A high-frequency custom CMOS S/390 microprocessor

by C. F. Webb
J. S. Liptay

The S/390® Parallel Enterprise Server Generation 4 processor is an implementation of the IBM ESA/390™ architecture on a single custom CMOS chip. It was designed on a blank slate after consideration of remapping either a prior CMOS design or a prior bipolar design. It uses a straightforward pipeline both to achieve a fast cycle time and to speed the design cycle. The complex instructions are implemented using highly privileged subroutines called millicode. To achieve high data integrity while maintaining a high clock frequency, the chip contains duplicate I- and E-units which perform the same operations each cycle and have their results compared.

Introduction

A major element of the transformation of mainframe computing is the transition from bipolar emitter-coupled logic (ECL) to complementary metal-oxide semiconductor (CMOS) logic. This change is nowhere more conspicuous than in the design of the central processing unit (CPU) of the S/390* Generation 4 (G4) CMOS system. CMOS technology has a tremendous advantage in circuit density, such that a high-performance CPU may be contained in a single CMOS chip, rather than hundreds of ECL chips. In addition, the low switching current of CMOS greatly reduces power consumption and eliminates the need for complex and expensive water-cooled packaging. The result is an S/390 G4 CMOS system with the CPUs, L2 caches, and bus-switching logic packaged in a single multichip module, which delivers performance comparable to that of a 9021 9X2 system in which the corresponding logic occupied 52 modules on 12 boards in six frames.

Designing a CPU, particularly for an architecture as rich as ESA/390*, is a major undertaking. One of the first questions to answer is whether the design should be based on an existing design or done on a clean slate. An existing design has the advantage that it is tested and proven, but the disadvantage that it embodies a set of design assumptions which may not be best for the new design. It is necessary to balance the characteristics of existing designs against the requirements for a new design and against the opportunities offered by a clean-slate design. The considerations which shaped the S/390 G4 CMOS design are discussed next.

Design considerations

Requirements

The requirements for this design were as follows:

0018-8646/97/$5.00 © 1997 IBM

- Full support for the ESA/390 architecture.
- Ability to fit on a single chip in CMOS 5X technology to allow cost-effective packaging of a 10-way symmetric multiprocessor system.
- Extendibility to exploit future CMOS technology improvements.
- Overall performance comparable to that of the 9021 9X2.
- Mainframe-class reliability, availability, and serviceability (RAS).
- Mid-1997 delivery to the marketplace.

Existing CMOS design point

The existing design point used in the first three generations of CMOS S/390 systems was intended to optimize chip area and cost rather than high-end performance [1]. Successive refinement of the design, culminating in the S/390 G3 CMOS system, has yielded significant performance improvements. However, this design utilizes a relatively long cycle time, shallow pipeline, and narrow dataflow, which limits the potential for further extension and, in particular, for exploiting custom high-frequency CMOS design.

Existing bipolar design point

The existing bipolar S/390 CPU designs offered different barriers to their use in this project [2]. These designs had been created for high-end server use, but the design points had been optimized for ECL technology. ECL and CMOS circuits have significantly different electrical characteristics. For example, ECL logic gates allow high fan-in and fan-out with little performance loss, whereas CMOS gates lose performance rapidly with more than three inputs and have much more limited drive capability. This allows complex controls in ECL to have significantly fewer stages of logic than controls in CMOS. Also, greater sensitivity to output loading in CMOS affects the ability to drive signals from one section of the chip to another.

Another difference is that ECL technology is much less dense than CMOS, which means that a processor designed in ECL is spread across multiple chips on multiple modules. Such a design must allow for the different delays involved in crossing such packaging boundaries.

These factors lead to differences between CMOS and ECL designs in overall processor structure, the partitioning of the pipeline into machine cycles, and the organization of dataflow and control logic.

Opportunity for custom design

The availability of custom design techniques for CMOS circuits adds another dimension to the design process. Custom design allows CMOS logic to be built in a smaller area, with faster timing, than is possible using a library of standard logic gates. This is done by designing at the device (transistor) level and carefully tuning each element. However, custom design requires considerable designer time and is difficult to modify once completed. Therefore, it is best suited to regular structures, such as arrays and arithmetic logic, where large functions can be built by repeated use of smaller components, and pieces can be physically arranged to minimize wiring delays between and within functional elements. The logic which controls the sequencing and execution of instructions is less suited to custom design because of its irregularity and the need to change it late in the design cycle to correct problems found during functional verification. Since it cannot be custom, but must run at the same clock cycle as the custom logic, it must be less complex than the control logic in recent bipolar S/390 processors.

Detection and recovery of hardware faults

One hallmark of "mainframe" computing is its robustness, even in the presence of hardware failures. A key aspect of this is the detection of such failures, generally through the use of redundant logic. In dataflow and address flow paths, this has traditionally been done via parity or error-correcting code (ECC). This is straightforward for some design elements (such as those where checking codes are passed through unchanged), but in other elements it adds complexity and adversely affects both chip area and the time required for the function. Custom circuit design exacerbates this situation, because the redundant checking logic disrupts the regularity of functional elements such as adders and bit rotators.

Fault-detection coverage is typically lower for control logic than for dataflow logic. This is because it is more varied in its structure, and commonly practiced procedures for checking it have never evolved. In some cases invalid-state checkers are used, but these usually cover only a small portion of the potential hardware faults. In other cases, functions are implemented twice and the outputs of the two copies are compared.

Once a fault has been detected, the processor must recover from it and continue processing (assuming the fault was transient). Past bipolar designs have used complex hardware controls interacting with a service processor. This has been a difficult area to design correctly and verify; for a new processor, a simple yet complete mechanism is needed.

