TANDEM

Multiple Instruction Issue in the NonStop Cyclone System

Robert W. Horst
Richard L. Harris
Robert L. Jardine

Technical Report 90.6
June 1990
Part Number: 48007

Multiple Instruction Issue in the NonStop Cyclone Processor[1]

Robert W. Horst Richard L. Harris Robert L. Jardine

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, CA 95014

Abstract

This paper describes the architecture for issuing multiple instructions per clock in the NonStop Cyclone Processor. Pairs of instructions are fetched and decoded by a dual two-stage prefetch pipeline and passed to a dual six-stage pipeline for execution. Dynamic branch prediction is used to reduce branch penalties. A unique microcode routine for each pair is stored in the large duplexed control store. The microcode controls parallel data paths optimized for executing the most frequent instruction pairs. Other features of the architecture include cache support for unaligned double-precision accesses, a virtually-addressed main memory, and a novel precise exception mechanism.

[1]A previous version of this paper was published in the conference proceedings of The 17th Annual International Symposium on Computer Architecture, May 28-31, 1990, Seattle, Washington.

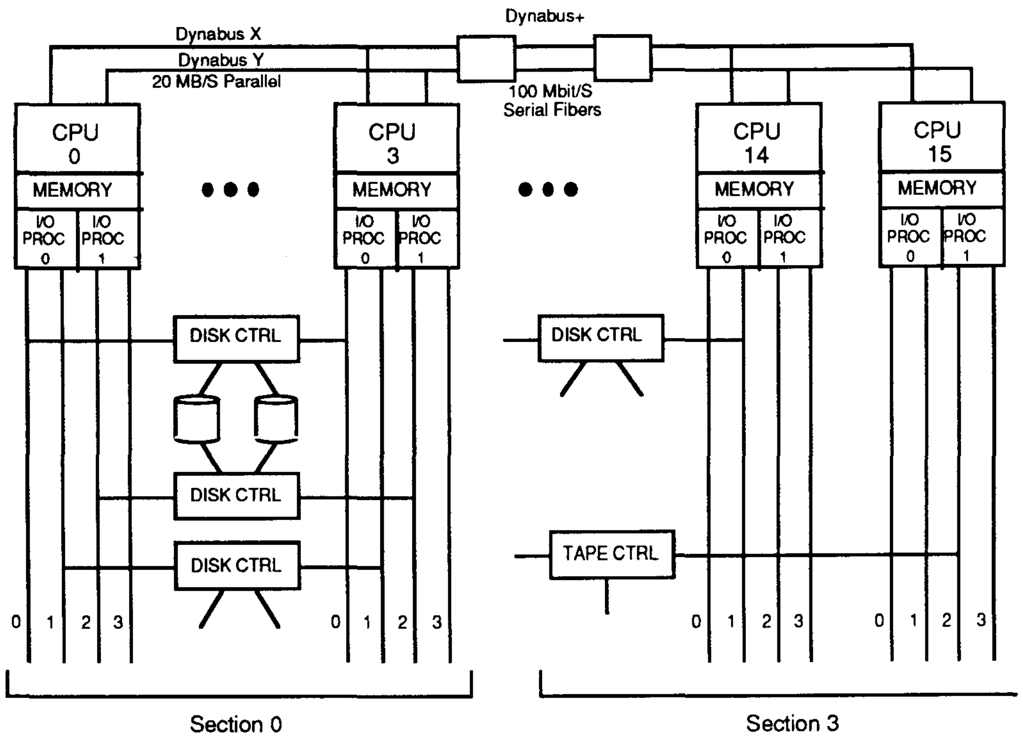

Figure 1. Cyclone System Architecture.

1. Introduction

The NonStop Cyclone system is a fault-tolerant mainframe targeted at transaction processing, query processing and batch. Each system consists of four to sixteen processors that are connected by dual high-speed busses (Figure 1). Sections of four processors may be geographically distributed and interconnected by fiber optic cables. Each processor has its own memory and drives two to four I/O channels. Fault detection is performed primarily by the hardware, and fault recovery is performed by the message-based operating system. The system can tolerate a single fault in a processor, peripheral controller, power supply, or cooling system. Failed components can be serviced on-line without disrupting processing.

Five generations of Tandem computers (NonStop II, TXP, VLX, CLX and Cyclone) are object-code compatible and have been kept current through microcode updates downloaded to writeable control store. The Tandem instruction set has approximately 300 fixed-length (16-bit) instructions, ranging from simple RISC-like instructions to very complex instructions, such as block moves and inter-processor sends, which may take hundreds of clocks to complete. Most operations are zero-address with operands on the top of an eight-word register stack. The basic memory reference instructions are load and store instructions with address displacements relative to a stack pointer or segment base register.

