Computer Architecture
Prof. Mainak Chaudhuri
Department of Computer Science and Engineering
Indi an Institute of Technology, Kanpur

Lecture - 34
Case study: Intel Pentium 4

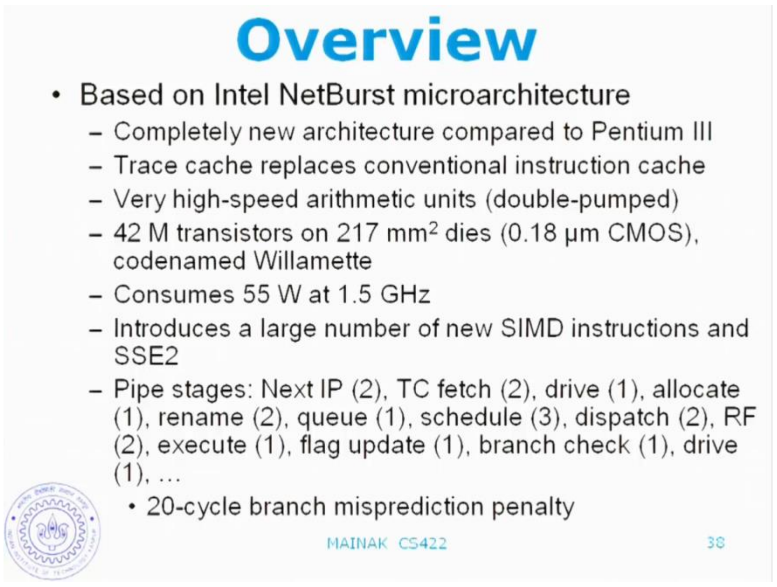
(Refer Slide Time: 00:23)

Intel Pentium 4

MAINAK CS422 37

So, today we will discuss last one in a single cases intel pentium 4. So, the one that we discuss first based on intel net burst micro architecture is a first one that increased on in a 42000 is a completely new architecture object to printing machine. So, in the mention that paintings to uses the physics micro architecture will be discuss the something till. So, Pentium four trace cachous they replace the conventional instruction cache, it uses very hard speed called double pumped, we use to discuss that.

(Refer Slide Time: 01:01)



Overview

- Based on Intel NetBurst microarchitecture
  - Completely new architecture compared to Pentium III
  - Trace cache replaces conventional instruction cache
  - Very high-speed arithmetic units (double-pumped)
  - 42 M transistors on 217 mm^2 dies (0.18 µm CMOS), codenamed Willamette
  - Consumes 55 W at 1.5 GHz
  - Introduces a large number of new SIMD instructions and SSE2
  - Pipe stages: Next IP (2), TC fetch (2), drive (1), allocate (1), rename (2), queue (1), schedule (3), dispatch (2), RF (2), execute (1), flag update (1), branch check (1), drive (1), …
    - 20-cycle branch misprediction penalty

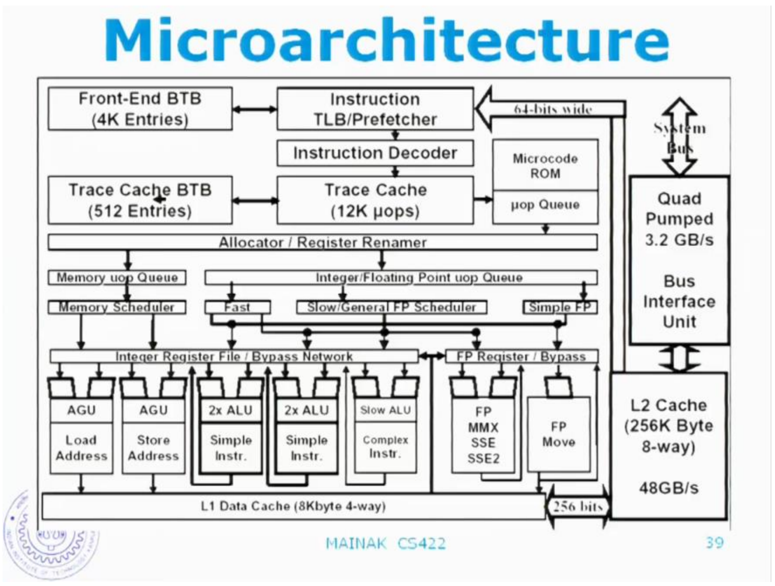
MAINAK CS422 38

So, essentially what this speaks is that if your processor also 3, 2 minutes reduce they will complete. They first act 42 milli transistors on 217 millimeters per dies on one must. So, this is a mix and the one that will be discussing now was coated. So, Intel Pentium 4 you may not know, because this is what come out, Pentium 4 has several. We will discuss three of this; the first one is next we discuss one, and the next we discuss. So, this we putting much show the new designs that introduced as you go next is the last one actually. So, we will consumes 55 what at 1.5 giga hertz and it introduce a large amount of the SIMD instructions and SSE2 instructions pipe stages. So, these a partial pipe show you the branch prediction. So, calculation of next instruction point at size. So, so you need tell this is you occasionally find.

So, this happens more as you try to either frequency. So, that is the as you reduce transistor size, but do not as past is that your communication becomes more less time as you study and higher frequency what happens is that you find out suddenly you just cannot communicate from function a to function b and actually accomplish the task. So, you should want to communicate b and b has to the new. So, suddenly falling that you cannot accommodate this communication and b in the side. So, there will be regarding two points. So, that is stage, you allocate queues ((Refer Time: 04:07)) etcetera, rename cycles samples, there this term actually schedule queues, these are schedules 100 micro ((Refer Time: 04:18)). We just say that that is items and then dispatch micro to the functionaries that is two cycles, and after execution is a there is something called a flag for example, by addition operation. So, there are many other flags that are mostly used by branch instructions if you if you decide. So, the flag a bit and then you decide to the branch and finally the branch outcome one more cycle to the weather safe or not.

So, if you count you will find that starting from the beginning of the time to the time to get to know that the branch prediction was correct or wrong. So, to that correct or not and that will move that cycle this particular processer. So, 20 cycle you essentially wastage. So, is that we need smart.

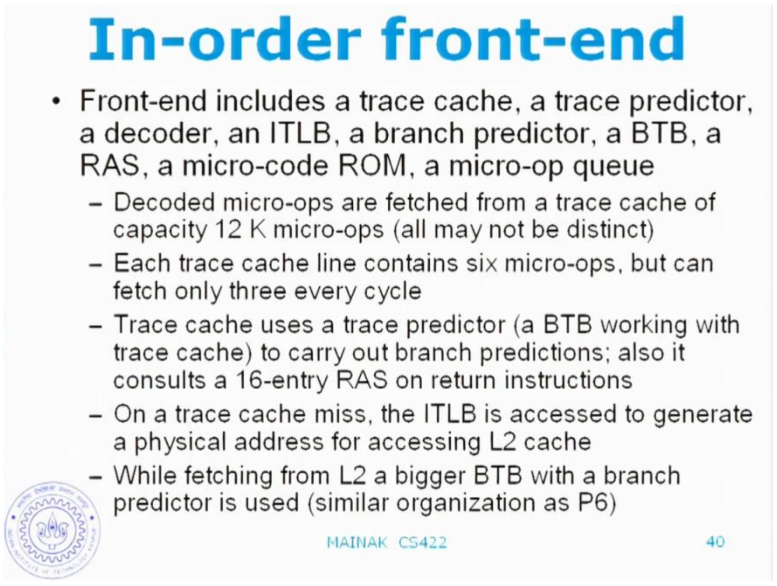
(Refer Slide Time: 06:13)



So, this is a block diagram of a micro architecture. So, there is a front end similarly in a trace cache. So, since trace cache and depending on the target. So, when you read from trace cache according to the trace will be very fine, there is a branches giving in the same what the branches there is no for trace cache a micro ops which is why you see that the decoder is not in the path of the trace cache it is actually. So, only you keep trace cache there only otherwise you do not know trace cache the micro queue structures can... So, what they do is that of course, rename instructions we should discuss already and the allocator allocates in two different queues what is s general purpose other what is the memory? From these two queues you said that to the respective scheduler queue.

