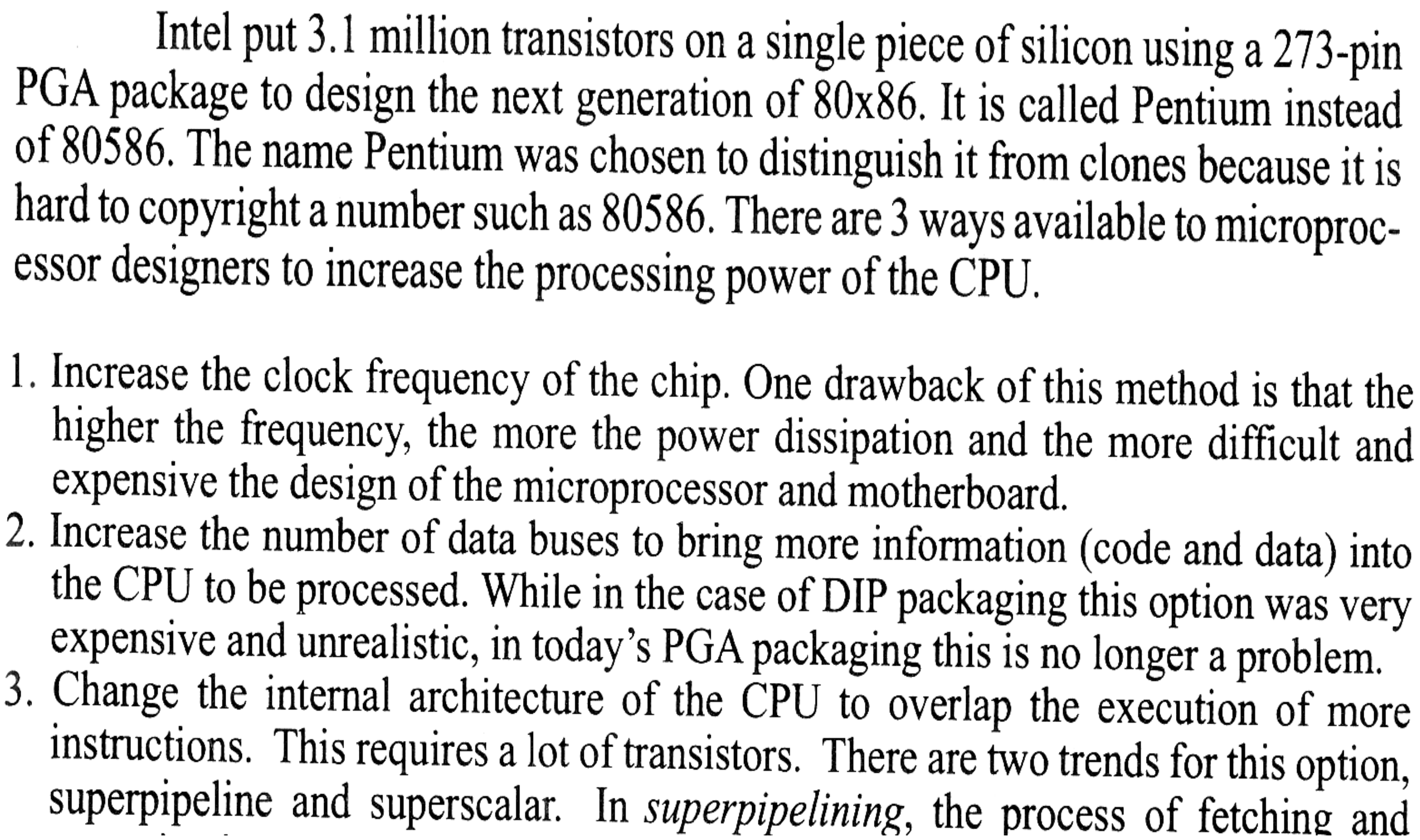
Department of Computer Engineering Techniques (Stage: 4)