Microcode control store

A key feature of bipolar S/390 processor designs is the use of horizontal microcode (a form of Licensed Internal Code) to perform complex functions. This code is contained in a special array known as the control store, which has grown as the ESA/390 architecture has grown. A study of remapping a 9121 CPU showed that the control store would have occupied over 30% of the available CMOS chip area. Further, the high performance of horizontal microcode depends on accessing the control store in one cycle, including the selection of one from a set of possible next instructions. This access path was roughly 50% greater than the longest path in the custom data flow in the 9121 remapping study.

- *Special register updates*

Implementing the ESA/390 architecture requires a large set of special registers to control instruction processing. These include not only the ESA/390 control registers, but also such elements as the prefix register, CPU ID, interruption controls, and interpretive execution controls. To minimize timing delays, a high-performance processor needs copies of parts of these registers distributed throughout the processor, close to the logic they control. Updates to these registers are infrequent, but must be propagated in a timely way. Bipolar designs have typically used dedicated wires for each register or field. If this were done on a CMOS chip, it would add significant wiring complexity; therefore, a more efficient distribution mechanism is needed.

- *Complex clocking*

Bipolar processors have often relied on complex clocking schemes to align and balance the various pipeline stages, particularly around large arrays. This increases the effect of clock skew on the machine cycle time, since each adjustment to clock arrival time adds noise to the clock signal, and since each critical interval between clock edges must account for skewed arrival times. For a CMOS processor, with greater variability in logic gate delay, this has a greater detrimental effect on machine cycle time; thus, a simpler clocking scheme is needed.

- *Cache access path*

A critical path in every processor is the access to the L1 cache, which includes the access of the translation lookaside buffer (TLB), the access register translation lookaside buffer (ALB), the cache directory, and the cache array itself. To fit with the simplified clock scheme, this path must be considerably different from previous bipolar implementations.

Design decisions

The net result of the above considerations was a blank-slate design, with its principal characteristic being a very fast cycle time of 3.5 ns in CMOS 5X (later changed to 3.2 ns in CMOS 6S). This was to be achieved even at the expense of an increased number of cycles per instruction executed, and dictated a simple pipelined design. In addition to its other benefits, this simple structure was expected to, and has, reduced the number of errors in the design; this has been instrumental in meeting the mid-1997 delivery to the marketplace.

Custom CMOS design is used throughout the array and dataflow portions of the design, and the simple pipeline has helped to keep the control logic paths within the target cycle time.

The I-unit and E-unit designs contain no checking, and instead are duplicated with their outputs compared. Although costly, this removes checking circuits which could have limited the cycle time and would have made the custom design more difficult, particularly in the arithmetic logic and bit rotator. It also provides more comprehensive checking than standard techniques, particularly for the control logic. The buffer control element (BCE) uses standard parity and ECC checking because it primarily moves data around without manipulating them.

The pipeline is one cycle longer than has been used in recent bipolar S/390 processors. This is necessary in order to have the cache clocking aligned with the rest of the processor. It also allowed some movement of function between cycles, which has helped to achieve the cycle time and to get all pipeline stages aligned to a single clock.

Since it was impossible to use a standard control store because of cycle time and space, complicated functions are implemented by highly privileged S/390 subroutines called millicode (a different form of Licensed Internal Code). The millicode has a separate set of general registers and additional instructions to manipulate system status which are not available in the ESA/390 architecture.

To make recovery straightforward and comprehensive, all system status which exists between instructions is collected in a checkpoint array. This array can be used to reinitialize the processor and retry an instruction whenever there is a hardware error, all under hardware control. Also, it can be accessed by millicode, which gives millicode great generality and power.

Figure 1 shows the major elements on the S/390 G4 CMOS chip. The outputs from the duplicate I-unit and E-unit go to a compare circuit before going to the register unit (R-unit) and BCE. All data produced by the E-unit, whether to be stored or written to an internal register, come over a single bus from the C-register, simplifying the compare. The storage addresses from the I-unit, and the control signals associated with the data and addresses, are also compared.

The BCE contains a single L1 cache, the address translation logic and lookaside buffer, the store queue and buffer, and the logic for communicating with the L2 cache.

The R-unit contains the checkpoint array and the logic for retrying in the event of an error. ECC is used for the data in the checkpoint array.

At the heart of the design decisions described here is a comparison of different design points. Such a comparison is difficult to make when the design points are in different technologies, since it requires a substantial effort to map

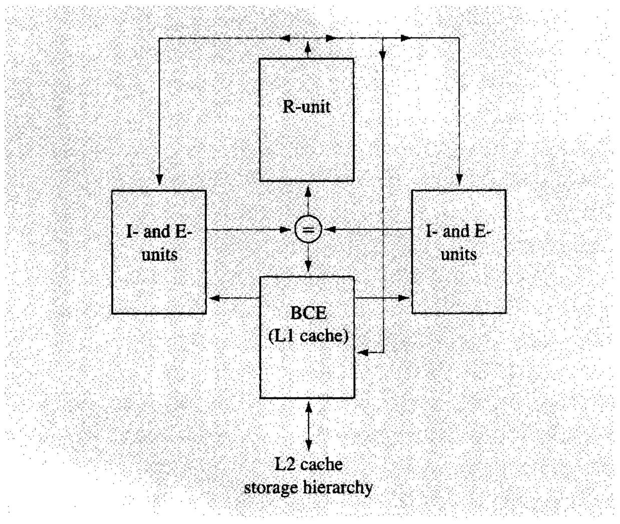

Figure 1

Overall chip structure.