So, these two these two are, but to be stored. So, these two queues said instructions to what they called schedules see that there are five schedules; one is the memory operated schedules which will schedule, there are four schedules here; on the fast scheduler one the slow schedule one the general floating point scheduler one the simple floating point scheduler. So, what they do is that all of them to the register path. For example, floating point schedulers that the other should send, and then you have several functions used there you can see their in the floating point on the interior side we have two address and there are two double ALU's that, and there is a slow ALU which execute complex instructions and upper that same there only operations continue after this address L2 cache micro architecture also you can see that there is a great path from L2 cache to the instruction t l b or instruction prefetcher yeah. So, in question about this general. So, let us start with the... So, I just that together of functional.

(Refer Slide Time: 11:03)



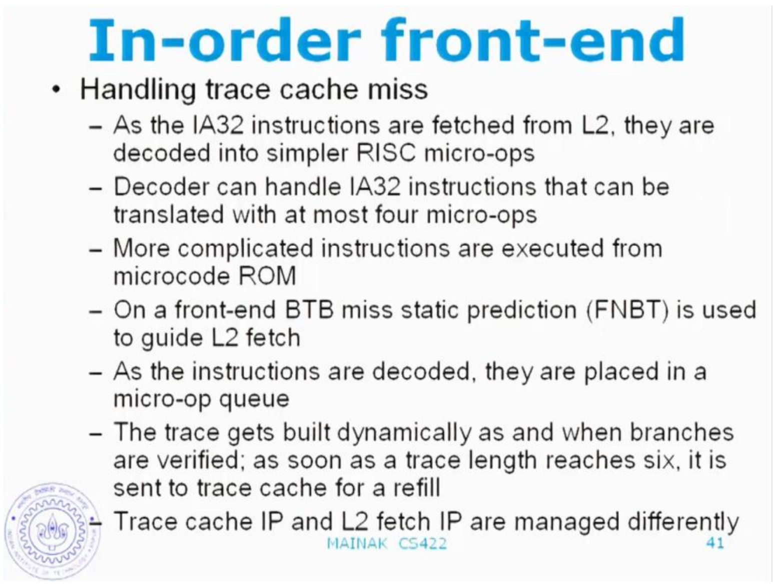
In-order front-end

- Front-end includes a trace cache, a trace predictor, a decoder, an ITLB, a branch predictor, a BTB, a RAS, a micro-code ROM, a micro-op queue
  - Decoded micro-ops are fetched from a trace cache of capacity 12 K micro-ops (all may not be distinct)
  - Each trace cache line contains six micro-ops, but can fetch only three every cycle
  - Trace cache uses a trace predictor (a BTB working with trace cache) to carry out branch predictions; also it consults a 16-entry RAS on return instructions
  - On a trace cache miss, the ITLB is accessed to generate a physical address for accessing L2 cache
  - While fetching from L2 a bigger BTB with a branch predictor is used (similar organization as P6)

MAINAK CS422 40

So, trace cache a trace predictor decoder and instruction, a branch predictor, Micro code ROM, and micro op queue. So, decoded micro ops are fetched from trace cache of capacity 12 micro ops and all each trace cache line contains six micro ops, but can fetch only three every cycle trace cache uses a trace predictor to carry out branch predictions. So, its own purpose of this is that during the consuming a trace it might contain several branches. You have to go also on a trace cache miss a ITLB is accessed to generate a physical address for accessing L2 cache while fetching from L2 a bigger BTB with a branch predictor is used means which has similar organization as the P6 micro architecture. So, you should happen you should imagine cache as the there you can see that because you just as if this is your first level of instruction cache. So, get you have to know what next instruction. So, this is the reason why you had a, but these two this one and trace predictor.

(Refer Slide Time: 12:31)



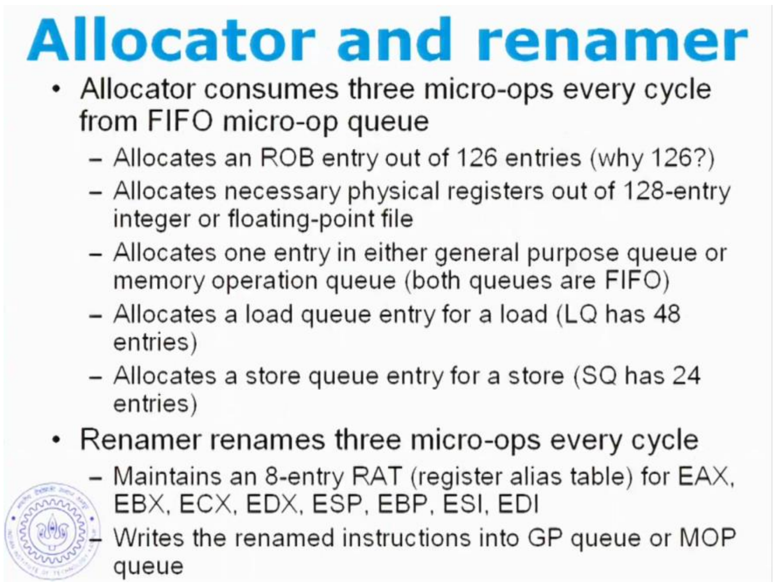
In-order front-end

- Handling trace cache miss
  - As the IA32 instructions are fetched from L2, they are decoded into simpler RISC micro-ops
  - Decoder can handle IA32 instructions that can be translated with at most four micro-ops
  - More complicated instructions are executed from microcode ROM
  - On a front-end BTB miss static prediction (FNBT) is used to guide L2 fetch
  - As the instructions are decoded, they are placed in a micro-op queue
  - The trace gets built dynamically as and when branches are verified; as soon as a trace length reaches six, it is sent to trace cache for a refill
  - Trace cache IP and L2 fetch IP are managed differently

MAINAK CS422 41

So, how you handle trace cache miss as the IA32 instructions are fetched from L2 cache, they are decoded into simpler risk micro ops. So, we talk about this particular part. So, missed trace cache. So, something for trace cache goes to the decoder and then… So, decoder can handle IA32 instructions that can be translated with at most four micro ops more complicated instructions are executed from micro code rom. So, also we discussed earlier to talking about the article on a front end BTB miss static prediction is used to guard the L2 fetch and what the static prediction says as the instructions are decoded, they are placed to the micro op queue the trace gets built dynamically as and when branches are verified as soon as a trace length reaches 6, it is sent to trace cache for refill, because a trace cache nine contains six micro ops the trace cache instruction pointed into cache instruction.

