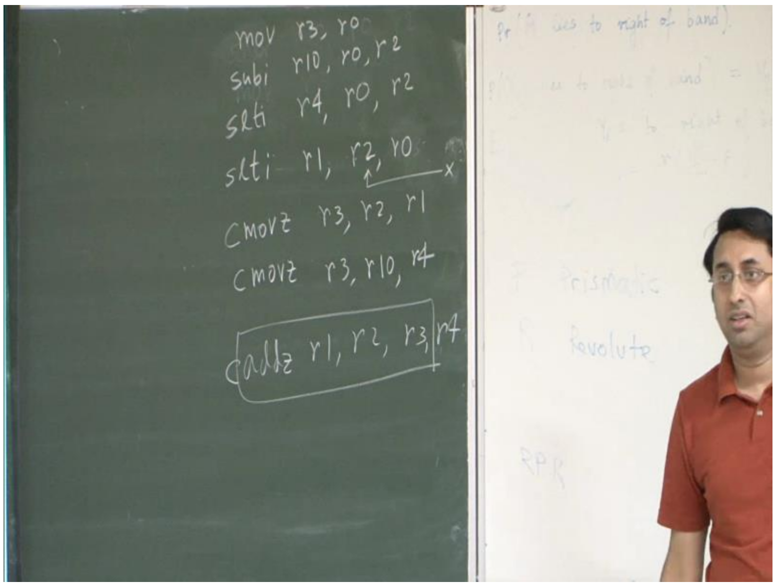
Student: ((Refer Time: 11:14))

Really, which stages?

Student: ((Refer Time: 11:22))

How many inputs, we have, you think? If I let it blank x, what is x? It should be equal to number bits. I can do in a cycle from this table. How many, I do, how many bits, forget about this number 16, give me some number ((Refer Time: 12:07)) to you.

(Refer Slide Time: 12:27)



```
mov r3, r0
subi r10, r0, r2
slti r4, r0, r2
slti r1, r2, r0
cmovz r3, r2, r1
cmovz r3, r10, r4
caddz r1, r2, r3, r4
```

So, let us suppose an inducting instruction here. Just say, I am renaming, thus add r 1, r 2, r 3. So, what we reach from the table, to rename this particular instruction?

Student: ((Refer Time: 12:45))

Sorry.

Student: ((Refer Time: 12:48))

Now tell me, which entries have to be from the table for this instruction?

Student: ((Refer Time: 12:56))

R 2 and r 3

Student: ((Refer Time: 12:58)

Right, I will read r 2 and r 3 to get the corresponding physical register ((Refer Time:

13:03)) for this. An exactly use physical register to r 1, I will update r 1. Then, I deal r 2 and r 3, I have ports certain instructions. So, how many ports, should I were?

Student: 8.

Sorry, 8. So, is there anybody, who thinks that, it is more; I just say that, you could have conditional and instructions c and z, which would have c sources. So, I can convert into this kind of we stopped here. Then, it will go to 12. Now, which does not have, only one time of conditional instructions? That is conditional CMOVZ; that is it. And conditional CMOVZ instructions as you can see have two sources, but I tell you that, the conditional CMOVZ instructions, we still require to read r 3 are also propagate. Why is that?

Student: ((Refer Time: 14:44))

Yes.

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

He has raised a separate point; he says that, if these instructions forget about the conditional part of it. I also need the old map of r 1. For example, suppose r 1 was map to p k previously. Now, I give you the new map, which is say p n. I need to read p k also, adjust around right over that p k with p n. Why is that? Let us somebody answer, why to need the whole map of the destination ((Refer Time: 15:45)).

So, what I am.

Student: ((Refer Time: 15:40))

Write, so currently r 1 is map to p k before that instruction is renamed. So, I asked preview to give me one new register and it gives me p n. So now, p k is replaced by p n. But, what he has saying is that, I also need to remember p k. So, I just now overwrite p k with p n forget about, why is that. So, why could we take p k, followed by the p k allocated?