The Cyclone processor is over three times faster than its predecessor. Approximately half of the performance improvement is due to higher clock rates, and the other half is due to the new micro-architecture. Much of the architectural improvement stems from the ability to issue up to two instructions per clock cycle. Other improvements are due to parallel data paths and new designs for the caches and main memory. This paper describes the architectural aspects of the NonStop Cyclone processor. In particular, it concentrates on the features that have been included to support multiple-instruction issue.

2. Overview

In recent years, advances in technology and computer architecture have allowed the design of processors in which simple instructions can be executed in a single clock cycle. Once that point is reached, further architectural performance improvements must be made by executing more than one instruction per clock. Some previous scientific machines were capable of issuing multiple instructions per clock, but this was done through simultaneous execution of integer and floating point operations. When the instruction set can be partitioned into independent operations that share few resources, then it is possible to design independent function units and to assign each instruction to one of these units. Several instructions can be issued to the function units simultaneously [1].

Issuing multiple integer instructions per clock is more difficult because most integer instructions require use of the same resources. Typically, nearly all instructions access the same register file, and there are many inter-instruction data dependencies. There is no simple partitioning that would easily allow execution of multiple instructions per clock.

Very Long Instruction Word (VLIW) machines use sophisticated compiler technology to generate wide object code to control parallel data paths [2]. Typically, each VLIW implementation has its own unique object code format. While VLIW is useful in some situations, our environment demands object-code compatibility between generations of machines. It was essential to find a way to detect the parallelism at run-time rather than at compile-time.

The term "superscalar" was recently coined to describe machines that issue multiple instructions per clock, yet produce the same results as machines that execute instructions sequentially [3]. At about the same time the NonStop Cyclone system was announced, superscalar microprocessors were announced by Intel and IBM. The primary difference between the Cyclone processor and other superscalar designs is in the selection of which sets of instructions are to be issued simultaneously. Other machines have divided the instruction set into categories, such as branches, memory reference, and execution operators. In those machines, at most one instruction from each category can be issued simultaneously.

During the design of the Cyclone processor, we recognized that there may be many cases where several sequential instructions from the same category (or even the same instruction) should be issued simultaneously. For instance, in our stack-based machine, it is common to sequentially load two literal constants onto the register stack with Load Immediate (LDI) instructions. This pair of instructions, LDI&LDI, could easily be executed in a single clock with appropriate data path flexibility and enough register file ports. However, there was no obvious way to partition the machine into independent function units to which instructions could be assigned. Some pairs could benefit from separate ALUs, while others could benefit from separate partitions for memory reference and ALU. A few operations even suggest a bit-partitioning; one frequent pair has separate instructions to load a full-word literal into a register from left and right half-word literals.

Rather than partitioning the processor into independent function units, we chose to use firmware control and to program the microcode routines for each unique pair individually. In this way, there are no artificial restrictions on which instructions can be paired. In addition, by using microcode control, we do not restrict pairable operators to ones that can execute in a single clock cycle. For instance, instructions that use indirect addressing make two sequential accesses to the data cache and require three clocks to complete. However, it is still beneficial to pair indirect operators with other instructions. It takes three clocks to perform an indirect load, yet takes no more clocks when the indirect load is paired with a branch, immediate, or add instruction.

Once we decided to control pair execution with unique microcode routines, we could decide on a case-by-case basis whether to include the hardware support to be able to execute a pair in a single clock cycle. A hardware performance monitor was built, and instruction-pair frequencies were gathered for transaction processing applications. We then examined the frequencies to determine which hardware would gain the most performance for the least cost.

Figure 2 shows the pairing matrix for some representative instructions. Of the pairs shown, all except those in the last row execute in a single clock. The indirect loads require three clocks. In the current microcode, the full table of 2014 pairs has 38 "first" instructions (out of a possible 64) and 53 "second" instructions (out of a possible 127). In future microcode releases, more pairs may be added for improved performance.

The most important data path additions for the support of pairing were the inclusion of a nine-port register file and two ALUs that could be controlled independently or linked together for double-precision arithmetic. The flexibility of the data paths also turned out to be of great benefit in the execution of long instructions, such as the those that move or scan blocks of data, and those that send or receive messages.

SECOND	**FIRST INSTRUCTION**					
INSTR	BCC	LDI	LOAD	STOR	DADD	RRM
BCC	-	X	X	X	X	X
LDI	X	X	X	X	-	X
LOAD	X	X	-	-	X	X
STOR	X	X	-	-	X	X
DADD	X	X	X	X	-	-
RRM	X	X	X	X	-	X
LOADI	X	X	-	-	X	X

Figure 2. Sample matrix of instruction pairs. X's indicate coded instruction pairs. BCC = conditional branch, LDI = load immediate, LOAD = memory load, STOR = memory store, DADD = double-precision add, RRM = register to register move, LOADI = load indirect.

