DESIGN OF ARITHMETIC ELEMENTS FOR BURROUGHS SCIENTIFIC PROCESSOR*

Daniel D. Gajski** and L. P. Rubinfield***

Abstract

The design criteria and implementation of the Arithmetic Element (AE) of the Burroughs Scientific Processor, a vector machine intended for scientific computation requiring speed of up to 50 million floating-point operations per second, is discussed. An array of 16 AEs operate in lockstep mode, executing the same instruction on 16 sets of data. The 16 AEs are one stage in a pipeline which consists of 17 memory modules, an input alignment network, and an output alignment network. The AE itself is not pipelined. It can perform over one hundred different operations including a floating-point addition, subtraction and multiplication, division, square root, among the others. Eight registers are provided for the storage of intermediate values and results. Modulo 3 residue arithmetic is used for checking hardware failures.

1. Introduction

The Burroughs Scientific Processor (BSP) is designed to provide very high-speed execution of algorithms used to solve complex scientific and engineering problems.[1,2,3]

Several high performance systems have been designed in the past decade; some using pipeline organization (CDC STAR, TI ASC, CRAY-1), while others used array organization (Burroughs ILLIAC IV, PEPE, and The Goodyear Aerospace STARAN). Pipeline machines perform very well on long vectors while the pipeline setup decreases the efficiency drastically for short vectors. Furthermore, partitioning of processing units into segments and then overlapping several instructions on one or more pipelines requires very sophisticated and complicated control. On the other hand, array machines perform very well on vectors that are of the same size as array or one of its multiples. The problem of array machines is availability of data for all processing units all of the time.

The BSP combines parallelism and pipelining. Since FORTRAN had been chosen to be the main programming language of the machine, the array-oriented memory and processing was adopted. A parallel memory system provides conflict-free access of frequently used patterns (rows, columns, diagonals, etc.) of a multidimensional array. An array of processing units provides high-quality numerical computation using approximate R* rounding schemes, and mod 3 arithmetic codes for error detection. Since array elements are not stored in the order they are processed, input and output alignment networks are used to bring the data to their corresponding processing units. Both alignment networks are full crossbar switches. The memory array, two alignment networks, and the processor array constitute five-stage memory-to-memory data pipeline, doing fetching, aligning, processing, aligning and storing of vectors. Each of these operations, including nonarithmetic processing, is performed in one 160 ns clock period. However, arithmetic operations require more than one clock period (floating-point addition (2), floating-point multiplication (2), floating-point division (8), floating-point square root (12)). A relatively long clock period was chosen in order to reduce the number of pipeline segments to avoid a complicated clock distribution, and to simplify manufacturing and testing by using standard procedures and tools.

To guarantee a high utilization of the memory-to-memory pipeline, each BSP instruction corresponds to an assignment statement of one to five vector or array arguments. This corresponds to simple or double nested loops in FORTRAN. The vectors or arrays are automatically sliced into 16 element slices and the slices are overlapped in the pipeline. Furthermore, the instructions are pipelined in the control unit, so that the setup time for each instruction is negligible. Up to four of these vector instructions can be in execution on the pipeline at the same time. This instruction overlapping results in high performance even for short vectors.

In addition to high performance, the machine instructions that resemble the source program statements simplify compiler design and control unit. On the other hand, the user is allowed to tune his problem to the machine and achieve better efficiency by using vector instructions that

* This work was supported in part by NSF Grant MCS73-07980.

** Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*** Burroughs Corporation
Great Valley Laboratories - P.O. Box 517
Paoli, Pennsylvania 19301

CH1412-6/78/0000-0245$00.75 (c) 1978 IEEE

FORTRAN has been extended with.

To support the high rate of computation, BSP uses a CCD file memory that serves as a buffer between the front-end processor and BSP. Usually, the standard I/O equipment may not be adequate for a high-performance CPU, since it makes the computation dependent on the I/O speed. The CCD file memory should smooth the flow of data between the BSP and the front-end processor, and provide uninterrupted high-speed execution of user programs on the BSP. The front-end processor is the system manager that provides the rest of the chores: compilation and linking of BSP programs, data communication and time-sharing services for the user, long-term storage and database management and other general-purpose data processing services.

The BSP system shown in a simplified block diagram in Figure 1 consists of three major parts: the control processor, the array processor and the file memory.

The control processor (CP) is a high-speed element of the BSP that provides the supervisory interface to the system manager in addition to controlling the parallel processor and the file memory. The control processor executes some serial or scalar portions of user programs utilizing an arithmetic element similar to the sixteen arithmetic elements in the parallel processor, but containing additional capabilities to perform integer arithmetic and indexing operations. The CP also performs task scheduling, file memory allocation, and I/O management under control of the BSP operating system. It consists of four units:

a) The scalar processor unit (SPU) processes all operating system and user program instructions, which are stored in control memory. It has a clock frequency of 12.5 MHz and is able to perform up to 1.5 million floating-point operations per second. All array instructions and some scalar instructions are passed to the parallel processor control unit, which queues them for execution on the parallel processor.

b) The parallel processor control unit (PPCU) receives array instructions from the SPU. The instructions are validated and transformed into microsequences that control the operation of the sixteen arithmetic elements in the parallel processor.

c) The control memory (CM) is used to store portions of the operating system and user programs as they are being executed. It is also used to store program data values that are operands for those instructions executed by the scalar processor unit. The control memory is a 4K bit NMOS memory with a 160 ns cycle time. Four words can be accessed simultaneously. Capacity of the memory is 262K words; each word consists of 48 data bits and 8 bits for error detection and correction.

d) The control and maintenance unit (CMU) serves as the direct interface between the system manager and the rest of the control processor for initialization, communication of supervisory commands, and maintenance. It communicates with the input/output processor of the system manager. The CMU has access to most data paths and registers of the BSP, so that it can perform state analysis and circuit diagnostics under control of maintenance software running on the system manager.

