MICROPROCESSOR REPORT
THE INSIDERS' GUIDE TO MICROPROCESSOR HARDWARE
VOLUME 7 NUMBER 14 OCTOBER 25, 1993

Cyrix Describes Pentium Competitor

x86-Compatible "M1" Design is Superscalar, Deeply Pipelined

by Linley Gwennap

Can a $70 million company develop a better processor than a $6 billion giant? Cyrix is attempting to reaffirm that David can beat Goliath at his own game. At the recent Microprocessor Forum, Cyrix's Peggy Herubin unwrapped the "M1" design, a superscalar, deeply pipelined, x86-compatible CPU that she claims will outperform both current and future generations of Intel's Pentium processor.

Cyrix's 20-person design team has pulled out all the stops to maximize the performance of the x86 instruction set. The M1 design uses advanced techniques such as branch prediction, speculative execution, out-of-order execution, register renaming, and memory bypassing to overcome bottlenecks in the CISC-style architecture and improve the efficiency of its pipeline. The complexity of the hardware design removes much of the need for the compiler to reorganize instructions; Cyrix believes that the M1 will execute existing code at high speeds without the need to recompile.

The company did not announce any specific products based on the M1 design; the first such products will probably debut in 2H94. By that time, Intel should be shipping its P54C version of Pentium, having boosted the chip's clock speed to near 100 MHz and reduced its manufacturing cost significantly. The M1 core could be used in parts that are pin-compatible with Pentium, or in different packages. Cyrix would not discuss pricing for these potential products, but the company has historically priced its products aggressively against Intel's.

Breaking the Register Bottleneck

The biggest deficiency in the x86 architecture is generally acknowledged (even by Intel) to be the small register set. All x86 processors, even Pentium, have just eight general-purpose registers, compared with 32 for most RISC architectures. As a result, program data must be constantly shuffled back and forth between memory and the registers. Sometimes, frequently used data is simply left in memory, since the x86 instruction set supports memory-to-memory operations.

These situations reduce performance in two significant ways. First, accesses to memory, even when buffered by a fast cache, are often slower than register accesses, which increases latencies. Second, reusing registers creates "false" dependencies, such as write-after-read (w-a-r) and write-after-write (w-a-w) dependencies. For example, in the sequence:

```
MOV [mem1], AX;  MOV AX, [mem2]
```

a data value is being read from the AX register and stored in memory so a second value can be written into AX. In a RISC architecture, this sequence can often be avoided entirely by allocating two registers for the two data values. This sequence slows down superscalar processors such as Pentium because the register conflict prevents the two instructions from executing in parallel.

Pentium overcomes this problem by having the compiler rearrange the code by placing an unrelated instruction between the two moves; this instruction can be executed in parallel with either MOV. Rearranging the instructions, however, requires a Pentium-aware compiler; existing code might not be ordered properly. Even if the program is recompiled, there is no guarantee that there will always be an unrelated instruction to place in the needed slot.

Cyrix solves w-a-r and w-a-w dependencies through register renaming. The M1 design maps the eight logical registers onto 32 physical registers. In the example above, the code sequence would be translated into:

```
MOV [mem1], AX;  MOV AX', [mem2]
```

where AX' is a different physical register than AX. The new sequence can be executed in parallel, since the w-a-r dependency has been removed. This mapping is done entirely by the hardware; no recompilation is necessary.

Register renaming is also used in IBM's POWER processors *(see **071301.PDF**)* to remove bottlenecks in the floating-point register file. IBM's design, however, performs renaming only on load operations. The M1 will perform renaming on every operation that updates the general registers. For example, in the sequence:

MOV [mem1], AX; AND AX, DX

the destination of the AND will be mapped to a different physical register, again avoiding the w-a-r dependency.

