Cyrix 5x86: Fifth-Generation Design Emphasizes Maximum Performance While Minimizing Transistor Count

Darrell Benke and Tom Brightman
Cyrix Corporation

Abstract

The Cyrix® 5x86™ processor is a new performance-optimized x86 design. Drawing on the experience gained in the development of Cyrix's sixth-generation M1 superscalar CPU, the 5x86 design incorporates many of the performance-enhancing techniques of the M1 in a novel way to meet the key requirement of low power system designs: maximum performance per watt of power consumed. The challenge of the 5x86 design was to achieve compelling system performance at no more than half the power consumption of competing solutions. That goal was achieved by critically evaluating the costs (measured in power and transistor count) versus the benefits (measured in performance) of key architectural features. This analysis enabled the careful selection of architectural features that deliver the desired performance at less than half the power consumption of competing fifth-generation alternatives.

Introduction

The driving force behind architectural enhancements is the demand for increased system performance. Modern processors achieve performance by exploiting the parallelism inherent in algorithms to the fullest extent possible. The obvious example is a superscalar processor that can execute multiple instructions concurrently. While concurrent execution increases performance, it does so at a substantial cost in design complexity and transistors required.

Unfortunately, initial superscalar designs have been frustrated by the problems of inter-instruction dependency checking and resource usage contention. To manage these conflicts, some designs imposed instruction-issuing constraints on the theory that application programs could be recompiled quickly and easily (based on knowledge of the instruction issuing rules and restrictions) to optimize code flow. Examples of this approach include the PowerPC 601™, HP-PA7100™ and the Intel Pentium™ processor – CPUs that can issue two instructions simultaneously but only under restrictive, special-case conditions. These conditions are more of a limitation for the Pentium since the majority of the software it executes will be from the installed base of x86 operating systems and applications. This limitation reduces the effectiveness of issuing and executing multiple instructions.

The cost of the second execution pipeline and the complexity of dual pipe control will probably never be justified by application performance benefits in fifth-generation processors that do not have guaranteed access to recompiled software. In the x86 market, recompilation simply has not occurred and will likely never occur. (Note: The sixth-generation

M1 avoids this by design – it does not assume recompilation to achieve a high level of multiple issue and execution.)

5x86 Architecture

The increased complexity, transistor count, and power consumption of superscalar designs led Cyrix engineers to re-examine the benefits of the superscalar approach. Clearly the power dissipated in a second execution pipeline plus the added power dissipated in the control logic to oversee two execution pipelines should be minimal to achieve performance that will justify the transistors added. Analysis has shown that the increased complexity of two execution pipelines can cost 40% in transistor count while providing an increase of less than 20% in instructions-per-clock performance.

Cyrix engineers analyzed the M1 performance features to identify those that could increase the performance of a single execution pipeline. The resulting list includes is shown below.

