# Microcode Engine Offers Enhanced Emulation

## IMS's 3250 Emulates 486, 68040, 6502

**By Mark Thorson**

International Meta Systems will soon announce its 3250, a microcoded processor for binary emulation of Intel's 486, Motorola's 68040, and the venerable 6502. (IMS has also talked about emulation of RISC processors, but this is not part of the plan for the 3250.) User-defined microcode will also be supported, allowing system designers to achieve differentiation by moving performance-critical routines into microcode. IMS offered earlier versions of this architecture as board-level products for the PC/AT and Macintosh II, and sold about 500 systems, mostly for running a microcoded Smalltalk interpreter. IMS was founded in 1985 by executives from The Aerospace Corporation. It went public in 1989, and now has six full-time employees. Currently, their silicon foundry is S-MOS (part of Seiko Epson).

Although microcoding is a common implementation technique used internally in many CPU designs, RISC architectures have largely eliminated microcode in high-end CPUs. Even Intel's 80x86 family, which traditionally has used extensive microcode, is expected to use a RISC-like microarchitecture in future implementations.

Microcode has always seemed to offer the possibility of application acceleration through custom microcode. It's often said that programs spend 95% of the time executing 5% of the code, and microcode held the promise of implementing that 5% (or whatever) directly in the on-chip control signals of the native machine.

In practice, this potential has seldom been realized. Very few processor implementations have ever allowed user-defined microcode, and in the few that did—such as DEC's VAX 11/780—the capability usually went unused. This is the result of the difficulty of writing microcode, the lack of software development tools, and the inefficiency of on-the-fly microcode reloading.

The strategy being pursued by IMS is to use the flexibility of microcode to emulate popular instruction sets, and to support custom microcode for system differentiation and performance enhancement. No microcode reloading for individual application support will be provided; instead, custom microcode will accelerate the standard system functions. For the present, IMS is focusing on pen-based applications, in which the 3250 can be used to accelerate the user interface and handwriting recognition software. Other potential applications cited by IMS include DSP for modems, compression/decompression algorithms for file I/O, and support for graphics routines such as BitBLT and drawing commands for Windows.

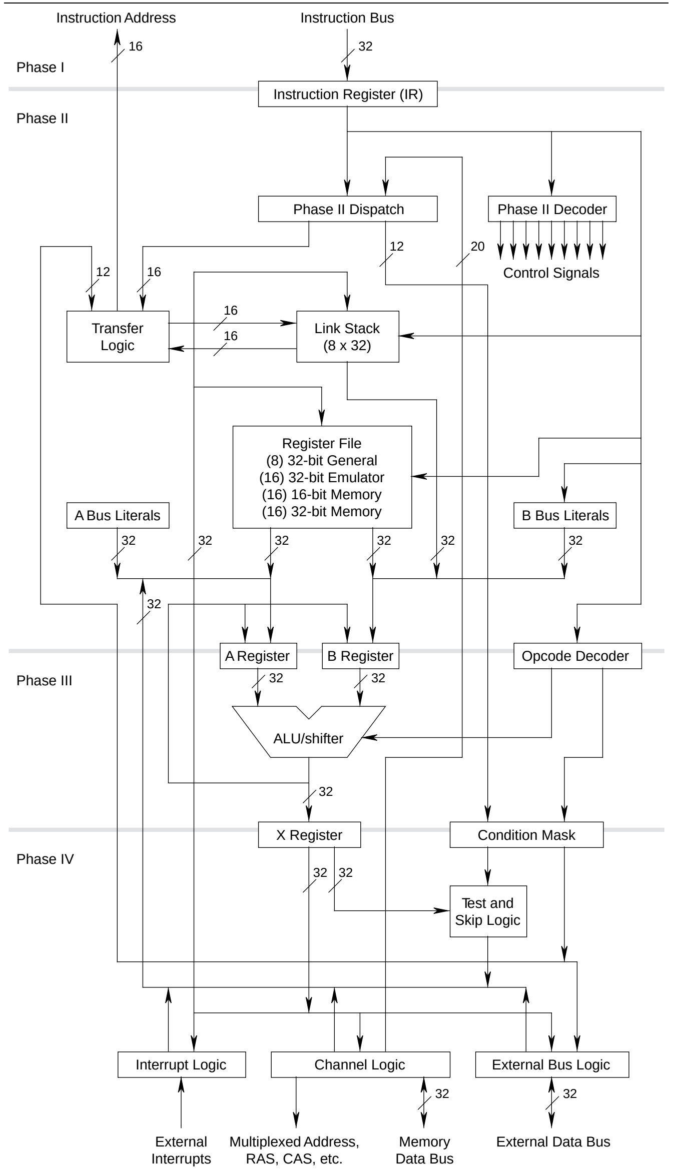

Figure 1. Block diagram of the IMS 3250P.