The parallel processor (PP) performs array-oriented computations at high speeds by executing 16 floating-point operations simultaneously in its 16 arithmetic elements of the parallel arithmetic. Data for the array operations are stored in a parallel memory consisting of 17 memory modules. Parallel memory is accessed by the arithmetic elements through an input and output alignment network. Thus, the memory-to-memory pipeline consists of four units:

a) The parallel memory (PM) consists of from .5 to 8 million words. Like the control processor memory, it is a 4K bit bipolar memory. Each word contains 48 data bits and 8 bits for error detection and correction. The rate of data transfer between the parallel memory and the arithmetic elements is 100M words per second. The parallel memory is organized internally into 17 modules, permitting simultaneous access to almost any consecutive 16 elements of the commonly referenced components of an array, such as rows, columns, and diagonals.

b) The input alignment network (IAN) is a full crossbar switch used to establish natural order since the adjacent elements of a row, column, diagonal or any other entity are not necessarily stored in adjacent memory modules. In addition, the IAN can broadcast scalars from SPU to all arithmetic elements in parallel arithmetic and resolve conflicts if several arithmetic elements request data from the same memory module.

c) The output alignment network (OAN) is similar to the IAN in switching the results to be stored in proper memory modules. Furthermore, the OAN is used to support communication between arithmetic elements in execution of instructions like min, max, n-ary sum, Fast Fourier transform, etc.

d) The parallel arithmetic unit (PAU) consists of 16 arithmetic elements (AE) that are the subject of this paper. At any time, all of the arithmetic elements are executing the same instructions on different data values. The arithmetic elements operate at a clock frequency of 6.25 MHz and are able to complete the most common arithmetic operations in two clock periods. Each arithmetic element can perform a floating-point add, subtract, or multiply in 320 nanoseconds, so the BSP is capable of executing up to 50 million floating-point operations per second. Each arithmetic element can perform a floating-point divide in 1280 ns and extract a square root in 2080 ns. Thus, the BSP can execute approximately 12.5 million divide instructions or 7.7 million square root instructions in one second.

The file memory (FM) is a high-speed secondary storage device. The FM is loaded by the system manager with BSP code and data files for a task. The task is then queued for execution by the control processor. The FM is also used to store scratch files and output files produced during execution of a BSP program. It is the only peripheral device under the direct control of the BSP; all other peripheral devices are controlled by the system manager.

The FM utilizes high-speed charge coupled devices (CCD) as its storage media. The CCD memory combines a .5 millisecond access time with a 12.5M word/second transfer rate. The high speed of the FM provides performance that balances the high speed computational units of the BSP. Since it is entirely electronic, the reliability of the file memory is much greater than that of conventional rotating storage devices.

The file memory control unit (FMCU) provides a further distribution of function within the BSP by performing logical address translation for accessing data file records. The FMCU also queues prioritized I/O requests and performs automatic error retry, thus avoiding interference with the control processor for routine I/O housekeeping operations. The FMCU is given logical record descriptors that are converted into physical memory addresses. This facilitates implementation of file allocation and protection of file memory against illegal access. The FMCU also contains buffer areas that balance the difference between the data rates of the file memory and the system manager. The rate of data transfer between the data file memory and the system manager is 250K words per second. The rate of data transfer between the file memory and either the control processor memory or the parallel memory is 12.5M words per second.

2. AE Considerations

2.1 AE Design Goals

The design and implementation of a parallel arithmetic unit for the execution of operations on vector sets by replication of an Arithmetic Element (AE) 16 times poses numerous problems.

Firstly, a simple and inexpensive method to control each AE needed to be developed. Although the same command is broadcast to all AEs, some operations are data dependent. For example, an addition may become a subtraction if one of the operands is negative. Thus, one AE may generate an overflow in its mantissa adder, shift one position to the right, and add 1 to the exponent, while the other AE may generate leading zeros during subtraction, shift to the left and subtract the shift amount from the exponent. If these operations are not executed in one clock (register-to-register transfer), a local control sequencer must be designed into each AE. Such a distributed control makes testing and maintenance more complex and it increases the cost by replicating unnecessary hardware. An opposite approach of completely centralized control was adopted in BSP with data-independent algorithms implemented in the AE logic and microprogram. This requirement of synchronous operation on different data sets may slow down operations that require several clocks for execution. For example, the double-precision addition is executed in 8 clocks instead of 7 needed to add any two double-precision numbers.

Secondly, an AE design with maximum speed of arithmetic without excessive hardware is an obvious requirement for a scientific processor. Since 48-bit floating-point format may be considered too short for some applications, fast-double precision arithmetic was required. A good, unbiased rounding and detection of hardware failures were desired, too.

Thirdly, the implementation was to be executed with 4 PC boards with each board containing 200, 24 pin MSI and SSI Burroughs Current Mode Logic integrated circuits.

Finally, an AE was planned to be used in SPU.

2.2 Data Representation

The data word size of 48 bits was chosen as a tradeoff between memory cost, accuracy of computation and compatibility with the front-end processor.

A B7700 that serves as a front-end processor uses 48-bit data format. Thus, the most efficient storage and transfer of programs and data between the BSP and B7700 would be in 48-bit increments. On the other hand, a high degree of accuracy is desirable in some scientific applications. Sixty-four-bit data format is considered adequate for these applications. However, this means a 25% increase in cost of memory, data paths, and a 40% increase in arithmetic. Furthermore, the higher precision is usually required to compensate for poor rounding available on present-day computers. A 48-bit data format with good and consistent rounding was the obvious compromise. A fast, double-precision arithmetic is available as a substitute for higher precision.