- memory bypassing
- branch prediction
- 16-KByte cache
- decoupled load/store

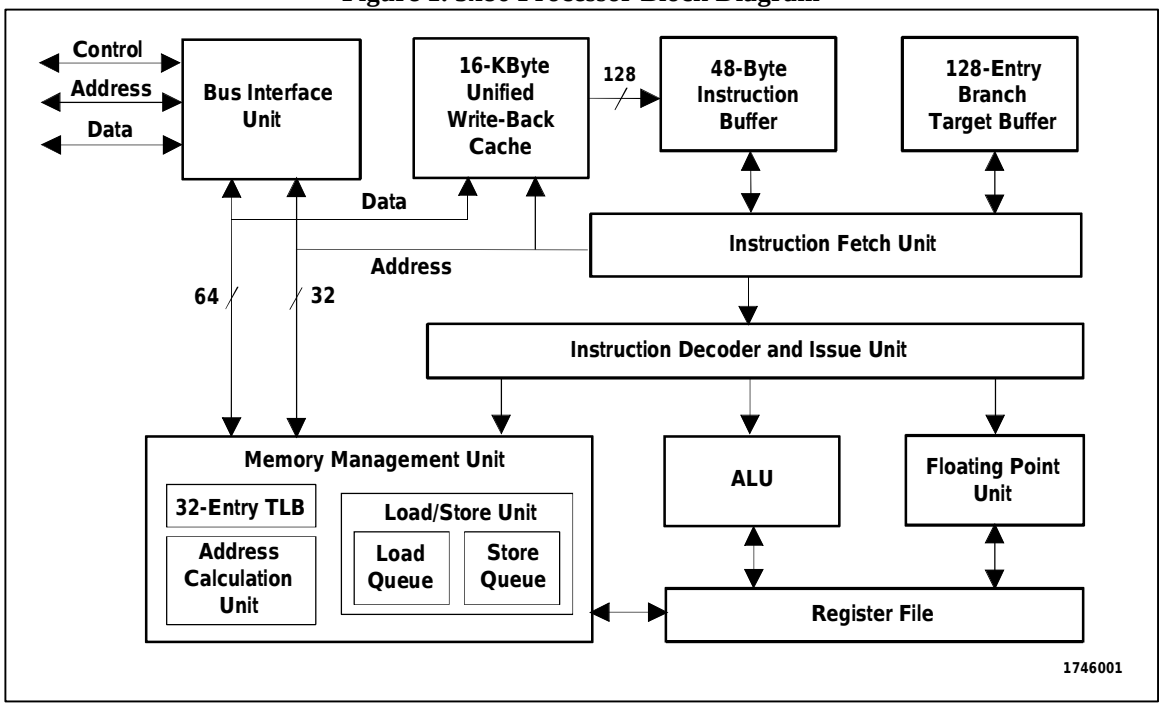
These features dramatically increase the utilization of a single execution pipeline without the added transistor count, power consumption, and complexity of a superscalar architecture.

Two facts were fundamental in identifying features for the 5x86:

(a) the x86 is a 32-bit architecture.

(b) the average instruction length is 2.7 bytes for existing 8/16-bit code and 4.4 bytes for 32-bit code.

Figure 1. 5x86 Processor Block Diagram



These two facts combine to reduce the bus width required to handle most data and code transactions to 32 bits. A key lesson from both fifth- and sixth-generation designs, however, is that inherent parallelism is most easily exploited through the use of decoupled units within the processor. These units are interconnected with multiple 32-bit, split-transaction busses so that the operational latency of one unit does not block actions by another.

The 5x86 CPU employs a dedicated branch unit including a branch target buffer (BTB), a 16-KByte unified write-back cache, a floating point unit (FPU), and an instruction fetch (IF) unit and an instruction decode (ID) unit. The memory management unit contains a 32-entry translation lookaside buffer, a load/store unit capable of managing concurrent operations, and the address calculation unit. The 5x86 functional units are interconnected by two 32-bit busses that permit non-blocking operation of the units. A 128-bit instruction fetch bus feeds 16 bytes of code per cycle to a three-line-deep buffer in the instruction decode unit.

Execution Pipeline

The 5x86 has a six-stage execution pipeline as shown in Figure 2. The instruction fetch pipe stage generates a continuous instruction stream from the on-chip cache and external memory for use by the instruction decode stage. The instruction fetch stage exploits the 5x86 branch prediction logic to fetch instructions at the predicted address. Up to 48 bytes of code are queued prior to the instruction decode stage. The instruction decode stage evaluates the code stream provided by the instruction fetch stage and determines the number of bytes in each instruction and the instruction type.

The address calculation function contains two superpipelined stages. If an instruction refers to a memory operand, Stage 1 calculates a memory address for the instruction. Stage 2 performs any required memory management functions, cache accesses, and register file accesses. If a floating point instruction is detected by Stage 2, the instruction is sent to the FPU for processing.