Advance Computer Technologies

Dr.: Hussein Ali Ameen

hussein_awadh@mustaqbal-college.edu.iq

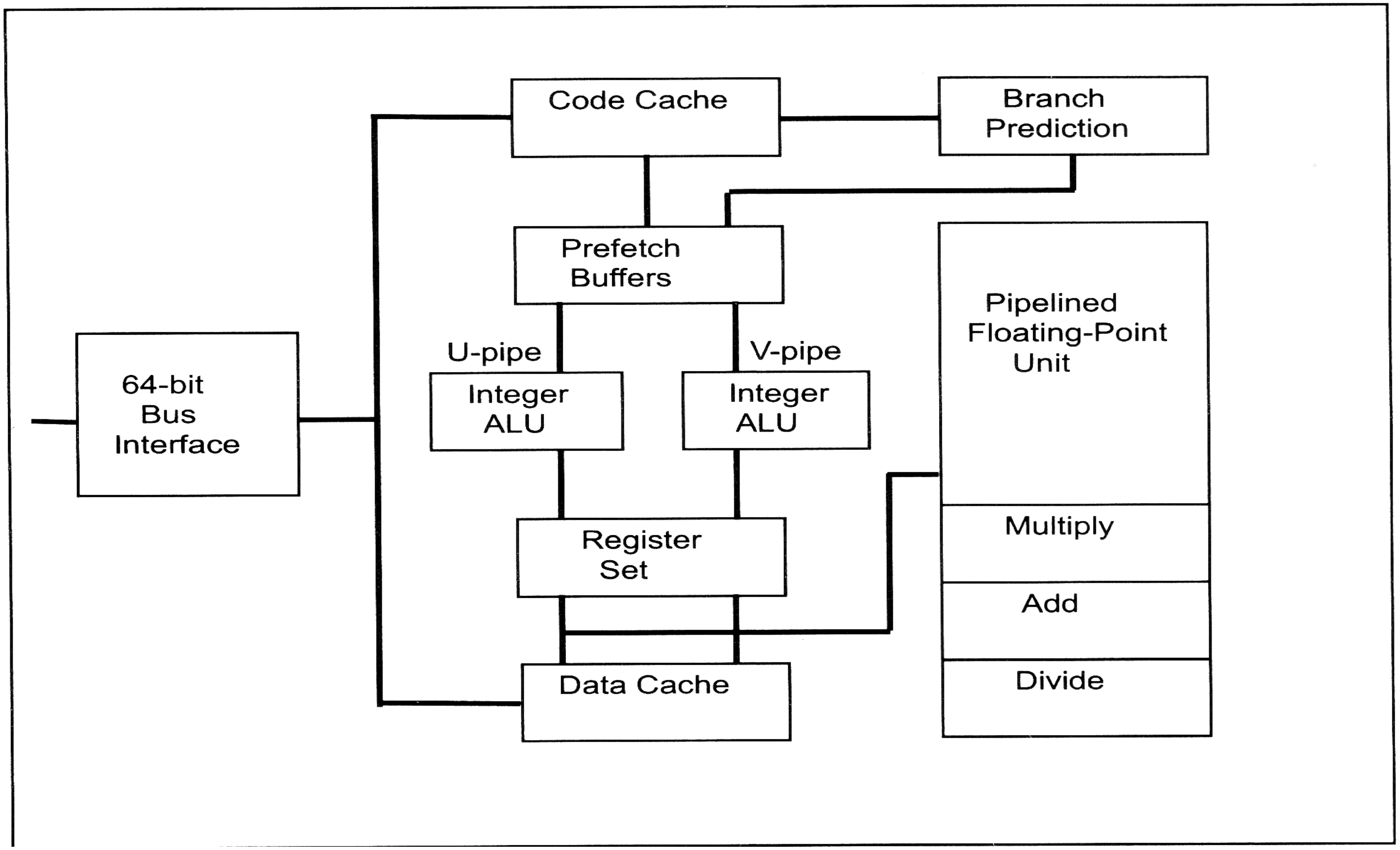
INTEL'S PENTIUM



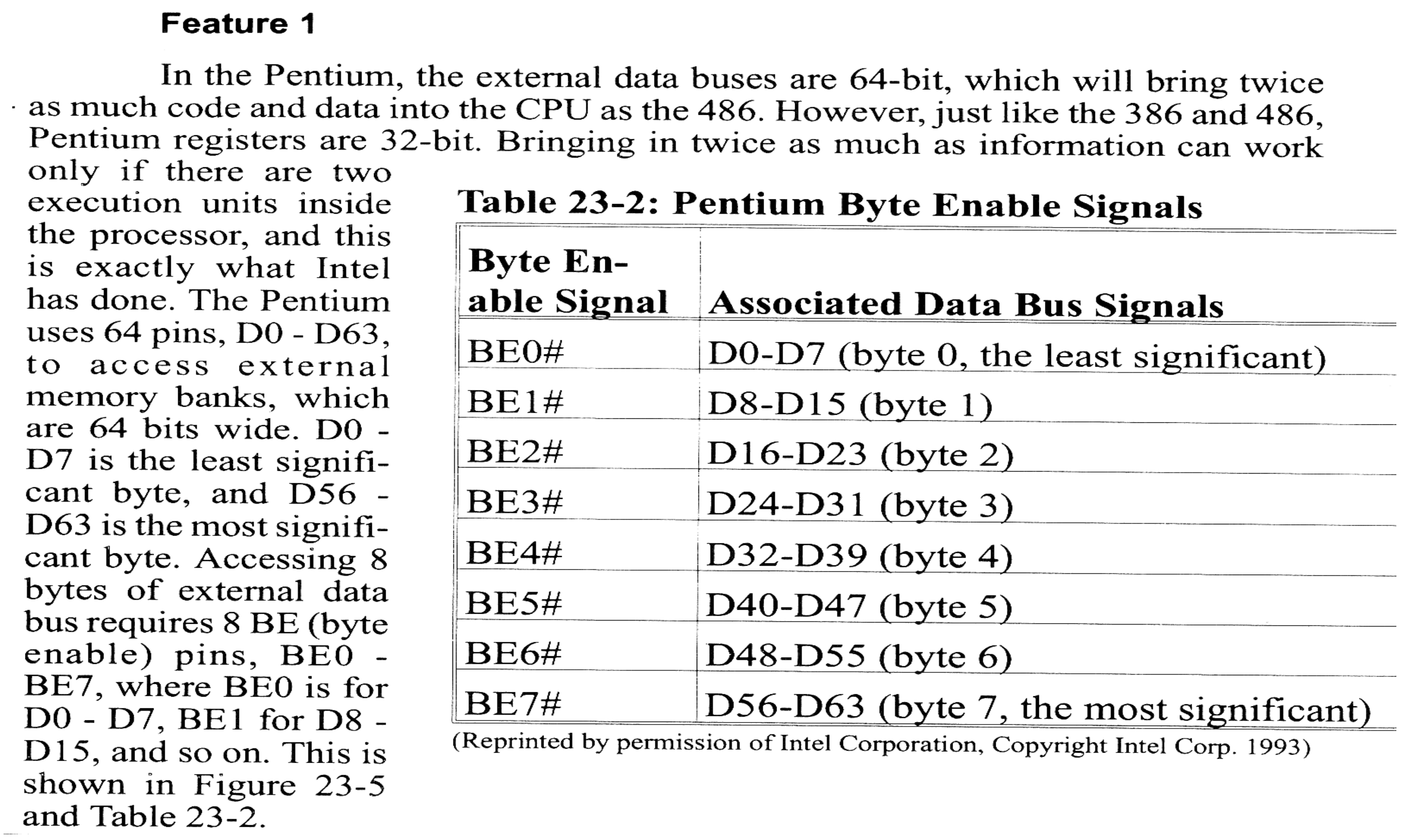
Intel put 3.1 million transistors on a single piece of silicon using a 273-pin PGA package to design the next generation of 80x86. It is called Pentium instead of 80586. The name Pentium was chosen to distinguish it from clones because it is hard to copyright a number such as 80586. There are 3 ways available to microprocessor designers to increase the processing power of the CPU.

1. Increase the clock frequency of the chip. One drawback of this method is that the higher the frequency, the more the power dissipation and the more difficult and expensive the design of the microprocessor and motherboard.
2. Increase the number of data buses to bring more information (code and data) into the CPU to be processed. While in the case of DIP packaging this option was very expensive and unrealistic, in today's PGA packaging this is no longer a problem.
3. Change the internal architecture of the CPU to overlap the execution of more instructions. This requires a lot of transistors. There are two trends for this option, superpipeline and superscalar. In *superpipelining*, the process of fetching and

INTEL’S PENTIUM



Features of the Pentium



Feature 1

In the Pentium, the external data buses are 64-bit, which will bring twice as much code and data into the CPU as the 486. However, just like the 386 and 486, Pentium registers are 32-bit. Bringing in twice as much as information can work only if there are two execution units inside the processor, and this is exactly what Intel has done. The Pentium uses 64 pins, D0 - D63, to access external memory banks, which are 64 bits wide. D0 - D7 is the least significant byte, and D56 - D63 is the most significant byte. Accessing 8 bytes of external data bus requires 8 BE (byte enable) pins, BE0 - BE7, where BE0 is for D0 - D7, BE1 for D8 - D15, and so on. This is shown in Figure 23-5 and Table 23-2.

Table 23-2: Pentium Byte Enable Signals

Byte Enable Signal	Associated Data Bus Signals
BE0#	D0-D7 (byte 0, the least significant)
BE1#	D8-D15 (byte 1)
BE2#	D16-D23 (byte 2)
BE3#	D24-D31 (byte 3)
BE4#	D32-D39 (byte 4)
BE5#	D40-D47 (byte 5)
BE6#	D48-D55 (byte 6)
BE7#	D56-D63 (byte 7, the most significant)

(Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1993)

While in the 486 there were four DP (data parity) pins, one for each of the 4 bytes of the data bus, in the Pentium there are 8 DP pins to handle the 8 bytes of data pins D0 - D63. The Pentium has A31 to A3 for the address buses. This is shown in Figure 23-6. Just like the 486, the Pentium also has the A20M (A20 Mask) input pin for the implementation of HMA (high memory area).