Pipelining "True" Dependencies

Renaming cannot resolve "true" dependencies, also known as read-after-write (r-a-w) dependencies. These often occur when a value is loaded from memory and immediately used in a calculation. A typical sequence is:

MOV DX, [mem]; ADD AX, DX

In this case, the data must be loaded into DX before it can be added to AX. This common situation stymies Pentium (and most superscalar RISC chips, as well), and forces the instructions to be executed one at a time. As with w-a-r dependencies, Pentium counts on the compiler to rearrange instructions to avoid r-a-w sequences. For unrecompiled code, however, these sequences can significantly reduce performance.

The M1 design resolves many r-a-w situations by organizing its pipeline as shown in Figure 1. The design includes separate stages for loading data from memory, performing arithmetic calculations, and storing data to memory. This eliminates the cache-access penalty common in most other processors. Figure 2 shows the instruction flow for the MOV, ADD sequence shown previously. The M1 will issue and execute these instructions in parallel due to the data bypassing between the AC2 stage and the execute stage.

Pentium, in contrast, must execute these instructions sequentially. The Pentium pipeline, shown in Figure 1, has a single stage that executes all load and arithmetic operations, forcing the ADD to wait until the MOV has completed.

Pairing Dependent Instructions

While the M1 pipeline will resolve many r-a-w dependencies, it cannot handle situations where two successive arithmetic instructions modify the same data. For example, the sequence:

ADD AX, BX; ADD AX, DX

which sums the values in AX, BX, and DX, will be executed serially in the M1. This is the case for most processors; only SuperSPARC (and Power2, in some situations) can execute such dependent operations in the same cycle.

Some dependent instruction pairs occur frequently enough to deserve special handling. For example, a Jcc (jump on condition code) instruction is often preceded by a compare instruction that modifies the condition codes. The M1 design will issue these two instructions together, and the Jcc instruction will actually complete in the writeback stage, one cycle after the compare is completed in the execute stage. Note that if the required condition code is already available when the Jcc is in the execute stage, it will complete at that time rather than be delayed. In any case, the Jcc does not cause any pipeline stalls or delay subsequent instructions.

Another common sequence is a series of PUSH or POP instructions. The M1 can issue two PUSH or two POP instructions together, even though both modify the stack pointer; special hardware detects and resolves this situation. Both compare-and-branch and PUSH/POP are also handled as special cases by Pentium.

One drawback to the longer M1 pipeline is an increased address-generation interlock. This interlock occurs when a value needed for an address calculation is being calculated in the execute stage of the previous instruction, for example:

ADD BX, DX; MOV AX, [BX].