(Refer Slide Time: 13:50)

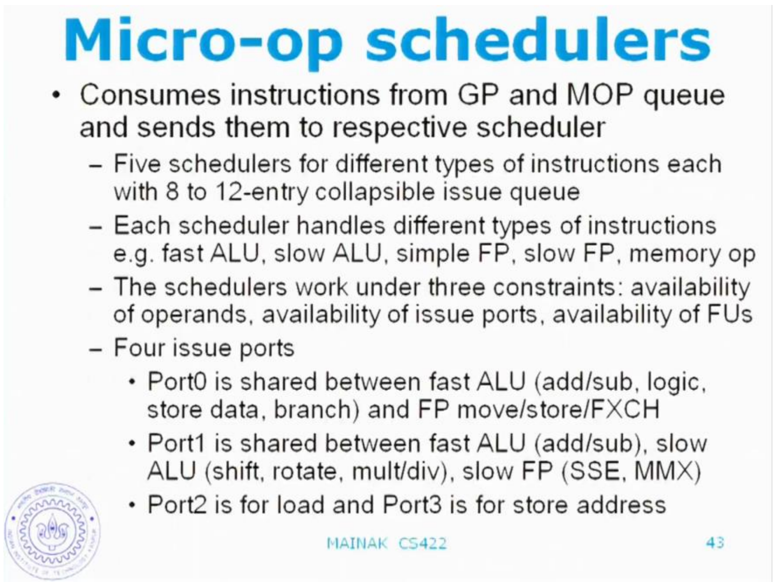


Allocator and renamer

- Allocator consumes three micro-ops every cycle from FIFO micro-op queue
  - Allocates an ROB entry out of 126 entries (why 126?)
  - Allocates necessary physical registers out of 128-entry integer or floating-point file
  - Allocates one entry in either general purpose queue or memory operation queue (both queues are FIFO)
  - Allocates a load queue entry for a load (LQ has 48 entries)
  - Allocates a store queue entry for a store (SQ has 24 entries)
- Renamer renames three micro-ops every cycle
  - Maintains an 8-entry RAT (register alias table) for EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI
  - Writes the renamed instructions into GP queue or MOP queue

Once this is done you will got the allocator, and the Renamer the allocator consumes three micro ops every cycle from FIFO micro op queues allocates an rob entry out of 126 entries. So, why using 126, because it wants something that is divisible 3, because you are allocating three allocates necessary physical registers out of 128 entry integer or floating point file. Because it has to be a part two allocates; one entry in either general purpose queue or memory operation queue both in the FIFO allocates a load queue entry for a load queue has 48 entries allocates a store queue entry for a store queue has 24 entries the renamer renames three micro ops every cycle maintains an 8 entry re register a l i as table writes the renamed instructions into general purpose queue or memory op queue.

(Refer Slide Time: 15:08)



Micro-op schedulers

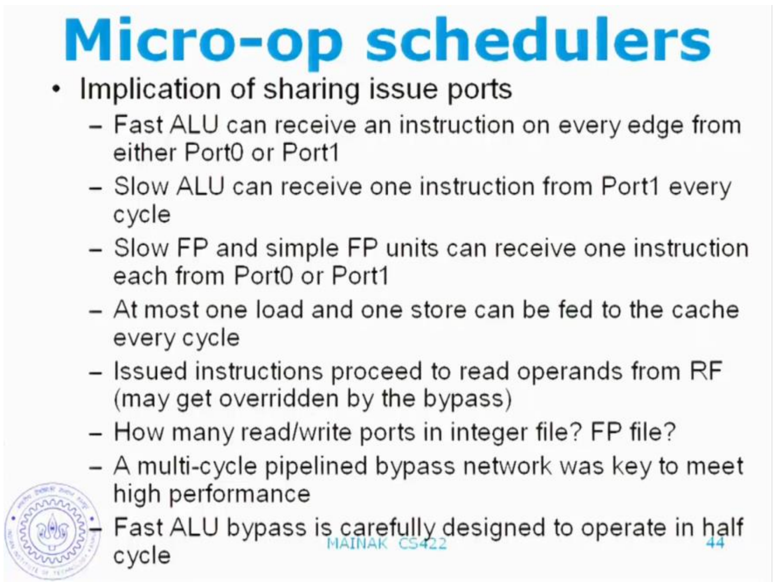
- Consumes instructions from GP and MOP queue and sends them to respective scheduler
  - Five schedulers for different types of instructions each with 8 to 12-entry collapsible issue queue
  - Each scheduler handles different types of instructions e.g. fast ALU, slow ALU, simple FP, slow FP, memory op
  - The schedulers work under three constraints: availability of operands, availability of issue ports, availability of FUs
  - Four issue ports
    - Port0 is shared between fast ALU (add/sub, logic, store data, branch) and FP move/store/FXCH
    - Port1 is shared between fast ALU (add/sub), slow ALU (shift, rotate, mult/div), slow FP (SSE, MMX)
    - Port2 is for load and Port3 is for store address

MAINAK CS422 43

So, the next head is micro op scheduling it consumes instructions from general purpose and memory op queue and sends them to respective schedulers; there are five schedulers for different types of instruction each with 8 to 20 entry collapsible issue queue, each scheduler handles different types of instructions there. What we mentioned about this fast ALU, slow ALU, simple FP, slow FP, and memory op the schedulers work under three constraints availability of operands availability of issue ports availability of functional units.

So, there are four issue ports in Intel Pentium four port zero is shared between fast ALU and floating point move store and floating point exchange. So, or you can see volume operations the store data and branch operations port one is shared between fast ALU slow. So, there are two fast op queues right, port two is for load and store, three is for store address. So, these are the four. So, this is these are ports two actually this took the other three other three, sorry other three ports.

(Refer Slide Time: 16:44)



Micro-op schedulers

- Implication of sharing issue ports
  - Fast ALU can receive an instruction on every edge from either Port0 or Port1
  - Slow ALU can receive one instruction from Port1 every cycle
  - Slow FP and simple FP units can receive one instruction each from Port0 or Port1
  - At most one load and one store can be fed to the cache every cycle
  - Issued instructions proceed to read operands from RF (may get overridden by the bypass)
  - How many read/write ports in integer file? FP file?
  - A multi-cycle pipelined bypass network was key to meet high performance
  - Fast ALU bypass is carefully designed to operate in half cycle

MAINAK CS422 44

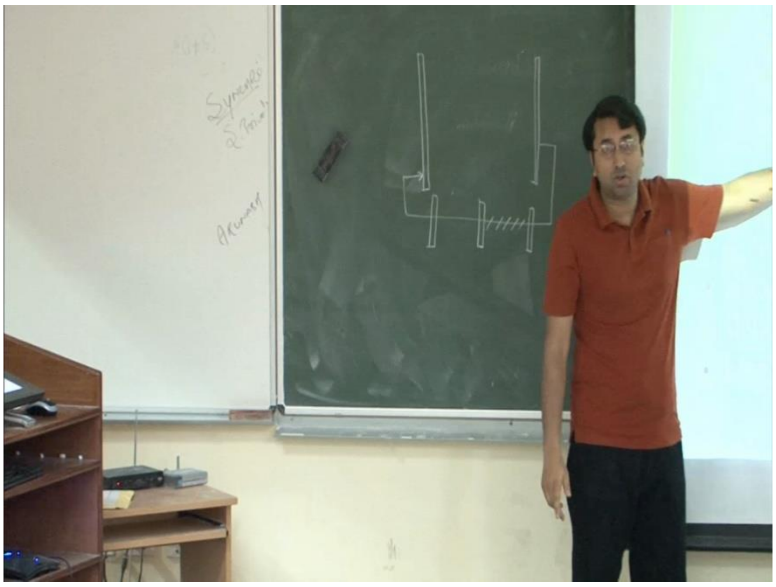
This is the one port this is the another port. So, what is the implication of sharing issue ports. So, the question me. So, that we have dedicated issue ports. So, again discuss also why last time we discussed that.

So, this is the instructions. So, these are very common, but these are also common. So, you look at you can assured that most of the time the fast ALU it will actually similarly here, these are common instructions, but these are common instructions of fast ALU instructions also the slow ALU instructions shift and rotated are actually common, but most of the time fast ALU. The reason for this is the interesting is that you find that memory operations are always given most of the time of the fast ALU's for, because has very small number of resistance those we have done that if you has small register number of registers.

You will have a lot loads to operations, you will you which has the compiler registers you speed data on the memory and then against to... So, sir what is the storing point exchange exchanges a floating point register with top of the floating points, but it is not a. So, fast ALU can receive an instruction on every edge from either port zero or port one, because more the ports are connected to fast ALU, the slow ALU can receive one instruction from port one every cycle the slow fp, and simple floating point units can receive one instruction each from port zero or port one at most one load, and one store can be fed to the cache every cycle, because issued instructions proceed to read operands from register which may get overridden by bypass.