In some cases, we chose not to include data path support for pairing. For instance, support for the pairing of memory reference instructions would have required more than twice the area and cost of a simpler cache. The frequency of successive memory references did not warrant such a cost. Instead, we determined that a greater payoff would result from supporting fast access to unaligned cache data for double-words.

The following sections describe in more detail the support for multiple instruction issue in key parts of the processor: the instruction fetch unit, the control store, the data paths, and the memory.

3. Instruction Fetch Unit

The Cyclone Instruction Fetch Unit (IFU) has four main functions: 1) to fetch instructions from memory, 2) to decode these instructions to determine whether they are candidates for paired execution, 3) to provide the beginning address for microcode execution of the instruction or pair, and 4) to assist in the execution of branching instructions and exception handling.

The IFU also maintains the macro-instruction pipeline. As shown in Figure 3, each stage (or rank) of this pipeline holds two instructions to support paired execution. During ranks 1 and 2, control store is accessed; during rank 3, operands are fetched; during rank 4, ALU operations are performed; during rank 5, queued stores are completed and certain exception handling is performed.

The IFU design requirements centered around the ability to fetch at least two instructions per cycle. The alternatives were quickly narrowed down to an asynchronous IFU that could fetch two instructions per cycle with a four-instruction queue acting as a buffer between the instruction fetch and execution units. Extensive modeling showed that an instruction fetch rate of less than two instructions per cycle or an instruction queue of less than four instructions would result in significant performance degradation, but IFU designs with more capability would not increase overall performance significantly.

The IFU contains an instruction cache, which holds 32K instructions. Two instructions can be fetched in each cycle, whether the address is aligned on an even or an odd instruction address. This cache is similar in structure to the separate data cache, which is described in section 6. A hardware state-machine initiates memory requests and supervises receiving an eight-instruction block from memory when a cache miss occurs. Modeling shows that about half of the time penalty associated with an instruction cache miss is masked by the execution of instructions already in the pipeline and instruction queue.

The instruction queue holds up to four instructions in preparation for decoding and execution. Instructions fetched from the instruction cache are placed directly in the queue. Either one or two instructions are fetched, depending on how much room is left in the queue. The instructions can be placed into any location in the queue, and the queue can shift (due to instruction initiation) on the same cycle as instructions are being fetched. In between instruction cache misses, the instruction queue fills up quickly and remains full with one or two instructions fetched to replace the one or two instructions promoted out of the queue at each instruction initiation. The queue typically becomes empty while the instruction cache is filling or while recovering from a branch misprediction.

The two instructions at the head of the instruction queue (in IQ0 and IQ1) are decoded and used to address two RAMs, called the Entry Point Tables (EPTs). The information in the EPT entries is combined to determine whether the two instructions will proceed into the execution pipeline as a