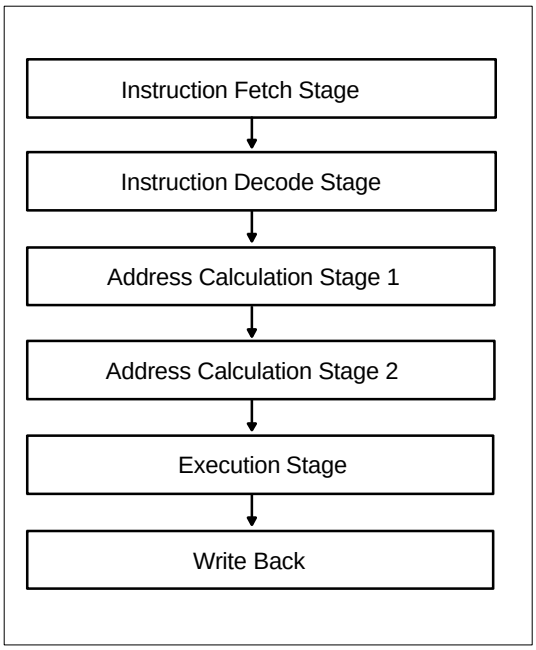


Figure 2. Six-Stage Execution Pipeline

The execution stage, under control of microcode, executes instructions using the operands provided by the address calculation stage. The last stage of the pipeline, write-back, updates the register file or writes to the load/store unit within the memory management unit.

Memory Bypassing

The six-stage pipeline of the 5x86 CPU is capable of bypassing memory operations, under certain conditions, to streamline processing. Memory bypassing can be illustrated by the instruction sequence below.

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

This sequence adds the value in CX to the value at [mem] and then subtracts the new value from the value at DX. Most processors wait for the first instruction to update the value at [mem] before fetching the operand for the second instruction. The 5x86 processor detects that the value being updated at [mem] is needed by the

second instruction and supplies the result of the first instruction to the second instruction directly, without an intervening memory read operation. Bypassing the memory read operation allows this sequence to complete in two clock cycles while other processors, without memory bypassing, may take at least four cycles.

Cache

The 5x86 implements a 16-KByte, four-way set associative, unified instruction/data cache that can operate in either write-back or write-through mode. The cache is arranged as four sets of 256 lines per set with 16 bytes per line. Each 16-byte cache line has an associated 21-bit tag and one valid bit. Each cache line also includes four dirty bits, one bit per double-word. The four dirty bits allow each double-word to be marked independently as dirty rather than marking the entire line as dirty. Marking each double-word as dirty minimizes the number of writes needed when a cache flush operation or line eviction occurs. When three or more double-words within a cache line are dirty and a cache flush operation or line eviction occurs, a burst write cycle is performed when writing back that line to memory to further minimize required bus bandwidth for cache management.

To increase cache bandwidth, the 5x86 cache architecture is surrounded by three buffers that allow an entire cache line to be read or written in a single clock cycle. The cache fill buffer assembles 16 bytes of data prior to requesting cache access to perform the actual line fill. The cache flush buffer holds dirty cache data that needs to be exported to the external bus (system memory) as a result of a cache flush or line replacement. The cache HITM buffer holds a cache line from an external inquiry that results in a cache hit. Because the 5x86 is scalar and has these buffers, it alleviates the need for more sophisticated cache banking techniques for concurrent accesses. This leads to a transistor reduction of approximately 20% relative to a banked implementation of equivalent size.

The cache bandwidth is further enhanced by a dedicated 128-bit port for transferring instructions to the IF unit. The 128 bits of instruction are transferred directly to a line in the instruction buffer. The cache data port is 64 bits wide and can be split into two 32-bit data paths. The ability to have two 32-bit data paths allows the 5x86 to simultaneously perform a 32-bit data transfer to or from main memory, and a 32-bit data transfer to or from the load/store unit. In addition, superpipelining the 5x86 address calculation stage allows cache accesses in a single clock cycle, identical to register accesses.