So, how many read write ports in integer files will be computed that based on this particular we know and the Pentium four had a multi cycled pipelined bypass network. So, among that designed bypass networks, we have pipelined, we have assumed that we can bypass values from one stage to another right as you go towards high frequency this is impossible. So, what do is that the destination stage of course, it is just because the communicated from here to there which may not. So, pipelining the bypass path...

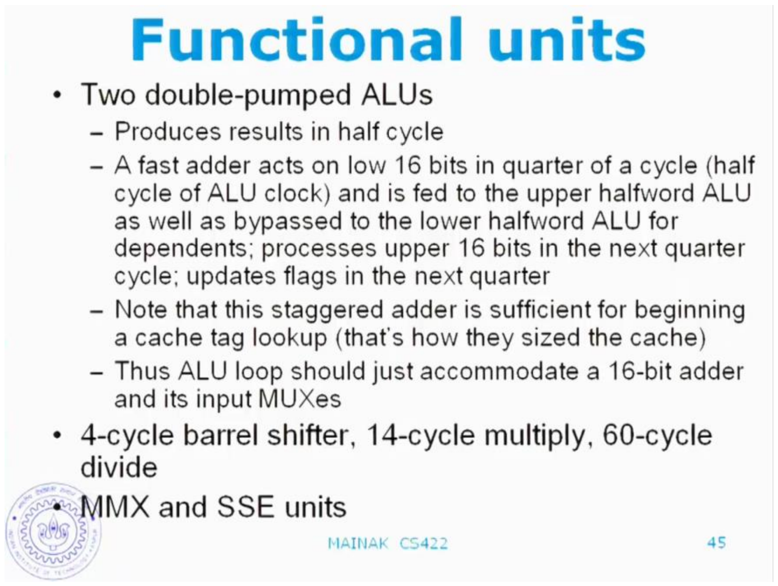
(Refer Slide Time: 19:54)



So, let us suppose this is my destination pipeline register that will bypass from here to here itself and this length of the wire is… So, long that you cannot bring to the single size right. So, what are the options, now you have well we can say that this value will be held for several cycles and during those cycles this register cannot any more bypass values we otherwise we value to right the other option is that put latches in between. So, you segment the wires. So, that this can be done in single side. So, then value is here. So, this particular segment new values.

So, that was the key requirement to build the high performance, otherwise which is almost impossible to have fast ALU bypass is carefully designed. So, there are two double pumped ALU's produces results in half cycle. So, how their keep that fast? So, essentially what happened is that in upper cycle we have done with the operation.

(Refer Slide Time: 21:31)



Functional units

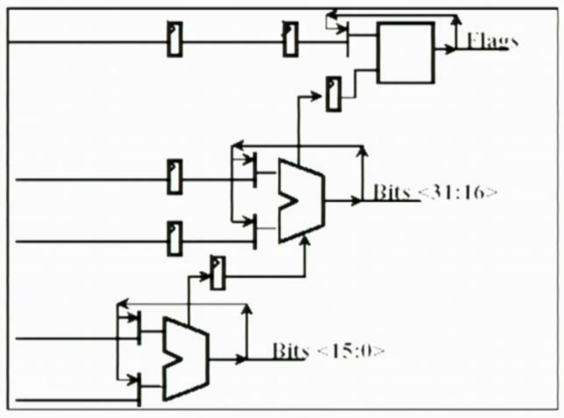
- Two double-pumped ALUs
  - Produces results in half cycle
  - A fast adder acts on low 16 bits in quarter of a cycle (half cycle of ALU clock) and is fed to the upper halfword ALU as well as bypassed to the lower halfword ALU for dependents; processes upper 16 bits in the next quarter cycle; updates flags in the next quarter
  - Note that this staggered adder is sufficient for beginning a cache tag lookup (that's how they sized the cache)
  - Thus ALU loop should just accommodate a 16-bit adder and its input MUXes
- 4-cycle barrel shifter, 14-cycle multiply, 60-cycle divide
- MMX and SSE units

MAINAK CS422 45

So, this staggered adder is sufficient for beginning of the cache tag lookup, because this is also used by the address sequence unit, and the index part the index the fast ALU is enough the ALU loop should just accommodate the 16 bit adder and it is input mixes.

(Refer Slide Time: 22:09)

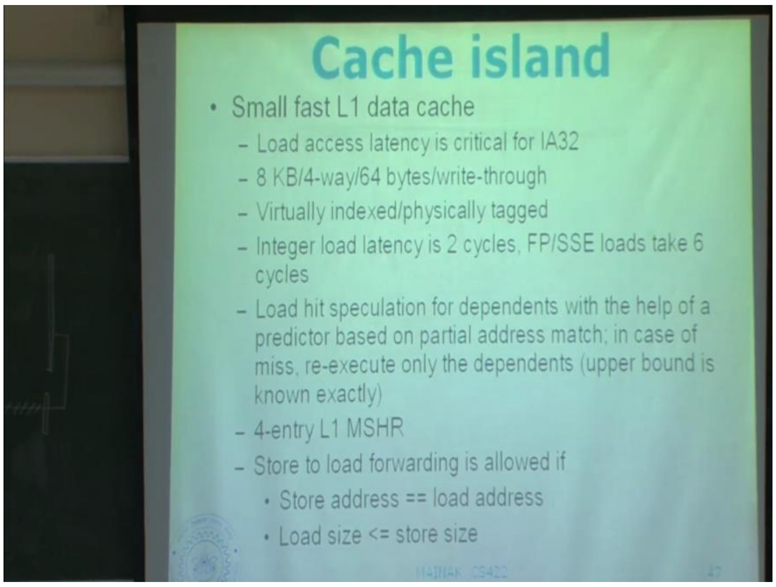
Staggered add



MAINAK CS422 46

So, also sent to the to the next ALU the upper for the adding of the next sixteen bits, and then the barrel shifter is a four cycle there is a 14 cycle multiply and 60 cycle divide and the...

(Refer Slide Time: 23:12)



Cache island

- Small fast L1 data cache
  - Load access latency is critical for IA32
  - 8 KB/4-way/64 bytes/write-through
  - Virtually indexed/physically tagged
  - Integer load latency is 2 cycles, FP/SSE loads take 6 cycles
  - Load hit speculation for dependents with the help of a predictor based on partial address match; in case of miss, re-execute only the dependents (upper bound is known exactly)
  - 4-entry L1 MSHR
  - Store to load forwarding is allowed if
    - Store address == load address
    - Load size <= store size

MAINAK CS422 47

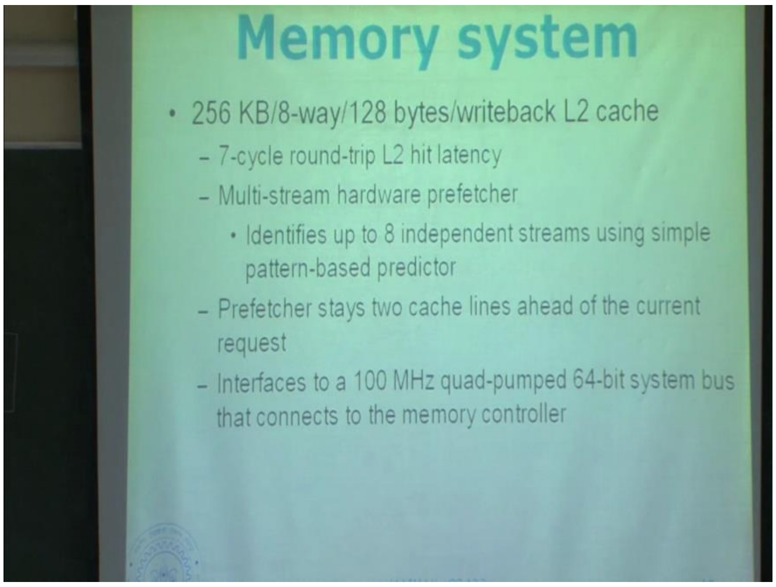
So, these are basic functional units, it will have the cache Island has a small and fast L 1 data cache load access latency is critical for IA32, as I have just mentioned Intel architecture is pay special attention to memory operations for obviously, data cache is 8 kilo byte 4 way 64 bytes and writes through virtually indexed physically tagged integer load latten latency is 2 cycles and floating point loads take six cycles load hit speculation for dependents. As we discussed with the help of predictor based on partial address match in case miss re execute only the dependents. So, here unlike the other architectures they actually do not re execute the dependents, and here how they do it is that they have estimate the upper bound of the number of dependents. If you because it exactly calculate the number of cycles that you take to know the exact out long of the of the of the hit of the cache hit.

