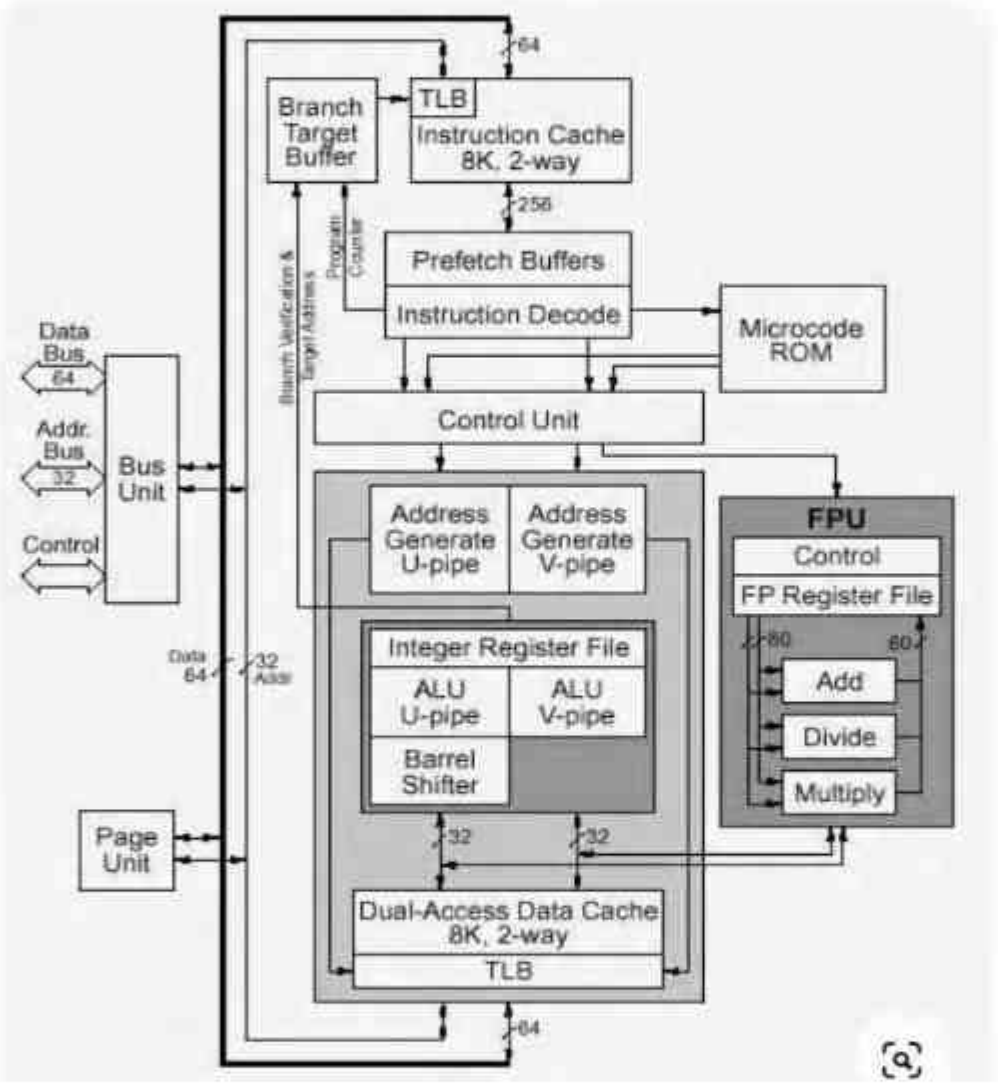
Academic Year: 2022-23 Semester: IV

Class/Branch: SE Subject: MP

Subject: Microprocessor **Semester: IV**

P5 Micro architecture/Superscalar Architecture



Academic Year: 2022-23 **Semester: IV**
Class/Branch: SE **Subject: MP**

To understand any processor architecture, trace the path of Fetch, Decode & Execute.

Integer Pipeline

① Fetch (1st stage of pipelining)

Where are the instructions fetched from?

Normally, instructions are stored and fetched from mdy But in Pentium recently used instructions are stored inside the processor in an L1 cache.