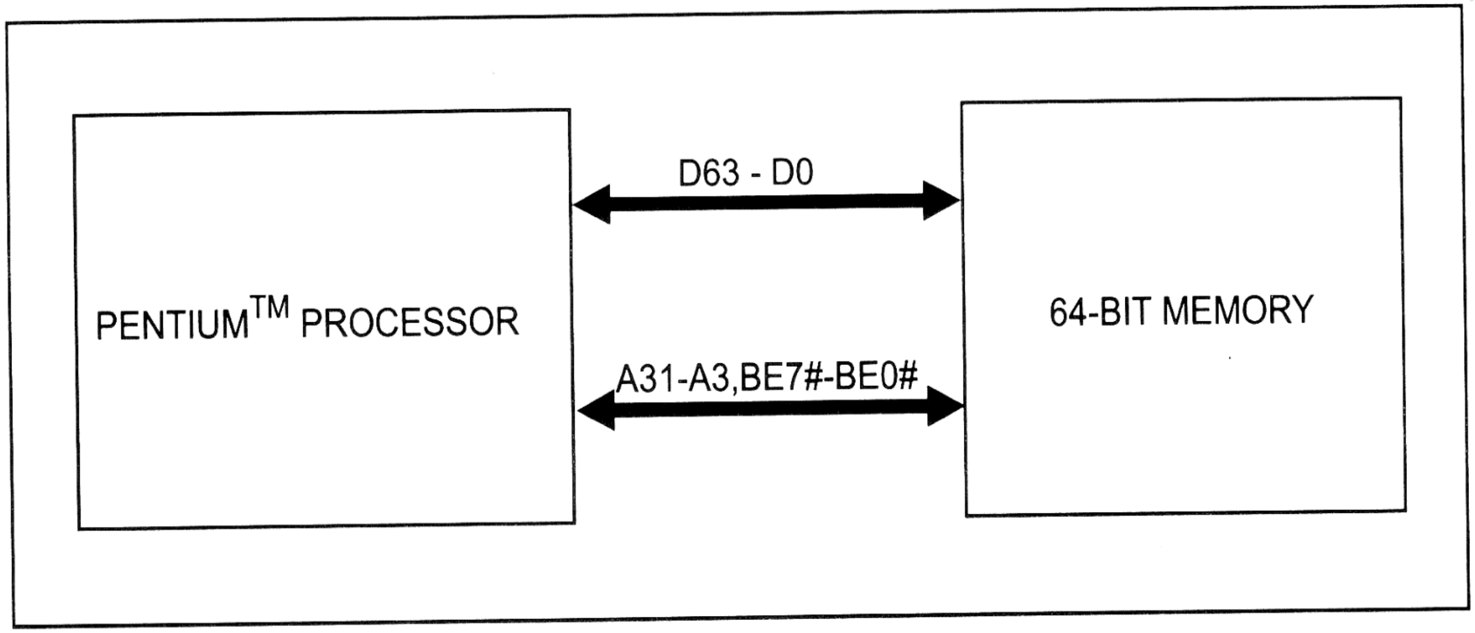


Figure 23-6. Pentium Address Buses
(Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1993)

Features of the Pentium

Feature 2

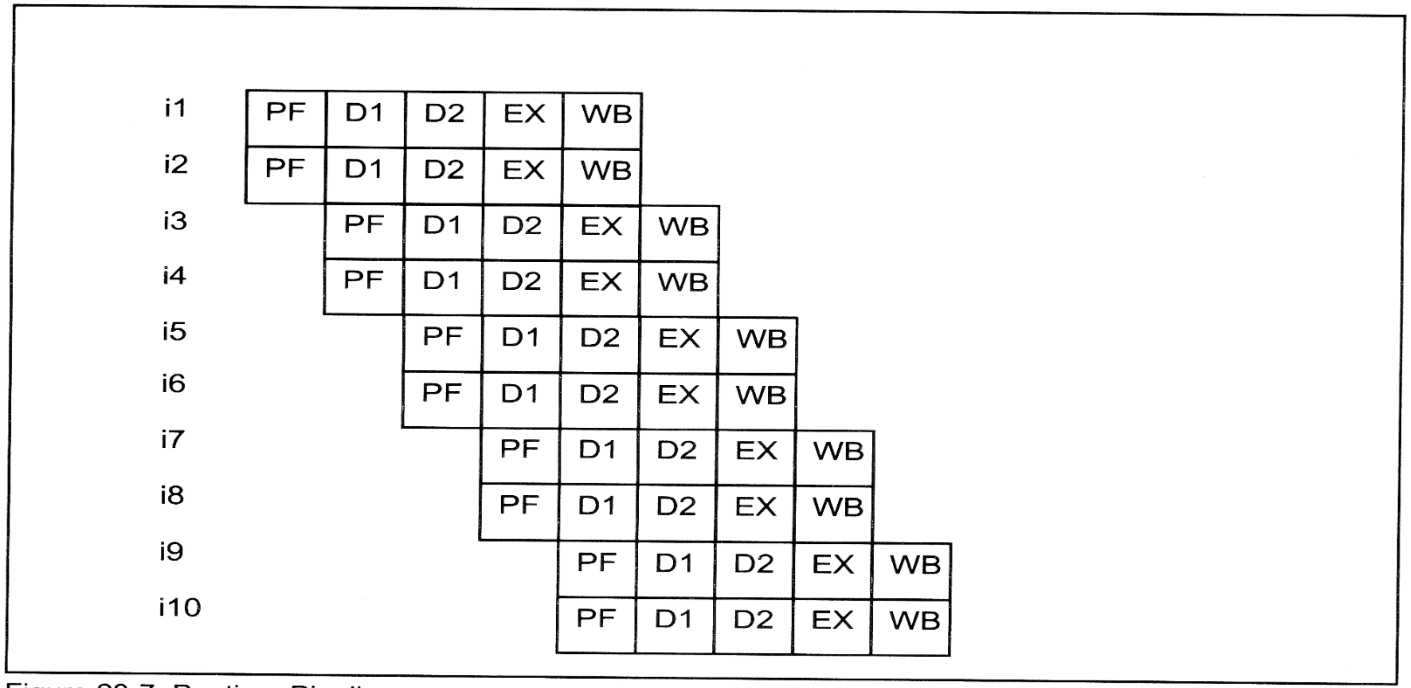
The Pentium has a total of 16K bytes of on-chip cache: 8K is for code and the other 8K is for data. In the 486 there is only 8K of on-chip cache for both code and data. The data cache can be configured as write-back or write-through, but to prevent any accidental writing into code cache, the 8K of code cache is write protected. In other words, while the CPU can read or write into the data cache, the code cache is write protected to prevent any inadvertent corruption. Of course, when there is a cache miss for code cache, the CPU brings code from external memory and stores (writes) it in the cache code, but no instruction executing in the CPU can write anything into the code cache. The replacement policy for both data and code caches is LRU (least recently used).

Both the on-chip data and code caches are accessed internally by the CPU core simultaneously. However, since there is only one set of address buses, the external cache containing both data and code must be accessed one at a time and not simultaneously. Some CPUs, notably RISC processors, use a separate set of address and data pins (buses) for the data and another set of address and data buses for the code section of the program. This is called *Harvard architecture* and will be discussed in the next section. The Pentium accesses the on-chip code and data caches simultaneously using Harvard architecture, but not the secondary (external) off-chip cache and data. The Pentium's cache organization for both the data and code caches is 2-way set associative. Each 8K is organized into 128 sets of 64 bytes, which means $2^7 \times 2^6 = 2^{13} = 8192 = 8\text{K}$ bytes. Each set consists of 2 lines of cache, and each line is 32 bytes wide.

