THE RCA 301 SCIENTIFIC COMPUTER

A. TURECKI
Palm Beach Operations
Electronic Data Processing
West Palm Beach, Florida

A new high-speed arithmetic unit is now available for the RCA 301 Computer to add scientific capability to the basic RCA 301 System. With a new total of 58 instructions, a user can apply the RCA 301 efficiently to both technical and business problems. The new arithmetic unit can be added to any existing RCA 301 system, and a variety of subroutines and programs are made available by RCA for scientific applications.

THE *high-speed arithmetic unit*, latest enhancement of the RCA 301 System, adds scientific capability to the low-cost, high-capacity RCA 301 Computer. With this capability, a customer can acquire greater efficiency for his rental dollars and schedule time for both technical and business assignments.

With the addition of the high-speed arithmetic unit, the present RCA 301 instruction repertoire is increased from 48 to 58 instructions, and the execution of arithmetic instructions is accomplished at least 100 times faster. The 10 new instructions are: *Fixed and floating add, subtract, multiply* and *divide, shift register*, and *store register*. The data format of a fixed-point operand consists of eight digits, and that of a floating-point operand consists of ten digits. The least-significant two digits are the exponent, and the remaining eight digits are the fraction of the operand. The sign and the overflow are indicated by the standard RCA 301 zone bits.

The instruction format is consistent with the present RCA 301 format, the only variation being the interpretation of the *N* digit. The two most significant bits of the *N* digit indicate that the operands *a* and *b* are taken either from the *A* and *B* address locations or from the accumulator. The next bit controls whether the result is stored in the accumulator only or is also transferred to the *A* address location. The last three bits of the *N* digit control the selection of the three index fields for the *A* and *B* address modification which is carried out during staticizing. Also, the modified tally instruction permits the incrementing or decrementing of the index fields on the consecutive runs through a program loop. This adds a very powerful tool to the RCA 301 Scientific Computer.

To be consistent with the original RCA 301 modular philosophy, the arithmetic unit can be added to *any* RCA 301 System in the field. The estimated time of the field installation, which includes some necessary modifications to the 301 basic processor unit, is 40 hours.

RCA makes available a variety of scientific subroutines for matrix operations, linear programming, statistical analysis, differential equations, and curve-fitting, plus scientific and Bell Interpreter Systems, UMAC (an algebraic compiler employing FORTRAN mathematical statements), and RCA 301 FORTRAN. Also available are some special industrial programs, such as lens design, bus scheduling, electrical load distribution, power requirements analysis, and others.

Current marketing reports indicate that the RCA 301 Scientific Computer *has no serious competitors in its price range.*

FUNCTIONAL SPECIFICATIONS

After the modifications required for the addition of the high-speed arithmetic unit, the basic processor unit models 304 and 305 become Models 354 and 355, respectively. The arithmetic unit provides the hardware for high-speed fixed- and floating-point arithmetic and associated address modifications. The unit occupies 12 rows (a half rack) of the control module rack.

The arithmetic unit consists of an addressable upper accumulator and lower accumulator, two nonaddressable arithmetic registers, and a high-speed parallel adder. Standard core memory locations are assigned for address modification. They are the three Index fields used during staticizing and the three increment fields used by the modified tally instruction.

Any quantity falling within the range of 10^{-99} to 10^{+99} can be represented in the floating-point format to the eight-digit precision. All the floating-point results are automatically normalized and rounded. To make double-precision programs possible, the results of fixed point arithmetic are not rounded.

Since the operands of both fixed and floating arithmetic are word-oriented, they must always begin at even locations in the core memory. The floating-point zero is represented by the zero fraction with the exponent of −99.

All the arithmetic-unit errors stopping the computer are indicated by the ARIE alarm on the processor console. There are six different conditions that will ac-

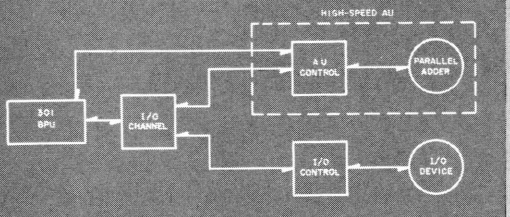

Fig. 1—High-speed AU interface.