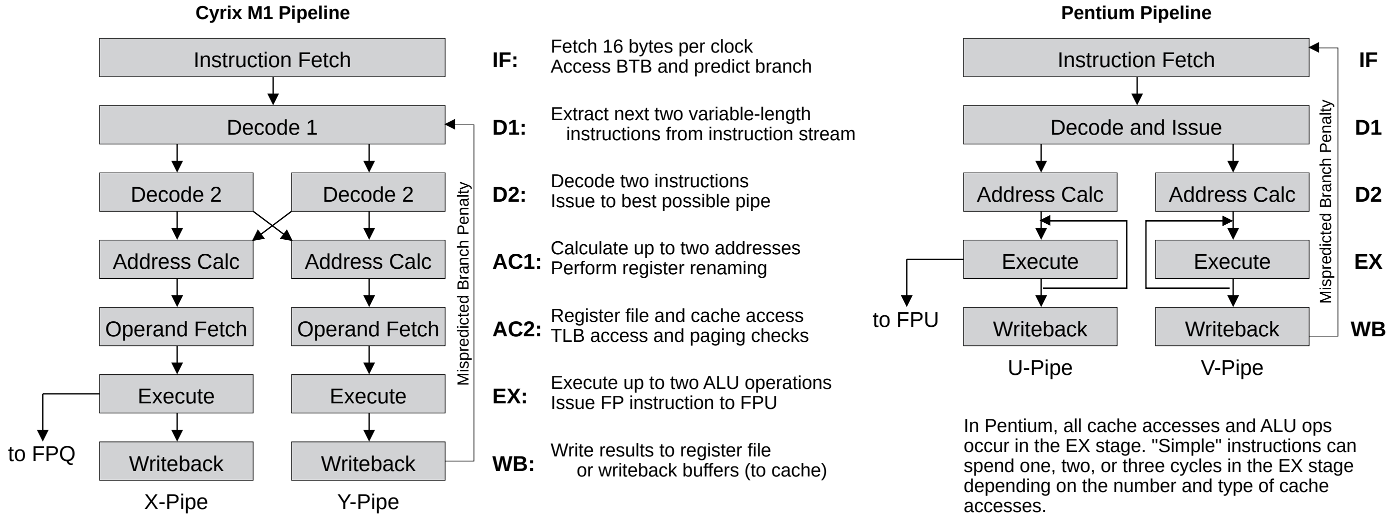
Cyrix M1 Pipeline
Instruction Fetch
Decode 1
Decode 2
Decode 2
Address Calc
Address Calc
Operand Fetch
Operand Fetch
Execute
Execute
Writeback
Writeback
to FPQ
X-Pipe
Y-Pipe
Mispredicted Branch Penalty
IF:
Fetch 16 bytes per clock
Access BTB and predict branch
D1:
Extract next two variable-length
instructions from instruction stream
D2:
Decode two instructions
Issue to best possible pipe
AC1:
Calculate up to two addresses
Perform register renaming
AC2:
Register file and cache access
TLB access and paging checks
EX:
Execute up to two ALU operations
Issue FP instruction to FPU
WB:
Write results to register file
or writeback buffers (to cache)
Pentium Pipeline
Instruction Fetch
Decode and Issue
Address Calc
Address Calc
Execute
Execute
Writeback
Writeback
to FPU
U-Pipe
V-Pipe
Mispredicted Branch Penalty
IF
D1
D2
EX
WB
In Pentium, all cache accesses and ALU ops occur in the EX stage. "Simple" instructions can spend one, two, or three cycles in the EX stage depending on the number and type of cache accesses.

Figure 1. The M1 pipeline contains separate stages for cache loads and arithmetic calculations, while Pentium does both operations in its execute stage, forcing these operations to be serialized.

As shown in Figure 2, this sequence causes the MOV to be held up for two cycles (actually three, if the instructions are paired). These sequences, however, occur less frequently than the r-a-w dependencies. In fact, the flexible addressing of the x86 architecture can eliminate the depicted situation entirely; most compilers would code that sequence with a single instruction, MOV AX, [BX+DX].

Pentium has a single-cycle address-generation interlock that is half as long as the M1's penalty. Most RISC processors, using a fetch-execute-load pipeline, do not have any address-generation interlock. The M1 designers accepted the address-generation penalty to eliminate the more common cache-access (load-use) penalty.

Memory Bypassing Prevents Stalls

Memory bypassing is a third technique that helps reduce the effects of the limited register set and prevent pipeline stalls. Like most processors, the M1 will route data that has been calculated (or loaded) in a previous cycle to a current operation, bypassing the register file. The new design goes beyond other implementations by performing this bypass even if the data is stored in memory rather than the register file.

Memory bypassing is useful in an x86 processor because the small register set forces the compiler to keep many variables in memory. Sequences such as:

ADD [mem], CX; SUB DX, [mem]

are fairly common when dealing with memory values. In this example, CX is being added to the value stored at [mem], and the result is being subtracted from DX. The M1 design detects that the address represented by [mem] is the same for both instructions and bypasses the result of the first calculation directly to the SUB instruction.

Although the instructions in this example cannot be executed in parallel, they will execute in successive cycles in the M1, as depicted in Figure 2. Other processors, including Pentium, have to wait for the memory update to complete before the second instruction can begin to load its operand. This sequence takes four cycles to execute on a Pentium.