them to a common technology, and there is never enough time to do this properly. Nevertheless, on the basis of a combination of measurement, analysis, and projection, the designers of the S/390 G4 CMOS processor believe that this design realizes a 70% cycle-time advantage compared to the S/390 G3 CMOS processor when normalized to the same CMOS technology, while requiring 30% more cycles to perform the same work. This gives the S/390 G4 CMOS design an overall CPU performance advantage of 30%. In comparison to the ES/9000* Model 9021 processor, were that mapped into CMOS, it is believed that the S/390 G4 CMOS cycle time is approximately 60% faster, and that S/390 G4 CMOS requires about 60% more cycles, yielding roughly equivalent performance. However, the ES/9000 Model 9021 is a complex, superscalar, out-of-sequence processor, while the S/390 G4 CMOS is a much simpler design, requiring less chip area and enabling a more aggressive product design schedule. It should be noted in both of these cases that this is only one component of system performance, which is also influenced by storage hierarchy effects and the particular workload being applied.

Buffer control element

The BCE provides access to ESA/390 storage via a single store-through 64-kilobyte L1 cache which holds both instructions and operands. It was not split into separate instruction and data caches in order to save space and simplify the controls. The line size is 128 bytes, and the lines are grouped into 128 sets of four lines. Even and odd doublewords are in separate data arrays (interleaves) that are separately accessible. There is also a 32-kilobyte read-only storage extension to the cache, which is described in the section on millicode. The BCE also includes hardware for performing ESA/390 address and access register translation, including support of guests executed under ESA/390 interpretive execution, and a 256-entry TLB to remember the virtual-to-absolute translations for recently accessed pages.

The conceptual requirements for fetching data from the cache are to translate the access-list-entry token (ALET) to a segment table origin (STO) using the ALB, translate the virtual address to an absolute address using the TLB, look in the cache directory to see where the data are, and then read the data from the cache. The practical problem is that if those four steps are done sequentially, the cache access will take much too long; therefore, cache designs usually do some of the steps at the same time. In the case of the S/390 G4 CMOS processor, the ALET translation is done during the cycle in which the address is being calculated, and the remaining steps are done simultaneously. At a minimum this means reading two entries from the TLB (because it is two-way associative), four entries from the cache directory (because it is four-way associative), and four doublewords from the cache data array (because it is four-way associative). Then the appropriate addresses are compared to see whether there was a cache hit and to control gating the correct data.

However, this is complicated by the fact that two of the address bits (18 and 19) needed to read the cache are subject to translation, which means that they are part of the TLB output. Since it would take too long to complete the TLB access before starting the cache, an alternative is needed. Two approaches have been used on past processors, and a third approach is used on the S/390 G4 CMOS processor.

The first approach is to read four times as many cache directory entries (sixteen), and four times as many doublewords of data (sixteen), to cover all four possible values that absolute address bits 18 and 19 could have. Then all needed information is available, and it is just a matter of additional compares and gating. Unfortunately, that extra logic would make it impossible to meet the target cycle time.

The second approach is to organize the data in the cache according to virtual address, in which case virtual address bits 18 and 19 are used for accessing the cache, and there is no need to read four times as much information out of the directory and data array. This has other drawbacks. When a cache miss occurs, it is not known whether the data are really not in the cache, or just in a different place because they were previously referenced using a different virtual address (called a synonym). While that is infrequent, it is still necessary to go looking for the data on every cache miss and to move them if they are found. This movement can be accomplished either by reading data from the first place and writing them to the second, or by invalidating data in the first place and loading them from the L2 cache into the second place. Also, in a multiprocessing system, when a cross-interrogate address (an absolute address) is received, there are now four times more places to look for it. There are several alternative ways to deal with cross-interrogation, which represent different trade-offs between speed and cost, none of which were acceptable.

The S/390 G4 CMOS design is different from the first two alternatives. Since the data are organized by absolute address, the problems associated with the second approach do not exist. To avoid the need to read from so many extra cache directory and data array locations, a new structure called an *absolute address history table* (AAHT) is introduced for operand requests. The AAHT uses the values of the base and index GRs specified for an address calculation to select a predicted value of absolute address bits 18 and 19. The base GR is used when one is specified; otherwise, the index GR is used. During the address add cycle, bits 12 to 19 of the selected GR value are used to read one of 256 entries in the AAHT. This entry contains a predicted value of absolute address bits 18 and 19, which are latched at the beginning of the cache access cycle and used to address the cache. For instruction fetches, a prediction for bits 18 and 19 is also made, but using a small set of registers and a simple algorithm; this is possible because instruction fetching is more localized in its addressing pattern.

As with any prediction mechanism, there is a possibility of misprediction. The predicted absolute address bits are compared to the actual absolute address bits in the matching TLB entry; if they do not match, any cache hit is suppressed, the AAHT is updated, and the storage request is recycled in the BCE using the correct absolute address bits 18 and 19.

In addition to accessing the cache with a logical address, it can be accessed by cache physical address. Whenever a fetch is made using a virtual or real address, the physical location of the line in the cache is remembered, and additional fetches can be made from that line without using the ALB, TLB, or cache directory. Fetches made using a remembered cache physical location are called "continuation fetches." This allows multiple requests to be processed by the BCE in a single cycle. For example, on the same cycle it is possible to make an operand test request to look for access exceptions, and also fetch doublewords of data for two different I-buffers.