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

Which instruction?

Student: ((Refer Time: 16:19))

Instructions do not have maps, registers have maps

Student: ((Refer Time: 16:26))

Write the last instruction produce r 1 called p k.

Student: Yes sir.

Student: ((Refer Time: 16:37))

Are you sure, at that time, I can pick p k, saying that, there are parallel instruction, which produce the r 1 and we mapped r 1 to p k at that time. And when that instruction commit is I can recycle p k and assign someone else.

Student: ((Refer Time: 17:05))

So, when does get bit in this case...

Student: ((Refer Time: 17:22))

Why you assigned?

Student: ((Refer Time: 17:25))

How do you use that?

Student: ((Refer Time: 17:35))

What would be a very simple cause recycle p k?

Student: ((Refer Time: 17:56))

You are right.

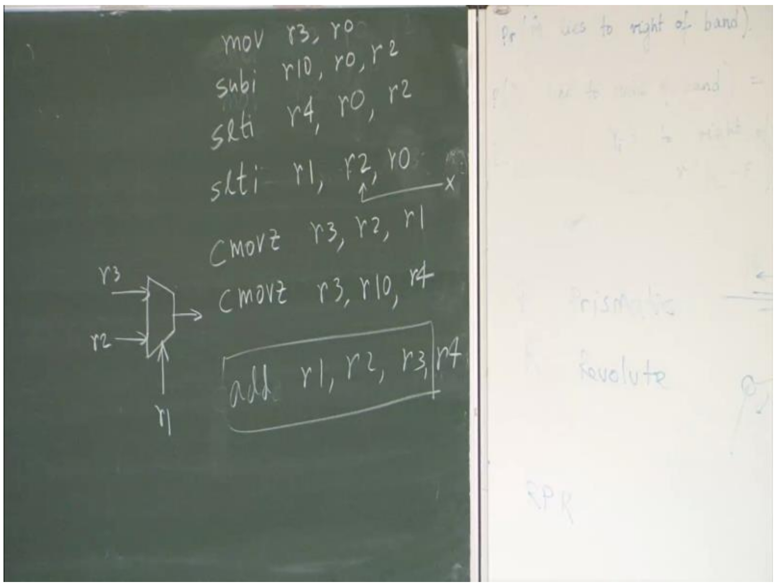
So, I should it, while your last instruction to use p k has completed. How do you know, what is the last instruction; that is used p k. As you mentioned that was these instruction is inside the pipeline, I know that, beyond this point p k is dash, for sure, because r 1 has now new in correction. But, these instruction has a final inside pipeline is not a point line, when it comes and goes. It starts here and goes to several pipe stages and then, completes.

How did know that, during this time, that is still some instructions the pipeline uses completed. That could, which is before that. So, whenever that, p k is sure will not be in that. Exactly, while these instructions comments, I know that everything before it has commented and anything after it should not return, because r 1 has now incurred. So, is very cause, I am getting, but is correct.

And this is the reason why, this instruction is to carrying p k with it. That finally commit is, it can recycle p k .You should move p k to the previous back. That means, I need four more extra read ports, just for every destination I read it four as well. So, that makes 12, I have two sources, which is writing for 12. Now, coming back to the previous you have seen that, I need to read r 3 here also for a different reason, not, what he is mentioned.

In this instructions and if you can answer correct that would make you in the parcel four conditional CMOVZ instructions, why it is so think about how to conditional CMOVZ. See, in front of memory register, you put a multiple x is that correct, will able to implement traditional CMOVZ the multiplex in selection will be r 1.