Memory bypassing can be complicated. It's relatively easy to compare 3-bit register descriptors to detect a match; to bypass memory, the complete memory address must be compared. Also, care must be taken for memory-mapped I/O addresses and other non-cacheable areas; these addresses must not be bypassed. Cyrix's Herubin said that the M1 design correctly handles these situations, but she did not provide additional details.

M1 Design Uses Long Pipeline

Cyrix calls the M1 superpipelined, a term which is also used by MIPS and Alpha with little technical meaning. Is it really superpipelined, or just complicated? One thing we can all agree on is that the M1 pipeline is long: its seven stages are two more than those found in most x86 or RISC processors. The M1 pipeline is comparable in length to the seven-stage Alpha or eight-stage R4000 pipelines.

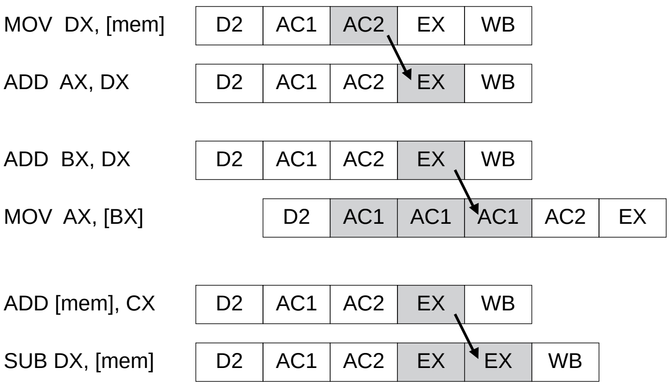

Figure 2. The first example shows how data bypassing allows a load operation and an add to execute in parallel. The second example shows the two-cycle address-generation interlock. The third demonstrates memory bypassing.

The M1 design uses two pipeline stages for instruction decode and dispatch, an undeniably complex task in a superscalar x86 processor due to the variable-length instructions. Pentium jams most of this operation into a single pipeline stage. Cyrix also spreads the segmentation checks across the AC1 and AC2 stages. By using two stages for these complex tasks, the M1 may be able to attain higher clock rates than Pentium.

The classic problem with long pipelines is the extended branch penalty. As shown in Figure 1, the branch penalty can be as long as five cycles in the M1 if a Jcc resolved in the writeback stage must change the instructions entering the first decode stage. If the Jcc is resolved in the execute stage, the penalty will be four cycles. Because of the dual pipelines, up to ten instruction slots can be lost on a branch.

Cyrix employs branch prediction to reduce the number of these long pipeline stalls. The M1 design includes a branch target buffer (BTB) nearly identical to Pentium's. The four-way set-associative BTB stores the target address of conditional branches. It uses the same four-state prediction algorithm as Pentium *(see **070402.PDF**)*. If the history bits predict that the branch will be taken, its target address is read from the BTB and used as the next fetch address. Thus, if the branch is correctly predicted, there will be no pipeline delays.

The Cyrix design goes beyond Pentium by implementing a separate target address stack for RET instructions. These instructions cannot be handled through the BTB because their target address can change. When a CALL instruction is encountered, its return address is pushed onto the address stack; it can then be popped when the corresponding RET is executed, which results in no branch penalty for the RET. While a similar feature is defined in the Alpha architecture, it has not yet been implemented in any Digital (or other) microprocessor.

For the programs in the SPEC89 suite, Intel claims that Pentium correctly predicts 75% to 85% of the dynamic branches, including non-taken branches that miss the BTB. Herubin would not reveal the size of the M1's BTB or return stack, but said that the BTB is "at least as large" as Pentium's. Thus, with its extra return stack, the M1 should exceed the prediction rate of Pentium. An improvement of about 5% would balance out the extra penalty cycle (compared with Pentium) that the M1 incurs on mispredicted branches.

The M1's prefetch buffer helps reduce the mispredicted branch penalty. As shown in Figure 3, the prefetch buffer passes instructions into the dual integer units. It has room for up to four predicted code streams, as well as the four code streams that are not predicted. Thus, even if the branch is mispredicted, the required instructions can be obtained from the prefetch buffer, eliminating one cycle of delay.