A single-precision floating-point number is represented with 11 bits of exponent and 37 bits of mantissa. The range of $10^{\pm 300}$ is considered adequate for scientific applications.[4] For a given word size and the exponent range, binary arithmetic provides potentially the greatest accuracy.[5,6] The argument that higher-radix representation requires less shifting during addition[7] does not hold since BSP uses a combinatorial barrel shifter. Furthermore a good rounding scheme requires three guard digits, which means only three additional bits for binary arithmetic and twelve additional bits for hexadecimal arithmetic.

Both exponent and mantissa use sign-magnitude representation which allows symmetric distribution around zero. Furthermore, it is easier to work with during the design, testing, maintenance and reading memory dumps. The frequently used biased exponent was considered, but no improvement in

hardware speed or cost was obtained. Furthermore, a biased exponent requires corrective computation in higher order arithmetic operations (multiplication, division, square root, etc.)

The floating-point numbers are normalized and the binary point is understood to be on the left of the mantissa. A double-precision value X is represented as two single-precision values X_1 and X_2 such that $X = X_1 + X_2$. It is always the case that $e(X_2) \leq e(X_1) - 36$ unless the mantissa of X_2 is zero, where $e(X_2)$ and $e(X_1)$ are signed exponents of X_2 and X_1, respectively. The signs of X_1 and X_2 must be the same, except when $X_2 = 0$.

Since the range of single-precision numbers was considered adequate for double-precision, the above double-precision format was chosen to simplify and speed up double-precision algorithms. Each part of a double-precision number can be handled independently, and no operation executed in one clock ever depends on any information from previous-clock operation. For example, during the normalization of a double-precision number the shift amount (the number of leading zeros) is never stored or in any other way passed from the first to the second part. This kind of consideration made arithmetic completely combinatorial and thus easily testable.

An integer has a zero exponent and assumed binary point on the right of the least significant bit.

The logical value true is represented by a word of all ones and false with all zeros.

A character is stored as an 8-bit byte using EBCDIC. Six characters may be packed into one 48-bit word.

To the basic BSP word of 48 bits, 12 other bits are appended to form a 60-bit AE word: a 1-bit null indicator, 3 error condition bits (underflow, overflow, and undefined), 4 guard bits, 2 bits of the mantissa modulo-3 residue, and 2 bits of the exponent modulo 3 residue. For input and output purposes only 49 bits are used: the 48-bit data word, and the null indicator. The AE data word is shown in Figure 2.

2.3 Instruction Set

A rich instruction set with over 100 instructions has been microprogrammed for the AE. The microinstruction register is 114 bits wide. A horizontal type of micro control is used to achieve the maximum amount of parallelism and allow for a variety of instructions. Some instructions have several hundred variations. Only a few of them are included in the present instruction set. The main goal of logic design was to group similar functions into independently controlled logic blocks, and thus provide clean design, simplified testing and easy change in the instruction set. For example, two operands in any instruction may have their sign sets in more than 100 different ways before they are operated upon. The setting of signs is performed independently of any other operation in the AE.

The instruction set is divided into eight broad categories:

Boolean
conditional
shift and extract
single-precision floating-point
double-precision floating-point
integer
conversion
miscellaneous

The set of conditional instructions may have two different consequences: the result generated by a conditional instruction may be operand A or operand B and/or a test bit T may be set or reset, depending on the condition specified. Some typical instruction in this class may look like:

$$\text{IF} \left\{ \begin{matrix} A \\ 0 \\ T \end{matrix} \right\} \text{relation} \left\{ \begin{matrix} B \\ 0 \\ T \end{matrix} \right\}$$

$$\text{THEN return} \left\{ \begin{matrix} A \\ B \\ \text{TRUE} \end{matrix} \right\} \text{and set T bit}$$

$$\text{ELSE return} \left\{ \begin{matrix} A \\ B \\ \text{FALSE} \end{matrix} \right\} \text{and reset T bit}$$

where relation may compare two operands, test one of the operands for underflow, overflow, undefined, evenness, or the lack of it, or test T bit. Note that min and max functions are executed within one microinstruction. The T bit is the mechanism used to communicate the status of AE to ACU or from one microinstruction to another inside AE. Shift and extract instructions provide for shifting or rotating of any number of positions, bit setting, complementing and testing, and field masking, extraction and insertion.

The miscellaneous group contains instructions for calculating addresses of operands that are randomly distributed in 17 memory modules and converting character strings into binary numbers and vice versa.

3. Subunit Definition

In the realization of the AE, we found it helpful to associate a subfield of the microword with a functional subunit. Eight such subunits evolved: the Register, Shift and Mask, Arithmetic and Logic, Exponent, Error, Multiplication Unit and Residue Logic (Figure 3). While microword fields could be easily defined to only one particular subunit, each subunit interplayed closely with others. Figure 3 does not show all the connection between the units.

The register unit (RU) forms the major

interface with the alignment network. It provides for storage of input data and contains the AE output registers. Furthermore, it provides the multiplexing capability for selecting either logic, arithmetic or multiplication results from the other AE subunits. These results can be stored in one of six temporary registers, an output register, or the two working registers RO and R1. Two operands used by all other subunits are always coming from registers R_0 and R_1. This way there is no time delay associated with reading the file of general-purpose registers, i.e., the access time is equal to zero.

The shift and mask unit (SMU) selects one of the two operands (R_0 or R_1) and rotates it a specified number of places. The specified fields of the rotated data can then be masked on or off, to yield a shifted result. The shift and mask amounts can be specified from the exponent difference, the number of leading zeros or various fields of the other operand. In case of arithmetic operations, the input to SMU is a single-precision, 36-bit mantissa and the output to ALU is the double-precision, 72-bit mantissa.