(Refer Slide Time: 20:39)



```
mov r3, r0
subi r10, r0, r2
slti r4, r0, r2
slti r1, r2, r0
cmovz r3, r2, r1
cmovz r3, r10, r4
add r1, r2, r3, r4
```

And, what you write, what was the input to the multiple x, adding two inputs, what are they? r 3 r 2. So, I either write same content back to the register or I write new contents that is what I mean. So, r 3 acts the source as well in this instruction. So, I have to need r 3, I may write r 3 or I may write r 2 to r 3 depending on finally, what r 1 results, who can know those what next, how do you design the register tell me, what is the hardware structure, there is no other way other than this actually.

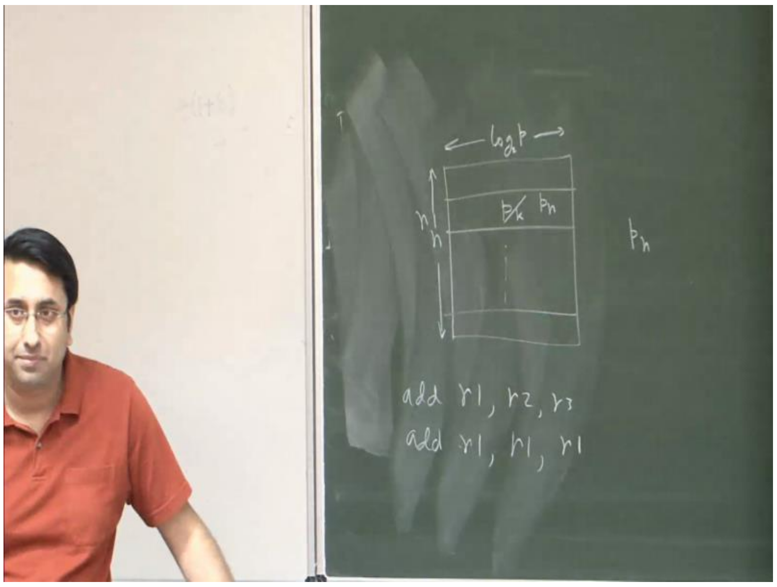
Student: ((Refer Time: 21:40))

Yes, the r 3 is being related always that is it will be, because this is the architecture this is the hardware. So, you have to prepare for the walls get that is partial content four conditional CMOVZ. First, what is mentioned that is what I am saying which one, how it is a hardware look at force that is all the way give the hardware. So, that is how you get 16 read ports, third operand is a condition bit.

So, we talk about this particular bit, we talked about registered pipe, but keeping mind that, this is also renamed. Even, if it is a bit, it is still renamed. So, quit handling, which has single bit value, but, it is actually a register describes this. The second problem normal activation, so the equation is not it clear 16 read ports 4 writes ports. And yes, if you did not have conditional CMOVZ instructions, you have 12 read ports, which have 16 read ports. But, because of this we have 4 extra CMOVZ.

The rename uses 24, 5 bit is comparators to resolve dependencies, what are these dependencies. So, we before try to actually account for this particular number 24 here can somebody tell, what are these dependencies?

(Refer Slide Time: 24:57)



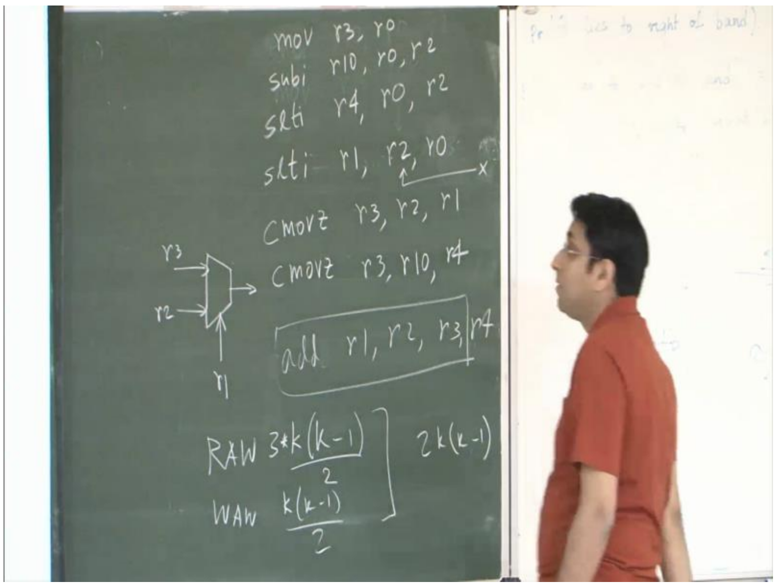
So, remember that, we are currently in a piece of hardware, which is tried to rename four instructions to parallel by looking at this particular table. Before that RAM have 16 read ports, yes, and the remaining has actually 4 instructions and these four instructions it iself, dependencies demand upon corresponds. How they actually these, suppose, that we deep all the input dependencies first started that, why did the, how does that.