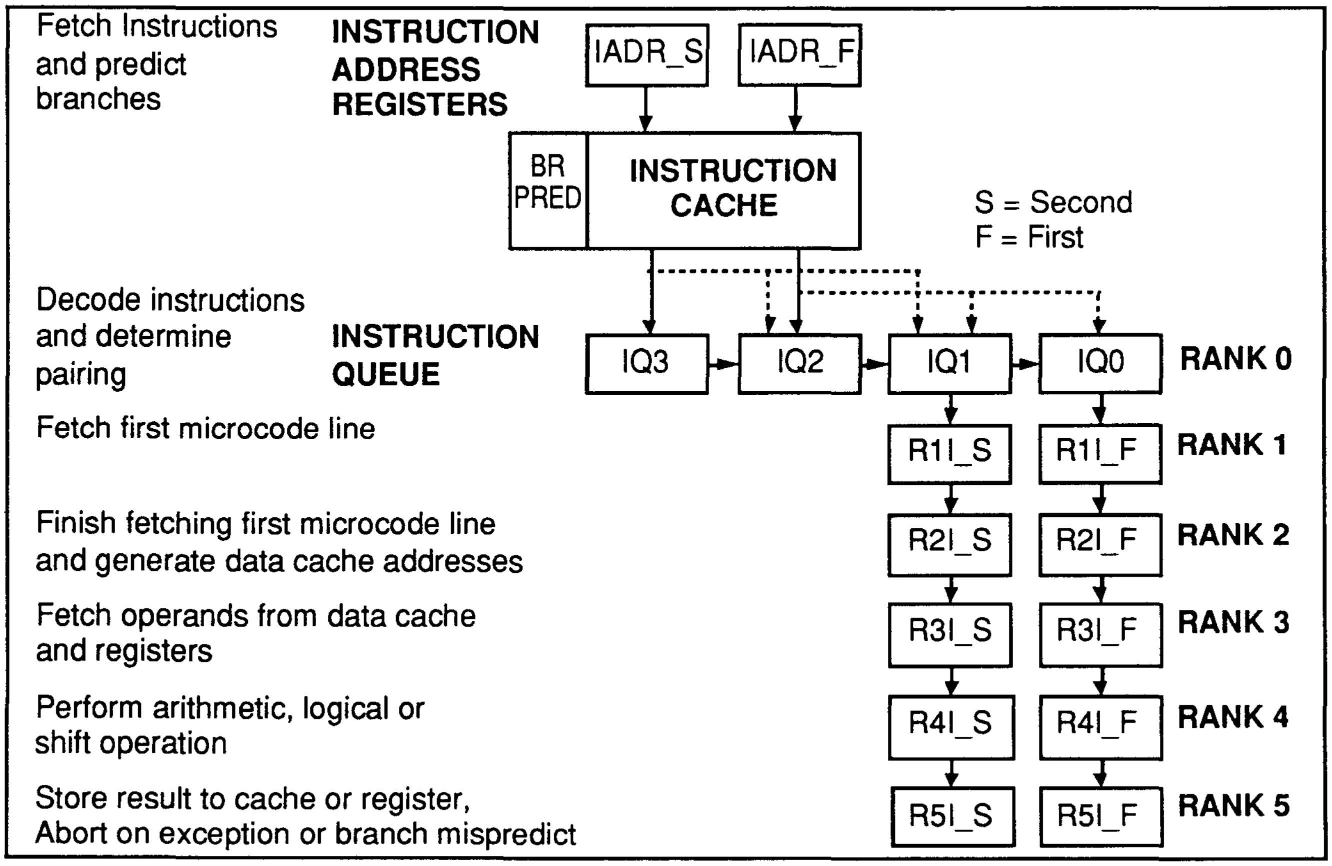
Fetch Instructions and predict branches
INSTRUCTION ADDRESS REGISTERS
IADR_S
IADR_F
BR PRED
INSTRUCTION CACHE
S = Second
F = First
Decode instructions and determine pairing
INSTRUCTION QUEUE
IQ3
IQ2
IQ1
IQ0
RANK 0
Fetch first microcode line
R1I_S
R1I_F
RANK 1
Finish fetching first microcode line and generate data cache addresses
R2I_S
R2I_F
RANK 2
Fetch operands from data cache and registers
R3I_S
R3I_F
RANK 3
Perform arithmetic, logical or shift operation
R4I_S
R4I_F
RANK 4
Store result to cache or register, Abort on exception or branch mispredict
R5I_S
R5I_F
RANK 5

Figure 3. Instruction Fetch Unit Pipeline.

ways leads to an even lower average cost per branch.

Another important aspect of total compatibility with the previous Tandem processors is exception handling. Some exceptions, such as arithmetic overflows, are reported after the execution of the instruction that encounters them, while other exceptions, such as page faults, are reported before the instruction. We wanted to handle all exceptions in such a way that the paired-instruction execution would be transparent.

In the Nonstop Cyclone implementation, a family of two instructions could have zero, one, or two exceptions; in the case of one exception, it could be reported before the family, between the two instructions of the family, or after the family.

Handling exceptions reported before or after the family is relatively easy to do with general mechanisms, but handling exceptions that should be reported between the two instructions of the family is difficult. If unique microcode were required to handle each individual case for every pair of instructions, the amount of microcode would be unmanageable. Instead, most exceptions in instruction pairs are handled in a uniform way: exception handlers are written only for single-instruction families. When an exception is encountered in a paired instruction family, the hardware initiates an "unpaired restart". It aborts the execution of the pair, refetches the first instruction, and issues the two instructions sequentially as single-instruction families. The exceptions are then encountered again and handled correctly in order.

4. Sequencer

The microcode sequencer controls the execution of both long and short instruction algorithms. It also includes extensive microcode to handle interrupts and exceptions. A large control store is required to hold enough microcode to individually code the algorithms for each instruction pair. Figure 4 shows the layout of the control store and its relationship to the entry point logic of the IFU. Fully half of the available addresses are reserved for paired instructions, including the case of a pairable instruction that executes unpaired.