Dual Instruction Dispatch

The M1 is a two-way superscalar design. Like Pentium, the Cyrix design can pair most simple instructions but must execute complex, multi-cycle operations serially. Each of these processors uses both pipelines to speed the execution of these multi-cycle operations.

The two M1 pipelines, which Cyrix calls the X-pipe and the Y-pipe, are not totally symmetric: the X-pipe is the only one that can handle branches and floating-point instructions. The two pipelines can swap instructions after the decode stage, as shown in Figure 1, allowing the processor to avoid most instruction-issue restrictions.

The biggest difference between the M1 and Pentium dispatch rules is that the Cyrix chip will combine many dependent instruction pairs that must be issued serially in Pentium. This advantage comes from the M1's deeper pipeline and register-renaming abilities described earlier. The performance difference will be most pronounced on code that has not been recompiled for Pentium.

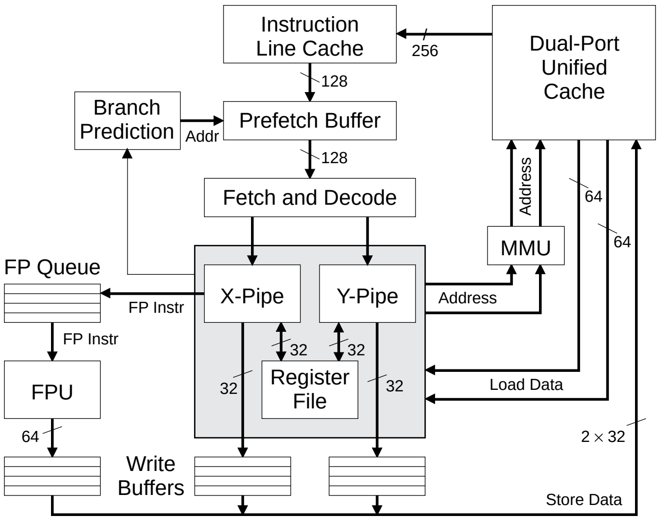

Figure 3. In the M1 design, instructions flow from the unified cache to the smaller instruction line cache and then into the prefetch buffer before being decoded and dispatched.

There are a few minor differences between these two processors' dispatch rules. Pentium cannot pair a branch instruction with its successor; the M1 design can handle this case. Also, the M1 has a shift unit in both pipelines, while Pentium has only one.

The M1 design can issue only one FP instruction per cycle. It does not pair floating-point operations with FXCH instructions, as Pentium does. On the other hand, the Cyrix design can pair floating-point instructions with integer instructions, while Pentium cannot. Thus, while the Intel processor will have an edge on code that has been recompiled to take advantage of FXCH pairing, the M1 will do well on existing binaries.

Bursting Pipeline Bubbles

After pairs of instructions are issued, "bubbles" can be created in the pipeline if the two instructions do not flow through the processor at the same rate. This is typically caused by instructions that take several cycles to execute, such as floating-point math operations and loads with cache misses. Pentium creates additional bubbles because it takes two or three cycles to execute ALU operations with operands in memory.

Cyrix's honed M1 design eliminates many of these bubbles, which improves the processor's efficiency. A significant savings comes from the added pipeline stages, which handle loading and storing memory operands.

For example, an instruction such as ADD [BX], AX requires the processor to load data from memory, add to it the value of AX, then store the result to memory. Pentium takes three cycles to execute this instruction; the second pipeline usually sits idle for two of these cycles. In the M1 design, the load occurs in the operand stage, the add takes place during the execute stage, and the store happens in the writeback stage; the second pipeline can be fully utilized during this process.

A more serious problem occurs during long-latency operations such as floating-point math or a cache miss. The M1 design handles the former case using the FP queue shown in Figure 3. Although its floating-point unit is not pipelined, it will queue up to four FP operations; integer instructions continue to execute in both pipelines as long as the queue does not overflow.