The strategy for handling stores is that the L1 cache is store-through and the L2 cache is store-in. Operand stores are written into the L1 cache as each store instruction is executed, and are merged with current data from the cache to form a full doubleword. These doublewords have ECC applied, are saved in the store buffer, and are sent to the L2 cache after the storing instruction has completed successfully. The store buffer can hold up to 64 doublewords from eight different cache lines. This store design allows the L1 cache to just be reset during a recovery action, and that in turn allows the L1 cache to have only parity checking on its data.

The connection to the L2 cache and the rest of the storage hierarchy is by a single quadword bidirectional bus. Parity checking is used for the transmission of data from the L2 cache, and ECC is used for the store buffer and the transmission of data to the L2 cache. All storage updates are forwarded to the L2 cache, which uses ECC. The clock rate of the processor and BCE is twice as fast as that of the L2. The L2 cache is described in the companion paper by Mak et al. in this issue [3].

Instruction and execution units

The I- and E-units consist of three parts: the instruction unit, the fixed-point unit (FXU), and the floating-point unit (FPU). There are two separate execution units because the specialized nature of the floating-point logic prevents it from being integrated with the FXU; however, only one of the execution units can be executing at a time. The FPU is described in a companion paper by Schwarz et al. in this issue [4].

Figure 2 is a simplified dataflow diagram of these units. The I-unit contains four I-buffers, each containing four doublewords. Instruction fetching can occur simultaneously along the decoding path, and along two alternate branch paths, with a different I-buffer used for each path. Each I-buffer operates with only a single cache line at a time, holding up to four consecutive doublewords from that line. The initial fetch for an I-buffer uses a logical address, but all subsequent fetches are continuation fetches. When the data in the earliest doubleword are used, the doubleword is reassigned 32 bytes farther on. Fetching proceeds in this manner until the end of the cache line is reached; then a different, "linked" I-buffer is started to fetch the next line, and the current I-buffer becomes inactive when its data are exhausted.

One instruction can be delivered to the I-register for decoding each cycle, and for most instructions, decoding takes only a single cycle. Instructions are always decoded in the order in which they appear in the instruction stream. When a branch is encountered, an I-buffer is started along the target path, and fetching continues along the sequential path. A guess is made as to whether the branch is taken, and decoding proceeds in whatever direction is guessed. When the branch is resolved, if the guess was wrong, the operations along the wrong path are discarded, and decoding changes to the correct path.

During the decode cycle, a variety of control signals are generated to control instruction processing, such as

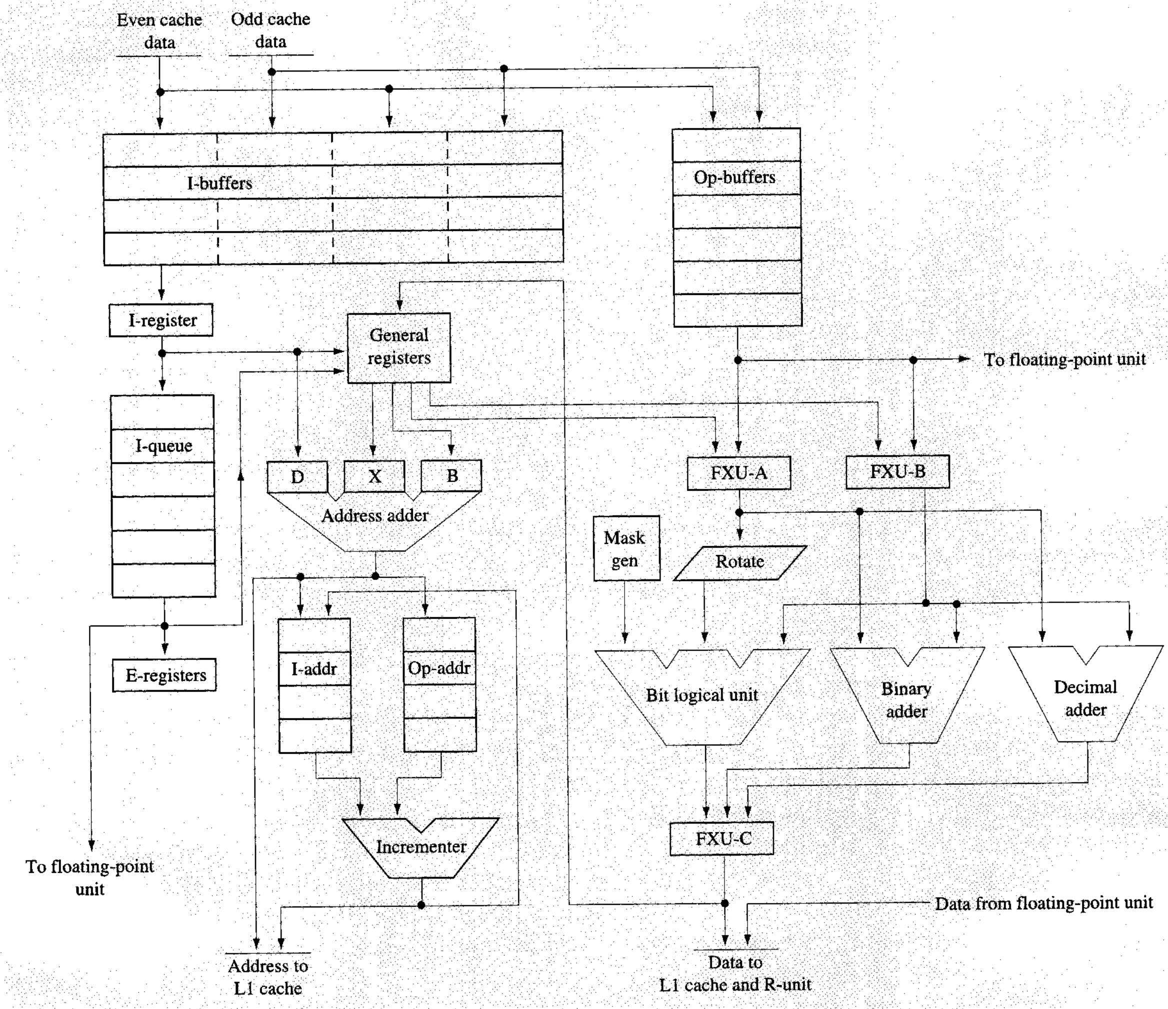

Figure 2

Dataflow diagram for I-unit and fixed-point unit.

