Cascade: Hardware for High/Variable Precision Arithmetic*

Tony Carter
UUCS-89-005

January 30, 1989

Abstract

The Cascade hardware architecture for high/variable precision arithmetic is described. It uses a radix-16 redundant signed-digit number representation and directly supports single or multiple precision addition, subtraction, multiplication, division, extraction of the square root and computation of the greatest common divisor. It is object-oriented and implements an abstract class of objects, variable precision integers. It provides a complete suite of memory management functions implemented in hardware, including a garbage collector. The Cascade hardware permits free tradeoffs of space versus time.

1 Introduction

Applications such as solid modeling of geometric objects [16], solving complex sets of equations using Gröbner bases [10], and encryption/decryption often involve the use of very high precision arithmetic operations that generally must be implemented using the relatively low precision arithmetic units available in today's computers. The numbers used in such calculations are of variable length. For example, in Gröbner bases calculations number lengths vary from a few to several hundred decimal digits. In Thomas' algorithm for combining b-spline surfaces [16], numbers vary in length from a few to over one thousand decimal digits. In encryption and decryption algorithms, the use of large primes is desirable.

Some systems make use of Common Lisp bignums [17] in algorithms which solve these problems, but bignum arithmetic operations implemented in software are slow. This slowness is necessitated by the digit-serial nature of high precision arithmetic computations using conventional processors with limited word-widths as well as by significant overhead for memory allocation and garbage collection. Digit-serial algorithms for addition and subtraction have $O(n)$ time complexity and those for multiplication and division have $O(n^2)$ complexity [9].

A way to significantly reduce the time required for arithmetic operations on high/variable precision numbers is to provide specialized, scalable arithmetic hardware in which the width of the memory used to store numbers and the width of the arithmetic unit can be increased to match the size of numbers used in solving a problem. To accelerate solutions to the problems of solid modeling, Gröbner bases solutions to systems of equations and encryption/decryption the hardware must scale from precisions of a few decimal digits to potentially thousands of digits, permitting constant time complexity for additions and subtractions and $O(n)$ time complexity for multiplications and divisions.

Hardware solutions using conventional techniques such as the two's complement number representation and fast carry lookahead are not viable

*This research was supported by DARPA through contract number DAAK11-84-K-0017.

given such constraints. Full carry lookahead is impractical at large word widths since it has an area complexity of $O(n^2)$. Furthermore the full-carry lookahead circuit actually slows down linearly with the number of bits in the operand due to gate fanin and fanout effects. Even with a very small constant on the $O(n)$ time complexity of full carry lookahead schemes, the area*time complexity of the full carry lookahead approach is unacceptable at $O(n^3)$. If block carry lookahead is used, the time complexity is reduced from $O(n)$ to $O(\log n)$, but the physical design is still complicated since a tree of block carry lookahead units must be used. In area, the block carry lookahead scheme has $O(n \log n)$ complexity which results in an area*time complexity of $O(n \log^2 n)$ [18] which is significantly better but still unacceptable.

The ideal area*time complexity for addition and subtraction is $O(n)$. This can be provided by digit-serial adders (using any implementation technique) with $O(1)$ space and $O(n)$ time or by redundant signed-digit methods (of which carry-save is one) with $O(1)$ time with $O(n)$ space.

Cascade is a hardware architecture being developed to accelerate high/variable precision arithmetic operations. As mentioned above, software packages for performing variable precision arithmetic exhibit two time related problems:

- digit-serial arithmetic based on very limited precision arithmetic units, and
- significant memory management overhead.

The architecture of Cascade takes both problems into account. In profiling a complex software system (Alpha_1 [1]) we discovered that memory management for objects required more time than the arithmetic. The cost of memory management frequently equals or exceeds the cost of arithmetic computation so the speed of memory management operations is at least as important as the speed of the arithmetic.

Cascade is based on a design for a variable precision processor proposed by Chow in [8]. The radix-16 digit slice in Chow's processor has been designed and implemented using VLSI [4], [7], [14].

As described hereafter, the digit-slice used in Cascade differs from the one proposed by Chow in some simple yet significant ways to better support division and extraction of the square root. Cascade provides linear scalability in space while maintaining constant addition time, but also directly supports multiple precision arithmetic operations when the physical word-width is not adequate. Cascade also provides specialized memory management hardware.

2 Representing Numbers

Cascade uses only redundant signed-digit number [2]; conversions to and from the two's complement number representation are performed as infrequently as possible and only when requested by an external agent. For reasons described in [8], Cascade uses radix 16 digits that represent numbers between -10 and 10 (for a total of 21 values).