And read those cycles how many dependents solution that we can also calculate from your, and they actually a buffer of that size that dynamically remember who are the dependents, there is 4 entry lone masher. We have discussed purpose of this particular structure in the class store to load forwarding is allowed is the store address matches, the load address and load size less than the store size.

(Refer Slide Time: 25:05)

So, suppose I have a load instruction which loads these bytes, and other storage instruction which stores these bytes. So, this is the store and this is the load and this is the program. So, the store comes before the load we are right. So, in this case the load can actually take the values to the store completely, because the load is contained completely store what I want is that the starting allied and the load size must be less than the stores, lone cache.

(Refer Slide Time: 25:43)

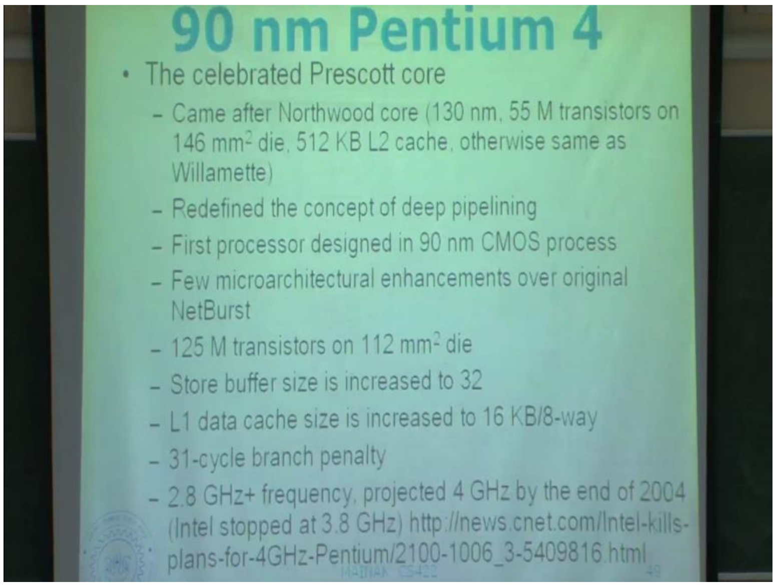


Memory system

- 256 KB/8-way/128 bytes/writeback L2 cache
  - 7-cycle round-trip L2 hit latency
  - Multi-stream hardware prefetcher
    - Identifies up to 8 independent streams using simple pattern-based predictor
  - Prefetcher stays two cache lines ahead of the current request
  - Interfaces to a 100 MHz quad-pumped 64-bit system bus that connects to the memory controller

We have 256 kilo byte 8 way 128 byte line size write back L2 cache seven cycle round trip L2 hit latency, it has a multi stream hardware prefetcher identifies up to 8 independent streams using simple pattern based predictors. So, most means stripe predictors which pick upped arithmetic progresses in addresses prefetcher stays two cache lines ahead of the current request interfaces to 100 milli hertz quad pumped 64 bit system bus that connects to the memory any question. So, we move on to the next one. So, what is the, oh I am sorry yeah. So, see actually maintains four different clocks at four different places and on each for every each of the based clocks transfer 64 bits. So, essentially sorry there would be there would be any more questions.

(Refer Slide Time: 27:21)



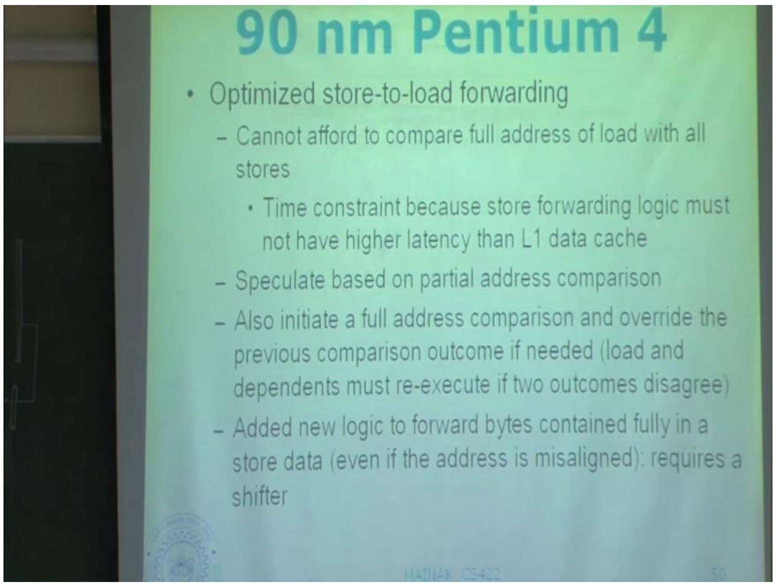
90 nm Pentium 4

- The celebrated Prescott core
  - Came after Northwood core (130 nm, 55 M transistors on 146 mm^2 die, 512 KB L2 cache, otherwise same as Willamette)
  - Redefined the concept of deep pipelining
  - First processor designed in 90 nm CMOS process
  - Few microarchitectural enhancements over original NetBurst
  - 125 M transistors on 112 mm^2 die
  - Store buffer size is increased to 32
  - L1 data cache size is increased to 16 KB/8-way
  - 31-cycle branch penalty
  - 2.8 GHz+ frequency, projected 4 GHz by the end of 2004 (Intel stopped at 3.8 GHz) http://news.cnet.com/Intel-kills-plans-for-4GHz-Pentium/2100-1006_3-5409816.html

MAINAK CS422 49

So, this one relevant was 180 nano meter. So, in between there was molecule which was 130 galore transistor 55 million transistors on 146 milli meter square die the only thing that the change from double the size. The next one was more interesting at that is what everyone discussed that is the 90 nano meter Pentium four that is the best core and we saw in the… So, this came after Northwood and redefined the concept of deep pipelining there is a first processor designed in a history units 90 nano meter CMOS process few micro architectural enhancements over original net burst. So, there is what we are we discussed it had 125 million transistors. So, you can see that... So, are something others smaller die 112 millimeter square store buffer size is increased to 32 which was 24. So, L 1 data cache size was doubled to 16 kilo bytes and was also made 31 cycle branch penalty. So, it really had a very weak pipeline the first question of Prescott that came out had two point 8 giga hertz plus frequency and it was projected to have four gigahurtz by the end of 2004, but intel could not achieved. So, Intel stopped at 3.8 giga hertz and this is what the main marketing the name of Intel Pentium four that 's intel Pentium four you can go and read on this one I do not know which history line itself it is a old name, we can read the story why the four giga hertz Pentium. So, this once the combination of the Pentium line, I am going to discuss what happened after that, of course the first next start you understand what is enhancements, that is all.