Features of the Pentium

Feature 3

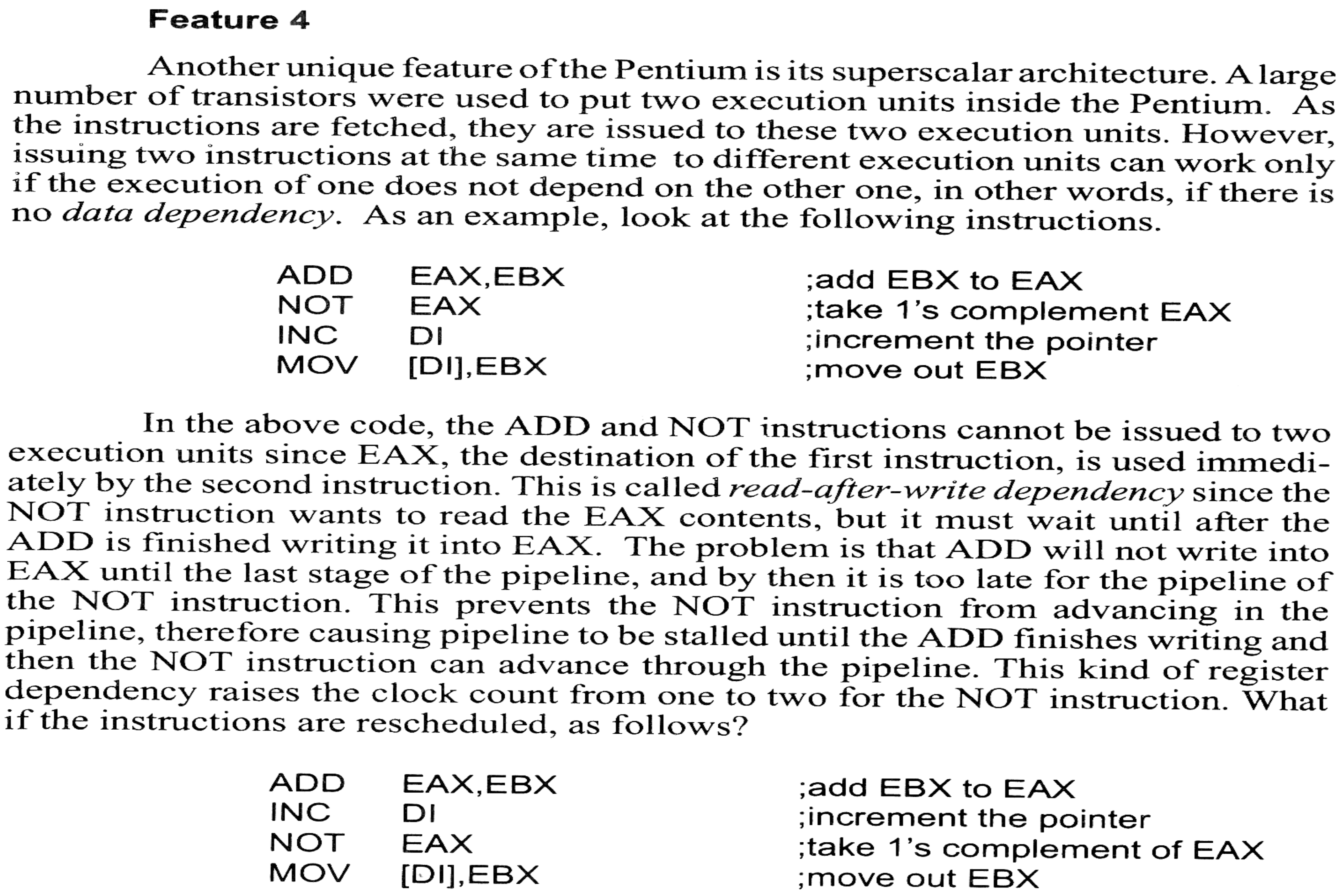
The on-chip math coprocessor of the Pentium is many times faster than the one on the 486. It has been redesigned to perform many of the instructions, such as add and multiply, ten times faster than the 486 math coprocessor. In microprocessor terminology, the on-chip math coprocessor is commonly referred to as a *floating point unit* (FPU) while the section responsible for the execution of integer-type data is called the *integer unit* (IU). The FPU section of the Pentium uses an 8-stage pipeline to process instructions, in contrast to the 5-stage pipeline in the integer unit.



i1	PF	D1	D2	EX	WB				
i2	PF	D1	D2	EX	WB				
i3		PF	D1	D2	EX	WB			
i4		PF	D1	D2	EX	WB			
i5			PF	D1	D2	EX	WB		
i6			PF	D1	D2	EX	WB		
i7				PF	D1	D2	EX	WB	
i8				PF	D1	D2	EX	WB	
i9					PF	D1	D2	EX	WB
i10					PF	D1	D2	EX	WB

Figure 23-7. Pentium Pipeline

Features of the Pentium



Feature 4

Another unique feature of the Pentium is its superscalar architecture. A large number of transistors were used to put two execution units inside the Pentium. As the instructions are fetched, they are issued to these two execution units. However, issuing two instructions at the same time to different execution units can work only if the execution of one does not depend on the other one, in other words, if there is no *data dependency*. As an example, look at the following instructions.

```
ADD     EAX,EBX                 ;add EBX to EAX
NOT     EAX                     ;take 1's complement EAX
INC     DI                      ;increment the pointer
MOV     [DI],EBX                ;move out EBX
```

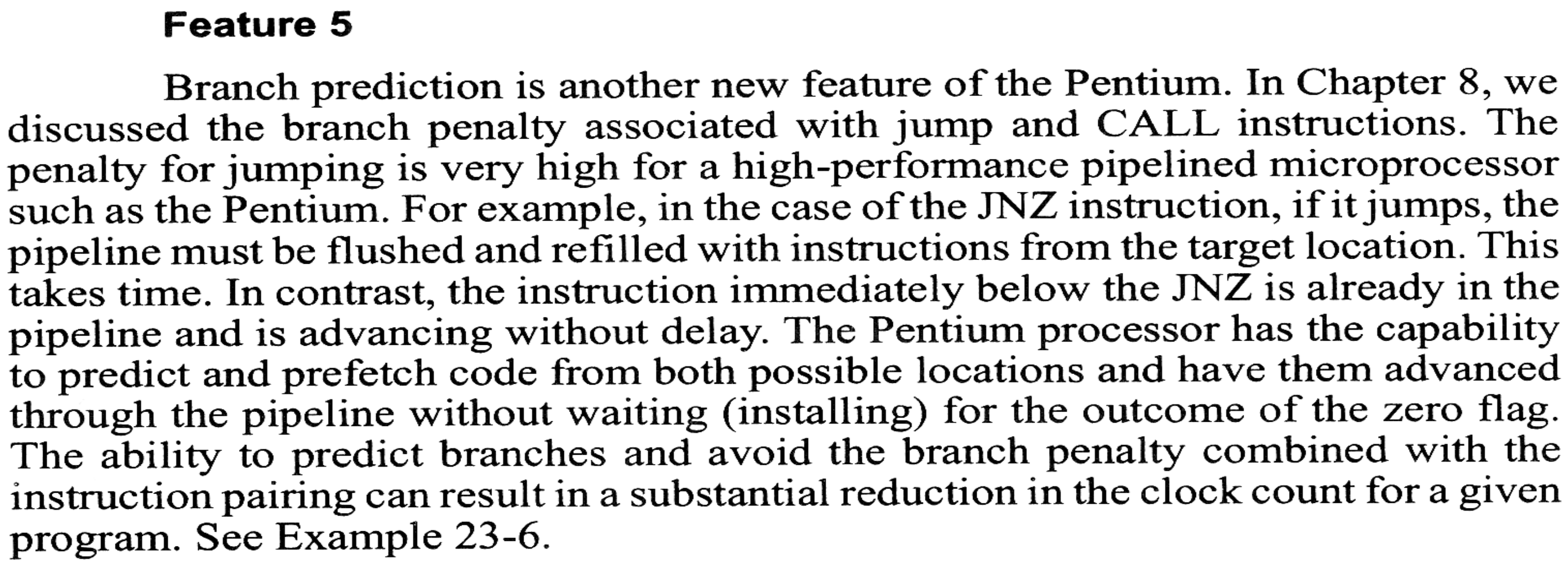
In the above code, the ADD and NOT instructions cannot be issued to two execution units since EAX, the destination of the first instruction, is used immediately by the second instruction. This is called *read-after-write dependency* since the NOT instruction wants to read the EAX contents, but it must wait until after the ADD is finished writing it into EAX. The problem is that ADD will not write into EAX until the last stage of the pipeline, and by then it is too late for the pipeline of the NOT instruction. This prevents the NOT instruction from advancing in the pipeline, therefore causing pipeline to be stalled until the ADD finishes writing and then the NOT instruction can advance through the pipeline. This kind of register dependency raises the clock count from one to two for the NOT instruction. What if the instructions are rescheduled, as follows?

```
ADD     EAX,EBX                 ;add EBX to EAX
INC     DI                      ;increment the pointer
NOT     EAX                     ;take 1's complement of EAX
MOV     [DI],EBX                ;move out EBX
```

