Arithmetic and Control Techniques in a Multiprogram Computer

N. LOURIE†, H. SCHRIMPF†, R. REACH†, AND W. KAHN†

IN THE DESIGN of a data processor for commercial applications, the designer is very often striving for better machine performance for little or no increase in cost. In the system design of the Honeywell 800 transistorized data processing system, several design concepts were utilized to help achieve this objective. One of these techniques involves the use of a small auxiliary memory to aid in the control of the high speed central processor. A second technique uses a new word organization that results in a faster and less costly arithmetic element.

In a modern transistorized computer, the speeds that are economically achievable in the central processor are very often much higher than necessary to keep up with peripheral devices. The concept of time sharing the central processor among several programs in order to utilize otherwise wasted time then becomes attractive. In order to achieve this time-sharing automatically without the use of cumbersome supervisory routines, at least one sequence counter per program is required. If a small coincident current memory running out of phase with the main memory were available, a relatively liberal number of programs could easily be run simultaneously by assigning these sequence counters to this control memory. Also, since additional memory locations become economical, it is now simple to assign each program two sequence counters for greater flexibility. These are known as the sequence and consequence counters. The Honeywell 800 has 8 pairs of sequence counters, thus allowing the simultaneous operation of eight independent programs.

TABLE 1

USE OF CONTROL MEMORY FOR SIMULTANEOUS PROGRAM ORATION

Program	Sequence Counter	Location	Cosequence Counter	Location
1	00050	2	00600	3
2	02090	34	03002	35
3	°	66	°	67
4	°	98	°	99
5	°	130	°	131
6	°	162	°	163
7	°	194	°	195
8	°	226	°	227

To illustrate how this is performed, Table 1 shows a possible state of the various counters. If, upon starting, the first order is specified from the sequence counter, 00050 will be read from location 2 of the control memory, and the contents of 00050 in the main memory will be read and performed as an order. The sequence counter will then be incremented by unity so that 00051 will be immediately reinserted into address 2 of the control memory as the location of the next order to be performed under control of the sequence counter in program 1. If the previous order in program 2 specified that the cosequence counter was to be used to obtain the next order, the contents of address 35 will then be read out of the control memory and 03002 will then be used as a main memory address to select the next order performed. Similarly the computer will then cyclicly perform one order from each program. Some orders that leave useful information in the central processor do not relinquish control to another program, so that occasionally, several orders from one program will occur before any orders from another program are performed. The multiply order is an example of an order that requires such treatment since there is still a low order product that may be required after the completion of the order. Because the control memory is running simultaneously but out of phase with the main memory, this multiple operation not only is extremely flexible, but is performed without loss of speed.

Each of the sequence and cosequence counters in the Honeywell 800 has associated with it in the control memory another register known as a history register. Whenever a sequence or a cosequence counter is modified because of a sequence change, the associated history register is changed so that it contains the address that the sequence or cosequence counter would have contained if there were no sequence change. With this feature available, the programmer can easily sequence change into a subroutine and then, at some later time, revert back to the main routine. Table 2 gives a numerical example of this use of a history register.

The same control memory can be extremely useful for control of information to and from peripheral devices and the main memory. Eight input registers and eight output registers have been reserved in the control memory for controlling the transfer of data between peripheral devices and main memory. Each of these control memory registers is uniquely associated with an input or output trunk. When an input trunk signals that it has a word available, the central

† DATAmatic Division, Minneapolis-Honeywell Regulator Company, Newton, Mass.

TABLE 2

USE OF SEQUENCE HISTORY REGISTER TO RELOCATE AFTER SUBROUTINE

	Sequence Counter	Sequence History Register
	00122	00000
	00123	00000
	01200	00124
Subroutine located from 01200 to 01280	01201	00124
	°	°
	°	°
	°	°
	01280	00124
	00124	01281

processor is interrupted at the end of the next memory cycle. The buffer control register in the control memory associated with this input trunk is read, and the contents used to select a main memory address into which the word from the input trunk can be inserted. The content of the buffer control register is then incremented by unity and immediately placed back into the same control memory location. Thus, the next word from the same input trunk will be inserted into the next highest memory location. In the case of a reverse tape read order, the content of the control memory register is decremented by unity prior to insertion back into the control memory so that the information from tape will be in correct order regardless of the direction of tape motion. Similarly, words are delivered from the main memory to a peripheral device in the case of a write order.