(Refer Slide Time: 29:53)



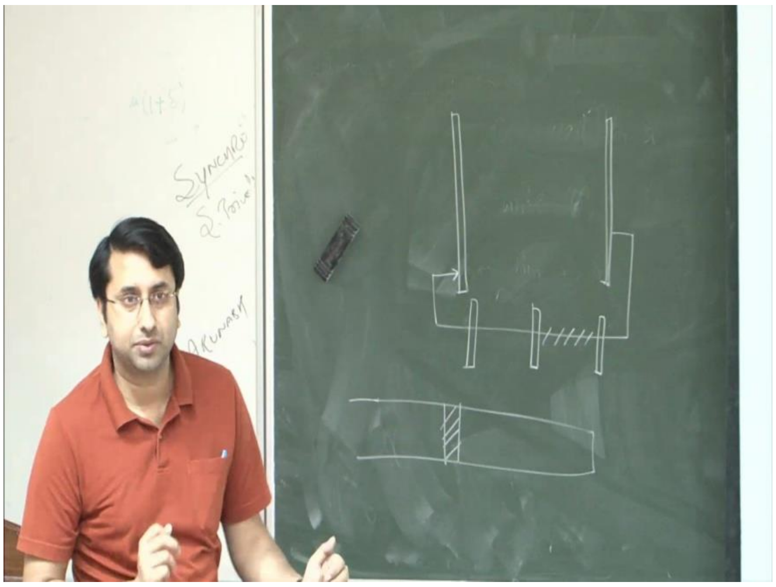
90 nm Pentium 4

- Optimized store-to-load forwarding
  - Cannot afford to compare full address of load with all stores
    - Time constraint because store forwarding logic must not have higher latency than L1 data cache
  - Speculate based on partial address comparison
  - Also initiate a full address comparison and override the previous comparison outcome if needed (load and dependents must re-execute if two outcomes disagree)
  - Added new logic to forward bytes contained fully in a store data (even if the address is misaligned): requires a shifter

MAINAK CS422 50

So, the optimized store to load forwarding. So, that found that it is impossible to have a complete address match between a load with all stores, which is needed for two partners is right. So, if you imagine simply queue of loaded store of instructions.

(Refer Slide Time: 30:18)



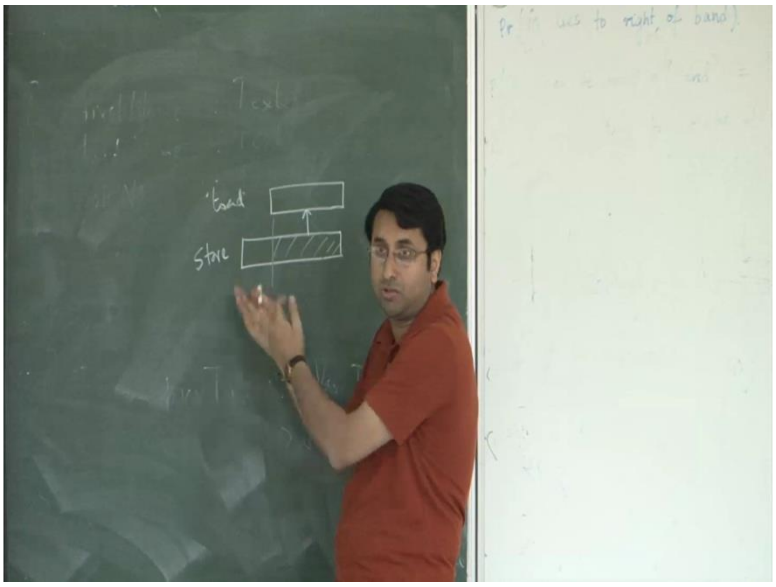
So, let us look at a load here right which has bunch of stores before and bunch of stores after all right. So, you did not to care about the stores before you, because you may have a match with the address. So, may have to get the data from here cache that is the we have discussed their alright You may have to care about yeah fine. So, load care about the stores before it for this either stores we have to care about loads after it for a right. So, when we stores the issues it has to make sure that these go around which has got a non right.

So, there fundamental question here is that failure issue in a load, whether you should get the value from a store or you should wait or it can get the value from it can get the value from a store, if the store alright, and it can if you go for sure that there is no and the data can be taken cache. So, for all this things the fundamental element that to require here is a match of address, that is once you compute the addressed load we have to compare this address with all the store addresses here alright and best was found that since your target is very that they just cannot accommodate of full address match only there. So, you cannot have cannot address of the load we can try to get the address here that is impossible.

So, at the same rows that were will be happy with the partial address match pick up or if you bits and just do the address match. So, (( )) big point here knows that the time constraint, because store forwarding logic must not have higher latency than L1 data, cache should not to be that. Of course you look at the L 1 data cache parallel with, but this should not become a. So, L 1 data cache latency and this should match, and this not should be smaller in latency full address match the latency of this particular match was actually data cache.

So, speculate at based on partial address comparison. So, if the partial address comparison matches the load should take a value from the store, which may we want actually, you are right. And other side could also may not that, if that partial address is do not right. So, you initiate ah full address comparison I am sorry yes. So, in the in the partial addresses do not match, then of course you sure that they are cannot they are cannot to be a match. So, initiate a full address comparison also and override the previous comparison outcome if needed. So, if there is there is an over override that keeps depends must needs outcome they added new logic to towards to bites contained fully in a store data even if the address is misaligned.

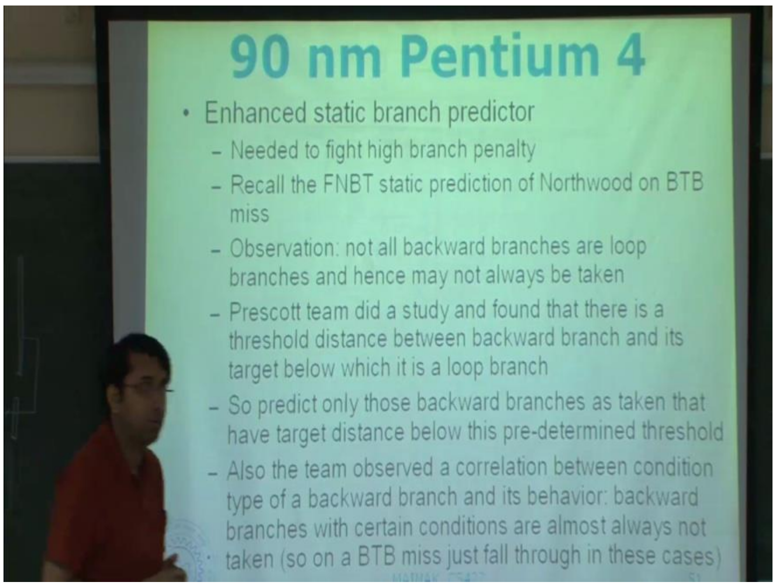
(Refer Slide Time: 34:01)



So, essentially take finally this particular case what a needs of that they would actually do a do a rotate of the board. So, that they first align the addresses, yes is in this cases essentially what will happens is that the door should get these point. So, there is actually shift the stone data to this, and take we can see the circuitries, such that it can only take the first two bites the circuitry remain are changed from your previous generation, that is there are to take leading bites of the store. So, here that came that I am changed and what the need was that actually shifted this whole data by whatever amount they want. So, that continued take this data, but now this will become this will come here alright.

So, that requires the other case that is here they talking about the situation that the third. So, that there could be two possible effects of this one is that the low could have confuse the wrong data from a store, because of a wrong partial and mismatch the other could be that the low the wrong data cache. Because there all the store which will actually pushed could not compute that the see at, but there all by passed there is a what they rolled is to here is that, it will only pickup the stores of his side which are already computed acts there could be store to each amp which are executed alright.

