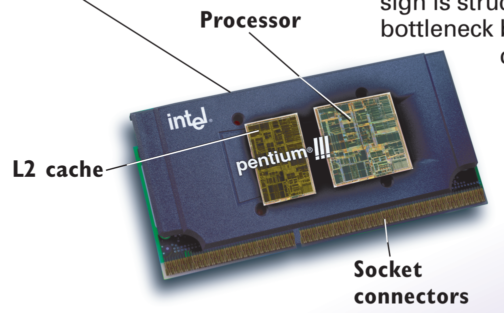
How a Pentium II Chip Works

1 Intel's Pentium microprocessor is made up of two slices of silicon. One is the 5.5-9.5-million transistor CPU, where the software's instructions are executed. The other is a *level 2* (L2), custom-designed high-speed memory cache. Its 15.5-million transistors store up to 512 kilobytes of data and code. Earlier processors typically used a cache separate from the processor, usually part of the computer's motherboard.

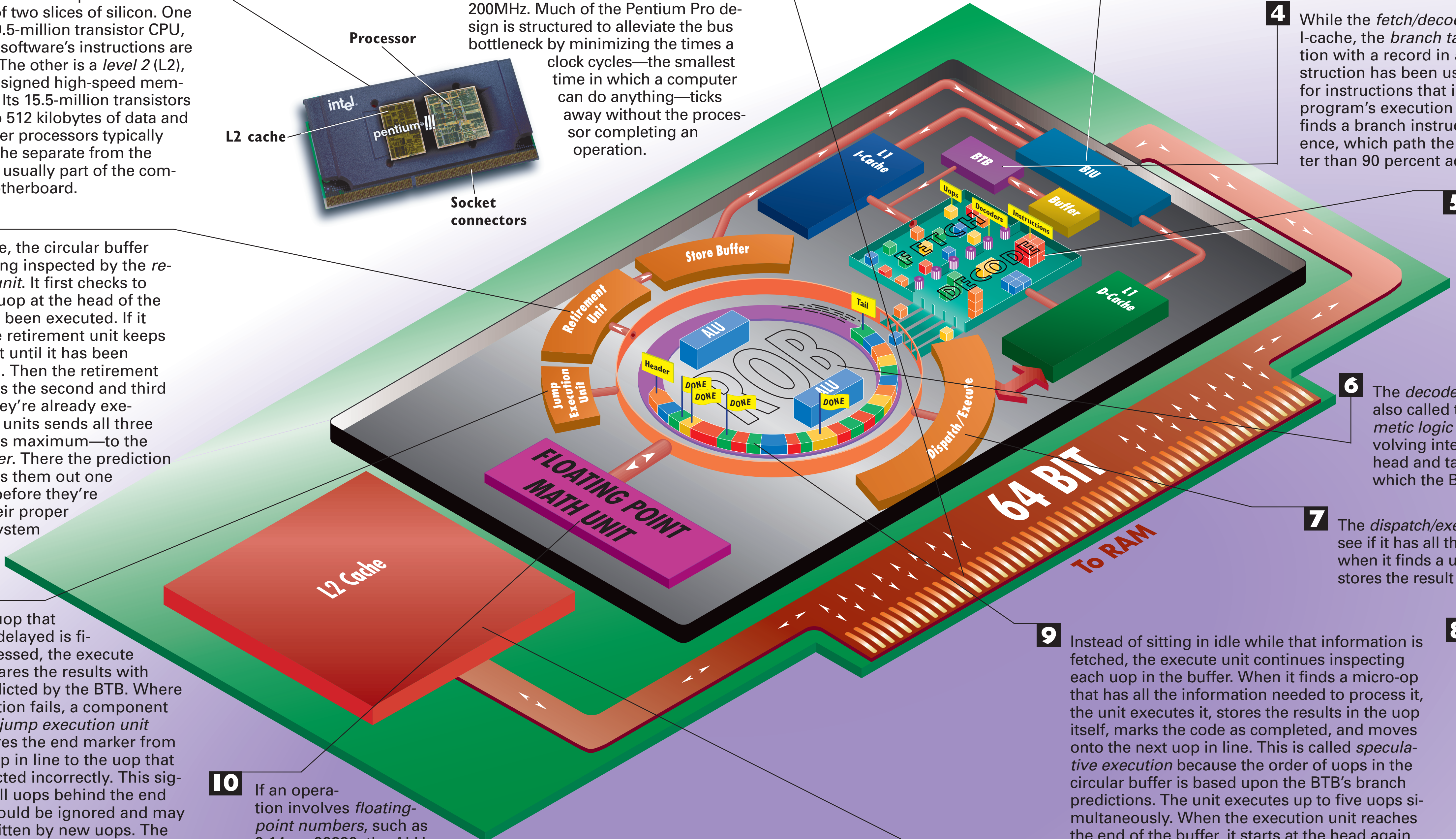


2 The processor and cache share the same 64-bit interface to the computer's information. Program code or data manipulated by that code move in and out of the chip at the PC's maximum bus speed, no more than 100Mhz even for processors that function internally at 200MHz. Much of the Pentium Pro design is structured to alleviate the bus bottleneck by minimizing the times a clock cycles—the smallest time in which a computer can do anything—ticks away without the processor completing an operation.

3 When information enters the processor through the *bus interface unit (BIU)*, the BIU duplicates the information, sends one copy to the CPU's closely linked L2 cache, and the other to a pair of *level 1* (*L1*) caches, ranging in size from 8-16K, built directly into the CPU. The BIU sends program code to the L1 *instruction cache*, or *I-cache*, and sends data to be used by the code to another L1 cache, the *data cache* (*D-cache*).

4 While the *fetch/decode unit* is pulling in instructions from the I-cache, the *branch target buffer* (*BTB*) compares each instruction with a record in a separate set-aside buffer to see if any instruction has been used before. The BTB is looking in particular for instructions that involve *branching*, a situation where the program's execution could follow one of two paths. If the BTB finds a branch instruction, it predicts, based on past experience, which path the program will take. The predictions are better than 90 percent accurate.

5 As the *fetch/decode unit* pulls instructions in the order predicted by the BTB, three *decoders* working in parallel break up the more complex instructions into *uops*, which are smaller, 274-bit *micro-operations*. The *dispatch/execution* unit processes several uops faster than it processes a single higher-level instruction.



6 The *decode unit* sends all uops to the *instruction pool*, also called the *ReOrder Buffer*. This contains two *arithmetic logic units* (*ALUs*) that handle all calculations involving integers. The ALUs use a circular buffer, with a head and tail, that contains the uops in the order in which the BTB predicted they would be needed.