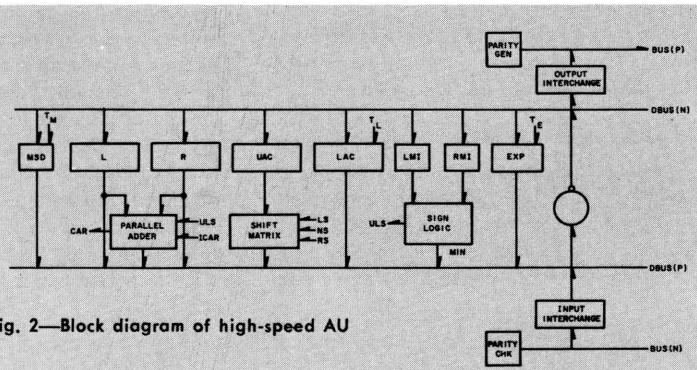

Fig. 2—Block diagram of high-speed AU

tuate the ARIE alarm: 1) *data parity error*, 2) *non-normalized operand*, 3) *dividend greater than divisor*, 4) *zero divisor*, 5) *zero operand with the exponent not equal to −99*, and 6) *modified address exceeding the core-memory capacity*.

The data characters in the arithmetic unit are numeric (straight binary-coded decimal) and therefore the 2^4 and 2^5 zone bits are ignored with the exception of the permissible 2^4 bit in the most-significant digit and the 2^5 bit in the least-significant digit of a quantity, signifying an overflow and negative sign, respectively. Both zone bits can be transferred from the arithmetic unit to the core memory, but only the zone bit signifying the negative sign can be transferred from the memory to the Arithmetic Unit.

Depending on the sign of the result, a corresponding PRI *(previous result indicator)* is set at the completion of any arithmetic instruction. SCAR, signifying an overflow, may be set only in *fixed add/subtract* and in any floating instruction indicating that the resultant exponent exceeds the range of ±99.

FUNCTIONAL DESCRIPTION

Intercommunication with 301 Basic Processor

The similarity between the high-speed arithmetic unit and any input-output control module (besides their common modular structure) can better be seen if the parallel adder is considered to be the arithmetic unit's peripheral device (Fig. 1). Some of the signals, not available from the input-output channel, are obtained through additional lines. These are codes of the new arithmetic unit instructions, new status levels, and special controls.

While the average input-output device speeds are far below the present processor speeds, a good balance was achieved between the 301 basic processor and the parallel adder in the high-speed arithmetic unit. Any further speeding up of the adder would not be justified by the performance-versus-cost relationship.

Although the high-speed arithmetic unit has its own eight-clock-pulse generator, it runs synchronously with the basic processor, BPU. The purpose of the arithmetic-unit clock is to subdivide status levels into more timing slots that are available from the 301 basic processor. This increases the number of performable functions per status level.

The High-Speed Arithmetic Unit

There are eight data buses (DBUS0 to DBUS7) connected, via read-in and read-out gates, to the corresponding digits of all the arithmetic-unit registers. The data from the core memory is transferred one dyad at a time via BUS2 and BUS3 of the input-output channel and the input interchange to the consecutive DBUS's starting from the least significant pair. The input interchange is switched by means of control signals IN1 to IN4 which are generated by a counter running during the loading of operands to the arithmetic unit. Similarly, the output interchange controlled by the signals OUT1 to OUT4 switches the consecutive pairs of DBUS's to the core memory via BUS2 and BUS3. There is a provision made to transfer all the four digits of *A* or *B* address at one time. Moreover, if the address has any zone bits in its most significant digit (signifying either 20,000 or 40,000 character memory) the input interchange generates an equivalent fifth digit. After performing an address modification, the resultant fifth digit is combined with the address most significant digit in the output interchange.

The parity of the incoming data is checked at the input interchange. All the DBUS's consist of four lines with the exception of DBUS0 and DBUS6 which also have 2^4 lines and DBUS7 which has a 2^5 line. After the execution of an arithmetic instruction, the four-bit numeric code is transferred back to the seven-bit code at the output interchange.