(Refer Slide Time: 35:37)



90 nm Pentium 4

- Enhanced static branch predictor
  - Needed to fight high branch penalty
  - Recall the FNBT static prediction of Northwood on BTB miss
  - Observation: not all backward branches are loop branches and hence may not always be taken
  - Prescott team did a study and found that there is a threshold distance between backward branch and its target below which it is a loop branch
  - So predict only those backward branches as taken that have target distance below this pre-determined threshold
  - Also the team observed a correlation between condition type of a backward branch and its behavior: backward branches with certain conditions are almost always not taken (so on a BTB miss just fall through in these cases)

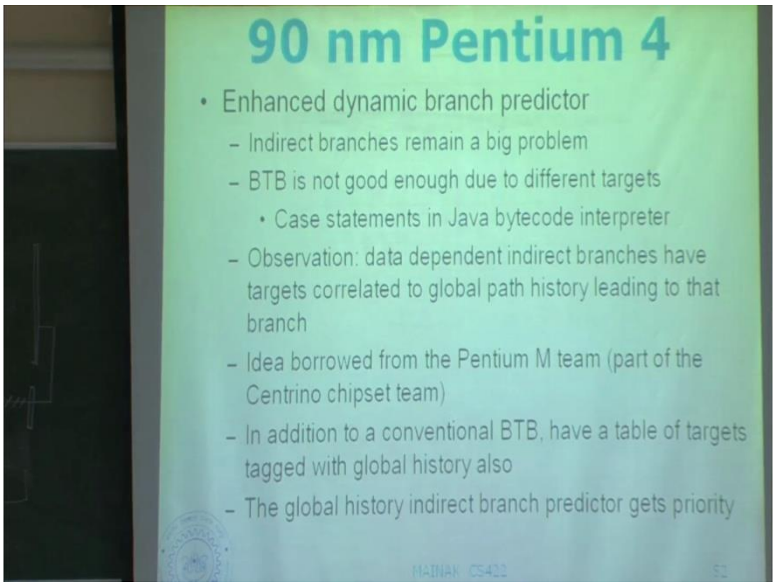
So, in most cases there would be a little execution of the load and the dependents they enhanced static branch predictor.

Student: Sir always the other the store is not executed.

Yeah, when the store issue it object all the loads behind it right. So, we recall that beats BTB the used in a static branch predictor which forward or backward technique, yes Northwood both. So, the observation that the need was that not all backward branches are loop branches and hence may not always be taken. So, I should mention here that these are not actually very frequent cases, these are mostly. And other cases were compiler optimizes certain pieces generating are backward branch which is not a loop branch alright. So, team did a studying at all that they will discuss between backward branch and its target below few exchanging branch towards says is that, if you take the distance between the target and the branch, if it is larger than a its unlikely to be which means the loop was the normal size.

So, what they concluded that you credit only the backward branch, that is taken that have target distance bellow this. So, this threshold will actually hardware inside the processor this was not pre determined also tem observed that there is a correlation between condition type of a backward branch and its behavior backward branches with certain condition are almost always not taken. So, an a BTB miss, we could just fall through in this cases. So, with this entire static there is no dynamic all you do is to you know analysis bench marks fix out the thresholds which separates these two sides you know ma maximum. And then you look at this conditions you decide statically which condition should be that which do practical which should not be and then you hardware that and do a stat explanation, that is it these are principle planning to alright.

(Refer Slide Time: 32:12)



90 nm Pentium 4

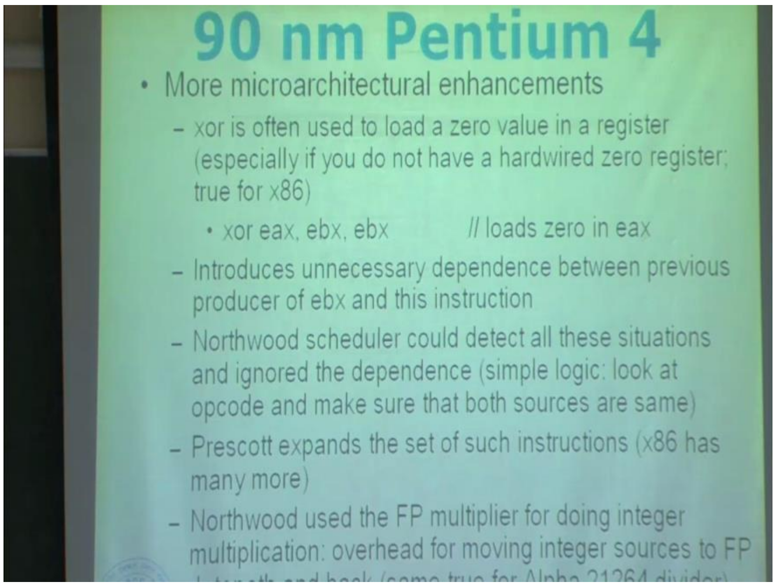
- Enhanced dynamic branch predictor
  - Indirect branches remain a big problem
  - BTB is not good enough due to different targets
    - Case statements in Java bytecode interpreter
  - Observation: data dependent indirect branches have targets correlated to global path history leading to that branch
  - Idea borrowed from the Pentium M team (part of the Centrino chipset team)
  - In addition to a conventional BTB, have a table of targets tagged with global history also
  - The global history indirect branch predictor gets priority

MAINAK CS422

So, there enhance the dynamic predictor also that particularly for engaged branches this case, because BTB is not good enough you must or homework. So, the observation that data dependent indirect branches have targets correlated to its global path history leading to that branch. So, such the forward and the hope fully we have seen that the normal BTB is what compare to this particular predictor. So, this idea was borrowed from the

Pentium team which was the part of the centrino chipset team in addition to a conventional BTB essentially they had a table of targets tagged with global history indirect branch predictor gets priority over the conventional BTB.

(Refer Slide Time: 39:02)



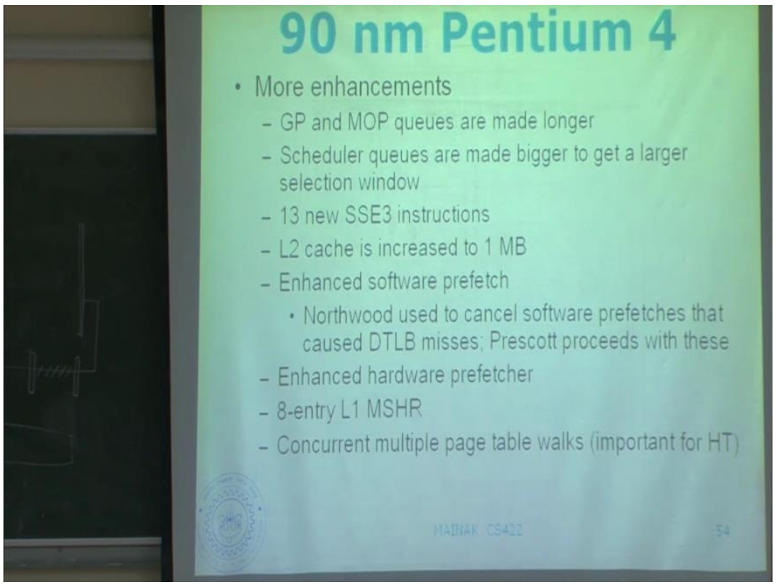
90 nm Pentium 4

- More microarchitectural enhancements
  - xor is often used to load a zero value in a register (especially if you do not have a hardwired zero register: true for x86)
    - xor eax, ebx, ebx // loads zero in eax
  - Introduces unnecessary dependence between previous producer of ebx and this instruction
  - Northwood scheduler could detect all these situations and ignored the dependence (simple logic: look at opcode and make sure that both sources are same)
  - Prescott expands the set of such instructions (x86 has many more)
  - Northwood used the FP multiplier for doing integer multiplication: overhead for moving integer sources to FP

Some more micro architectural enhancements x or is often used to load a zero value in a register especially if you do not have a hardwired zero register likely x 86. So, you typically come across this kind of core in 86, where you are essentially loading zero e b x alright an e b x is just earning as some register which could be anything else alright, the problem is that counselor and instruction that produce e b x before it alright. So, that we made instruction which produce e b x instruction that compete e b x this instruction is also appear in to e b x, but that is not actually truth, because the is in to this. So, it introduces unnecessary dependence between previous producer of e b x and this instruction the point here is that this instruction will probably be held up and the instruction producing will be x and executed it just make new sense, there is actually independence problem.