Size of L1 cache = 8 KB

8 KB = 8000 Bytes of most recently used instructions & 8000 Bytes of most recently used data is present inside the processor itself

What are the odds of fetching an instruction (1 out of last 8000 used bytes)? Very High

Principle of temporal locality → You tend to access locations of the mly which you have accessed more recently as compared to the ones which you have not accessed since a long time.

So the first attempt to fetch an instruction will be from L1 code cache. If it is found in cache (Cache hit) ← 95% it is taken from cache.

If it is not found in the cache (Cache Miss), a request will be made to the bus interface unit to get the instruction from the external memory.

If the instruction is fetched from the external mdy, then it is first stored in the Cache, so that for the next attempt it can be accessed from the cache itself.

From code cache, instruction is fetched into the Prefetch Queue.

Size of the prefetch queue = 32 bytes.

8086 → Biggest instruction → 6 bytes ∴ Prefetch Queue → 6 Bytes

Pentium → Maximum size of an instruction → 15 Bytes

In Pentium, there are 2 prefetch queues of 32 bytes.

Note:- Only one of the 2 queues are active at a time. So that single queue has to feed 2 pipes. So it should be able to hold 2 full instructions at any given point of time. Since the biggest instruction is 15 bytes, the queue should be atleast 30 bytes.

Nearest power of 2 ⇒ 32 bytes.

No of cycles required to fetch 30 bytes & 32 bytes are same. So if queue is 32 bytes, it is possible that we may fetch 2 extra (1 byte instructions) instructions.

From L1 cache, instructions come into the prefetch queue using 256 bits bus.

$\frac{256}{8} = 32$ bytes

So even if the whole queue has to be flushed (in case of branching) and refilled, then it can be refilled in 1 cycle (1 t-state).

Here comes the advantage of having an on-chip cache. If this cache was outside the chip, there is no way we could have made a 256 bit bus because external buses are physical lines & 256 bit bus nearly needs 25 cm ie to carry data only and we want smaller computers.

Inside the processor, everything is microscopic. We can even make a 1000 bit bus in nanometers space.

(2) Decode I (Second Stage of pipelining)

From prefetch queue, instruction will go to the instruction decoder. 2 instructions can be decoded in one cycle. While 2 instructions are getting decoded, 2 or more instructions are fetched in the queue.

Instructions in Pentium are categorised as < Simple instruction / Complex instruction

Simple → MOV, ADD, SUB, INC, DEC, AND, OR, XOR

Complex → MUL, DIV, String instructions etc.

The instruction decoder is made using Hardwired method

U-coded ROM is made using Micro Program Method

Note - 2 types of control units - Hardwired & Microprogrammed

Hardwired	Microprogrammed
Fast	Slower
Good for simple instructions	Good for complex instruction

→ So simple instructions are directly decoded by hardwired decoder and will be passed to control unit and to U-pipe and V-pipe for the further stages.

→ Complex instruction will go to the U-coded ROM from where it will go either to U-pipe or V-pipe

(3) Decode 2 | Physical Address Translation (3rd stage of pipelining)

Instructions will go in U-pipe and V-pipe for physical address generation of the operands (data).

Note - Physical address will not be sent on the bus to get the data from external mly. Those are those days!
This data will be fetched from L1 data cache.

from L1 cache, data will be released for U-pipe and V-pipe through the (32 bit + 32 bit) 64 bit data bus.

(4) Execute (4th stage of pipelining)

For ALU to do execution, it needs operands. These operands can either come from memory (data cache) or from GPRs

Note - In a 2 operand instruction, operands can be
- → Reg, Reg
- → Reg, Mem
- → Mem, Mem

Since there can never be 2 mly operands in an instruction, requirement of data will be only 32 bits for one pipe

Barrel Shifter - U pipe has a barrel shifter
V pipe does not have a shifter

Means U pipe can do operations which need shifter like Rotate & Shift, Mul, Div etc.