There are eight registers in the arithmetic unit (Fig. 2). Starting from the left, the registers are:

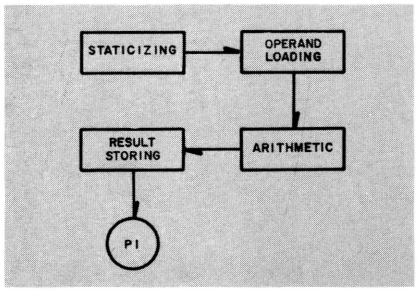

Fig. 3—General fixed point arithmetic flow chart.

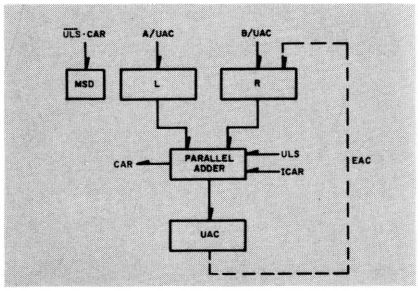

Fig. 4—*Add/sub* data flow chart.

Fig. 3—Fixed-point arithmetic instructions, general functional flow chart. The RCA 301 Computer, being a two-address system, requires *Ai* for the transfer of the result to the memory. This is accomplished by loading the *B* operand into the arithmetic unit and transferring *Ai* to the *B* register. This register is then used in the final step of the arithmetic instructions, which is loading of the result into memory.

Fig. 4—Add-subtract, fixed point. After loading the L and R registers, a minimum of 3 μsec is allowed for the adder to propagate carry. The output of the adder is then transferred to UAC. The result has an overflow if ULS·CAR. This condition is used for triggering up MSD. If MSD = 0, the 2^4 bit is set in the most significant digit of UAC. The condition ULS·CAR initiates the EAC procedure. In this case, L and R registers are reset, UAC→R, and $0 + \bar{R} \rightarrow$ UAC, thus correcting the result in the manner explained above.

Fig. 5—Multiply, fixed point. The loading of operands proceeds in a similar manner as in *fixed add-subtract*, but the destination of operands is different. The multiplicand *a* is transferred into R and the multiplier *b* into LAC. The multiplication algorithm consists of two cycles; a consecutive addition (generating partial product) and a right coupled shift. The consecutive addition proceeds as follows:

L + R is transferred to UAC.
If there is a carry, MSD is triggered up.
LAC7 (l.s.d. of LAC) is triggered down.
UAC is transferred to L and the cycle is repeated until LAC7 = 0.
When LAC7 = 0, the shift cycle performs a coupled right shift of MSD, L, and LAC, and triggers the digit counter down.
If the digit counter XC ≠ 8 and LAC7 ≠ 0, the first cycle is selected again.

After eight shifts, the multiplication is completed and the final product, consisting of 15 or 16 digits, is stored in UAC and LAC. For single precision, only the part in UAC is used. For double precision, the parts in UAC and LAC are used.

Fig. 6—Divide, fixed-point. The loading of operands is the same as in *fixed add-subtract*. The divident *a* is transferred into L and the divisor *b* into R. The division algorithm consists of two cycles; a consecutive subtraction (generating partial remainder) and a left coupled shift. The procedure is as follows:

A left shift of L is executed, transferring the m.s.d. of L into MSD. This step enables multiple precision.
The subtraction cycle is then started with L—R being transferred to UAC.
If there is no carry, MSD is triggered down.
If MSD ≠ 15, LAC7 is triggered up and UAC is transferred to L.
The cycle is then repeated.
When MSD = 15 (which means the difference L—R is negative), the left coupled shift of MSD, L, and LAC is performed, the digit counter is trig-

MSD—*most significant digit*. This is a single-digit register triggerable up or down. As its name implies, it is the most significant digit of sums, partial products, and partial remainders.

L and R—*left operand* and *right operand*. Both registers are eight-digit registers and their contents provide the input to the parallel adder.

UAC—*upper accumulator*. This register stores results and is associated with the shift matrix.

LAC—*lower accumulator*. This register, with its least-significant digit LAC7 (triggerable up or down), is used for generating the quotient in *divide* and exhausting the multiplier in *multiply* instructions. It is also used to store the second part of the product in *fixed multiply* and remainder in *fixed divide*.