The nonshifted operand may be extended to 72 bits with various constants in the least significant part. These constants are used in rounding, and in instructions like floor, ceiling, and fraction.

The arithmetic and logic unit (ALU) operates on the two output operands from SMU. The ALU field of each microinstruction specifies a pair of operations that can be executed in ALU. Some of the frequently used pairs are: <add, subtract>, <TRUE, FALSE>, <left operand, right operand>, and <AND, OR>. One of the operations in the pair is selected depending on the operands' signs, the validity of a prespecified relation between the two operands or the state of the TEST flip-flop. The ALU is 72 bits wide.

The exponent unit (EU) operates on the exponents of the two operands. Two results are available--one for each part of the double-precision result.

The error processing unit (EPU) is the central point to which error conditions are reported. Underflow, overflow and undefined status bits are appended to each result and are reported to the Array Control Unit whenever an output data is read. Because it was often desirable to continue a computation even after underflow occurred, hardware is included to set a result which has underflowed to zero.

The residue unit (RU) has no microinstruction field associated with it. In that respect, it can be thought of as a simple extension of hardware in other units. Because of its unique functional role and the problems the RU created, it is shown as a separate unit. For every floating-point number, the exponent and the mantissa part each are extended in the IAN with 2-bit modulo 3 residue. In arithmetic microinstructions, valid residues are generated, concatenated with the result, and stored. If the result is an intermediate value, then the residue is checked when its dataword is again operated upon. In other words, only data in registers R_0 and R_1 has its residue checked. If the result is sent to an OAN, then the OAN checks the residue and reports check back to the AE. Logic microinstructions do not use residues for checking. A code of two ones (11) in the residue field of a dataword indicates the absence of the valid residue and prevents the error-detection logic from detecting an error. All errors are reported to ACU on a clock-to-clock basis.

The sign unit (SU) is a very simple unit that generates proper signs in arithmetic microinstructions.

The multiplication unit (MU) includes logic for execution of a 38×18-bit multiplication. A A full 36×36-bit product requires two cycles through (MU). In addition to XY, the MU generates 2 - XY and (3-XY)/2 which are used in calculation of 1/N and $1/\sqrt{N}$.

The microinstructions with numeric operands (single-precision floating-point and integer numbers) use all units to generate the result. Non-numeric data is operated upon in SMU and ALU only.

4. Algorithms and Implementation

Within the combinational section of the hardware, two major functions exist, namely, addition and multiplication. All the other hardware is in reality an appendage to one of these two areas. The basic flow and implementation for these two areas is now detailed.

4.1 Addition/Subtraction Logic

The block diagram of the logic that executes 90% of all microinstructions, including mantissa addition and subtraction, is shown in Figure 4. During addition (subtraction), the most significant 36 bits of mantissa A remain unchanged passing through the SMU. The least significant 36 bits of A are created inside SMU from 4 guard bits and predefined constants of the CONSTANT SELECTOR. For example, during rounding of the operand B, zero is entered as the operand A. The constant $2^{-37}(1-2^{-35})$ is created in CONSTANT SELECTOR and added to B. If the least significant 36 bits of B have value greater than 2^{-37}, a carry is propagated from ALU2 into ALU1 rounding up B. If the value is equal or less than 2^{-37}, no carry-propagation occurs. However, the least significant bit of ALU1 is forced to 1 if no carry occurred and the most significant bit of ALU2 is 1.

The same trick with a different constant is used during calculation of ceiling and floor functions ($\lceil x \rceil$, $\lfloor x \rfloor$) and their inverses ($\lceil x \rceil - x$, $x - \lfloor x \rfloor$).

During addition and subtraction, the B

operand (which is always the smaller of the two operands) passes through the ROTATOR that only rotates right from 0-47 positions. The extended shift right of 0-72 positions is created by the MASK GENERATOR.

The MASK GENERATOR is a circuit that produces a mask vector, that is, a sequence of zeros followed by ones, which is ANDed with data from the ROTATOR. The number of most significant zeros is determined by the MASK AMOUNT (MA). Using two control lines, a MASK GENERATOR may output:

a) a zero on all output lines;

b) the mask vector as a function of the MA;

c) the complement of the mask vector; or

d) a one on all output lines.

During addition, the MA is the same as ROTATION AMOUNT (RA). However, during the field manipulation instructions like field transfer from one dataword into another, a speedup of 2-3 times can be obtained by properly selecting RA and MA that are not equal in that case.[8]

The ALU is a 72-bit wide adder/subtractor performing a one's complement calculation using two levels of carry-lookahead. If during addition overflow should occur, the COMPLEMENTER/SHIFTER shifts the sum to the right by one position. If the ALU is performing a one's complement, subtraction and operand A is less than the shifted value of operand B, then to return the magnitude of the difference, a bit-by-bit complement of the result is performed in the COMPLEMENTER/SHIFTER.

The two operations are specified in each microinstruction. One of them is selected according to the validity of a relation between A and B, a test on either A or B, the status of the TEST flip-flop, the value of the signs or the command from ACU. The COMPARATOR does the comparison between the floating-point numbers, their magnitudes, or checks only one of them for a specific value. The result of the text can be stored in the TEST flip-flop and used in the next microinstruction. If exclusively-ORed signs are interpreted as a relation between A and B, all of the microinstructions realized by SMU and ALU are given by the following microinstruction form:

IF $\left\{\begin{array}{l} A(n) \text{ relation } B(n) \\ A(n-1) \text{ relation to } B(n-1) \end{array}\right\}$

THAN A(n) op_1(B(n)SHIFTED)

ELSE A(n) op_2(B(n)SHIFTED)

where A(n), B(n) are the values of operands on inputs A and B at n^{th} clock.