Features of the Pentium

If they are rescheduled as shown above, each can be issued to separate execution units, allowing parallel execution of both instructions by two different units of the CPU. Since the clock count for each instruction is one, just like the 486, having two execution units leads to executing two instructions by pairing them together, thereby using only one clock count for two instructions. In the case of the above program, if it is run on the Pentium it will take only 2 clocks instead of 4 as is the case of 486 microprocessor, assuming that two instructions are paired together. This reordering of instructions to take advantage of the two internal execution units of the Pentium is the job of the compiler and is called *instruction scheduling*. Currently, compilers are being equipped to do instruction scheduling to remove dependencies. The role of the compiler to reschedule instructions in order to take advantage of the superscalar capability of the Pentium must be emphasized. The process of issuing two instructions to the two execution units is commonly referred to as *instruction pairing*. The two integer execution units of the Pentium are called "U" and "V" pipes. Each has 5 pipeline stages. While the U pipe can execute any of the instructions in the 80x86 family, the V pipe executes only simple instructions such as INC, DEC, ADD, SUB, MUL, DIV, NOT, AND, OR, EXOR, and NEG. These simple instructions are executed in one clock as long as the operands are "REG,REG" or "REG,IMM" and have no register dependency. For example, instructions such as "ADD EAX,EBX", "SUB ECX,2000", and "MOV EDX,1500" are simple instructions requiring one clock, but not "ADD DWORD PTR [EBX+EDI+500],EAX", which needs 3 clocks.

Features of the Pentium



Feature 5

Branch prediction is another new feature of the Pentium. In Chapter 8, we discussed the branch penalty associated with jump and CALL instructions. The penalty for jumping is very high for a high-performance pipelined microprocessor such as the Pentium. For example, in the case of the JNZ instruction, if it jumps, the pipeline must be flushed and refilled with instructions from the target location. This takes time. In contrast, the instruction immediately below the JNZ is already in the pipeline and is advancing without delay. The Pentium processor has the capability to predict and prefetch code from both possible locations and have them advanced through the pipeline without waiting (installing) for the outcome of the zero flag. The ability to predict branches and avoid the branch penalty combined with the instruction pairing can result in a substantial reduction in the clock count for a given program. See Example 23-6.

Features of the Pentium

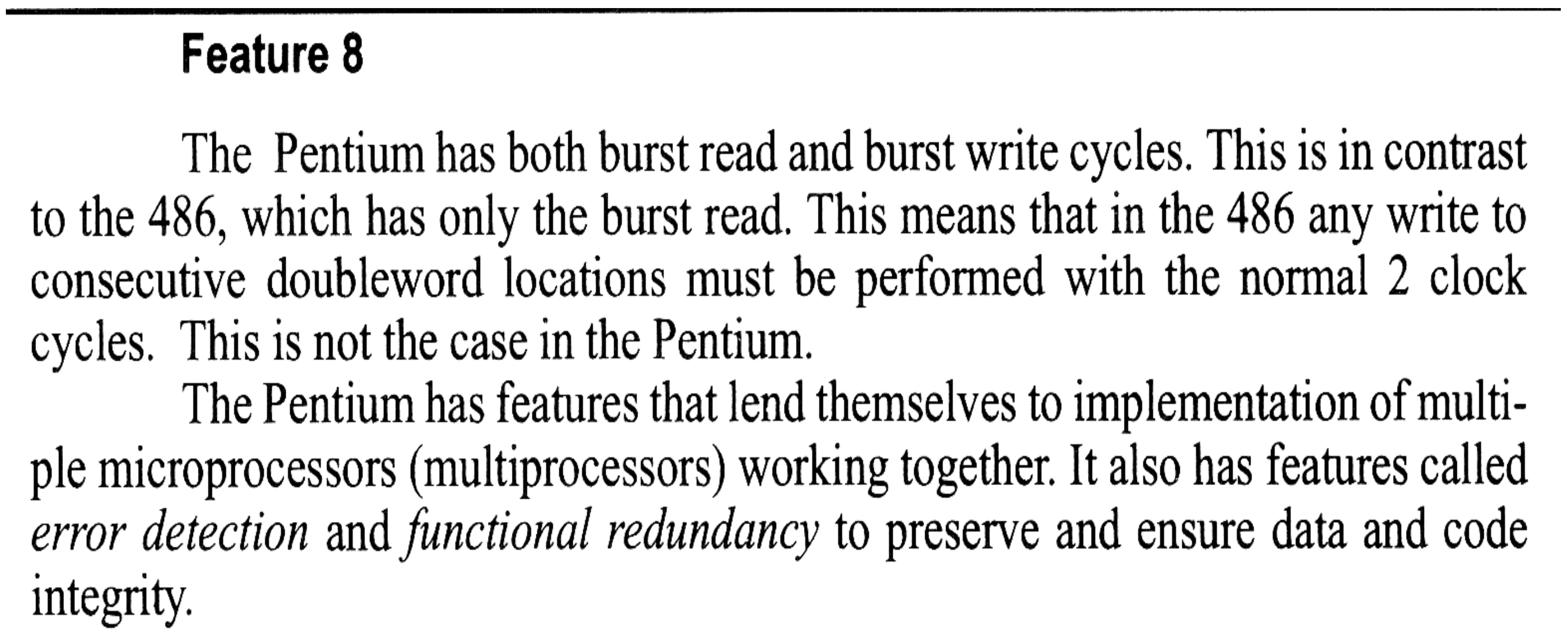
Feature 6

As discussed in Chapter 21, the 386/486 has a page size of 4K for page virtual memory. The Pentium provides the option of 4K or 4M for the page size. The 4K page option makes it 386 and 486 compatible, while the 4M page size option allows mapping of a large program without any fragmentation. The 4M page size in the Pentium reduces the frequency of a page miss in virtual memory.