determining whether millicode is required to execute the instruction, what should be done with the effective address, and whether certain exceptional conditions are present. In addition, the X and B general registers are read in preparation for an address calculation. At the end of the decode cycle, those values are in registers at the input to the address adder, and the address calculation occurs on the next cycle if there are no interlocks to prevent it. If there are interlocks on the address calculation, further decoding is also stopped.

Also at the end of the decode cycle, the instruction is placed in the I-queue, where it remains until it is executed. Instructions are placed in the I-queue, and removed from it, in program order; there is no out-of-order execution.

The address adder feeds four I-address registers, and four op-address registers. The I-address registers are associated on a one-for-one basis with the I-buffers. Whenever an I-buffer is started, its associated I-address is also set. For a branch target, it is set from the address adder, and for a linked I-buffer, it is set from the address incrementer.

The op-addresses are used for operand requests. They are assigned to individual instruction operands, and are used to make the type of references needed for that operand. If the operand is only stored into, one reference is made for each cache line to notify the BCE that stores will be coming. If data are needed from the operand, fetches are made to each doubleword; the first fetch in each cache line uses a logical address, and subsequent fetches are continuation fetches. If an instruction references two operands (as many SS instructions do), two op-addresses are assigned. An op-address fetches the entire operand to which it is assigned, regardless of how many cache lines it occupies. An op-address is never used for two successive instructions: even if they require the same or consecutive data, a second op-address is assigned for the second instruction.

The order of operand fetching is strictly defined. All fetches for instruction N occur after operand fetches for instruction $N-1$. Within an operand, fetches occur from left to right; if an instruction has two fetch operands, doublewords from those operands are fetched alternately. Operand data are held in the op-buffers until they are needed for execution. The op-buffers are assigned as needed to hold fetched data, and do not have a special relationship with any op-address.

The I-queue can pass one instruction per cycle to either execution unit (but not both). Instructions are executed in program order by them, and whichever unit is executing must finish before an instruction can go to the other.

The ESA/390 instructions are of three types: 1) those executed in the FPU, 2) those executed in the FXU, and 3) those executed by millicode. There are also special instructions, available only in millicode and not directly available to ESA/390 programs, which for the most part allow the manipulation of the system state. The FXU executes all of these except the floating-point instructions.

The FXU includes a binary adder, binary-coded-decimal (BCD) adder, bit rotator, mask generator, and bitwise logical and merge elements; all of these except the BCD adder are 64 bits wide. Since the internal architecture includes a mixture of 32-bit and 64-bit operations, the high and low 32-bit words are interleaved in the layout of the 64-bit dataflow, allowing efficient handling of both types of operation, including the forwarding of results from one operation to the next. The FXU also handles condition code and interrupt processing.

Register unit

The register unit (R-unit) performs a variety of functions associated with ESA/390 control operations, hardware fault detection, and error recovery. It is dominated by a custom CMOS register file known as the checkpoint array, which buffers all control states, including various mode controls, interrupt controls, and interpretive execution controls. It also buffers instruction addresses used by millicode and hardware, and all architected facilities, including ESA/390 GRs, access registers (ARs), floating-point registers (FPRs), and control registers (CRs). All of these are mapped into an 8-bit register address space, with room for 128 32-bit and 128 64-bit registers.

As instructions (either ESA/390 or millicode) are executed, their updates to the processor state are sent to the R-unit, where the results from the two sides are compared for equality and placed in an update buffer. When an instruction completes with no hardware faults detected, the contents of the update buffer are written to the checkpoint array.

The R-unit also includes special logic for the ESA/390 timing facility, instruction address and exception registers, and interfaces with the service processor and an auxiliary processing unit; the state registers for these are handled in a manner analogous to the main checkpoint array.

Because some of this state information is needed to control hardware operations in the I-unit, E-unit, and BCE, local copies of that state information are provided there. To maintain correct values in these copies, the E-unit output bus is routed to each unit along with the R-unit register address being written. Logic in each unit monitors this address and updates any local copies of R-unit registers from data on the E-unit output bus. Thus, a single bus conveys updates for all register and state updates to all units in the processor, and has a much smaller impact on chip wiring than would have been required for many small dedicated mode and control lines.

Millicode mode

In a complex instruction set (CISC) architecture such as ESA/390, it is simply not feasible to use hardware controls for all of the instructions; design time and debug time both preclude it. Also, as was noted above, conventional horizontal microcode was also not feasible for the S/390 G4 CMOS processor, because of both chip area required and cycle time. This led us to use highly privileged subroutines in 390-like code, which we call millicode. For this to be successful, it is necessary to have the processor run efficiently while it is in millimode, enter and leave millimode rapidly, and give the millicode the capabilities it needs without adding excessive complexity to the hardware.

To run efficiently, the millicode is provided with its own set of general registers, access registers, and control registers. It therefore has immediate access to its own set of registers and has no overhead related to saving and restoring the S/390-architected registers. For those instances when the millicode must access an S/390-architected register, there are special instructions which allow it to be done quickly. Also, there are other special instructions which allow millicode to perform important operations quickly.