The two instructions at the head of the instruction queue, in IQ0 and IQ1, are used to address separate entry point table RAMs, EPT_F and EPT_S, respectively. Bits from these tables indicate groups of resources required by the instructions; intersections of these bits exclude unpairable combinations. If the instructions are pairable, a control store address is formed from six bits taken from the EPT_F table and seven bits from the EPT_S table, thus providing for a matrix of 64 "firsts," each a candidate for pairing with up to 127 "seconds." If the first instruction is pairable but resource conflicts exist, the six bits from EPT_F are concatenated with seven 0 bits. Multiple-clock pair algorithms reduce the number of available "seconds" by 1 for each sequential word of microcode that they use. A 13-bit entry point address is taken from EPT_F for instructions that are not pairable. A fourteenth bit is computed by the logic that determines whether to access the paired or non-paired region of control store.

The highly pipelined nature of the processor allows for access of each microinstruction in two stages. The Vertical Control Store (VCS) contains 16K words of 48-bits each. The VCS is responsible for control of the operand access portion of the data unit and also provides microsequencer branching control. The Horizontal Control Store (HCS) contains 16K words of 112 bits each. The HCS provides control for the main ALU and register storage functions, as well as a variety of other functions.

The Jump Control Store (JCS) is a copy of the VCS that provides a fast conditional jump mechanism within the sequencer. For a conditional jump in the microcode, the not-taken path is addressed in the VCS while the jump-taken path is simultaneously accessed in the JCS. The outcome of the condition evaluation is used to choose whether the VCS or the JCS is loaded into the control register. The condition evaluation and microcode access overlap considerably, providing a faster branch mechanism than is possible with a serial mechanism.

In addition to the fast conditional jump mechanism, a slower but less expensive "trap" mechanism handles exception conditions related to

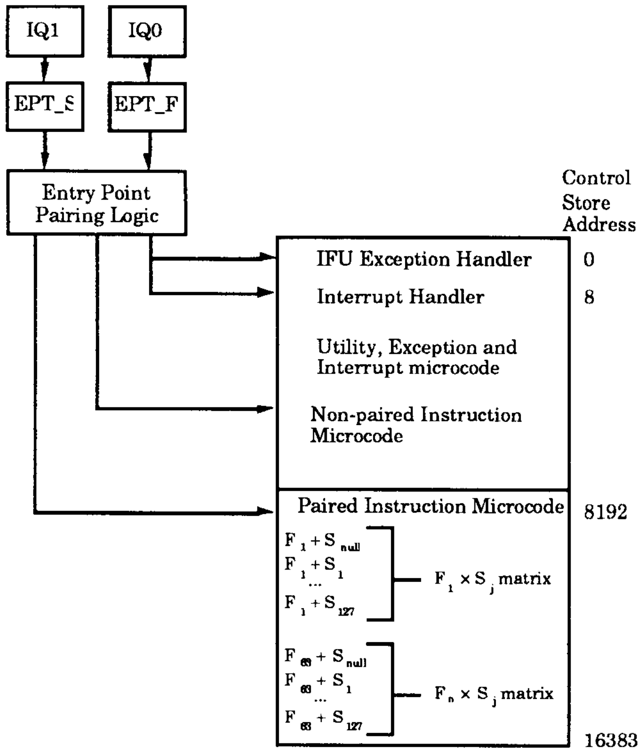

Figure 4. Microcode Layout

data cache access. A base address register is combined with a small offset field to vector into a set of utility routines. This trap address supersedes the JCS address of the successor microinstruction, thus implementing a one cycle slower jump for the cost of only a register and an address multiplexer. Traps have the unique feature of acting as a special type of microcode call, saving both the current and successor microcode addresses. This function is needed because the microinstruction that incurred the exception must be re-executed after resolution of the problem.

Duplicate copies of all three control stores (VCS, JCS, HCS) are included for performance and reliability. Two full cycles can be allowed for each access by reading alternate copies of each control store on successive cycles. This technique allows the use of 16Kx16 CMOS RAM modules for the control store, which would ordinarily not be fast enough for the control store of an ECL machine.

The second copy of each control store is also used to recover from RAM failures. Any error detected in one bank is overcome by retrying the access from the other bank, at a cost of two additional cycles to recirculate the addresses. In addition to this error recovery mechanism, spare RAMs are included. An exception handling routine can programmatically substitute a spare in place of a failing device. This mechanism, an extension of the one used in the NonStop VLX processor [5], provides substantial MTBF improvement at a relatively low cost.

5. Data Path

The design of the main data paths was driven by the requirements of the most frequent instruction pairs, tempered by reasonable cost limitations on logic. Two parallel ALUs were included for cases that required independent computation and to accommodate relatively frequent doubleword (32-bit) operands.