4.2 Multiplication

Figuve 5 shows a block diagram of the multiplication hardware. The multiplication hardware is capable of performing a 38-bit by 18-bit multiplication using the modified Booth's algorithm.[9] Two iterations through the hardware are required to obtain a normalized 36-bit product from two 36-bit operands.

The MULTIPLIER SELECTOR selects the appropriate half of the multiplier. For the first iteration the least significant 18 bits are used; for the second iteration the most significant 18 bits are selected. The selected multiplier bits control the selection of a multiple of the multiplicand. Each multiple is either +2, +1, 0, -1, or -2. Ten such multiplies are developed. These multiplies are then partially added via a modified Wallace tree adder called a COLUMN COMPRESSOR. Each bit slice of the COLUMN COMPRESSOR adds 7 bits without propagating carries. The three-bit sum is then reduced to two bits. The partial product and a rounding constant, can also be added in to the COLUMN COMPRESSORS, when appropriate. The final product is developed by summing the remaining two bits of each bit slice with a CARRY PROPAGATE ADDER. Since the fractional mantissas can yield a product in the range [1/4,1) and since numbers in the range [1/4,1/2) need to be normalized via a shift left, two products are developed, one for the nonshifted case [1/2,1) and one for the shifted case [1/4,1/2). The two products are not necessarily shifted copies of one another since rounding may change one radically. At the end of the first iteration, the most significant 36 bits of the product are saved in the PARTIAL PRODUCT REGISTER. These bits are then added in on the second iteration.

4.3 Reciprocal and Square Root

Division in the AE is accomplished by first taking the multiplicative inverse (reciprocal) of the denominator and then multiplying this by the numerator.

The reciprocal of A is found using Newton's method to solve the equation $F(X) = \frac{1}{X} - A = 0$. Newton's method for solving an equation required the selection of the first approximation, X_0, (the seed value) and the formation of X_1, X_2, ... using the following recurrence relationship

$$X_{n+1} = X_n - \frac{F(X_n)}{F'(X_n)}$$

For reciprocation, the above recurrence relationship becomes

$$X_{n+1} = X_n\ (2-A\ X_n).$$

Thus the development of each X_{n+1} requires two steps. The first produces $Y = 2-(A\ X_n)$, and the second $X_{n+1} = Y\ X_n$. The above recurrence formula can be easily obtained from series expansion. Since $0.5 \leq A < 1$ then $1/A = 1/1-\varepsilon$ where $0 < \varepsilon \leq 0.5$. Therefore,

$$\frac{1}{A} = \frac{1}{1-\varepsilon} = \frac{(1+\varepsilon)}{1-\varepsilon^2} = \frac{(1+\varepsilon)(1+\varepsilon^2)}{1-\varepsilon^4}$$

$$= \ldots = \frac{1}{1-\varepsilon^{2^{(i+1)}}} \prod_{k=0}^{k=i} (1+\varepsilon^{2^k})$$

For all practical purposes, the $(i+1)^{th}$ approximation of $\frac{1}{A}$,

$$X_{i+1} = \prod_{k=0}^{k=i+1} (1+\varepsilon^{2^k})$$

$$= [\prod_{k=0}^{k=i} (1+\varepsilon^{2})](1+\varepsilon^{2^{(i+1)}})$$

$$= [\prod_{k=0}^{k=i} (1+\varepsilon^{2^k})](2-(1-\varepsilon^{2^{(i+1)}}))$$

$$= [\prod_{k=0}^{k=i} (1+\varepsilon^{2})](2-(1-\varepsilon)\prod_{k=0}^{k=i} (1+\varepsilon^{2^k}))$$

$$= X_i(2-A\,X_i)$$

This proves the equivalence of methods of series expansion and additive iteration defined by Flynn.[10]

The seed value, X_0 is an approximation of 1/A, and is found through ROM-based table lookup. The ROM table gives X_0 with 7 most significant bits correct. Since each iteration almost doubles the number of bits of accuracy of the reciprocal three iterations of two cycles, each are performed to develop a 36-bit result.

The square root of A is found by first finding $1/\sqrt{A}$ and then multiplying by A. $1/\sqrt{A}$ is found through the recurrence

$$X_{n+1} = X_n(3-X_n^2\,A)/2$$

Again, the subtraction is done by complementing the multiplier and forcing the constant 3 into the COLUMN COMPRESSOR. Each iteration of the above equation requires three passes through the multiplication hardware and three such iterations provide a 36-bit result.

4.4 Rounding

It was desired to select a rounding scheme which produces statistically unbiased results. For single precision operations, rounding is performed as follows: If an operation produces a 36+X bit resultant normalized mantissa, m(R), which is composed of three parts $m_1(R)$ is the 35 most significant bits; $m_0(R)$ is the bit with weight 2^{-36}, $m_X(R)$ are the remaining X bits of the result, then the following table gives the value of the rounded mantissa, m(C),

$m_X(R)$	m(C)
$< 2^{-37}$	$m_1(R) + m_0(R)$
$= 2^{-37}$	$m_1(R) + 2^{-36}$
$> 2^{-37}$	$m_1(R) + m_0(R) + 2^{-36}$

4.5 Double-Precision Algorithms

This section describes the double-precision algorithms for addition/subtraction and multiplication only. Others have been omitted because of space. Only three basic microinstructions are used in double-precision algorithms:

ADD microinstruction performs addition or subtraction of two single-precision floating-point numbers. The result is a double-precision floating-point number. Both parts of the double-precision result are stored in registers. If overflow occurs during addition, the magnitude parts of the result are shifted to the right and both exponents adjusted accordingly. Otherwise, the result is left unnormalized.

NORMALIZE performs normalization of a single-precision floating-point number. The result is a single-precision number obtained by shifting left end-off all leading zeros and adjusting exponent accordingly.