LMI and RMI—These are single-bit registers used for storing the signs of operands. In association with the sign logic, they generate the ULS *(unlike signs)* signal and MIN (the sign of the results).

EXP—*Exponent Register*. This register is a two-digit algebraic up or down counter.

Parallel Adder and Sign Logic

The eight-digit parallel adder operates on straight binary-coded decimal digits.

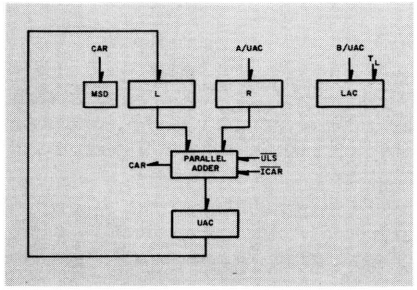

Fig. 5—*Multiply* data flow chart.

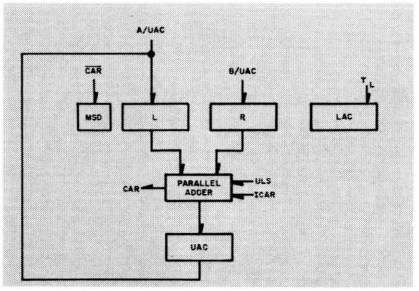

Fig. 6—*Divide* data flow chart.

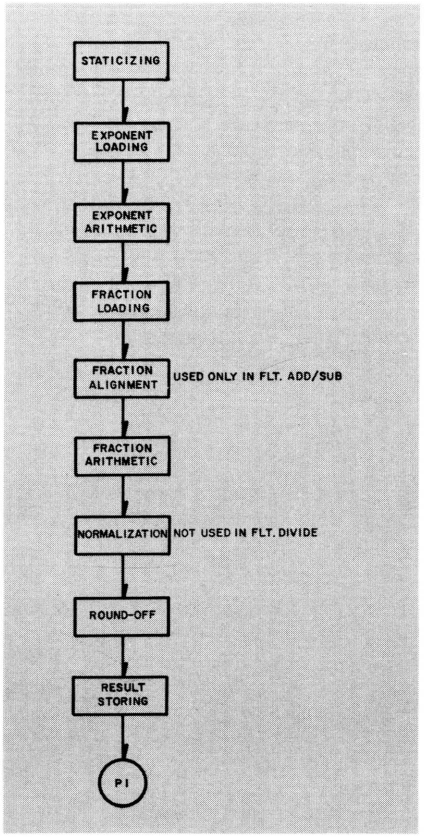

Fig. 7—General floating point arithmetic flow chart.

gered up, and the subtraction cycle is selected unconditionally.

The execution is terminated in the subtraction cycle when MSD = 15 and XC = 7.

Fig. 7—Floating-point arithmetic instructions. Floating-point arithmetic uses only the algorithms previously described and is similar to fixed-point arithmetic with some additional operations. The additional operations are exponent arithmetic, alignment of fractions (used in *floating add-subtract*, only), normalization of results, and round-off. The exponent arithmetic is a straight addition or subtraction depending on the instruction. The normalization is a left or right shift of the result with the corresponding correction of the exponent. The latter is done by triggering the exponent register algebraically up or down. The round-off is carried out on the basis of the least significant digit outside the eight digit precision. The general flow charts contained herein do not indicate any special paths. The charts are of a general nature because, depending on the instruction options and the result itself, some of the functions may be redundant. These procedures are bypassed, whenever applicable and justified by the involved cost, by means of special case status level selections.

Add or Subtract, floating point. The exponent arithmetic carried out three independent functions, and they are as follows: 1) it stores the algebraically larger exponent; 2) computes the exponent difference; and 3) determines which fraction is to be shifted to the right for the alignment purposes. During the alignment, the shifted-off digits from L and R are stored in LAC in a straight or complemented form, depending on the ULS signal state. This is done in order to provide the information for rounding-off. The fraction arithmetic is, in principle, identical to the *fixed add-subtract* arithmetic. During the normalization, the result is either shifted right (if MSD $\neq$ 0) or shifted left (if there are any zeros in the most significant portion of the result and MSD = 0). The rounding-off is performed by adding 1 to the result if LAC0$\geq$5. The solution of the case of an overflow produced during the round-off is left for the readers.