As noted by Robertson [12], there are two critical parameters that describe the set of values (digit-set) that can be represented by a single digit. They are the diminished cardinality δ — the number of distinct arithmetic values that a digit can represent, minus one, and the offset ω — the distance of the most negative value from zero. In this paper we denote digit-sets using the notation $< \delta.\omega >$. In particular, Cascade uses $< 20.10 >$ digits which can be represented as $4 < 4.2 > + < 4.2 >$. This makes it possible to model division using two radix-4 steps rather than one radix-16 step [5]. Each radix-4 digit is implemented as $2 < 1.1 > + < 2.0 >$, making the design of the Cascade arithmetic circuitry possible using Robertson's Theory of Decomposition [12], [13] and its physical counterpart, Structured Arithmetic Tiling [3], [6]. Thus, there are three useful views of digits in Cascade:

$$< 20.10 >$$
$$4 < 4.2 > + < 4.2 >$$
$$8 < 1.1 > +4 < 2.0 > +2 < 1.1 > + < 2.0 >.$$

3 Cascade's Architecture

The Cascade hardware is essentially an opaque physical implementation of an abstract class of objects, *variable precision integers*. (Some minor

modifications to its control chip would permit it to operate on normalized fractions as well). It contains its own storage for both numbers and memory management information. The sole interface to the outside world is through a self-timed request/acknowledge message interface. Cascade normally returns *handles* to variable precision integers, although it can return the value of a variable precision integer in an extended two's complement form if necessary.

Cascade is composed of two distinct types of modules as shown in figure 1. The first is a single *control module* with associated number management memory. The second is a set of N *arithmetic modules* each of which contain a single *arithmetic chip* with associated digit memory.

The control chip in the control module contains a multi-faceted controller that:

- interacts with external agents via the message port,
- manages the available digit memory through a hardware-resident garbage collection and memory allocation scheme,
- controls single and multiple precision arithmetic operations ($+$, $-$, $*$, $\div$, $\sqrt{}$, and gcd) on variable precision integers, and
- optimizes both memory management and common arithmetic operations.

The control chip also contains model division hardware for the two-stage, radix-16 division algorithm described in [5].

The Cascade hardware supports the use of arithmetic futures. At the option of the sender of a message, the Cascade hardware can return a handle to the new variable precision integer result as soon as storage has been allocated but before the arithmetic operation has been completed. This permits external modules that use Cascade to proceed without necessarily having to wait for the value to be computed.

At the top of each module there is a memory interface. The control module originates all memory control signals to both management memory and digit memory. As seen in figure 1, there are six signal loops in the system. The top two (*sp0* and *sp1*) are digit-wide shift paths capable of shifting left or right by whole digits (radix-16) or half digits (radix-4). They are used during multiplication for shifting the partial product and the multiplicand and during division and square root extraction for normalization and shifting the partial remainder (partial radicand). The next three are transfer digit paths. The top transfer digit path (*dbl*) is used only during extraction of the square root. The next one (*ml*) is used during multiplication, division and extraction of the square root and the bottom one (*ol*) is used during all operations except shifting. The bottom loop is labeled "±Ndp". It has three uses. First (±), it computes the sign of a signed-digit number (for which the sign is given by the sign of the most significant non-zero digit). Second (N), it is used to report on whether a number is normalized (for radix-16, radix-4 or radix-2) so that the control module can decide whether further shifts are necessary. Third (dp), it is used during square root extraction to signal where the root digit just produced should be inserted into the accumulating root.

On the lower left of the control module there is the message port which consists of request/acknowledge lines and a twenty-bit data bus through which handles, values and other information is passed between external agents and the Cascade system. There is a system clock which is directly used only by the control chip. There is a master reset signal so the system can be restored to a known state. The control module generates a ten-bit instruction word that is broadcast to the arithmetic modules where it is decoded and applied to control points within the arithmetic chips. The strobe signal is generated by the control chip and sent to all the arithmetic chips, causing registers to capture data from digit memory or the arithmetic unit. The *sdv* signal is an open-drain bus driven by all the arithmetic chips. It is used by the control chip to detect when an arithmetic operation results in zero or a value that is represented as a single digit. The control chip can then optimize storage use and future arithmetic operations that involve very common values such as 0, 1 and -1. Each arithmetic chip also has two signals indicating whether or not it contains the most or least/significant digit.

4 Arithmetic Modules

Each arithmetic module consists of a 16-digit (80 bit) wide digit memory and a custom arithmetic chip containing a 16-digit slice of the arithmetic datapath (roughly equivalent to 64-bits). Figure 2 shows the structure of the arithmetic chip. The XL and LX boxes encode and decode each six-signal < 20.10 > digit as a five-bit < 31.10 > digit for storage in digit memory. Thus the storage overhead in Cascade over what would be required in a normal two's complement system is only 25%.

At the top, there are the two shift paths which permit right or left shifts by whole (radix-16) or half (radix-4) digits. Each of these shift paths interfaces directly to four 16-digit (96-bit) registers enabling any of these to be shifted.

The addition of two large numbers with opposite sign results in a number of much smaller magnitude. The size of this result cannot be predicted before the addition, so the number of leading zeros must be computed following each addition or subtraction. This calculation is done in the *sign computer/leading zeros counter* by having each arithmetic chip count the number of leading zeros on one of the arithmetic unit buses (input or output). This will be a number between 0 and 16. The collection of arithmetic chips then shifts these counts to the left using a shift path and the control chip accumulates the number of leading zeros until a count of less than 16 is encountered.