To enter millicode quickly, all instructions which are implemented with millicode subroutines are treated by the hardware as unconditional branches to fixed addresses in

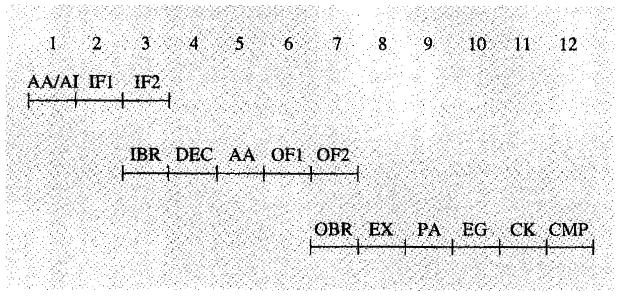

Figure 3

Pipeline timing.

hardware system area storage. They take about five cycles to execute, since they save a variety of status information, such as its effective addresses and the instruction text, in millicode general registers. Thus, the processor executes one "branch," and it is in millimode, executing the first instruction in the subroutine, with its needed status information loaded into its general registers. The execution of this "branch" into millicode occurs overlapped with both preceding and following instructions to minimize pipeline disturbance.

The exit from millicode is accomplished with a millicode end instruction which is an unconditional branch to the updated S/390 instruction address. This is also an overlapped instruction which causes minimal pipeline disturbance, and is even faster than millicode entry because there is no status to save.

The millicode can make use of the full set of hardware-implemented ESA/390 instructions. Additional instructions are defined to allow direct millicode access to specific hardware facilities and to accelerate specific millicoded operations. These instructions are modeled on ESA/390 instruction formats, so that little unique hardware is required to support them. One group of special millicode instructions provides both read and write access to the R-unit register space, using the 8-bit register addresses. Since this register file contains all processor states, a relatively small number of special instructions provide millicode with access to and control over the entire processor state. This is an economical mechanism which gives the millicode generalized, powerful control over the processor.

Millicode is also used for interruption processing. When an interruption occurs, the processor stops executing the current instruction stream and forces the processor into millimode at a fixed address. This is essentially the same as millicode entry to simulate an instruction, but it is caused by detecting an interruption condition rather than by encountering an instruction that must be simulated.

One disadvantage of millicode is that the code occupies space in the cache, potentially displacing ESA/390 data and increasing the number of cache misses. This effect is minimized by adding a 32KB read-only storage (ROS) array in the BCE, as an extension to the cache, which contains frequently executed millicode routines. When a fetch is made from the portion of HSA storage corresponding to the ROS, the BCE takes data from the ROS rather than bringing that line into the cache. ESA/390 instructions for which the millicode is in ROS are detected when they are decoded, and the appropriate ROS address is used rather than the default millicode routine address for that opcode. The reason for using ROS rather than increasing the size of the cache is that the ROS has a much greater density than the cache data array. While there was room on the chip to add 32 KB of ROS, there was not room to expand the cache by 32 KB. If there is an error in a ROS routine, the erroneous routine may be patched (millicode entry reverts to the default address for that opcode) without affecting any other ROS routines.

Instruction pipeline

The basic pipeline of the S/390 G4 CMOS processor is shown in **Figure 3** for an RX-format add- or store-type instruction. The three lines in the figure represent instruction fetching, I-unit processing, and E-unit processing. Normally, instruction fetching occurs well in advance of decoding, and there is a significant time gap between the first and second lines in the figure. The illustrated timing occurs when the processor starts with the instruction buffers empty.

In the AA/AI cycle, the I-unit passes the starting address through the address adder or incrementer and sends an instruction fetch request to the BCE. This request will return up to 16 bytes of data, depending on cache interleave availability.

In the IF1 cycle, the BCE accesses the cache, TLB, and directory and also latches the instruction data. These data are normally returned to the I-unit in cycle IF2, but are subject to the same delays described below for operand fetches.

In the IBR cycle, the instruction is gated from the I-buffer, or bypassed directly from the cache bus, to the I-register.

In the DEC cycle, the instruction in the I-register is decoded and sent to the address adder and I-queue. Also, the GRs are accessed to obtain the base and index address values designated by the instruction (if any), and the base AR is read for use in selecting the operand address space as needed.

In the AA cycle, the operand address is computed and sent to the BCE with a fetch request. In parallel, the base AR is used to read the ALB, which with the operand address controls selects the address space for the access. Also in cycle AA, bits from the base or index GRs are used to access the absolute address history table (AAHT) to predict absolute address bits 18 and 19.

In the OF1 cycle, the TLB, cache directory, and cache data arrays are read to obtain the data and check for access exceptions. If all is well, the data are latched at the end of this cycle and sent to the operand buffers in cycle OF2. If a TLB or ALB miss occurs, the BCE initiates a translation sequence, at the end of which the TLB or ALB is updated and the cache re-accessed. If a cache miss occurs, the BCE initiates a fetch from the L2 cache. In either case, the BCE signals the I-unit, and the pipeline is extended until the miss is resolved. If the TLB and ALB hit, but absolute address bits 18 and 19 are not as predicted by the AAHT, the request is recycled, and data return is delayed by two cycles (or more if a cache miss is then detected).