Precise exceptions are maintained by checkpointing the processor as each FP instruction is queued. Checkpointing saves most of the processor state in shadow registers; if an exception occurs, the processor state can be restored quickly by loading from the shadow registers.

The M1's large physical-register file avoids the need for shadowing the general registers; during a checkpoint, the current registers are locked, and future writes are mapped to different physical registers via renaming. The segment registers and a few control registers are not shadowed; writes to these registers force a stall until the FP queue is drained.

Out-of-Order Execution

The Cyrix design is unusual in its ability to continue dispatching instructions while a cache miss is processed. Digital's Alpha chips and HP's PA7100 also accomplish this feat, but Pentium and most others do not.

If a cache miss occurs on a load, the M1 will continue executing instructions using the other integer unit. Execution will continue until:

- An instruction requires the data being loaded.
- A complex instruction requires both pipelines.
- A non-shadowed register is written.
- An instruction requires access to the external bus.
- A branch is encountered (if the miss is in the X-pipe).
- An FP op is encountered (if the miss is in the X-pipe).

The dispatch logic will attempt to route memory instructions to the Y-pipe to avoid the latter two situations.

By continuing to execute past a cache miss, instructions can be completed out of order. Most processors do not allow out-of-order execution because of the complexities of data dependencies and of maintaining precise exceptions. Dependency checking is simplified in the M1 by allowing only a single instruction to be stalled; each out-of-order instruction must check only one dependency.

Peggy Herubin of Cyrix describes the M1 microarchitecture.

Even when instructions are executed out of order, the Cyrix design ensures that everything appears in order from a software standpoint. The processor is checkpointed before beginning out-of-order execution, so even an unexpected trap will not disturb the in-order model. To avoid confusing external devices, bus cycles will always be issued in order.

Dual-Ported Unified Cache

The M1 designers chose a unified on-chip cache because it has a higher hit rate than a split cache of equal total size. The drawback of a unified cache is the potential conflict between data accesses and instruction fetches. The Cyrix design adds a small instruction-line cache to ease these conflicts. Like Pentium, it also uses a dual-ported design to allow two load/stores per cycle.

A complex algorithm allocates bandwidth among multiple uses. If an instruction fetch misses the prefetch buffer, it directly accesses the unified cache with the highest priority, using both ports; Cyrix estimates that this situation will consume only 2% of all cache cycles due to the high hit rate in the prefetch buffer. Data loads are second priority, followed by data stores; write buffers queue stores until adequate bandwidth is available. Finally, excess bandwidth is allocated to instruction prefetching using one or both ports to deliver up to 32 bytes per cycle.

The cache itself is not truly dual-ported but uses a banked structure similar to Pentium's data cache. Herubin would not disclose the number of banks but said that the organization will be "better than Pentium's" for existing code.

The unified TLB design is fully dual-ported to generate two independent physical addresses per cycle. The unified cache is virtually indexed, allowing the cache access to proceed in parallel with the address translation. Herubin would not reveal the sizes of the cache, buffers, or TLB, but said that the M1 TLB will have more entries than the combined 96 entries in Pentium's split TLBs.

More Details to Come

Herubin's Microprocessor Forum presentation covered only the M1 microarchitecture, and she would not discuss the details of future Cyrix products. Exact performance information, cache sizes, power consumption, and die size will not be available until these future parts are officially announced. Herubin also did not describe the bus interface other than to say that it is 64 bits wide.

Cyrix did confirm that the M1 core will fit onto a single chip, and that the number of transistors will be roughly comparable to the two million used by Pentium, although the Cyrix design uses more on the integer side and fewer on the floating-point unit. (This reflects the company's view that, for most markets, integer performance is more important than floating-point.) Given a similar core size, it's unlikely that M1-based chips will be able to fit more than 16K of on-chip cache, and Cyrix may go to an 8K cache to reduce its manufacturing cost.