Each of the four registers can serve as input to either port of the arithmetic unit and can latch the output of the arithmetic unit. Under control of the token replicating *root digit position register*, any given digit in a register can store the value of the current root digit during square root extraction. At each digital position there is flip-flop. Initially all flip-flops are cleared. When the flip-flop at a digital position is clear but the flip-flop immediately to its left is set, the next root digit generated is stored at that digital position. This permits the unknown root to be accumulated in position, left to right.

The *sign computer* uses a self-timed priority encoding scheme to find the sign of the most significant non-zero digit. The sign is propagated from the most significant non-zero digit to the control module at the left. If a digital position contains a non-zero value it reports its own sign, otherwise it reports the sign of the digital position immediately to its right.

The *normalization sensor* examines the three most significant digits of a number to see if any more normalization operations are required. Under control of the instruction received from the control module, the arithmetic chip can detect radix-16, radix-4 or radix-2 normalization.

Below the normalization sensor and the arithmetic unit is a *distribution box* in which routing is performed. It also contains a set of switches to connect the output of the arithmetic unit directly to the memory bus so the result of an addition or subtraction can be stored to memory without first being moved to a register.

At the bottom is the arithmetic unit. It is composed of sixteen identical radix-16 digit slices (described in section 4.1). The arithmetic unit contains zero detecting ROMS at all digital positions. It also contains a small *single digit value* detecting ROM at the least significant digital position to enable the detection of results that are represented as a single digit. This permits the control module to detect values like 1, 0 and -1 as the result of an arithmetic operation. When such values can be detected, storage need not necessarily be used for them and operations such as multiplication by zero or one can be dynamically optimized in the hardware.

Note that the transfer digit at the multiplier level (mlc and mlo) is recoded from a five-bit to a four-bit representation, saving two pins.

4.1 The Radix-16 Digit Slice

The heart of the arithmetic chip is the radix-16 digit slice pictured in figure 3. It has a conditional doubling circuit (an adder plus a multiplexor) used during extraction of the square root. During square root extraction using completion of the square; all root digits are doubled except the most recently generatedas indicated by the root digit position register. This doubling circuit is not present in the digit-slice proposed by Chow in [8].

A < 20.10 > digit is broadcast to all digital positions of the arithmetic unit for use dur-

ing multiplication, division and square-root extraction. There, an elementary multiplier performs the radix-16 multiplication of the multiplier digit by the multiplicand digit. The output of the elementary multiplier is sent to the *m0 adder* which transforms the result into a 16 < 12.6 > transfer digit, a < 8.4 > sum digit that is recombined in the *m1 adder* with the incoming < 12.6 > transfer digit to form a < 20.10 > digit, and a 4 < 2.1 > digit that is passed on to the normal addition circuitry. Just below the *m1 adder* is a pair of multiplexors. During multiplication, division and extraction of the square root these multiplexors pass on the output of the multiplication circuitry. During addition and subtraction they do not.

There is a pair of conditional complementing circuits just below these multiplexors to permit subtraction by addition of the complement, assisting in division by permitting the recurrence equation $p_{j+1} = rp_j - q_{j+1}d$ to be computed in a single step. The location of these conditional complementing has been changed from Chow's proposed digit slice. Below the conditional complementers is the two-level signed-digit addition circuitry, the *a0 adder* and the *a1 adder*.

5 Control Module

5.1 Memory Management

One of the most significant functions of the control module is the management of variable precision integer objects. Figure 4 shows the complete memory organization of the Cascade hardware. There is a bifurcated management memory that contains descriptors and descriptor pointers. Descriptor pointers are directly referred to by *handles* and never move. Descriptors are referred to by descriptor pointers and may be moved as part of the memory allocation/garbage collection scheme. Descriptor pointers are allocated from the bottom up while descriptors, like digit memory locations, are allocated from the top down. Descriptors are maintained such that there are no crossing pointers into digit memory, facilitating garbage collection.

The memory management strategy attempts to avoid garbage collection if at all possible. If a number has been destroyed but has not yet been reclaimed by the garbage collector and if it has adequate storage for the next result it will be reused immediately without garbage collection. Up to one megaword of digit memory and one megaword of management memory can be addressed. A setup register in the control chip permits lesser amounts of real memory to be installed in the system.

5.2 External Message Port

The external message port consists of a request/acknowedge signal pair and a twenty bit data bus. When an external agent wishes to send a message to the Cascade hardware, it first asserts its data and then raises the request line. When the Cascade hardware is ready, it latches the data and interprets it. When done, it raises the acknowledge line until the external agent lowers the request line.