Multiply, floating point. The exponent arithmetic generates an algebraic sum of the operand exponents. The fraction arithmetic is identical to the one of the fixed multiply. As the final product may consist of 15 or 16 digits, the normalization, if any, produces one coupled left shift of UAC and LAC. In the rounding-off, as before, the result is incremented by 1 if LAC0$\geq$5.

Divide, floating point. The exponent arithmetic produces an algebraic difference of the operand exponents. In the fraction arithmetic, if $a<b$, the dividend is shifted one place to the right, thus cancelling the need for the normalization of the result. The following consecutive subtractions and shifts are the same as in the *fixed divide*. For the rounding-off purposes, the ninth cycle of the consecutive subtractions is carried out, but the ninth quotient digit is not allowed to exceed five. If the digit is five, the rounding-off takes place, otherwise, the execution of the division is terminated.

It can be shown that the excess-three code has hardly any advantage over the straight binary code in decimal adders. The excess-three self-complementing code requires some decoding at the adder output and, as a result, is more expensive than the complementing of the straight code. The complementing circuit is associated with the R input only.

The most fundamental operation in the arithmetic unit is addition. The adder performs subtractions using the following algorithm.

Case 1: $a>b$

$$a-b \equiv a+\tilde{b} =$$

$$a+(10^n-b) = 10^n+(a+b)$$

Instead of subtracting b, add its ten's-complement $\tilde{b}$. The presence of 10^n, being a carry of n-digit precision, indicates that the result is correct. In this case, the sign of the result is taken to be the same as one of the operand a.

Case 2: $a<b$

$$a-b \equiv a+\tilde{b} = d$$

As there is no carry, the following procedure, known as EAC *(end around carry)* is initated:

$$0+\tilde{d} = 0+[10^n-(a+\tilde{b})] =$$

$$0+10^n-[a+(10^n-b)] = -a+b$$

In this case, the sign of the result is taken as the opposite of the operand a.

In the adder, the complementing of b operand is initiated by the ULS signal. It is carried out by the complementing of each digit to 9 and then with the generated ICAR *(initial carry)* the result is effectively increased by 1:

$$\tilde{b} = 99999999 - b_6b_7b_2b_3b_4b_5b_6b_7 +$$

$$1 = 10^8 - b$$

The sign logic is mechanized to follow the fundamental rules of the arithmetic.

The parallel adder is also used for the arithmetic on exponents in the floating-point instruction. It was planned originally to provide a special two-digit adder for the exponent arithmetic, but the additional cost would not justify a negligible saving of time.

INSTRUCTIONS

Figs. 3 through 7 and their accompanying captions and text describe both the fixed-point and floating-point arithmetic instructions of the RCA 301 Scientific Computer.

ANATOLE TURECKI received his BSc degree in Engineering from the University of London in 1949. From 1949 to 1956 he was employed by Harris Lebus, Ltd. (London) as a Research Engineer; was employed by Tasma Pty. (Sydney) as a Development Engineer; and was a Lecturer in the Radio School of the Royal Melbourne Technical College. He joined RCA in 1956 in the Broadcast Engineering Division where he designed and tested TV transmitting antennas. He transferred to RCA Electronic Data Processing in 1960 where he assisted in design and testing of the RCA 301 System. He completed all course requirements for his MSEE at the University of Pennsylvania in 1960, and his thesis, "Ternary Numerical System in Digital Computers", is presently being submitted. He is a Member of IEEE and holds a U. S. Patent.

SUMMARY

This article has given a general picture of the high-speed arithmetic unit to readers who are familiar with the RCA 301 System.[1] Some of the minor details, often far from being easy problems for an elegant solution, were omitted. Readers interested in more-complete details of the high-speed arithmetic unit are welcome to request, through the normal channels, the functional specifications for Processor Models 354 and 355, status level flow charts, and logic drawings.

BIBLIOGRAPHY

1. H. Kleinberg, "The RCA 301: Flexibility at Low Cost—An Engineering Challenge"