MULTIPLY produces a double-precision product of two single-precision floating-point numbers. Both parts of the double-precision result are stored in registers. The most significant portion is a normalized number.

To achieve a comparable speed with single-precision operations, a "carriage" double-precision operation has been implemented. That means that a full double-precision accuracy has not been achieved in double-precision operations, and an error has been allowed to appear in 1, 2 or even 3 least significant binary digits.

a) Addition/Subtraction Algorithm

There are only two different positions of two double-precision operands A and B (Figure 6)

The following algorithm gives the correct result in either case. Note that a guard bit is required on the least significant portion of the partial result C, which in turn requires implementation of more than 72 bits adder. This guard bit is omitted, which will cause the error in two least significant bits. The subscript G denotes the presence of 4 guard bits in the operand.

$A = A_1 + A_2$, and $B = B_1 + B_2$ are double-precision operands. $S = S_1 + S_2$ is the double-precision result.

1. $A_1 + B_1 = C_1 + C_2$
2. $A_2 + B_2 = D_1 + D_2$
3. $C_{2G} + D_{1G} = E_1 + E_2$
4. $C_1 + E_{1G} = F_1 + F_2$
5. NORMALIZE F_1
6. $F_1 + F_2 = S_1 + S_2$
7. NORMALIZE S_1
8. NORMALIZE S_2

b) Multiplication Algorithm

Multiplication of two double-precision numbers produces quadruple-precision results. Only the most significant half of the result is retained. That involves 4 multiplications and at least 6 additions, altogether 14 microinstructions. To speed up the double-precision multiplication to 10 microinstructions, low-order portions of the product are neglected. Let $A = A_1 + A_2$ and $B = B_1 + B_2$ be multiplicand and multiplier, respectively. Figure 7 represents 8 different partial products that have to be added to obtain a final product. The double-precision partial product $K = K_1 + K_2$ is neglected. The error made that way is equal to K and less than $2^{e(A_2) + e(B_2) + 72)}$, which is less than $2^{e(A_1) + e(B_1)}$. Since the product $P = P_1 + P_2$ may have one leading zero, the error $2^{e(A_1) + e(B_1)}$ may be multiplied by 2 during normalization process. Therefore, the negligence of K may result in errors that are less than two least significant binary digits. To minimize the negligence of C_2 (D_2), the four most significant bits of C_2 (D_2) are attached to C_1 (D_1) as guard bits. So, the transaction of C_2 and D_2 generates an error that is less than $2^{e(A_1) + e(B_1) + 1}(1 + 2^{-3})$; that is, 2 least significant bits.

$A = A_1 + A_2$, and $B = B_1 + B_2$ are double-precision multiplicand and multiplier. $P = P_1 + P_2$ is the double-precision product.

1. $A_2 \times B_1 = C_1 + C_2$
2. $A_1 \times B_2 = D_1 + D_2$
3. $C_{1G} + D_{1G} = E_1 + E_2$
4. $A_1 \times B_1 = F_1 + F_2$
5. $F_2 + E_{1G} = G_1 + G_2$
6. $F_1 + G_{1G} = P_1 + P_2$
7. NORMALIZE P_2

5. Performance

The AE has been built, tested and operative for more than a year. It uses Burroughs Current Mode Logic (BCML) family of standard MSI and SSI 24 pin leadless packages. For nonstandard functions in AE, two master-slice logic arrays with a maximum of 100 and 150 gates per package have been used. A dozen different types of circuits have been implemented using logic arrays. Without logic arrays, the original goal of 4 board AE could not be met.

Each microinstruction requires 160 ns for execution. That is translated into 320 ns for single-precision addition, subtraction and multiplication, 960 ns for reciprocal, 1280 ns for division, 1000 for square root reciprocal, 1280 for double-precision addition and subtraction, 1600 for double-precision multiplication and 2400 for double-precision reciprocal. Most nonnumeric instructions are executed in 320 ns.

References

1. R. A. Stokes, "The Burroughs Scientific Processor," in High Speed Computer and Algorithm Organization, ed. by D. J. Kuck, D. H. Lawrie, and A. H. Sameh, pp. 85-89, Academic Press, Inc., 1977.
2. D. J. Kuck and R. A. Stokes, "The Burroughs Scientific Processor (BSP)," submitted for publication.
3. C. Jensen, "Taking Another Approach to Supercomputers," Datamation, Vol. 24, No. 2, pp. 159-172, Feb. 1978.
4. W. J. Cody, "Desirable Hardware Characteristics for Scientific Computation," SIGNUM Newsletter, Vol. 6, No. 1, pp. 16-31, 1971.
5. R. Brent, "On the Precision Attainable with Various Floating-Point Number Systems," IEEE Trans. on Comp., Vol. C-22, No. 6, pp. 601-607, June 1973.
6. H. Kuki and W. J. Cody, "A Statistical Study of the Accuracy of Floating-Point Number Systems," Comm. ACM, Vol. 16, No. 4, pp. 223-230, April 1977.
7. D. W. Sweeney, "An Analysis of Floating-Point Addition," IBM Syst. Journal, Vol. 4, pp. 31-42, 1965.
8. D. D. Gajski and B. R. Tulpule, "High-Speed Masking Rotator," Digital Processes, Vol. 4, pp. 67-81, 1978.

9. L. P. Rubinfield, "A Proof of the Modified Booth's Algorithm for Multiplication," IEEE Trans. on Comp., Vol. C-24, No. 10, pp. 1014-1015, Oct. 1975.