7 The *dispatch/execute unit* checks each uop in the buffer to see if it has all the information needed to process it, and when it finds a uop ready to process, the unit executes it, stores the result in the micro-op itself, and marks it as done.

8 If a uops needs data from memory, the execute unit skips it, and the processor looks for the information first in the nearby L-1 cache. If the data isn't there, the processor checks the much larger L2 cache. Because the L-2 cache is integrated with the CPU, information moves between them 2-4 times faster than between the CPU and the external bus. At a chip speed beginning at 150MHz, information is retrieved at the rate of 1.2 gigabytes a second, compared to the 528MB a second if the CPU has to go to external memory to get the information.

9 Instead of sitting in idle while that information is fetched, the execute unit continues inspecting each uop in the buffer. When it finds a micro-op that has all the information needed to process it, the unit executes it, stores the results in the uop itself, marks the code as completed, and moves onto the next uop in line. This is called *speculative execution* because the order of uops in the circular buffer is based upon the BTB's branch predictions. The unit executes up to five uops simultaneously. When the execution unit reaches the end of the buffer, it starts at the head again, rechecking all the uops to see if any have finally received the data it needs to be executed.

10 If an operation involves *floating-point numbers*, such as 3.14 or .33333, the ALUs hand off the job to the *floating-point math unit,* which contains processing tools designed to manipulate floating-point numbers quickly.

11 When an uop that had been delayed is finally processed, the execute unit compares the results with those predicted by the BTB. Where the prediction fails, a component called the *jump execution unit* (*JEU*) moves the end marker from the last uop in line to the uop that was predicted incorrectly. This signals that all uops behind the end marker should be ignored and may be overwritten by new uops. The BTB is told that it's prediction was incorrect, and that information becomes part of its future predictions.

12 Meanwhile, the circular buffer also is being inspected by the *retirement unit*. It first checks to see if the uop at the head of the buffer has been executed. If it hasn't, the retirement unit keeps checking it until it has been processed. Then the retirement unit checks the second and third uops. If they're already executed, the units sends all three results—its maximum—to the *store buffer*. There the prediction unit checks them out one last time before they're sent to their proper place in system RAM.