There are two types of message cycles in Cascade, `REQUEST`s and *Results*. Each message may consist of any number of request and result cycles, depending on what data must be transferred. Most messages that cause arithmetic operations to be performed have three request cycles and one result cycle (for the returned handle). The conversion messages have unbounded message lengths to permit values and digit sequences of variable precision integers to be returned to the invoking external agent. External agents must follow message protocols exactly.

The arithmetic message codes (indicated by an @ following the message name) have two flags that modify the behavior of the Cascade hardware. The first indicates that a future is to be returned and the second indicates that the numbers passed as arguments to the arithmetic operation should be destroyed at the completion of the operation. Multiple messages cycles are indicated by enclosing the message name in brackets (e.g., `[multiple cycle]`).

5.3 Model Division

The SRT division algorithm used in Cascade is described in [5]. The model division used a three digit estimate of the divisor and a two digit estimate of the partial remainder. The three digit estimate of the divisor is stored in a special register since mul-

tiples of it must be constantly computed as part of the model division. The two digit estimate of the partial remainder can be obtained by the model in one of three ways:

1. keep the two most significant digits of the divisor and partial remainder in the control chip, or

2. keep the most significant digit of the divisor and partial remainder in the control chip and have the most significant arithmetic chip send the next most significant digit to the control chip via a shift path, or

3. keep all of the divisor and partial remainder in the arithmetic chips and have the most significant arithmetic chip shift its two most significant digits to the control chip via the shift paths.

Of these three, the third is clearly the least desirable since it requires special shift path connections at the second most significant digital position. The first is fastest and the most desirable since it requires no off chip communication for forming the estimates of the partial remainder. The second is a tradeoff that can be made if there is inadequate room on the control chip.

6 Performance Estimates

A software model of this architecture has been simulated to verify correctness of the control algorithms for the arithmetic operations. Detailed spice simulation of the individual arithmetic circuit modules (called operators) that make up the arithmetic unit have been done. Table 2 shows the propagation delays used to estimate the speed of the arithmetic modules. The arithmetic registers are edge-triggered rather than clocked using the normal two-phase MOS clocking scheme. This avoids slowing them down to the speed of the system clock. Table 2 does not include time required for memory management functions preliminary to the arithmetic operations.

The quotient/root digit selection hardware has not yet been fully designed so the speed of division and square-root extraction cannot be accurately estimated. For division and square root, the table contains only the time necessary for a single-precision recursion step.

A critical issue in addition and subtraction is memory access time since three or four memory cycles are required (two to fetch the operands and one or two to store the result). If fast (25 ns) static RAM is not used as suggested in table 2 then the times for addition and subtraction slow down dramatically. In general, memory cycle time has less effect on single-precision multiplication and division than it does on addition and subtraction.

7 Conclusion

Cascade is a hardware architecture designed specifically for performing arithmetic operations on high/variable precision integers. Relative to Common Lisp bignums running on a professional workstation, it is possible to realize speed improvements of several orders of magnitude in arithmetic computations involving bignums of around 256 digits of precision.

The Cascade hardware permits free tradeoffs of time versus space since it is linearly scalable in space with no time cost incurred for additional word-width. The combination of high speed arithmetic function and memory management capabilities is a unique feature of this object-oriented processor.

References

[1] CAGD Research Group, *Alpha_1 User's Manual*, Univ. of Utah, Dept. of Computer Science, 1983.

[2] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic", *IRE Trans. on Electronic Computers*, vol. EC-10, No. 9, Sep. 1961, pp. 389-400.

[3] T. M. Carter, *Structured Arithmetic Tiling of Integrated Circuits*, Ph.D. Diss., Univ. of Utah, Dept. of Computer Science, Dec. 1983.

[4] T. M. Carter and L. A. Hollaar "The Implementation of a Radix-16 Digit-Slice Using a Cellular VLSI Technique", *Proceedings of ICCD 1983*, Nov. 1983, pp. 688-691.

[5] T. M. Carter and J. E. Robertson, "Radix-16 Signed-Digit Division", Report UUCS-88-004, Univ. of Utah, Dept. of Computer Science.

[6] T. M. Carter, "Structured Arithmetic Tiling of Integrated Circuits", *Proc. 8th Symp. on Computer Arithmetic*, May 1987, pp. 41-48.

[7] C. Y. F. Chow, "A Variable Precision Processor Module", *Proc. IEEE Int'l Conf. on Computer Design*, Nov. 1983, pp. 692-695.

[8] C. Y. F. Chow, *A Variable Precision Processor Module*, Ph.D. Diss., Univ. of Ill. at Urbana-Champaign, Dept. of Comp. Science, July 1980.

[9] D. W. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, Reading, MA: Addison-Wesley, 1981, pp. 250-265.

[10] H. Melenk, H. M. Möller and W. Neun,"On Gröbner Bases Computation on a Supercomputer Using REDUCE", Preprint SC 88-2, Jan. 1988, FB Mathematik und Informatik der Fernuniversität Hagen.