(5) Store/Write Back (5th stage of pipelining)

Result of instructions are always stored in registers.
ie. Source → Memory operand or Register
Destination → Registers always

* After the first 5 T-states are over, in every T-state, 2 instructions will get completed

		T1	T2	T3	T4	T5		
I_1	u pipe	PF	D1	D2	EX	ST	I_1	completed in T_5
I_2	v pipe	PF	D1	D2	EX	ST	I_2	
I_3	u pipe		PF	D1	D2	EX	ST	completed in T_6
I_4	v pipe		PF	D1	D2	EX	ST	
I_5	u pipe			PF	D1	D2	EX ST	completed in T_7
I_6	v pipe			PF	D1	D2	EX ST	

Pentium works at 100 MHz (66-99 MHz)
100 MHz = 100 Million T states per second
(clock pulses)

* After the first 5 cycles, Pentium can finish 200 million instructions per second.

In case of simple instructions, they are decoded by hardwired decoder and are sent for execution in u-pipe and v-pipe.
But complex instructions are decoded by μ-code ROM.

* 2 simple instructions can be decoded at a time. But only 1 complex instruction can be decoded at a time and sent to control unit.

Control unit will dispatch the instruction to the u-pipe and v-pipe

* Instructions are not always given simultaneously to control unit. They are issued on the basis of (Instruction Pairing Rule) or Instruction Issue Algorithm.

FP unit / Floating Point Pipeline

At the decoding unit, it will come to know whether it is an integer instruction or floating point instruction

11011 ← Escape ← is the prefix for any floating point instruction.

So a floating point instruction will come to the FP unit.
FP unit has its own control unit.
FP registers are also different from integer registers.

Integer Register → 32 bit (8, 16 bits)
Floating point Registers → 32 bit, 64 bit, 80 bits

There are 8 → 80 bit registers in FP unit.
floating point unit has floating point ALUs.
Registers will provide the operands to ALUs (Add-Sub, Mul, Div unit) and the result will be stored in registers.

Again, the operands can come from mly also, but at a time only one mly operand is fetched.
But here, operand size is much bigger than 32 bits (80 bit max)
So both the buses 32 bit + 32 bit = 64 bit is used.

* When a floating point instruction is going on, an integer instruction cannot be executed, because at that time FP instruction uses all 64 bit of buses.
So if there are 2 integer units, 1 FP unit
Simultaneously,

2 Integer instruction ? → YES
1 Integer, 1 FP ? → NO
2 FP instruction ? → NO

Floating point stages:

Stage	Description
Prefetch	Identical to integer prefetch
Decode 1	Identical to integer D1 stage
Decode 2	Identical to Integer D2 stage

Academic Year: 2022-23 **Semester: IV**

Class/Branch: SE **Subject: MP**

Execution stage	Register read, mdy read or mdy write performed as required by the instruction to access an operand.
FP Execution 1 stage	Information from register or mdy is written into a FP register. Data is converted to a FP format before being loaded into the FP unit.
FP Execution 2 stage	Floating point operation is performed within FP unit.
Write FP result	FP results are rounded and the result is written to the target FP register.
Error Reporting	If an error is detected, an error reporting stage is entered where the error is reported and FP unit status word is updated.

Academic Year: 2022-23 Semester: IV
Class/Branch: SE Subject: MP

Branch Target Buffer (BTB)

Pipeline fails in case of branching. So all the instructions which are there in the queue have to be flushed.
The solution is to find in advance where there is a jump.

When do you come to know that it is a jump?
- In decoding stage.

location 1 JMP 2000
2
:
2000

In the decoding stage you come to know that it is a JMP. So give a signal to prefetcher to discard all the instructions and go to location 2000.

Here the penality is only in the prefetch queue and to fill the whole queue you need only one cycle which is negligible penality. Here JMP 200 is an unconditional jump.

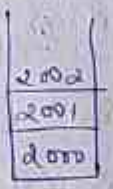
In case of JC 2000 → conditional jump → branch prediction algorithm.

location 1 JC 2000
2
:
2000

Academic Year: 2022-23 **Semester: IV**
Class/Branch: SE **Subject: MP**

When you are decoding IC 2000, in the prefetch queue, there are instructions from location 2, 3 ...

IC 2000 → 9 will jump to 2000 only if there is a carry.
So you need to prefetch.