In the OBR cycle, the next instruction from the I-queue is sent to the E-unit for execution. The E-unit performs additional decoding of the instruction and sets up controls for the dataflow elements. During this cycle, the operands for the instruction are read from the registers (GRs, ARs, or FPRs) and from the operand buffers as needed; data just being sent from the BCE are bypassed by the operand buffers directly into the E-unit operand registers, including any merging and aligning that may be required. If the storage operands are not yet available, the OBR cycle is allowed to complete, but the execution of the instruction is held at the next stage.

In the EX cycle, the actual execution takes place, and the instruction results (including the condition code) are generated.

In the PA cycle, these results are forwarded to the R-unit and/or BCE, and are broadcast to the rest of the processor to allow updates to local copies of R-unit registers.

The remaining cycles of the pipeline affect only the R-unit and BCE, and are never subject to pipeline stalls. In the EG cycle, the ECC checking code is generated for R-unit register updates, and store operands are written to the cache and merged with the unchanged bytes to form a full DW.

In the CK cycle, the data from the two copies of the E-unit are compared in the R-unit, and parity is checked on all bytes of the merged DW for operand stores in the BCE. ECC is then generated for the doubleword in preparation for writing it into the store buffer on the next cycle. For hardware instructions requiring multiple execution cycles, results are buffered at this stage until all results have been generated.

The last CK cycle for each hardware instruction (ESA/390 or millicode) is followed by a CMP cycle, in which the buffered results are written to the checkpoint array in the R-unit, and stores held in the BCE store buffer are marked as eligible to be forwarded to the L2 cache. The CMP stage is blocked if any hardware fault is detected in any prior cycle.

All stages of this pipeline are the same length, and all are driven by the same system clock. The main portion of the pipeline is one cycle longer than in the 9021 and 9121 processors because of uneven scaling between bipolar and CMOS technology, because of the need for a simple clocking scheme to minimize clock skew, and to move some cache-related functions to the cycles before and after the cache access.

Handling of exceptional conditions

As a rule, high-performance processor designs are optimized for "normal" operations, with minimal circuitry devoted to interruptions and special hardware actions. However, when unusual conditions arise, the processor must respond according to the architectural definition, which in the case of ESA/390 is very precise. This precision allows for highly reliable software, but imposes a significant burden upon the hardware design. In a pipelined processor, information about unusual conditions is often not available until the affected instruction is being executed. Furthermore, with precise program interruption as in ESA/390, the condition causing an interruption must be associated with a specific instruction, and often requires that substantial information be passed from the hardware to the software. All of this must be dealt with without adding significant complexity to the logic or affecting cycle time.

In the S/390 G4 CMOS processor, this is addressed with a relatively simple concept. During normal pipelined operation, instructions are monitored for conditions which might require an interruption or special hardware action. These conditions include program exceptions, program-event-recording (PER) events, and certain operand destructive-overlap conditions for which millicoded execution is required. They are combined into a single exception summary bit which is carried in the instruction queue and passed to the E-unit with each instruction. When the E-unit begins execution in normal pipelined mode of an instruction with this bit set, it blocks the results from that instruction, and the I-unit and E-unit discard that and any subsequent instructions in the pipeline. Then the I-unit is restarted at the current instruction address, in single-instruction mode. In this mode, a single instruction is passed through the pipeline by itself. Detailed exception information is accumulated for that instruction, and the appropriate interruption or special hardware action is taken, after which the I-unit resumes fully overlapped operation. In some cases, the instruction executed in single-instruction mode may complete with no exceptional condition detected, and then the processor simply resumes normal operation. Even with the application of this mechanism to a broad range of conditions, the net effect of single-execution mode on overall performance is about 0.1%.

Detection and recovery of hardware faults

To provide thorough hardware fault detection without complicating the custom design elements and without impact on the cycle time, the I-unit and E-unit are duplicated *in toto* on the S/390 G4 CMOS processor chip. There is no checking at all within these units other than a small number of functional checkers intended mainly to detect millicode errors. The outputs of these units [storage addresses (instruction and operand from the I-unit), operand results (storage and register updates from the E-unit), and exceptional condition status (E-unit)] are sent to the R-unit and/or the BCE, where the values from the two copies are compared. A mismatch is considered a hardware fault. The BCE and R-unit are not duplicated, but are protected from hardware faults by parity and ECC (double-error detection, single-error correction) in the data flow, and consistency checks and local duplication in the controls. Since most BCE operations are byte-coherent, including parity merely adds a bit in the path and usually does not complicate the custom design elements. The checkpoint array in the R-unit is protected by ECC, since it must be reliably correct for hardware fault recovery to be effective. All storage updates are forwarded to the L2 cache immediately after completion of the storing instructions. The entire data path for stores, including the store buffer, the interface to L2, the L2 store stack, and the L2 cache itself, is protected by ECC. Thus, storage updates are reliably preserved across hardware faults, and L1 cache contents do not have to be preserved through recovery from hardware faults.

When a hardware fault is detected anywhere, instruction completion is halted immediately, which blocks updates to the checkpoint array and inhibits operand stores from being forwarded to the L2 cache. The timing of error detection is such that any error which could affect the resulting state of the processor is detected prior to the completion of the instruction (ESA/390 or millicode) which would update that portion of the state. Thus, all instructions that have completed at the point at which an error is detected did so without error, and both the processor state in the checkpoint array and the storage state in the storage hierarchy (including the store buffer, but not the L1 cache) are correct.