Now, first of all can you give me an example, where there might be problem? So, let us suppose to get you started, let us suppose the first instruction is this yes can you give me. Do you mind if I write side register.

So, let us take this two instructions rename them parallel, So, what is a problem, so as he has mentioned I cannot get go head and treat each instruction individually and say that I am going read r 2 r 3 and going to read r 1 and r 1. But, that this instruction lead a wrong map, it should get the map that this instruction gets actually. So, this is the dependence we are talking about here, so now can somebody account for there.

Here, four instructions and first so discussing this particular example you also to make sure that this map finally, survives in the in the sort of r 1, this should not, then this order has also should be maintained. Even though, you have renaming four parallel there are certain order restriction that have to be obeyed within this parallel depending on the dependence. So, these comparators dependencies can somebody example that now what we did.

(Refer Slide Time: 27:03)



So, let us suppose that r 1 model size of k is not and k instructions and for worst case, we have three sources for instruction final I can have k conditional CMOVZ instructions. So, how many comparators do, I need exactly, so k is to k minus 1 component, I have 3 components, how I get this.

So, if I take the first target, I compare it with next k minus 1 instruction take this target, I am compare with next k minus 2 instructions and for each instruction. So, if I planning to change 4, I still do not get 24 by 2, what are the extra six comparators? This is the raw depends, what is that extra 6 comparators? I still to make sure that, in this case, this map survives not this one.

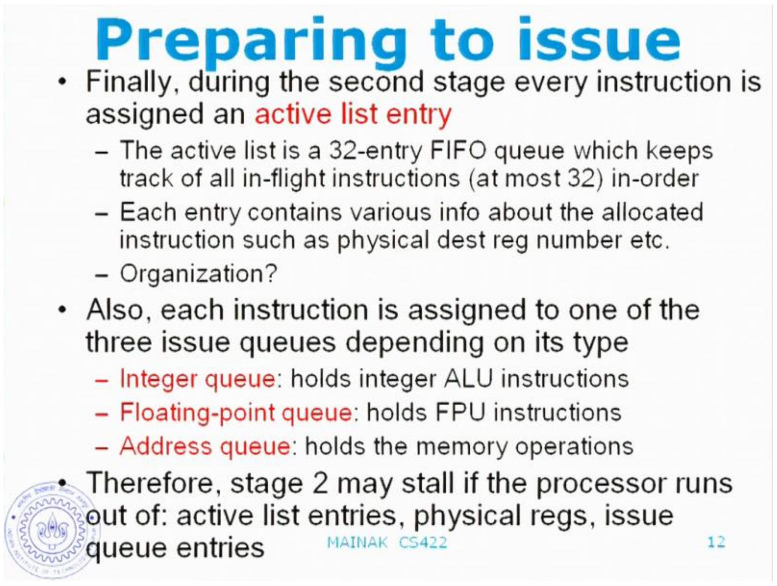
Student: ((Refer Time: 28:34))

So, k is to k minus 1 by 2 for the first target, compares the next minus 1 target this target we compares next k into k minus 1, so total is twice k into k minus 1. So now, the question is how many implements? I know how many comparators I need in deserves, so you can guess that there will be sudden bypass path within the rename stage that are actually operated which would actually pass on this r 1 map to this guides.