We found that many pair algorithms could benefit from simple "selector" paths that could read and write the main register file without performing data transformations. The selector paths were also used for the interface to other sections of the processor outside the main data path, such as the integer multiplier, I/O subsystems, and maintenance processor.

A nine-ported register file was needed to support the desired register access flexibility. There are four write ports (two ALUs and two selector paths) and five read ports. Four of the read ports are for operand registers used directly in the ALUs and selector paths, while the fifth read port provides limited access to base registers for the address generation logic. Bypass paths are provided to allow families to access registers modified by previous families without stalling the processor.

The block diagram of Figure 5 includes a high-level view of the main data path logic. The execution of a simple LOAD instruction, which comprises a single line of microcode, fetches data from cache while in rank 3 of the pipeline and stores to the register file logic in rank 4. Address generation can come either from a decoding of the rank-2 instruction for short instructions, or from the ALU(s) under microcode control during long algorithms.

This provision of four operand registers and dual ALUs allowed a paired implementation of a double-word load (LDD) with double-precision integer add (DADD). Figure 6 shows schematically the register stack transitions involved. If issued as single instructions, they would require two execution cycles. Data from cache would be loaded to R2 and R3 in the first cycle and then added to R0 and R1 in the second. Issued as a pair, however, LDD&DADD takes only a single execution cycle. In that cycle, data from cache can be simultaneously written to R2 and R3 while the sums of R0+R2 and R1+R3 are computed and stored to R0 and R1. This single-cycle family uses both ALUs, all four of the register file write ports, two of the read ports, and both of the selector paths. It also takes advantage of the double-word cache access.