Herubin presented some performance simulations comparing the M1 to Pentium. At the same clock rate, she says that the M1 will be about 40% faster on the Power Meter benchmark and 110% faster on Norton SI. These are 16-bit DOS programs in binary form, so these figures should be representative of the M1's performance on code that has not be recompiled for Pentium. Cyrix admits that on recompiled benchmarks, such as SPEC, Pentium's performance will be much closer to the M1's at the same clock speed.

Assuming that the M1 design achieves the performance indicated by these simulations, the key question is, will it be able to match Pentium in clock speed? Cyrix claims that the M1 is designed to operate at 100 MHz in initial products, with higher clock speeds in the future. Pentium, on the other hand, is limited to 66 MHz today but should be close to 100 MHz by the time the M1 begins shipping. Thus, if Cyrix achieves its design goals, M1 should be faster than the fastest Pentium.

Clock frequency goals can be difficult to reach. Pentium is rumored to have started life with a 100-MHz design goal but is still struggling to ship 66-MHz parts. Herubin says that spreading the instruction decode across two stages will allow higher clock frequencies, but ALU operations and cache accesses must still complete in a single cycle. Furthermore, Cyrix's 0.65-micron CMOS process appears to be potentially slower than the 0.6-micron BiCMOS process that Intel has planned for future Pentiums.

Cyrix may find it difficult to match, much less exceed, the clock speed of Intel's Pentium processors. If the M1 can even come close, however, it could match Pentium's performance, particularly on unrecompiled code.

Competition For Pentium

The M1 design uses a variety of advanced techniques to execute x86 code at high speeds. These techniques are particularly advantageous for existing x86 code; Pentium, in contrast, loses 20–30% of its peak performance unless programs are recompiled. Cyrix's initial performance simulations seem to demonstrate the superiority of its approach.

The new design raises questions about Intel's Pentium, which is said to have cost as much as $100 million to develop. Even with this enormous effort, Pentium ended up being little more than two 486 pipelines glued together, with a fast floating-point unit. Cyrix will spend perhaps a tenth of that amount and has developed a much more innovative design, using register renaming and other advanced tactics to solve critical problems in the x86 architecture.

For More Information

The M1 microarchitecture is not a product. Cyrix expects that processors implementing the M1 design will be available in 2H94. For more information, contact Cyrix at 2703 N. Central Expressway, Richardson, TX 75080; 214.994.8491, fax 214.994.8404.

Intel is rumored to be using similar techniques in its next-generation P6 processor, which is expected to reach the market 6–12 months after the M1. The P6 probably will also dispatch three or four instructions per cycle, giving it a significant performance advantage over the M1. Thus, even if Cyrix achieves a lead in performance, it may not last for long.

Whether or not it outperforms Pentium, the M1 should provide a means for Cyrix to compete with Intel up and down the product line. Until now, Intel has always had a lock on the high-performance products; the x86 leader has thus placed a premium price on these chips and used the profits to fund its massive R&D efforts. By holding the key to maximum performance, Intel has also been able to enforce the loyalty of its customers.

If, during 1995, Cyrix can offer processors similar to Intel's premium products, Intel may have to drop prices on its high-end chips, jeopardizing its legendary profitability. Alternatively, Intel could try to boost prices on its 486 chips, or cut back on R&D or other spending. Either of these tactics could open the door for other x86 chip vendors. Furthermore, Cyrix could become a "one-stop shopping" alternative to Intel.

Before this scenario can develop, Cyrix must move M1 from a fancy design to a potent reality. The fabless vendor must then demonstrate an ability to obtain enough chips to make a dent in the overall market; while Pentiums are in short supply today, Intel will probably be shipping a million per quarter by the time the M1 is available, raising the bar for Cyrix. Finally, the fledgling company must withstand inevitable legal challenges. While the recent revelations show that Cyrix has a talented and efficient design team, only time will tell whether the M1 is truly a Pentium killer. ♦