So here the decoder has to make a prediction (95% success rate)

Programs tend to repeat their behaviour ie, if the previous time this instruction caused a jump, most likely it will jump this time also.

Consider a loop with 100 iterations

1st iteration finish count = 99 so jump
2nd " count = 98 jump
count = 97 jump

previous time jumped, so predict that it will jump again. → 2nd →

→ last iteration ⟶ count = 0 no jump

So only one (the last one) wrong prediction occurs out of 100 predictions.

In modern computing, everything in scale of MB, GB, TB etc. So one wrong prediction out of million ones doesn't matter.

So the history of previous predictions is maintained in a data structure called Branch Target Buffer (BTB).

BTB has a history of last 256 jump instructions encountered by the program.

So when processor sees JC 2000, it will go to seek history and compare this entry in the BTB. BTB will tell whether the branch was taken or not taken previously.

So accordingly, next instruction is fetched in the prefetch queue.

On the previous eg. if location 1 was JC 2000, we had 2 options. fetch the instruction from location 2 onwards or from location 2000 onwards.

Instructions from location 2 are already prefetched. So now, if it is predicted that there will be a jump, instructions from location 2 will have to be discarded and the queue will have to be refilled with instructions from location 2000 onwards.

Now, during the execution of location 1, if you come to know that the branch was not actually taken, then the instructions from 2000 onwards will have to removed from the prefetch queue and again instructions from location 2 onwards will have to be put in the queue, which was actually there in the queue earlier but was removed because of the prediction that branch will happen.

⇒ That is why, there are 2 prefetch queues. Only one queue is active at a time.

So the orginal queue is the queue which contains instruction 2 onwards.

2 predictions are possible : Branch taken
or
Branch not taken.

If the prediction is branch will not be taken, we continue with the orginal queue.

If the prediction is branch will be taken, then instructions from location 2000 is required, then we dont disturb the original queue. We just activate the new queue, put instructions 2000 onwards in that queue and make it the active queue.

Original queue then becomes the passive queue.
If during the execution, you come to know that the prediction is wrong. Then we have to discard whatever is done in the stages after decoding and that is the penality for wrong prediction. But atleast we dont have to fetch instructions again. We have to just switch back to the previous queue.

At execution time, whether the prediction is right or wrong is the latest history of the instruction and this has to be updated in BTB.

Instruction Issue Algorithm / Instruction Pairing Algorithm

Integer pipeline stages -

1. Prefetch (PF)
2. Decode1 (D1) → Instruction pairing algorithm
   → Branch Prediction.
3. Decode 2 (D2) → Segment translation
   → Page translation
   → Protection
4. Execute (EX)
5. Write Back (WB)

In the decoding phase,

If it is a simple instruction, it is decoded by hardwired decoder
it is a complex instruction, it is decoded by µ-coded ROM.

2 simple instructions can be decoded together, not 2 complex instructions.

In the architecture, after the decode stage, the u and v pipeline separation begins.

So after the decode stage, the u and v pipeline separation begins. So after the decode stage comes the decision whether 2 instructions is issued simultaneously to the u-pipe and v pipe or not.

For this, an algorithm called Instruction Issue Algorithm is used.

Algorithm:-

Consider I_1 and I_2, two consecutive instructions

If all the following rules are true:

- I_1 is a simple instruction
- I_2 is a simple instruction
- I_1 is not a branch instruction
- Destination of I_1 ≠ Source of I_2
- Destination of I_1 ≠ Destination of I_2

Then they can be paired.
Issue I_1 to u-pipe, I_2 to v-pipe

Else,
Issue I_1 to u pipe.
Hold back I_2.

On the next cycle, pair I_2 and I_3. If it satisfies the rules otherwise issue I_2 to u-pipe and hold I_3 and so on.

Academic Year: 2022-23 **Semester: IV**

Class/Branch: SE **Subject: MP**

Data dependency check -

I_1 : ADD BL, CL

I_2 : MOV DL, BL

Here Dest of I_1 = Source of I_2

So can't pair them together.

because I_2 should execute once I_1 is executed.

I_1 is updating the value of BL and if I_1 doesn't execute first the old value of BL may get stored in DL.