So, that take out whatever their rate for the table will finally take over it by whatever they have did to be able to do given right so. So essentially, what I am saying is that the final map that is good to get this equal to the register source will either whatever it has rate from the map table or something coming from one of this destination used. So, if multiplex which will be having these two inputs and it is selection will be taken by one of the comparators these are, how the comparators are finished.

So, it is four way banked each bank is 8 entry FIFO, so each bank's read pointer gives you a free register id, that is the simplest free list that can give you 4 registers at a side 4 FIFO’s. So that see, 4 peoples so cause of the 8 point these.

(Refer Slide Time: 30:34)



Preparing to issue

- Finally, during the second stage every instruction is assigned an active list entry
  - The active list is a 32-entry FIFO queue which keeps track of all in-flight instructions (at most 32) in-order
  - Each entry contains various info about the allocated instruction such as physical dest reg number etc.
  - Organization?
- Also, each instruction is assigned to one of the three issue queues depending on its type
  - Integer queue: holds integer ALU instructions
  - Floating-point queue: holds FPU instructions
  - Address queue: holds the memory operations
- Therefore, stage 2 may stall if the processor runs out of: active list entries, physical regs, issue queue entries

MAINAK CS422 12

So, after renaming we are almost ready to issues, so what will do is you assigned an active list entry there instruction. So, this is essentially the reorder profit it means the active list is a 32 entry FIFO queue which keeps track of all in flight instructions which gets in order.

Each entry contains various information about the allocated instruction such as physical register number extra, how do you organized the activities is, it a single FIFO should be broken down to much of FIFO's, how did decide that? So here, how do will decide this one, four way banked each 8 entry FIFO. What you three way banked 8 entry FIFO higher 60 way banked 280 entry FIFO, two way banked 60 entry FIFO, why are the one FIFO.

So, that, here we says that, the case of files 64 registers, exceptive says to the 64 and 6 as yes.

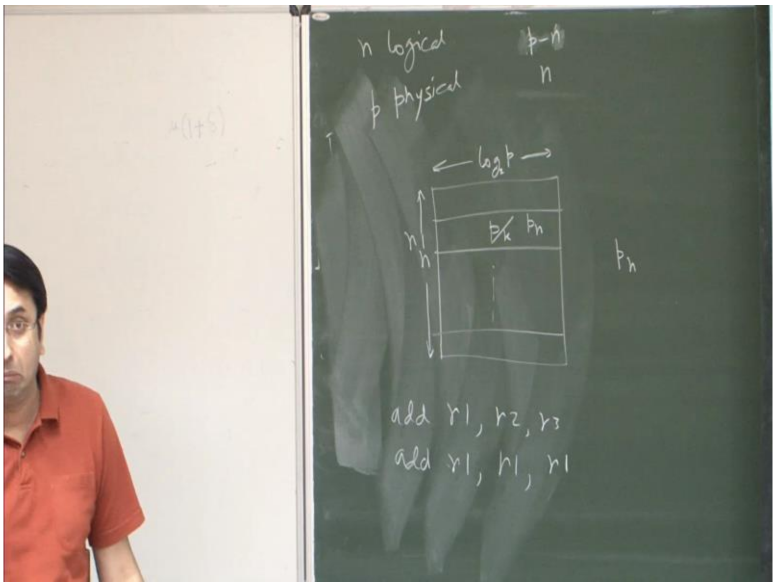
Student: ((Refer Time: 32:26))

We are no each various 64 register.

Student: ((Refer Time: 32:41))

Files tell me up...

(Refer Slide Time: 33:01)



Lets we see, I will ask some their n logical register and p physical register, how many is free list, what is minus in n

Student: ((Refer Time: 33:15))

P minus n, which one is bigger p is bigger, what?

Student: ((Refer Time: 33:41))

P is deduct would you like to devise p is p minus n why study as what is it. So, what is it, so what is free list dating something, it is an integer contents an integer which could be used.