After a hardware fault has been detected, a recovery sequence is performed under direction of the R-unit:

1. The BCE forwards to the L2 cache any remaining stores for instructions that have completed.
2. The I-unit, E-unit, and BCE are reset.
3. The ALB, TLB, cache directory, and cache data arrays are purged and written with good parity.
4. An asynchronous interrupt is made pending to log the recovery event and check for any interrupts lost while performing recovery. This interrupt is taken at the next interruptible point, which is after the recovery sequence.
5. Each element of the checkpoint array is read, checked, and corrected via ECC, sent to the E-unit, staged to the FXU output register, and written back to the same address in the checkpoint array, thus cleaning up any correctable errors in that array which might otherwise cause repeated invocations of processor recovery. This uses the same data and control paths as are used for special millicode instructions which read and write registers in the R-unit space. As part of this operation, any local copies of that register in the I-unit, E-unit, or BCE are updated with the known good value.
6. The entire checkpoint array is read again and checked for errors. If any error is found, it is presumed to be a solid hardware failure, since the entire array was just written with good ECC; the recovery sequence is aborted, and the processor is placed in the check-stopped state.
7. The E-unit forces a hardware serialization interrupt, which restarts the I-unit at the current instruction address.
8. Instruction processing resumes, using the state just loaded from the checkpoint array.
9. If another hardware fault is detected immediately, it is presumed to be a solid failure, and the processor is placed in the check-stopped state; otherwise, recovery is deemed successful, and normal processing continues.

This entire sequence is performed under hardware control, using relatively simple control logic and the same data paths that are used in normal operation. Because the processor state is maintained on a hardware instruction (either ESA/390 or millicode) basis, recovery can be performed for faults that occur during millicode operation without any special action by millicode. This eliminates by design the majority of special recovery scenarios which have made instruction retry very complex in bipolar S/390 processors. The only exceptions are millicode sequences that interact with system components outside the CPU, in which case responses may be lost while recovery is being performed. To handle this, a means is provided for millicode to detect that recovery has been performed within a given interval in the millicode routine, so that special action can be taken as necessary.

Conclusion

Through innovative design, the S/390 G4 CMOS processor implements the complex ESA/390 instruction set and achieves high performance while maintaining or improving upon traditional S/390 strengths of reliability, availability, and serviceability. Millicode routines are used to implement complex ESA/390 functions, minimizing the chip area and limiting the control complexity impact from these operations. The duplicate instruction and execution units and straightforward hardware instruction retry mechanism not only achieve high reliability and availability, but also enable efficient custom design. Through application of custom CMOS design, the S/390 G4 CMOS microprocessor has become the high-frequency design leader in IBM, with a clock speed of up to 400 MHz in the laboratory.

Acknowledgments

The logic design technical leaders were Charles F. Webb, John S. Liptay, Barry Krumm, Eric Schwarz, and Mark Farrell, who respectively led the E-unit, the I-unit, the BCE, the floating-point unit, and the millicode portions of the design. Other key contributors included Joe Braun, Mark Check, Phil Emma, Bruce Giamei, Jin Ji, Chris Krygowski, Wen Li, Kai Ann Mueller, Jennifer Navarro, Steve Risch, Kevin Shum, Tim Slegel, Scott Swaney, David Wong, and Mike Wright.

*Trademark or registered trademark of International Business Machines Corporation.

References

1. H. Schettler, K. Getzlaff, K. Klein, C. W. Starke, L. Wilczynski, and A. Bhattacharyya, "A CMOS Mainframe Processor with a 0.5 μm Channel Length," *IEEE J. Solid State Circuits* **25,** 1166 (1990).
2. J. S. Liptay, "Design of the IBM Enterprise System/9000 High-End Processor," *IBM J. Res. Develop.* **36,** No. 4, 713–731 (1992).
3. P. Mak, M. A. Blake, C. C. Jones, and P. R. Turgeon, "Shared Cache Clusters in a System with a Fully Shared Memory," *IBM J. Res. Develop.* **41,** No. 4/5, 429–448 (1997, this issue).
4. E. M. Schwarz, L. J. Sigal, and T. J. McPherson, "CMOS Floating-Point Unit for the S/390 Parallel Enterprise Server G4," *IBM J. Res. Develop.* **41,** No. 4/5, 475–488 (1997, this issue).

Received March 19, 1997; accepted for publication June 30, 1997

Charles F. Webb *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (cwebb@vnet.ibm.com).* Mr. Webb received his B.S. degree in 1982 and his M.Eng. degree in 1983, both from Rensselaer Polytechnic Institute. He joined IBM in 1983 at the IBM Product Development Laboratory in Poughkeepsie in the Processor Performance Analysis organization. In 1987 he joined the Processor Development organization, where he has remained since. Mr. Webb has worked on the ES/9000 processor and the S/390 G4 CMOS processor in the areas of performance analysis and CPU design. He has received seven IBM Invention Achievement Awards and an IBM Outstanding Innovation Award. Mr. Webb is an IBM Senior Technical Staff Member.

John S. Liptay *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (LIPTAY at PK705VMA).* Mr. Liptay received his B.E.E. degree in 1962 and his M.E.E. degree in 1966, both from Rensselaer Polytechnic Institute. He joined IBM in 1965 at the Thomas J. Watson Research Center, transferring shortly thereafter to the IBM Product Development Laboratory in Poughkeepsie, where he has remained since. Mr. Liptay has worked on the System/360 Models 65 and 85, the System/370 Model 168, the 3033, the ES/9000 processor, and the S/390 G4 CMOS processor, all in the area of CPU design. He has received five IBM Invention Achievement Awards, an IBM Outstanding Contribution Award, an IBM Division Award, an IBM Technical Excellence Award, and an IBM Outstanding Innovation Award. Mr. Liptay is an IBM Senior Technical Staff Member and a member of Tau Beta Pi, Eta Kappa Nu, the Institute of Electrical and Electronics Engineers, and the Association for Computing Machinery.