[11] J. E. Robertson, "Normalization and Quotient Digit Selection in a Variable Precision Arithmetic Unit", Report UIUCDCS-R-86-1229, Univ. of Ill. at Urbana-Champaign, 1986.

[12] J. E. Robertson, "A Theory of Decomposition of Structures for Binary Addition and Subtraction", Report UIUCDCS-R-81-1004, Univ. of Ill. at Urbana-Champaign, Jan. 1983.

[13] J. E. Robertson, "A Systematic Approach to the Design of Structures for Arithmetic", *Proceedings of the 5th Symp. on Computer Arithmetic*, May 1981, pp. 35-41.

[14] J. E. Robertson, "Design of the Combinational Logic for a Radix-16 Digit-Slice for a Variable Precision Processor Module", *Proc. of ICCD 1983*, Nov. 1983, pp. 696-699.

[15] G. L. Steele, *Common Lisp.*

[16] S. W. Thomas, *Modeling Volumes Bounded by B-Spline Surfaces*, Ph.D. Diss., Univ. of Utah, Dept. of Computer Science, 1984.

[17] J. L. White, "Reconfigurable, Retargetable Bignums: A Case Study in Efficient, Portable Lisp System Building", *Proc. ACM Conf. on Lisp and Functional Prog.*, 1986, pp. 174-191.

[18] D. Zuras and W. H. McCallister, "Balanced Delay Trees and Combinational Division in VLSI", *IEEE J. of Solid-State Circuits*, vol. SC-21, no. 5, Oct. 1986, pp. 814-819.