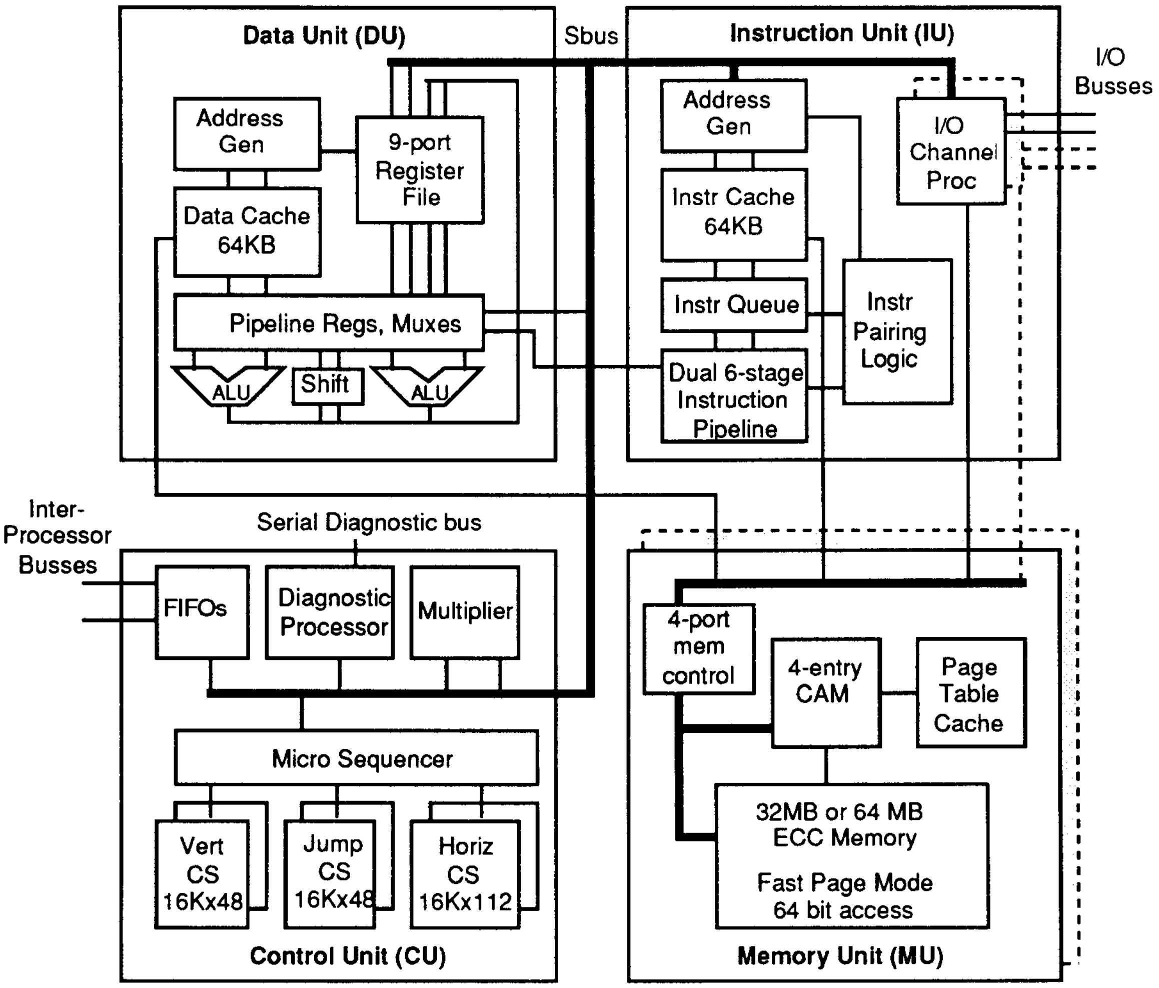

Figure 5. NonStop Cyclone Block Diagram. The processor is partitioned into four 18"x18" boards plus optional boards to add memory and I/O channels.

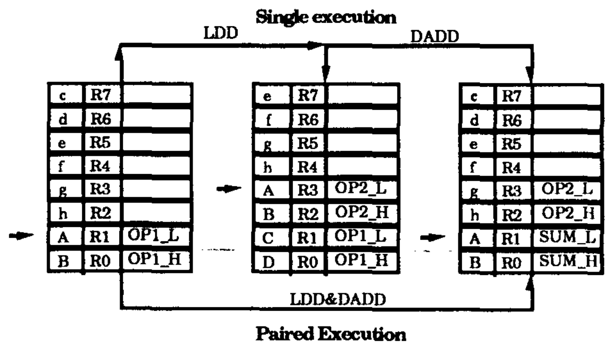

Figure 6. Register stack transition for Load-Double & Double-Add. Each stack shows the register name, the physical register number, and the contents.

Some candidate instruction pairs would have required even more data path resources, but their frequency of occurrence did not warrant the inclusion of these resources. An example is a double-add followed by a load-immediate (DADD&LDI), which is not executable as a pair. Implementation of this pair would have required a fifth operand pipeline register (four for the DADD operands and one for the immediate value). On the other hand, the reverse sequence, LDI&DADD, can be executed as a pair; the immediate value is one of the DADD operands, and only four operand registers are required. Fortunately, due to the frequency of computing 32-bit address offsets, LDI&DADD occurs over 10 times more frequently than DADD&LDI.

6. Cache and Memory

Data gathered by the hardware performance monitor was critical for the design of the caches. The frequency of unaligned doubleword references was higher than the frequency of pairs with two memory references. Therefore, we concentrated on efficient support for unaligned accesses rather than support for two arbitrary memory references per cycle.

The unaligned cache design is shown in Figure 7. The data storage is partitioned into halves that store the data from even and odd addresses. The tag store is duplicated because an unaligned data access could cross cache block boundaries, requiring two different tags to be accessed. For aligned accesses, the same doubleword address is applied to both even and odd caches, and the first word is read from the even cache and the second is read from the odd cache. For an unaligned access, the doubleword address of the first word is applied to the odd cache, and that address is incremented and sent to the even cache. For unaligned accesses, the first data word comes from the odd cache, and the second comes from the even cache. Doubleword cache accesses require only one cache cycle as long as both words reside in the same page.

The unaligned cache design is also used in the instruction cache to allow pairs of arbitrarily-aligned instructions to be fetched each clock. Both the instruction and data cache are direct-mapped 64 Kbyte caches with address hashing, 16-byte blocks, and support for unaligned access.

The even-odd structure is maintained all the way to main memory. The data cache is a store-through design. All cache writes are sent to main memory through a one-entry write buffer. The memory can accept an average of one aligned or unaligned doubleword write every two cycles. Most memory reads are aligned, because full cache blocks are returned. After initial latency, the memory can supply one doubleword per cycle to either of the two caches or to an I/O subsystem.

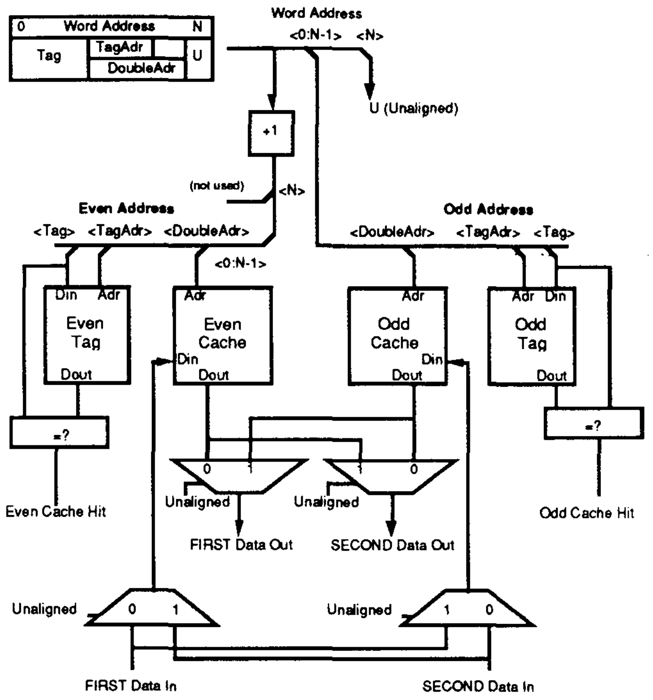

Figure 7. Block diagram of cache for unaligned access.