Branch Prediction

The 5x86 minimizes the performance impact of latency in branch instructions by using branch prediction. Branch instructions occur, on average, every five instructions in x86-compatible programs. When the normal sequential flow of a program changes due to a branch instruction, the pipeline stages may stall while waiting for the CPU to calculate, retrieve, and decode the new instruction stream. The branch unit is composed of logic to boost performance by predicting the set of instructions that are most likely to be executed. The 5x86 uses a 128-entry BTB to store branch target addresses and branch prediction information. This feature allows the processor to predict, on the basis of recent history, which branch will be taken. Correctly predicted branch instructions execute in a single clock. Incorrectly predicted branches require five clock cycles to flush the instruction pipeline. The decision to follow one branch or the other is based on a four-state branch prediction algorithm that achieves approximately 80% prediction accuracy with a 128-entry BTB.

If an unconditional branch instruction is encountered in the fetch stage, the 5x86 accesses the BTB to check for the branch

instruction's target address. The BTB actually contains a pointer to a line in the cache containing the instructions at the desired address. If the branch instruction finds a matching address in the BTB, the 5x86 begins fetching at the cache line specified by the BTB.

In the case of conditional branches, the BTB also provides history information to indicate which branch is more likely to be taken. If the conditional branch instruction finds a matching branch address in the BTB, the 5x86 begins fetching instructions at the predicted target address. If the conditional branch does not find a matching address in the BTB, the 5x86 predicts that the branch will not be taken and may prefetch both the predicted and the non-predicted path, eliminating the cache access cycle on misprediction. Once fetched, a conditional branch instruction is decoded and then dispatched to the pipeline. The conditional branch instruction continues through the pipeline and is resolved in the execute stage.

Since the target address of a return (RET) instruction is dynamic rather than static, the 5x86 caches the target addresses for RET instructions in a return stack rather than in the BTB. The return address is pushed on the return stack during a CALL instruction and popped during the corresponding RET instruction.

Instruction Fetch Unit

The instruction fetch unit in the 5x86 fetches instruction bytes from cache or memory and delivers them to the ID unit. Because of the variable-length nature of x86 instructions and the time required to access external memory, the IF unit implements a 48-byte buffer for temporary storage of fetched instruction bytes. Instructions from memory are loaded, one 16-byte line at a time, into the three-line buffer. The instruction fetch unit keeps the instruction buffer full by issuing fetch requests ahead of instructions being sent to the ID unit. The IF unit provides eight bytes of instruction to the ID unit each cycle. When enough bytes have been sent to the ID unit to free up a 16-byte line in the instruction buffer, the instruction fetch unit requests an instruction fetch. The 48-byte instruction buffer conserves the required instruction bandwidth to the cache and frees up cache bandwidth for data accesses. In addition, the instruction buffer can store small code loops, making them easily accessible to the ID unit and allowing increased execution performance.

A special feature of the IF unit allows short change-of-flow actions to execute without accessing memory if the target address has already been fetched and stored in the instruction buffer. The process combines the capabilities of the IF and branch target prediction. This capability enhances performance and saves power since the cache and internal busses are not activated for the fetch.

Instruction Decode Unit

The instruction decode unit in the 5x86 decodes the variable-length x86 instructions. The instruction decode involves determining the length of each instruction, separating immediate and/or displacement operands, decoding addressing modes and register fields, and creating an entry point into the microcode ROM. As previously discussed, the input to the instruction decoder is eight bytes of instructions supplied by the IF unit. These bytes are shifted and aligned according to the instruction boundary of the last instruction decoded. The ID unit can decode and issue instructions at a maximum rate of one per clock. Instructions with one prefix and instructions of length less than or equal to eight bytes can be decoded in a single cycle.