Feature 7

As discussed in Chapter 21, the 386 (and 486) has only 32 entries for the TLB (translation lookaside buffer), which means that the CPU has instant knowledge of the whereabouts of only 128K of code and data. If the desired code or data is not referenced in the TLB, the CPU must go through the long process of converting the linear address to a physical address. The Pentium has two sets of TLB, one for code and one for data. For data, the TLB has 64 entries for 4K pages. This means that the CPU has quick access to 256K (64 × 4K =256K) of data. The TLB for the code is 32 entries of 4K page size. Therefore, the CPU has quick access to 128K of code at any give time. Combining the TLBs for the code and data, the Pentium has quick access to 384K (128 +256) of code and data before it resorts to updating the TLB for the page miss. Contrast this to 128K for the 486. If the page size of 4M is chosen, the TLB for the data has 8 entries while the TLB for the code has 32 entries.

Features of the Pentium

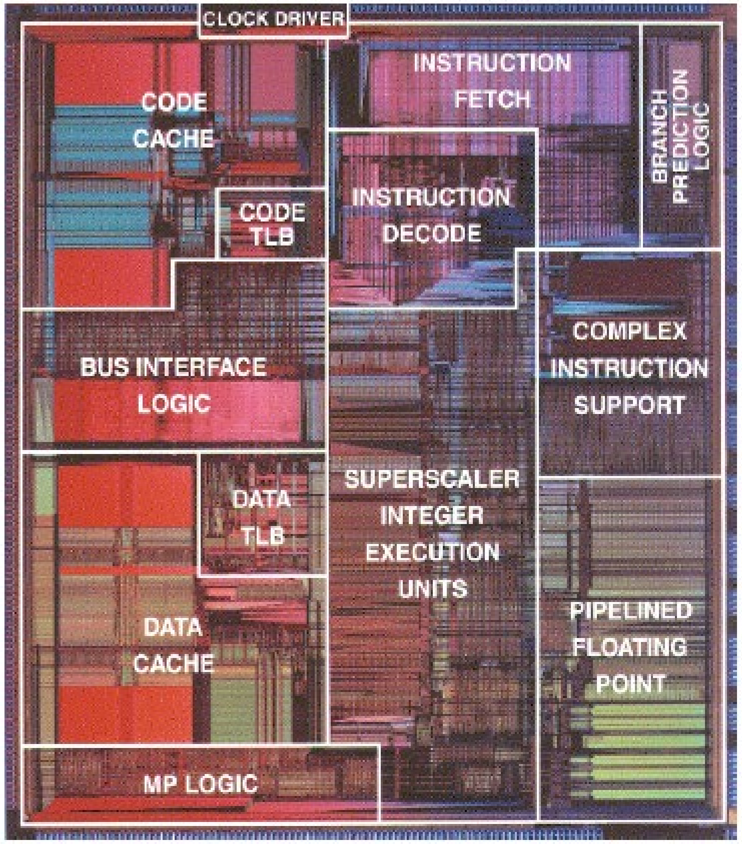


Feature 8

The Pentium has both burst read and burst write cycles. This is in contrast to the 486, which has only the burst read. This means that in the 486 any write to consecutive doubleword locations must be performed with the normal 2 clock cycles. This is not the case in the Pentium.

The Pentium has features that lend themselves to implementation of multiple microprocessors (multiprocessors) working together. It also has features called *error detection* and *functional redundancy* to preserve and ensure data and code integrity.

Pentium Die Photo



- 3,100,000 transistors
- 296 mm^2
- 60 MHz
- Introduced in 1993
  - 1st superscalar implementation of IA32

Intel’s overdrive technology

To increase both the internal and external clock frequency of the CPU requires faster DRAM, high-speed motherboard design, high-speed peripherals, and efficient power management due to a high level of power dissipation. As a result, the system is much more expensive. To solve this problem, Intel came up with what is called *overdrive technology*, also referred to as *clock doubler* and *tripler*. The idea of a clock doubler or tripler is to increase the internal frequency of the CPU while the external frequency remains the same. In this way, the CPU processes code and data internally faster while the motherboard costs remain the same. For example, the 486DX2-50 uses the internal frequency of 50 MHz but the external frequency by which the CPU communicates with memory and peripherals is only 25 MHz. This allows the instructions stored in the queue of 486 to be executed at twice the speed of fetching them from the system buses. With the advent of the 32- and 64-bit

external buses, on-chip cache, and the burst cycle reading (reading 16 bytes in only 5 clocks), the amount of code and data fetched into the queue of the CPU is sufficient to keep the execution unit of CPU busy even if it is working with twice or three times the speed of external buses. This is the reason that Intel is designing processors with clock triplers. In that case, if the CPU’s external buses are working at the speed of 33 MHz, the CPU works at 99 MHz speed. The design of a system board of 33 MHz costs much less than that of a 100-MHz system board. With slower memory and peripherals one can get instruction throughput of three times the bus throughput. As designers move to wider data buses, such as 128-bit-wide buses, the use of clock doublers and triplers is one way of keeping the system board cost down without sacrificing system throughput. The Intel 486DX4 is an example of a clock-tripler CPU. Note that "X4" does not mean that the external frequency is 4 times the internal frequency.

Pentium Pro is both superpipelined and superscalar

As mentioned above, in the Pentium Pro all x86 instructions are converted into micro-ops with triadic formats before they are processed. This conversion allows an increase in the pipeline stages with little difficulty. Intel uses a 12-stage pipeline for the Pentium Pro. In contrast to the 5-pipestage of the Pentium, although each pipestage of the 12-pipestage Pentium Pro performs less work, there are more stages. This means that in the Pentium Pro, more instructions can be worked on and finished at a time. The Pentium Pro with its 12-stage pipeline is referred to as *superpipelined.* Since it also has multiple execution units capable of working in parallel, it is also *superscalar*. Another advantage of the 12-pipestage is that it can achieve a higher clock rate (frequency) with the given transistor technology. This is one reason that the earliest Pentium chips had a frequency of only 60 MHz while the earliest Pentium Pro has a frequency of 150 MHz. Intel also used what is called *out-of-order execution* to increase the performance of the Pentium Pro. This is explained next.

What is out-of-order execution?

In Pentium architecture, when one of the pipeline stages is stalled, the prior stages of fetch and decode are also stalled. In other words, the fetch stage stops fetching instructions if the execution stage is stalled, due for example to a delay in memory access. This dependency of fetch and execution has to be resolved in order to increase CPU performance. That is exactly what Intel has done with the Pentium Pro and is called *decoupling* the fetch and execution phases of the instructions. In the Pentium Pro, as x86 instructions are fetched from memory they are decoded (converted) into a series of micro-ops, or RISC-type instructions, and placed into a pool called the instruction pool. See Figure 23-9. This fetch/decode of the instructions is done in the same order as the program was coded by the programmer (or compiler). However, when the micro-ops are placed in the instruction pool they can be executed in any order as long as the data needed is available. In other words, if there is no dependency, the instructions are executed out of order, not in the same order as the programmer coded them. In the case of the Pentium Pro, the dispatch/execute unit schedules the execution of micro-ops from the instruction pool subject to the availability of needed resources and stores the results temporarily. Such a speculative execution can go 20 - 30 instructions deep into the program. It is the job of the *retire unit* to provide the results to the programmer's (visible) registers (e.g., EAX, EBX) according to the order the instructions were coded. Again, it is important to note that the instructions are fetched in the same order that they were coded, but executed out of order if there is no dependency, but ultimately retired in the same order as they were coded. This out-of-order execution can boost performance in many cases. Look at Example 23-8.