So, when we compare the... So, it just should it just use e b x same problem repeats and whatever register rules, you are setting up a different exchange. So, the Northwood scheduler could actually detect, all these situations that ignored the dependence. So, the logic was very simple they had taken up such opcodes, which could be and you just look at the instructions locates the table, and make sure that the both sources are same that one who is medico, it is not wording tough the pay is not expense to sell of such instructions. So, x 86 has many more such instructions o, where you could do such interesting things, where there is no discuss such the Northwood also used the fourteen not multiply for doing integer modification that is had an overhead for moving integer sources to FP data path and back which was same true alpha 2 1 2 6 4 divide. So, last time discussed about this which say that 264 divided. So, Prescott implements a dedicated integer multiplier.

(Refer Slide Time: 41:25)



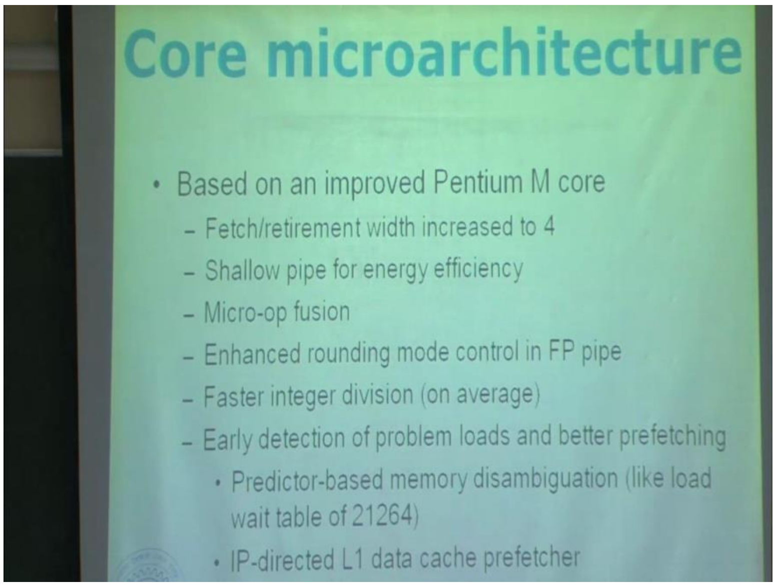
90 nm Pentium 4

- More enhancements
  - GP and MOP queues are made longer
  - Scheduler queues are made bigger to get a larger selection window
  - 13 new SSE3 instructions
  - L2 cache is increased to 1 MB
  - Enhanced software prefetch
    - Northwood used to cancel software prefetches that caused DTLB misses; Prescott proceeds with these
  - Enhanced hardware prefetcher
  - 8-entry L1 MSHR
  - Concurrent multiple page table walks (important for HT)

MAINAK CS422 54

Some more enhancements the GP, and MOP queues are made longer, scheduler queues are made bigger to get a larger selection window, 13 new SSE3 instructions, L2 cache is increased to one mega byte enhance software prefetch Northwood used to cancel software prefetches, that caused DTLB misses Prescott proceeds with these enhance hardware prefetcher 80 entry L1 MSHR concurrent multiple page table walks important for hyper discuss next week.

(Refer Slide Time: 42:00)



Core microarchitecture

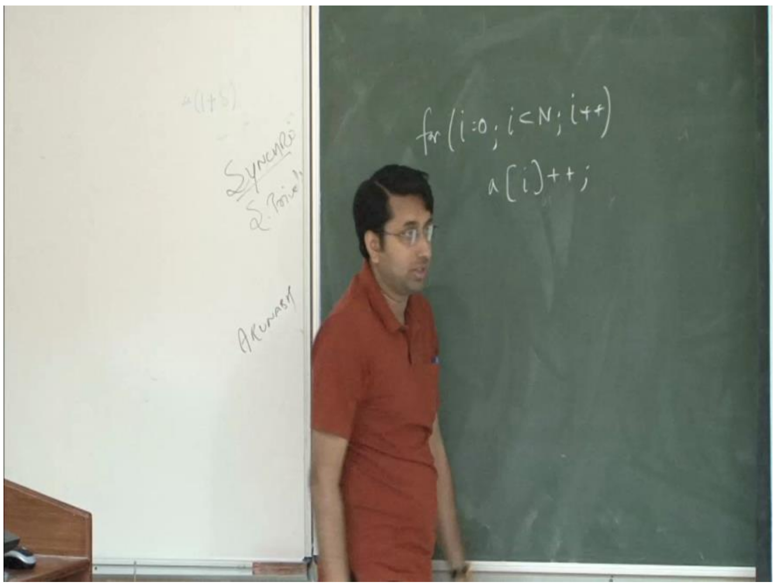
- Based on an improved Pentium M core
  - Fetch/retirement width increased to 4
  - Shallow pipe for energy efficiency
  - Micro-op fusion
  - Enhanced rounding mode control in FP pipe
  - Faster integer division (on average)
  - Early detection of problem loads and better prefetching
    - Predictor-based memory disambiguation (like load wait table of 21264)
    - IP-directed L1 data cache prefetcher

What happened? So, was the Pentium plan h that formulated Intel essentially took their Pentium m core which was designed for one devices, and the improve it to be used in the desktop, and this was called the improve architecture was called a core micro architecture based on which you have all your processor starting from today the core that use this basic macro architecture as such nothing much changed this fundamental processor some numbers changed fetch retirement width increased from 3 to 4. So, now gets matches and core processor the pipeline was met shallow you say an energy. So, Prescott that you on plus stage pipeline a much more pipeline actually introduced something called micro, these are very interesting. So, what was that?

So, that there actually two types of macro op fusion micro op fusion. So, macro op fusion actually takes multiple x 86 instructions, and fuses them in to a sink operation at all time. So, instruct and micro op fusion, what it does is that it takes the micro ops of an instruction and increases them to reduce an about micro ops in certain pipeline. So, that that actually saves the lot of energy, and also save execution type enhanced rounding mode control in point pipes of actually well actually say that this one actually done, because they did not want to change that report, because enhance the to actually not generate sonly extract this platform.

After that important we put this optimizer which actually examine the micro op fusion that remote control and modified faster integer division on average. So, these two are very in that actually where we take very high performance boost in that processor early detection of problem loads. So, this one instruct you about loads that bypass load and later they caught, because that was a mistake. So, it seen in 2 1 2 6 4 that they use a low wait table to remember such loads right, which has cost problem that passed. So, the whole micro architecture actually to on such predictor a similar such predictor which tries to identify the loads that continuously calls such problems, they do not actually issue that step they wait until all us stores before it has completed they also incorporated at L 1 data cache prfetcher, which is instruction point will be directed. So, this one is very interesting what the was that that actually this have already will observing that it will take a program. And look at a particular load instructions, you find that the data addresses that the load instruction generates a highly correlated they require.

(Refer Slide Time: 45:34)



So, I am what I am talking about is that. So, this one will be booked down into a load than a store and it apart with an addition operation alright. So, if we look at the addresses that this load instruction generates that will be right. So, and this is just one instruction that will monitor, that is exactly what they did they essentially make in a team where for each loads to instructions instruction address, there will be a to actually find out this in this correlation. So, I want arithmetic progression. So, the advantage of that instruction point directed prefetcher exist this prefetcher exactly it keeps you lot of complex information just for single program, you completely compares the whole pattern instead of actually looking at bunch of streams and loading each of the streams separately, that is what the which as a. So, those stream primates steel there, because all the for caching other complicated patterns the cache until work well very instructions.