## Architecture

Figure 1 shows a block diagram of the IMS CPU. It is a 4-stage pipeline, as indicated by the division lines. Initially, IMS plans to have two versions of the 3250. The first version, called the 3250P, will be a gate-array implementation with external microcode. This is intended as a microcode development vehicle that will be sold as a board-level product for the PC/AT and Macintosh II. The second version will be a standard-cell implementation with an on-chip 16K × 32 microcode ROM. The CPU itself (not including ROM) is about 50K gates.

A limited ability to correct errors in the microcode ROM is provided in the form of patch registers. Eight 15-bit patch address registers and eight 32-bit patch data registers allow replacement of up to eight microinstruction words.

The figure shows the 3250P architecture, with its ports for an external microcode memory and an external host bus interface. These are deleted in the ROM-based version, which communicates with both its DRAM and expansion bus through the block marked “Channel Logic.” This block includes an MMU with a 32-entry TLB. The MMU handles base and extent checking in hardware, as well as 486-compatible privilege-level checking; microcode is responsible for reloading the TLB on a TLB miss. The TLB has tags for the complete 32-bit virtual address but provides only 24 bits of physical address, limiting the physical address space to 16 Mbytes.

The register file contains eight general-purpose 32-bit registers, sixteen 32-bit emulator registers for use as the general registers of the emulated instruction set architecture, and 32 “memory” registers (sixteen 16-bit and sixteen 32-bit) for the internal registers of the emulated machine.

Microsubroutine call and return is implemented using the link stack, which is an 8-level stack for holding return addresses. It is 32 bits wide, even though addresses are 16 bits, so that it can also be used as an argument stack.

Power management is available through a clock-scaling mechanism, which allows software to slow or stop the clock to 98% of the chip (about 2% of the chip requires a constant-frequency clock). No specifications for active or standby power are available at this time. The clock-rate target for the initial version is 40 MHz.

## Instruction Set

The instruction formats are shown in Figure 2. It is based on a three-operand assignment of the form T := A LOp B, in which T is the destination operand, A and B are source operands, and LOp is an operator. T and A must be registers; B can be a register or a 16-bit literal. When B is a register, the right-hand side of the format can be a separate instruction with its own ROp operator and 12-bit address operand. The right-hand side instructions are mostly conditional branches, data movement instructions between the CPU and DRAM or other external devices, and assignment operations involving registers outside of the general-purpose and emulator register sets (e.g., the “memory” registers).

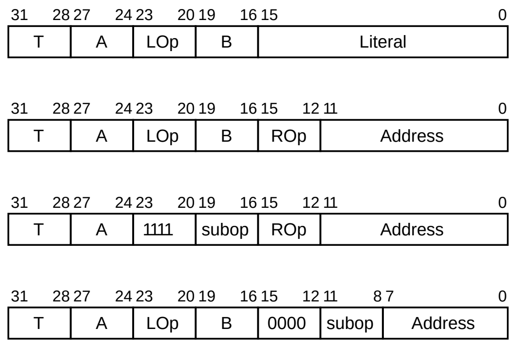

Figure 2. IMS 3250 instruction formats.

One LOp encoding, 1111 (binary), takes over the B field as an extension to the opcode. This format is used for unary operators. Similarly, an ROp encoding, 0000 (binary), extends the selection of right-hand side instructions by taking four bits from the address operand.

The left-hand side instructions include a complete set of arithmetic and logical operators, with a particularly rich selection of rotate, shift, sign-extension, and fill operators. All operations are basic ALU functions that can be performed in a single cycle; no multiplication or floating-point arithmetic is provided. There is, however, an instruction designed to accelerate floating-point emulation. The count-leading-repetitions instruction returns the number of leading bits that have the same state as bit 31; this instruction is useful in normalizing a floating-point result.

## Emulation Support

The 3250 has some unusual features for faster emulation of foreign instruction sets. Specifically, these instructions support emulation of the target machine’s condition code register and acceleration of the instruction decode mechanism.

Condition codes are a major problem for rehosting binary instruction code. For example, the 80x86 family includes parity and half-carry (i.e., carry out of the low four bits of the ALU) flags in its condition codes register. These flags are rarely needed for their intended functions, but they must be synthesized any time software reads the condition codes register. Worse yet, some software uses the fast condition code testing available for these obscure flags to implement performance-sensitive state machines.