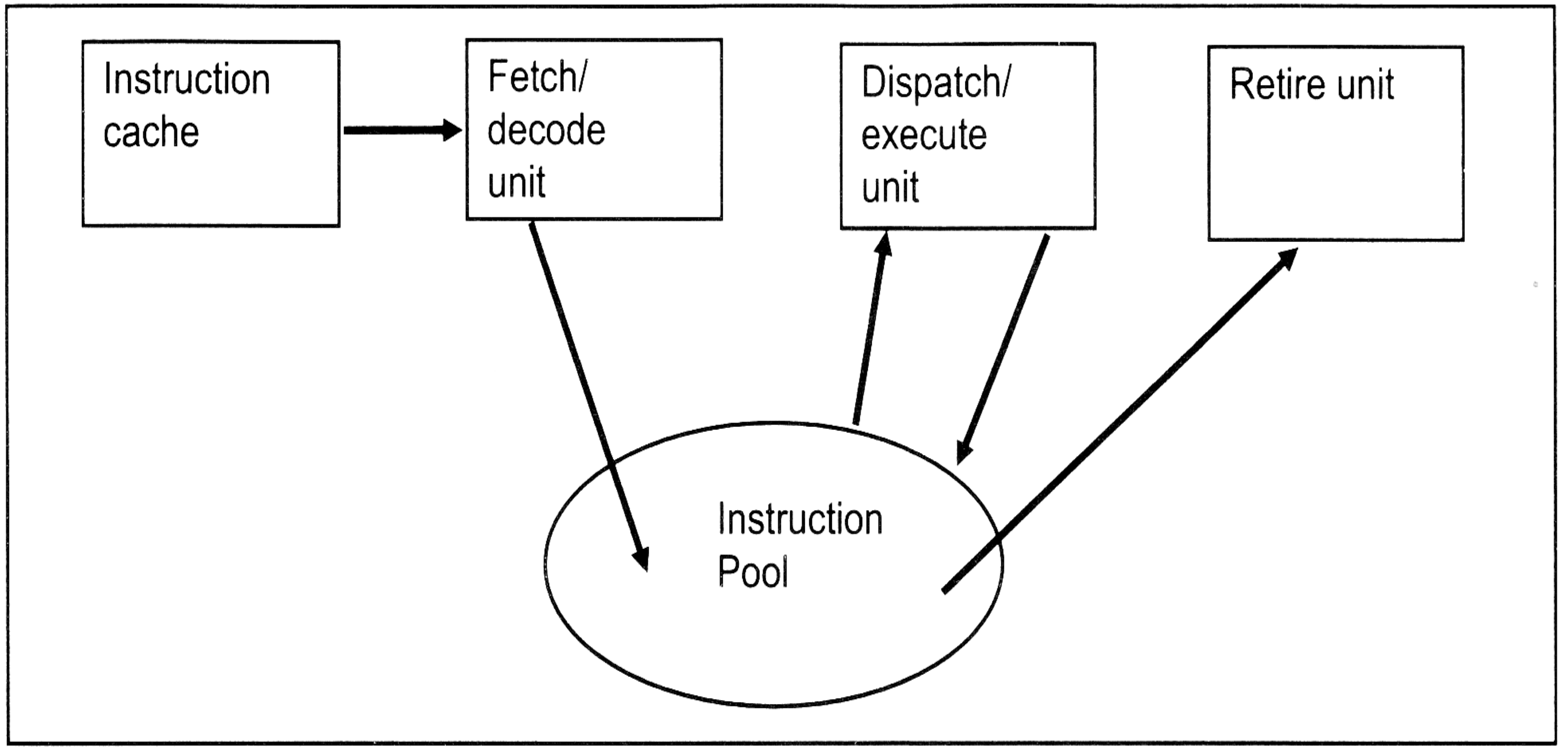


Figure 23-9. Pentium Pro Instruction Execution

performance in many cases. Look at Example 23-8.

Example 23-8

For the following code, indicate the instructions that can be executed out of order in the Pentium Pro.

```
i1) LOAD (R2), R4        ;LOAD R4 FROM MEMORY POINTED AT BY R2
i2) ADD  R3,R4,R7        ;R3+R4--->R7
i3) ADD  R6,R8,R10       ;R6+R8--->R10
i4) SUB  R5,R1,R9        ;R5-R1--->R9
i5) ADD  R6,1,R12        ;R6+1---->R12
```

Solution:

Instruction i2 cannot be executed until the data is brought in from memory (either cache or main memory DRAM). Therefore, i2 is dependent on i1 and must wait until the R4 register has the data. However, instructions i3, i4 and i5 can be executed out of order and in parallel with each other since there is no dependency among them. After the execution of i2, all the instructions i2, i3, i4, and i5 can be retired instantly since they all have been executed already. This would not be the case if these instructions were executed in the Pentium since its pipeline would be stalled due to the memory access for R4. In that case, instructions i3, i4, and i5 could not even be fetched let alone decoded and executed.

Table 23-4: Comparison of Pentium and Pentium Pro

Feature	Pentium	Pentium Pro
Year introduced	1993	1995
Number of transistors	3.3 million	5.5 million
Number of pins	273	387
External data bus	64 bits	64 bits
Address bus	32 bits	36 bits
Physical memory (maximum)	4 GB	64 GB
Virtual memory	64 TB	64 TB
Data types (register sizes)	8, 16, 32 bits	8, 16, 32 bits
Cache (L1)	16K bytes (data 8K, code 8K)	16K bytes (data 8K, code 8K)
Cache (L2)	External	256KB/512KB
Superscalar	2-Way	3-Way
Number of execution units	3	5
Branch prediction	yes	yes
Out-of-order execution	no	yes