Memory Management Unit

The 5x86 memory management unit contains three primary functional units: the load/store unit, the 32-entry translation lookaside buffer, and the address calculation unit. The address

calculation unit performs all address calculations, maintains instruction pointers for each pipeline stage, and initiates load and store transfers.

The 5x86 CPU implements an advanced load/store unit to reduce the typical bottlenecks associated with load/store processing. The pipelined load/store unit is capable of managing concurrent operations and of processing loads and stores out of order while maintaining a three-deep load queue and four-deep store queue. The load/store unit is also responsible for handling all read/write requests from the address calculation unit, managing read-after-write dependencies for memory accesses, performing data forwarding, and checking self-modifying code.

Execution and Floating Point Units

The execution unit consists of functional units (logical, adder, constant ROM, shifter, and multiplier/divider), register files, and the microsequencer and associated ROM. The execution unit is an efficient implementation since performance gains are achieved in other elements of the design. The 5x86 executes the majority of widely used instructions in Windows® and other common applications in a single clock cycle. As with previous Cyrix processors, the 5x86 includes a hardware integer multiplier that significantly reduces integer multiply latencies.

The 5x86 FPU is based on the same core as the FPU in Cyrix's sixth-generation M1 processor. The FPU interfaces to the integer unit and the cache unit through a 64-bit bus. It is x87-instruction-set compatible and adheres to the IEEE-754 standard. Because most applications contain FPU instructions mixed with integer instructions, the 5x86 FPU achieves high performance by completing integer and FPU operations in parallel.

FPU instructions are dispatched to the pipeline within the address calculation unit. The address calculation stage of the pipeline checks for memory management exceptions and accesses memory operands for use by the FPU. The load/store unit is responsible for managing FPU operands. Once the instructions and operands have been provided to the FPU, the FPU completes instruction execution independently of the ALU and load/store unit.

Power Management

The 5x86 was engineered with several advanced power management features. The processor monitors and automatically powers down the FPU and other internal circuits when they are not in use. The activation of internal sense amplifiers is minimized by enabling them only during cache accesses and by optimally organizing the microcode. Each 32-bit section of the 64-bit internal data bus is driven only when needed. The core design of the 5x86 is completely static to allow for easy clock manipulation, a feature commonly used to adjust processor power consumption. At 100 MHz, the 3.45-volt 5x86 dissipates a maximum of 4.3 watts, with a typical dissipation of about 3 watts.

Additionally, software can automatically reduce the core bus frequency to one half the external bus frequency by simply writing to on-chip registers. The System Management Mode (SMM) software implementation is compatible with all existing and planned Cyrix processors and can be used for systems management functions such as power conservation.

Bus Interface Unit

The 5x86 internal 64-bit bus is tapered down to a 32-bit external bus to allow the processor to be dropped into existing platforms. This is an example of Cyrix's strategy to leverage existing sockets/designs to minimize customers' development cycles. The 5x86 pinout is a superset of the DX4 pinout since pins are necessary to support the 5x86 write-back cache, a feature not found on DX4s. The eight buffers

allow sufficient buffering of write activity to maintain bandwidth for read operations, reducing pipeline stalls. The 5x86 supports both clock doubling and clock tripling. The bus protocol is standard except for an optional linear burst mode which can be implemented instead of the Cyrix one-plus-four mode. The one-plus-four mode is compatible with all existing chipsets. Operating the CPU in linear burst mode minimizes bus activity and results in higher performance.

Conclusion

The 5x86 is clearly an innovative design in identifying and utilizing superscalar architectural features in a scalar configuration to significantly improve performance while minimizing transistor count. The branch prediction and branch target cache, decoupled load/store unit, and data forwarding capabilities are just a few of the fifth-generation features Cyrix brings to a scalar x86 design.

Cyrix Corporation
P.O. Box 850118
Richardson, TX 75085-0118
Tel: (214) 968-8388
Fax: (214) 699-9857