A right-hand side instruction, the alternate registers instruction, redirects the left-hand side instruction from the general-purpose register set to the emulator register set. It also selects an alternate condition codes register that has the same format as the emulated machine. Control bits in a system configuration register select whether the emulated machine has the condition code format of the 486 or the 68040. The native condition codes of the 3250 continue to be updated, even while operations are performed on emulation registers.

Instruction decoding is simplified by the decode and dispatch instruction. In both the 486 and 68040 the opcode fields are not always contiguous. The decode and dispatch microinstruction reads a high-level instruction from the application's code space, extracts the opcode bits, compacts and right-justifies the opcode bits, then does a multiway branch. The decode and dispatch instruction is a right-hand side instruction that includes three separate fields for directing the T, A, and B operands of the left-hand side instruction to either the general-purpose or emulation register sets.

There are separate flavors of the decode and dispatch instruction, depending on whether the system is configured for 486 or 68040 emulation. In addition, within each emulation mode, there are two flavors of the instruction called DD0 and DD1. The DD0 instruction resets the instruction fetch mechanism, while DD1 continues the instruction decode from where DD0 left off. The DD0 instruction is intended to initiate the interpretation of the next emulated instruction, so it typically appears on the right-hand side of the last microinstruction in the interpretation of the previous emulated instruction.

If the emulated instruction is a 486 instruction, it may have a scaled-index byte or a mod-R/M byte (i.e., a general-addressing mode specifier). In this case, the DD0 instruction will interpret the s-i-b or mod-R/M byte so that argument prefetching can occur, and the DD1 instruction will interpret the core instruction.

## Memory Controller

The memory controller supports two channels: a read-only channel for reading instructions from the application code space and a read/write channel for access to operands. The channels do their own address generation, allowing single-cycle access to consecutive addresses after the channel is set up. External data is byte-addressable in 8-, 16-, 32-, or 64-bit words, but it is made available to the CPU in a form that is right-justified and zero-extended to 32 bits. Channel operations such as prefetching can proceed in parallel with the CPU.

A channel is set up by programming the channel registers. There are base and extent registers for implementing bounds checking, and an address register for autoincrement or autodecrement addressing. A parameter register allows programming the address step size and data width, and a status register reports details about the previous channel transfer (used chiefly to support the 486's debugging features).

The channel logic includes a DRAM controller that supports up to 16 Mbytes of DRAM. On-chip drivers for the RAS, CAS, write enable, and multiplexed address signals minimize external package count.

## Price & Availability

General availability of the 3250P is planned for third quarter as part of a development system. No pricing is available on the development system alone; it will be provided as part of a technology licensing agreement. In quantities of 10,000, the ROM-based 3250 is expected to cost about $45.

IMS, 23842 Hawthorne Blvd., Torrance, CA 90505; 310/375-4700; fax 310/378-7643.

## Conclusion

IMS expects the 40-MHz version of the 3250 to be roughly equivalent to a 386DX running at 33 MHz on the execution of 80x86-family binary code, which seems optimistic. They expect 10–20× improvement on functions moved into microcode.

The 3250's likely real-world level of performance on x86 code is adequate for laptops and portables, but it is slow by the current standards of the desktop market. IMS is considering a high-end cache-based version of their architecture, but competing with future Intel (and Intel-compatible) CPUs on execution of 80x86 code is not likely to be a winning strategy. By the time they could bring a high-end chip on-line, the next generation of 80x86-compatible chips will be available, raising the goal that must be reached to be competitive.

IMS is addressing a niche where emulated performance at the low-end of the 386 range is acceptable, and fast native performance in the execution of compute-intensive proprietary features is desirable. This strategy is attractive because it could enable hardware vendors to differentiate their products with proprietary microcode, and differentiation is sorely needed in the intensely competitive PC-compatible market. Another potential market is in systems that could run both DOS and Macintosh applications, assuming that one of the "clean room" Mac toolbox suppliers succeeds in getting its product to market.

While the IMS approach is innovative, competing with the spectrum of x86 and RISC microprocessors is a daunting challenge. IMS lacks the resources to compete head-on in the microprocessor market. Its chance for success lies in finding one or two well-heeled system makers that might fund the continued development so they can gain access to some unique technology in an increasingly commodity-like marketplace. IMS might also find customers among semiconductor suppliers seeking 386-compatible CPU core technology. In any case, prospective users are likely to wait for IMS' promised demonstration of 386 protected-mode operation, planned for September.♦