10. M. J. Flynn, "On Division by Functional Iteration," IEEE Trans. on Comp., Vol. C-19, No. 8, pp. 702-706, Aug. 1970.

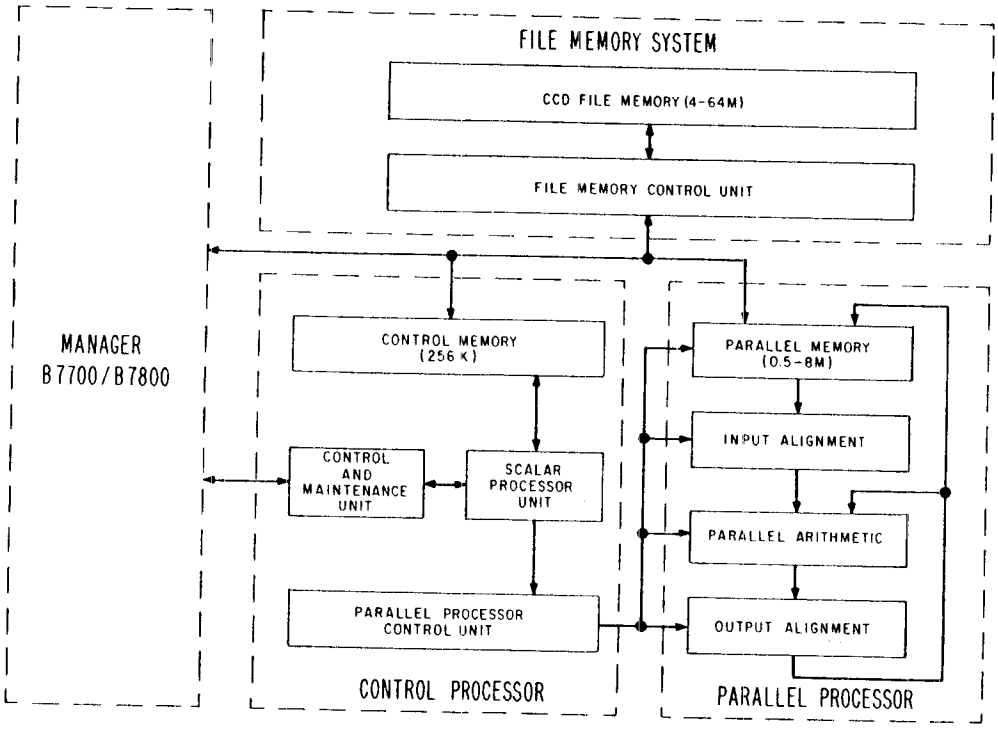
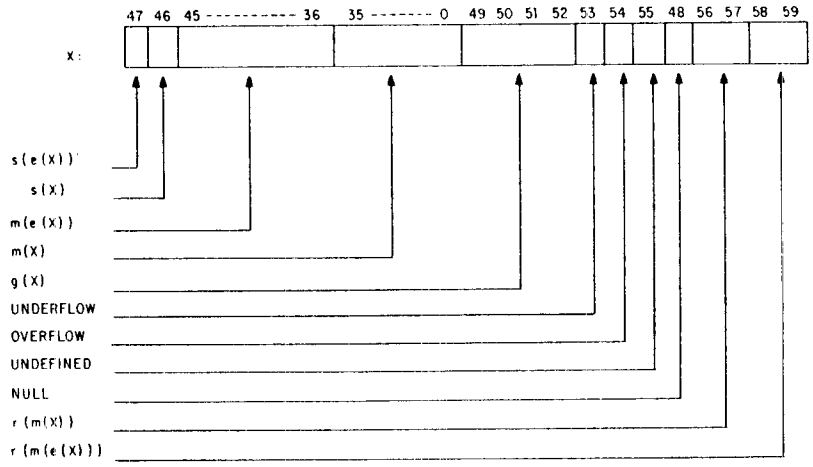


Fig. 1. BSP system



s(X) IS SIGN OF X
m(X) IS MAGNITUDE OF X
e(X) IS EXPONENT OF X
g(X) IS GUARD EXTENTION OF X
r(X) IS RESIDUE OF X MOD 3
$X_{00}-X_{48}$ IS INPUT (OUTPUT) TO (FROM) AE UNIT