Example 23-9

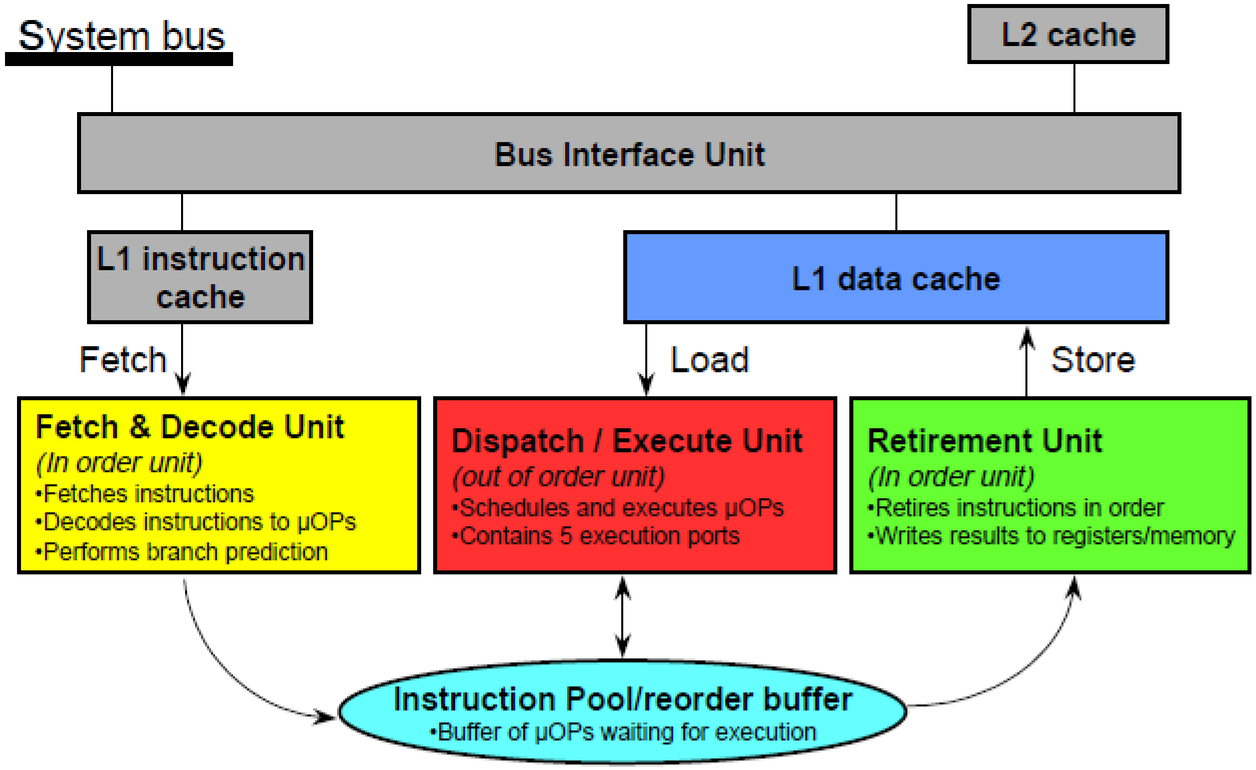
The following x86 code (a) sets the pointer for three different arrays, and the counter value, (b) gets each element of ARRAY_1, adds a fixed value of 100 to it, and stores the result in ARRAY_2, (c) complements the element and stores it in ARRAY_3. Analyze the execution of the code in light of the out-of-order execution and branch prediction capabilities of the Pentium Pro.

```
i1)                 MOV  EBX,ARRAY_1 ;LOAD POINTER
i2)                 MOV  ESI,ARRAY_2 ;LOAD POINTER
i3)                 MOV  EDI,ARRAY_3 ;LOAD POINTER
i4)                 MOV  ECX,COUNT   ;LOAD COUNTER
i5)  AGAIN:         MOV  EAX,[EBX]   ;LOAD THE ELEMENT
i6)                 ADD  EAX,100     ;ADD THE FIX VALUE
i7)                 ADD  EBX,4       ;UPDATE THE POINTER
i8)                 MOV  [ESI],EAX   ;STORE THE RESULT
i9)                 ADD  ESI,4       ;UPDATE THE POINTER
i10)                NOT  EAX         ;COMPLEMENT THE RESULT
i11)                MOV  [EDI],EAX   ;AND STORE IT
i12)                ADD  EDI,4       ;UPDATE THE POINTER
i13)                LOOP AGAIN       ;STAY IN THE LOOP
i14)                MOV  AX,4C00H    ;EXIT
i15)                INT 21H
```

Solution:

The fetch/decode unit fetches and converts instructions into micro-ops. Since there is no dependency for instructions i1 through i5, they are dispatched, executed and retired except for i5. Notice that the pointer values are immediate values; therefore, they are embedded into the instruction when the fetch/decode unit gets them. Now i5 is a memory fetch which can take many clocks, depending on whether the needed data is located in cache or main memory. Meanwhile i6, i8, i10, and i11 must wait until the data is available. However i7, i9, i12 can be executed out of order knowing that the updated values of pointers EBX, EDI, ESI are kept internally until the time comes when they will be committed to the visible registers by the retire unit. More importantly, the LOOP instruction is predicted to go to the target address of AGAIN and i5, i6, ... are dispatched once more for the next iteration. This time the memory fetch will take very few clocks since in the previous data fetch, the CPU read at least 32 bytes of data using the Pentium Pro 64-bit (8 bytes) data bus and the burst read mode, transferring into the CPU 4 sets of 8-byte data. This process will go on until the last round of the LOOP instruction where ECX becomes zero and falls through. At this time due to misprediction, all the micro-instructions belonging to instructions i5, i6, i7, ... (start of the loop) are removed and the whole pipeline restarts with instructions belonging to i14, i15, and so on.

Figure 1-1 The Complete Pentium II and Pentium III Processors Architecture


System bus
L2 cache
Bus Interface Unit
L1 instruction cache
L1 data cache
Fetch
Load
Store
Fetch & Decode Unit
(In order unit)
•Fetches instructions
•Decodes instructions to µOPs
•Performs branch prediction
Dispatch / Execute Unit
(out of order unit)
•Schedules and executes µOPs
•Contains 5 execution ports
Retirement Unit
(In order unit)
•Retires instructions in order
•Writes results to registers/memory
Instruction Pool/reorder buffer
•Buffer of µOPs waiting for execution

Caches of the Pentium II and Pentium III Processors

The on-chip cache subsystem of Pentium II and Pentium III processors consists of two 16-Kbyte four-way set associative caches with a cache line length of 32 bytes. The caches employ a write-back mechanism and a pseudo-LRU (least recently used) replacement algorithm. The data cache consists of eight banks interleaved on four-byte boundaries.

Level two (L2) caches have been off chip but in the same package. They are 128K or more in size. L2 latencies are in the range of 4 to 10 cycles. An L2 miss initiates a transaction across the bus to memory chips. Such an access requires on the order of at least 11 additional bus cycles, assuming a DRAM page hit. A DRAM page miss incurs another three bus cycles. Each bus cycle equals several processor cycles, for example, one bus cycle for a 100 MHz bus is equal to four processor cycles on a 400 MHz processor. The speed of the bus and sizes of L2 caches are implementation dependent, however. Check the specifications of a given system to understand the precise characteristics of the L2 cache.

What is a Core?

A standard processor has one core (single-core.) Single core processors only process one instruction at a time (although they do use pipelines internally, which allow several instructions to be processed together; however, they are still run one at a time.)

What is a Multi-Core Processor?

A multi-core processor is composed of two or more independent cores, each capable of processing individual instructions. A dual-core processor contains two cores, a quad-core processor contains four cores, and a hexa-core processor contains six cores.

Why do I Need Multiple Cores?

Multiple cores can be used to run two programs side by side and when an intensive program is running (AV Scan, Video conversion, CD ripping etc.) you can utilize another core to run your browser to check your email etc.

Multiple cores really shine when you’re using a program that can utilize more than one core (called Parallelization) to improve the program’s efficiency. Programs such as graphic software, games etc. can run multiple instructions at the same time and deliver faster, smoother results. So if you use CPU-intensive software, multiple cores will likely provide a better experience when using your PC. If you use your PC to check emails and watch the occasional video, you really don’t need a multi-core processor.

How many cores do i3, i5, and i7′s have?

A Dual-core processor has two cores.
A Quad-core processor has four cores.
An i3 processor has 2 cores.
An i5 processor has 2 or 4 cores
(depending on the model you have.)
An i7 processor has 2, 4 or 6 cores
(depending on the model you have.)