In Tandem processors, all memory references are mapped to a single virtual address space shared by all processes. This allows the caches to use virtual addressing without the problems of invalidating cache entries on process switches or requiring a reverse translation table to handle synonyms. In the Cyclone processor, the virtual addresses are sent directly to the main memory. A four-entry content-addressable memory (CAM) compares each access to the row address that previously addressed a bank of dynamic RAM. When the addresses match, the dynamic RAM column address is generated by a few bits from the CAM plus the address offset. Translation from virtual to physical address is performed only on a CAM miss. Because the translation is performed infrequently, it was possible to implement the Page Table Cache (translation lookaside buffer) in relatively slow CMOS static RAMS.

7. Performance

The performance gains from issuing multiple instructions per cycle are quite program-specific. Programs that execute only RISC-like instructions show much more benefit from pairing than programs that spend large amounts of time in long instructions (such as block move and scan instructions). The programs using long instructions benefit from many of the data path and memory enhancements made to support pairing, but do not gain much performance from multiple instruction issue because the instruction issue rate is relatively low.

No experiments have yet been run to determine the effects of pairing on large transaction processing benchmarks. However, modeling has shown that the ET1 debit-credit benchmark should run with half of the families issued as instruction pairs. This means that approximately 2/3 of the instructions are issued in pairs. In typical transaction processing tasks, the Cyclone processor averages about 1.5-2 clocks per native instruction (CPI), including all overhead for cache misses and the effects of averaging many short paired instructions with a few very long instructions.

Some simple benchmarks have been analyzed in detail on the microcode simulator. For instance, the inner loop of the simple Sieve benchmark has nine short instructions--a memory load, two register-to-register moves, an integer add, two immediates, an indirect store (3 clocks), and two branches. Without pairing, this loop takes 11 cycles. With pairing enabled, the nine instructions are issued in five families and require seven clocks, for a rate of 0.78 CPI. Running the Sieve benchmark on a real machine shows a performance improvement of 37% when pairing is enabled. Other benchmarks show less improvement with pairing, varying from 12% to 37%. Benchmarks showing smaller improvements with pairing generally make heavy use of block move instructions. If these had been coded instead with simple load and store instructions, the apparent benefit of pairing would increase, but the overall benchmark speed would generally decrease.

8. Conclusions

We have described many of the innovations in the architecture of the NonStop Cyclone processor. New superscalar techniques were developed and applied in the design of an object-code compatible, fault-tolerant mainframe.

While some of the techniques may be peculiar to Tandem's instruction set, most could be applied to either RISC or CISC architectures. The smaller instruction set of a RISC architecture would greatly reduce the number of instruction pairs and simplify the instruction decoding. The new microsequencer ideas and methods of attaining object-code compatibility are especially applicable to CISC designs. Finally, many commercial machines could benefit from new cache designs that allow rapid access to unaligned data.

9. Acknowledgements

The authors wish to thank the outstanding engineers and programmers who contributed their ideas and energy to the design of the NonStop Cyclone system.

References

[1] Acosta, "An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors," *IEEE Trans Computers*, vol. C-35, no. 9, pp. 815-828, September 1986.

[2] R. P. Colwell, R. P. Nix, J. J. O'Donnell, D. P. Papworth, P. K. Rodman, "A VLIW Architecture for a Trace Scheduling Compiler," in *proc. Second International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, MA, pp. 180-192, October 1987.

[3] N. P. Jouppi, D. W. Wall, "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines," in *proc. Third International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, MA, 1989.

[4] J. E. Smith, "A Study of Branch Prediction Strategies," in *proc. 8th Annual Symposium on Computer Architecture*, pp. 135-148, May 1981.

[5] R. W. Horst, "Reliable Design of High-speed Cache and Control Store Memories," in *proc. Nineteenth International Symposium on Fault Tolerant Computing*, Chicago, IL, pp. 259-266, June 1989.

Distributed by
TANDEM
Corporate Information Center
10400 N. Tantau Ave., LOC 248-07
Cupertino, CA 95014-0708