Fig. 2. AE data format

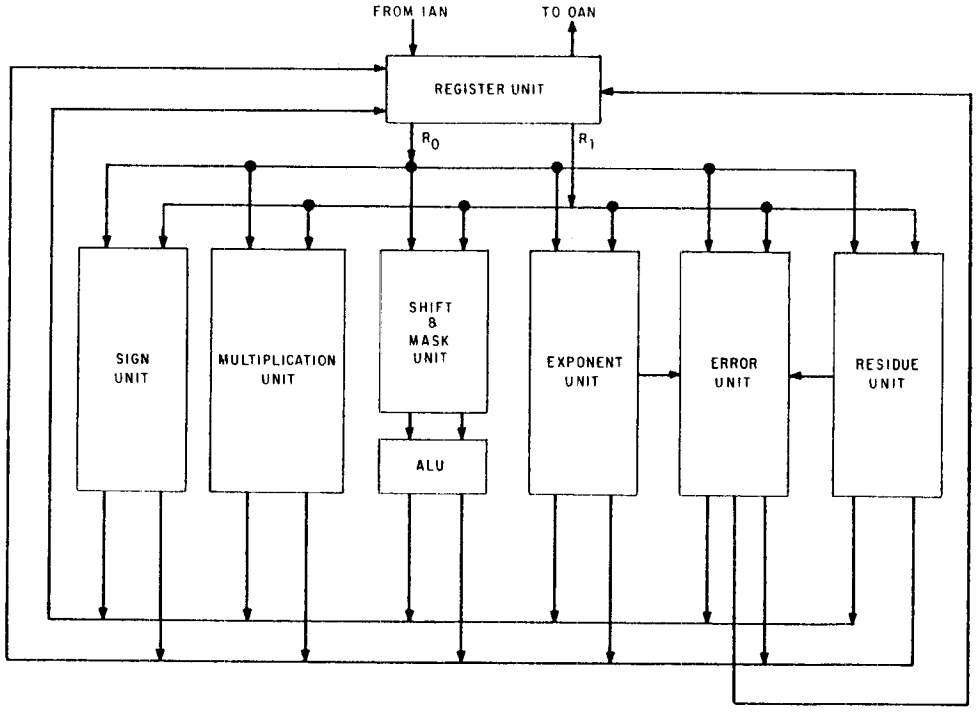


Fig. 3. AE block diagram

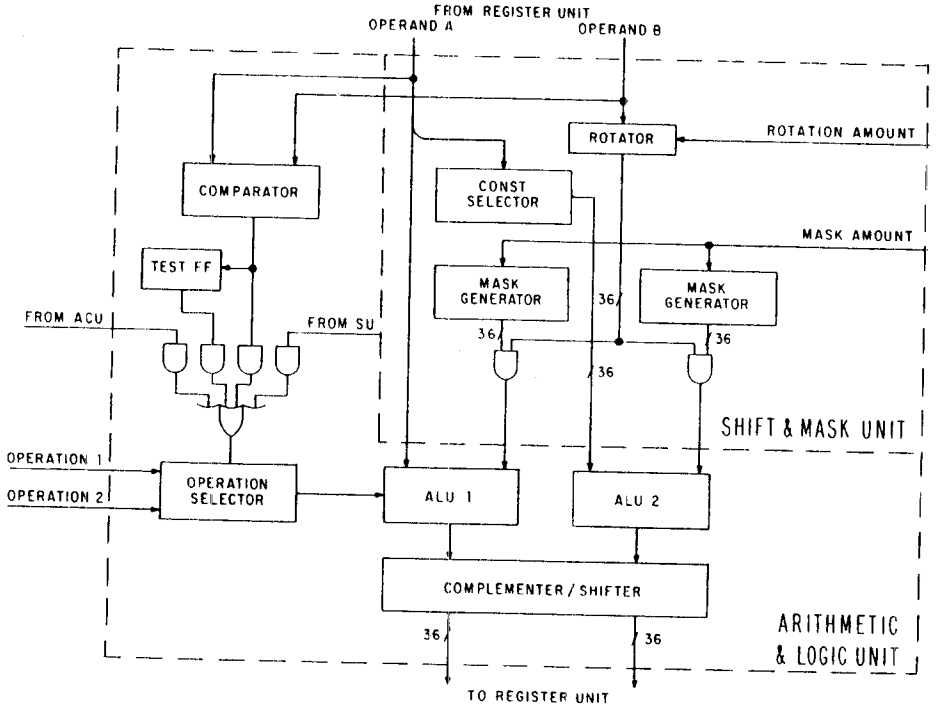


Fig. 4. SMU and ALU block diagrams

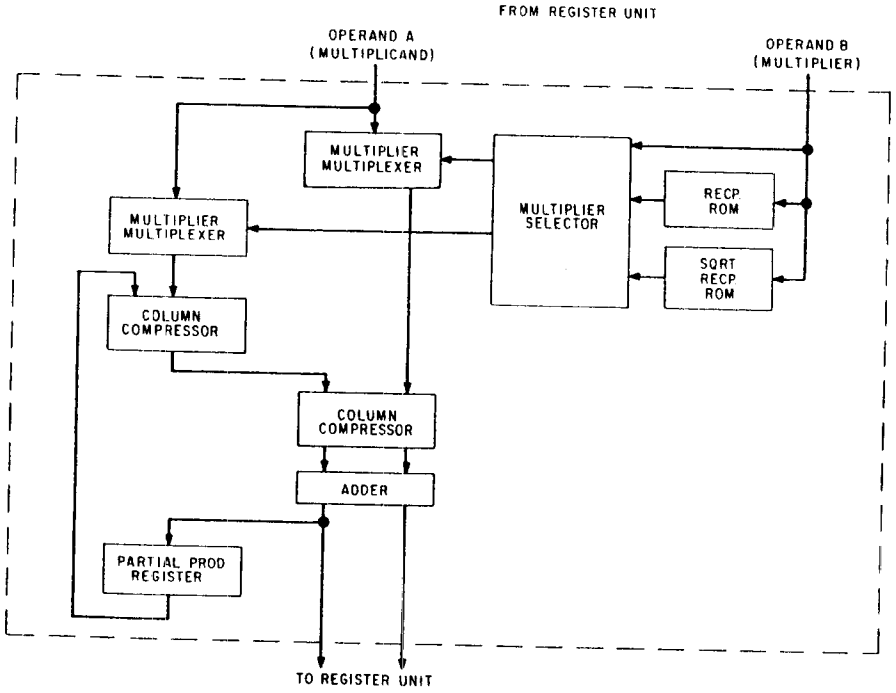


Fig. 5. MU block diagram

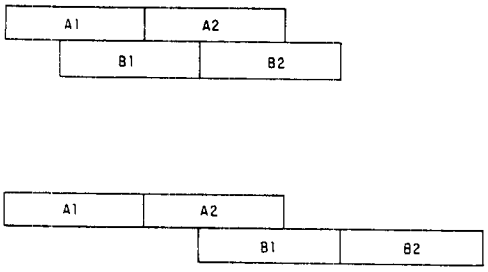


Fig. 6. Positions of operands in double-precision addition

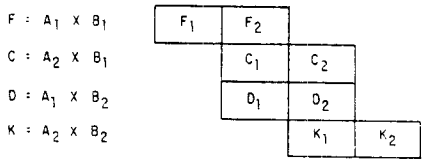


Fig. 7. Position of partial products in double-precision multiplication