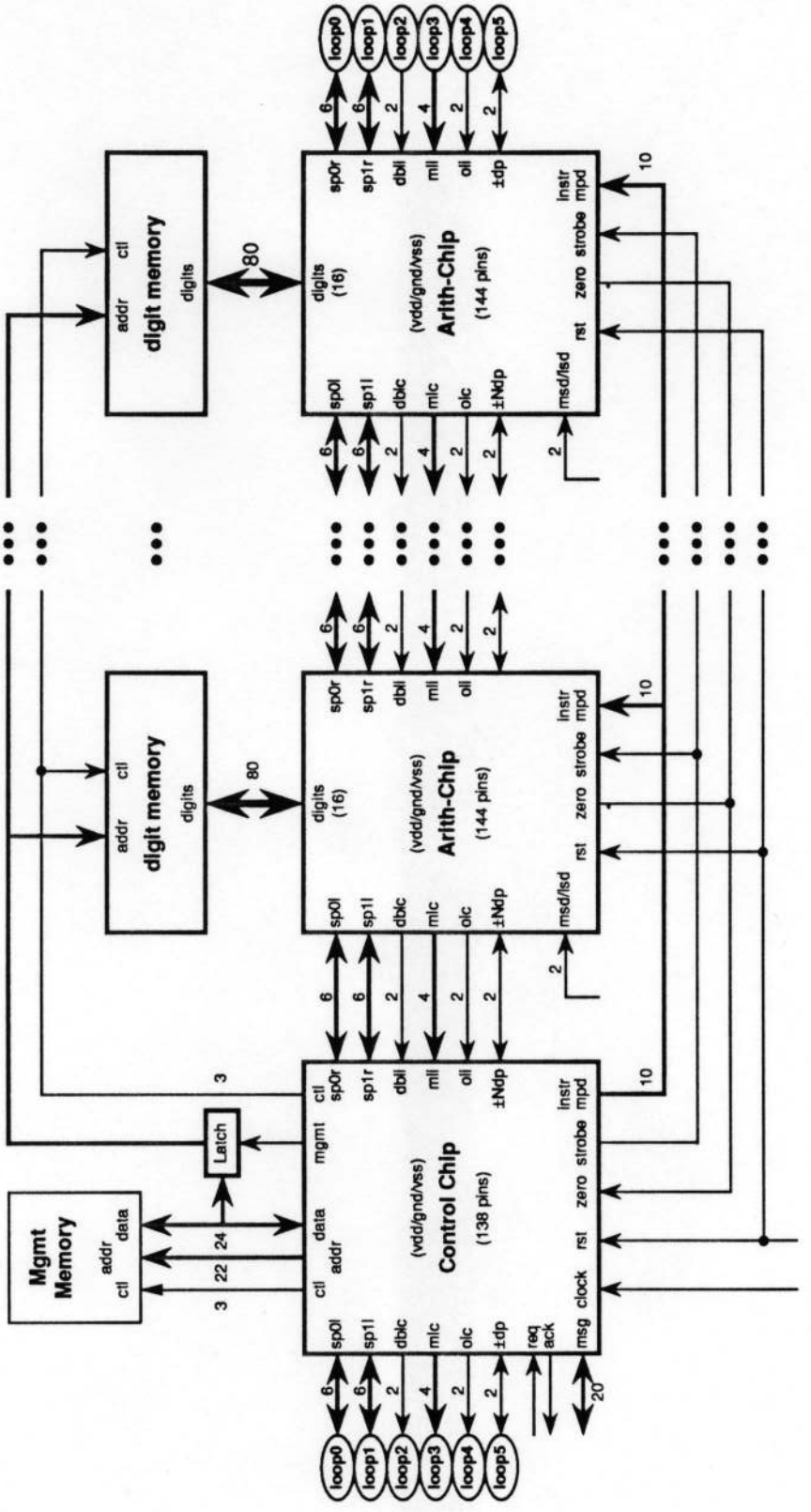

Figure 1: The Cascade Architecture

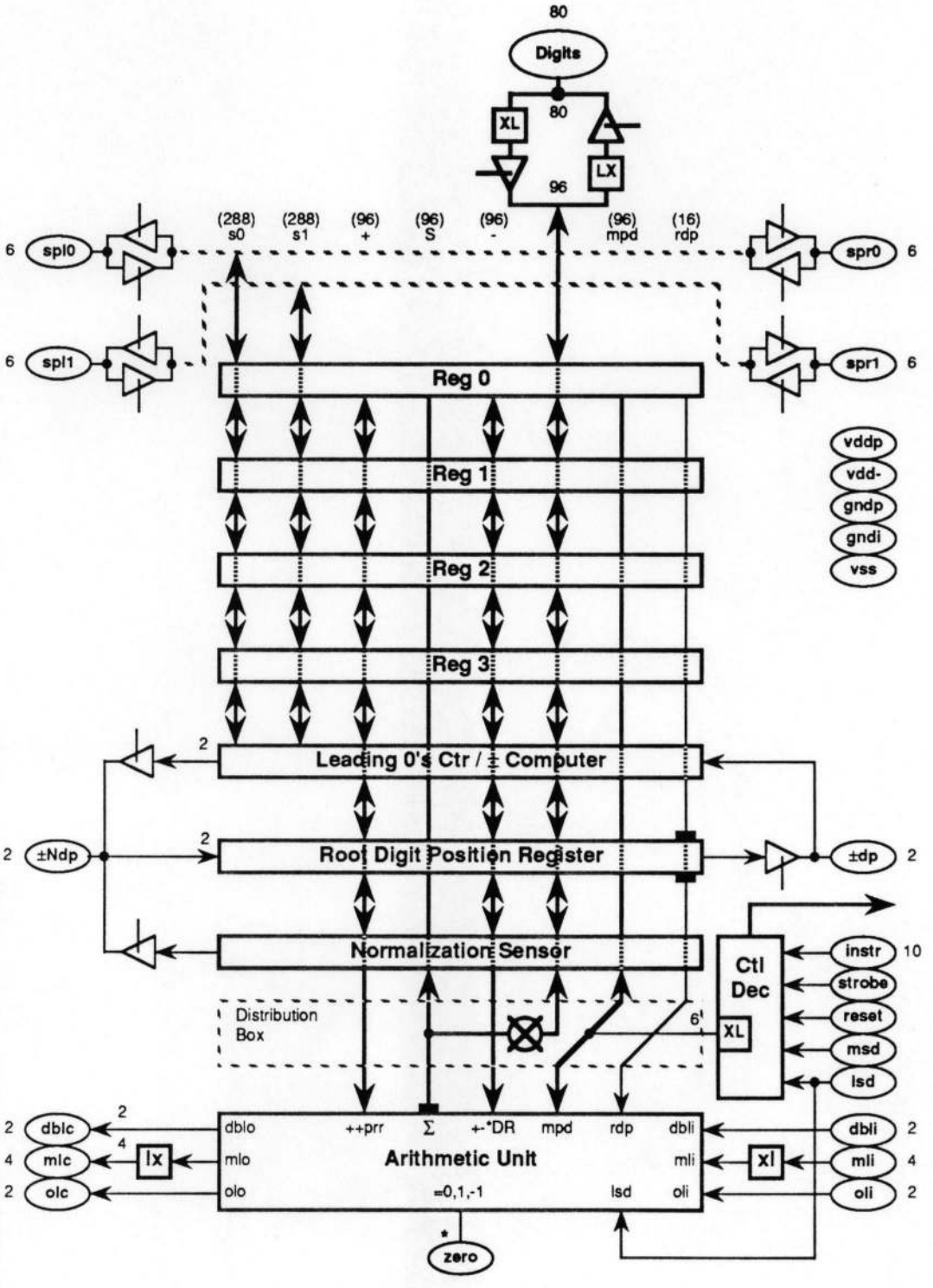

Figure 2: Cascade's Arithmetic Chip

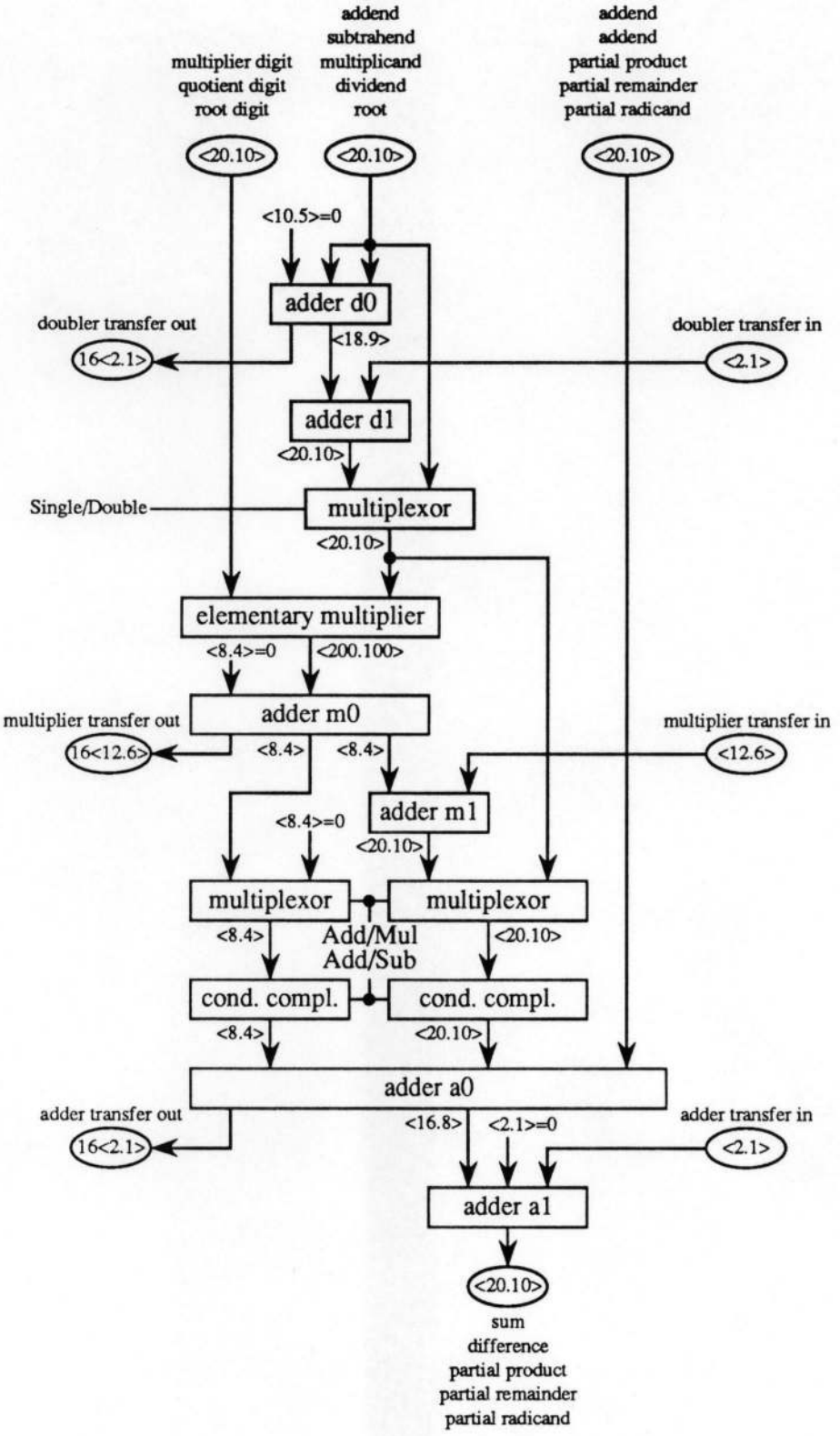

Figure 3: Cascade's Digit Slice

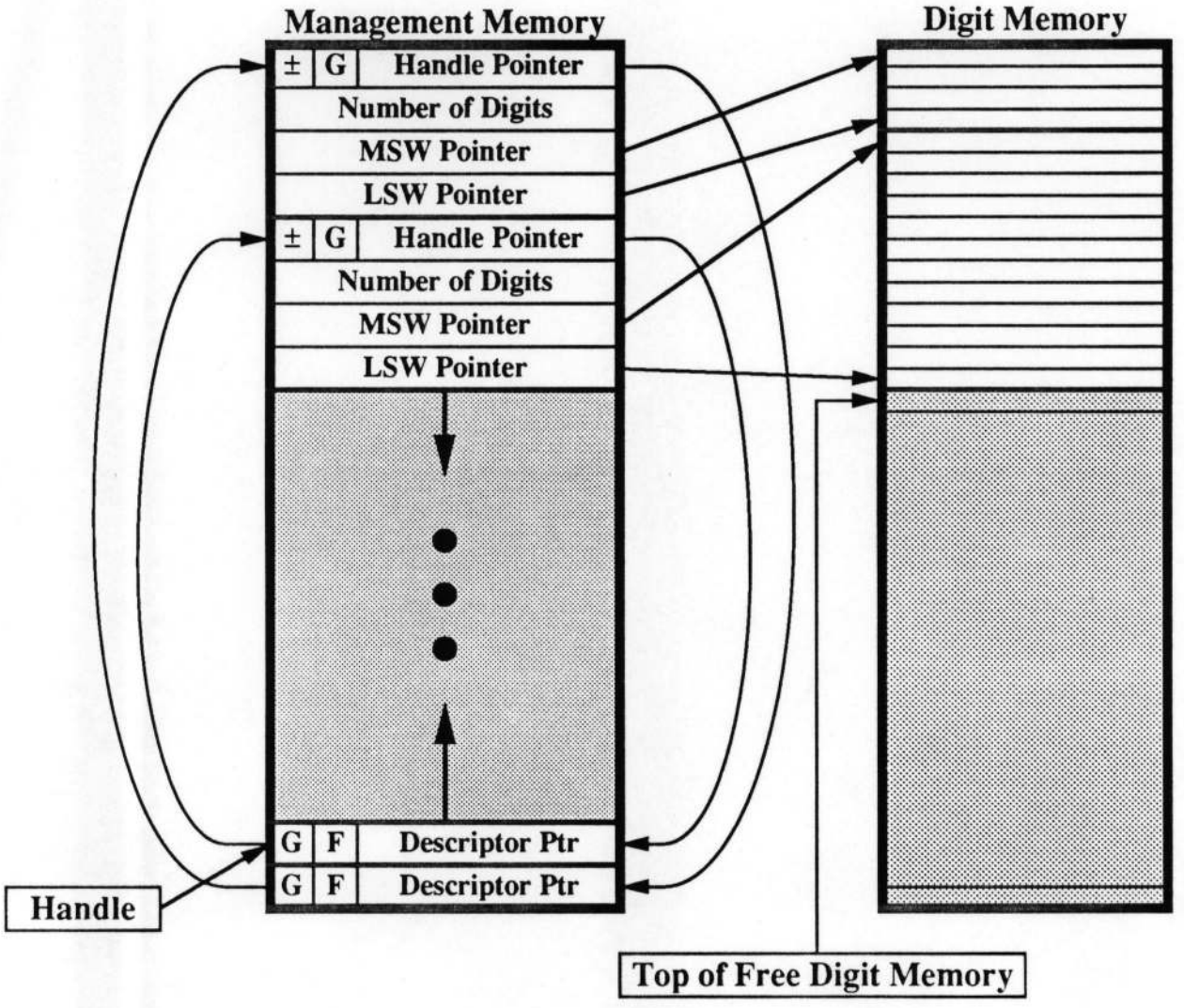

Figure 4: Cascade's Memory Organization

Table 1: Summary of Message Protocols

REQUESTS, *Results*, [Repeated Message Cycle]

CREATE	MS 16-BITS	LS 16-BITS	*Handle*
DESTROY	Handle		
RESTORE	NTRANSFERS	[4 DIGITS]	*Handle*
SAVE	HANDLE	*Ntransfers*	*[4 Digits]*