So, question how size it is free list, what is the maximum size.

Student: ((Refer Time: 33:40))

Initial map.

Student: ((Refer Time: 34:47))

So, what is the wrongness side of the free list, how do you prove that p minus n, you are saying that certain point which had historically listed mapped atmosphere exactly.

Student: ((Refer Time: 35:03))

Say the pipeline, there could be multiple instructions that have mapped same logical register yes possible. So, there would be there sudden pointing time, there could be more than n logical registers that are mapped possible which means that more than n free lists exists there are occupied.

But, what is the worst case that, what you design for it, the machine put's up something will be mapped, yes.

Student: ((Refer Time: 36:04))

Why? Because, there are three physical register n already in the map can you prove that n will always be mapped at least 10, looking to write worst case for all the time that is what we need really the large possible free list size. Then, when do you these two others balanced, will you lint to some others physical balances?

Student: ((Refer Time: 36:35))

So, essentially document is that at any pointing time at least in register will rename mapped that if your pipeline is completely empty as he has doing nothing, there will have n registers mapped, some physical registers. So, the large possible free list size is about more than p minus n and that is what we need to size.

And that explains, why it is 32, because 64 minus 32, why is it organized like this, there are two free lists and just forget about both are organized in the same group. One integer free list, one floating point free list, why is organized like this?

Student: ((Refer Time: 37:37))

4.

Student: ((Refer Time: 37:46))

Why, cannot I do that in a single?

Student: ((Refer Time: 38:03))

I can design my hard ware’s with 4 read ports, I will give 4 read pointers that will most interesting, I will just think tell them that you can activate this four lines give me the contents, what will I use organization? I use something this. There I have 32 8 g ram which is more important here. I have 4 rams which have single port each of them that may huge difference. That is why; it is organized like this.

Why four because I needs to four registers at a side as he has mentioned it is not about writing four p's, otherwise g 1 is the reinterring these four are recycle. Now this works, why did, I know the return four things to the list at a side that also determines here, how many bit is. Because, we cannot registers to the to the free list for instructions commit is like I say while these instructions commit is, I will see the old map of r 1. That will now return to the free list, I put the write point of each this.

So, four registers will returned every cycle to the free lists at most. So, I hope you remember FIFIO works which is lead point which is right point, so as the hidden theory, now carrying this forward, how do you organize this thing, that things what determines the organization, what is the parameter, so how we balance four banks 18 piece per bank, 16 bank, what is it that two bank for bank, what is this cycle.

So, why do you read that for pipe stage decode, what is need form the decode stage

Student: ((Refer Time: 41:08))

How do you that, why do you at least stage, which stage exactly. So, that is what says right, during the second stage in which instructions assigned the activities which means it becomes right into the things in the activities.

So, how many I need to write in the cycle, how many instructions come into the stage in cycle, four instructions coming four instruction read, I need write 4 the information. So, I read 4 write ports, why you read from that port, take that quite it here, integer here. For example, it says the physical destination register number, r 1 here, it has stole it is old map in that the place to remember, what is going to read, it renaming to check the remaining.

Student: ((Refer Time: 43:00))

Exactly, so, when the instruction comments, I require this particular number certainly that, so number read port telling by the how many instructions that commit and mean; that means that is also four. So, that basically it has four banks each is an integer bank each is an. At the time of issue also keep no, while carrying relevant instruct relevant information along with the store this need to check.

Student: ((Refer Time: 43:39))

So, those information will be stored organizing will also contains for example, your branch out come. So, that you can update instruction comes. So, anything that you write to this particular accumulistic will be required at count that is why you read that. So, also each instruction assigned to one of the three issue queues, depending on it is type integer queue holds integer all instructions, floating point queue holds floating point instructions address, queue holds the memory operations. Therefore, stage two may stall if the processor runs out of any of these resources active list entries physical registers issue queue entries.