ASSIM	HANDLE	*NTransfers*	*[2's Comp]*
NEG@	HANDLE	*Handle*	
ADD@	HANDLE-ADD	HANDLE-ADD	*Handle-sum*
SUB@	HANDLE-ADD	HANDLE-SUB	*Handle-diff*
MUL@	HANDLE-MPY	HANDLE-MCD	*Handle-prod*
DIV@	HANDLE-NUM	HANDLE-DEN	*Handle-quot*
SQRT@	HANDLE-RAD	*Handle-root*	
REM	*Handle-rem*		
COMPARE@	HANDLE-ADD	Handle-sub	*Comparison*
SIGN	HANDLE	*Sign-ind*	
DIGITS	HANDLE	*MS ndigits*	*LS ndigits*
SETREG	BIT-PATTERN		
GETREG	*Bit-pattern*		
GC			

Table 2: Propagation Delays

Memory Cycle	t_m							25 ns
Arithmetic Operators	t_a							2 ns
Input Pad	t_i							2 ns
Output Pad	t_o							10 ns
Register	t_r							5 ns
Shift-Right	$t_<$	t_i	+	t_o	+	t_r	=	17 ns
Shift-Left	$t_>$	t_i	+	t_o	+	t_r	=	17 ns
Operand Fetch	t_f	t_m	+	t_i	+	t_r	=	32 ns
Result Save	t_s	t_m	+	t_o			=	35 ns
Add (N digit)	t_+	$2t_f$	+	$15t_a$	+	t_s	=	127 ns
Sub (N digit)	t_-	$2t_f$	+	$15t_a$	+	t_s	=	127 ns
Mul (per digit)	t_x	$t_>$	+	$23t_a$	+	t_r	=	68 ns
Mul (N by 1 digit)	t_*	$2t_f$	+	t_x	+	$2t_s$	=	202 ns
Div (recursion)	t_d	$t_<$	+	$23t_a$	+	t_r	=	68 ns
Sqrt (recursion)	t_t	$t_<$	+	$35t_a$	+	t_r	=	92 ns