If more than one trunk is on demand at the same time, a simple buffer traffic control system establishes priority, and the trunks are processed one at a time. After all input and output trunks are processed, the machine control then reverts back to normal order processing.

To increase the average data rate from tape and improve the utilization of tape space, it is desirable to place more than one item on a block of tape. When this is done, it would be desirable to be able to place each individual item in a different section of the memory. In order to accomplish this "distributed reading or writing", a set of address locations that will serve as the starting locations for each of the consecutive items after the first item, is inserted into the main memory. The starting location of this group of beginning item addresses is placed into a control memory address called a distributed item counter. There is one distributed item counter for each input trunk and each output trunk. As before, when a block of information is read into the memory, the initial item is placed into main memory locations as specified by the associated buffer control register. However, when a special bit configuration representing an end of item is sensed, the content of the main memory location as specified by the distributed item counter is read into the buffer control register, thus creating a new starting address for the next item. The distributed item counter is incremented by unity prior to reinsertion into the control memory, to prepare for the next item. When this change of item location occurs, one extra memory cycle is required for all the associated housekeeping. Distributed writing is performed in a similar manner.

A numerical example of the handling of a four-item block is shown below:

Read Buffer Control Register contains	01400
Distributed Read Item Counter contains	00100
Main Memory Address 0100 contains	01500
Main Memory Address 00101 contains	01600
Main Memory Address 00102 contains	01700

With these constants located as shown, the first item would be placed in consecutive memory address locations starting with 01400, the second item starting with location 01500, the third item starting with location 01600, and the fourth item starting with 01700.

Two locations in the control memory are reserved for each of the eight programs to serve as counters for such orders as multiply and multiple transfer orders between groups of memory locations.

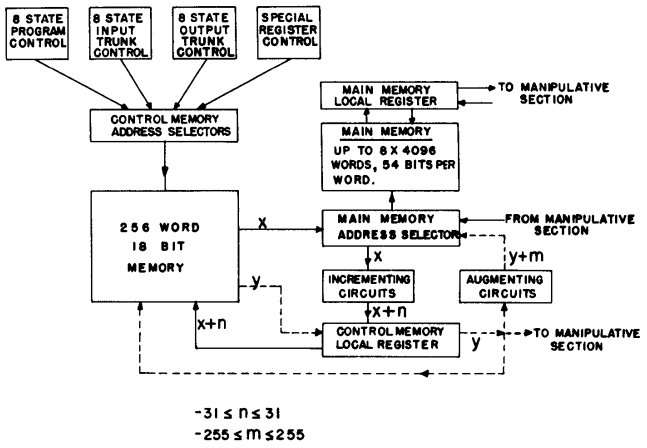

Fig. 1—Control memory in a multiprogram computer.

The control memory also functions as an aid to indexing addresses. Referring to Fig. 1, the base address "y" is read out of one of 64 index registers, eight of which are available for each of the eight programs. An eight-bit augmenter "m" specified by the order is either added to or subtracted from this base address to form the indexed address for the main memory. The base address "y" is reinserted into the control memory unmodified. One extra memory cycle time may be required to perform an order if any of the addresses in that order are indexed.

Indirect addressing is another feature that can easily be accomplished by use of a control memory. In this mode of operation, a number "x" is read from

a specified control memory location and used as an address in the main memory. The number "x" is incremented by a constant "n" prior to reinsertion in the control memory, so that next time this control memory content is used as a main memory address, a different main memory location will be addressed. Thus, with one order it is possible to operate on a whole series of main memory addresses without the necessity for order modification.

The control memory also serves to store a constant U that will give rise to an unprogrammed transfer of control if special situations such as end of tape, addition overflow, or read error occur. When one of these situations arises, the constant U is incremented by n, and an unprogrammed transfer of control to address $U + n$ occurs. The constant "n" is a function of the type of situation that calls for the unprogrammed transfer of control. Since there is an unprogrammed transfer register for each of the eight programs, eight independent U constants can be stored.

A mask index register is available for each program, such that any one of 64 mask constants can be called out of main memory by using one of the mask type orders and an incrementing constant.

A summary of the assignment of control memory locations is shown below:

Address	*Description*
0	AU-CU Control Counter No. 1
1	AU-CU Control Counter No. 2
2	Sequence Counter
3	Co-Sequence Counter
4	Sequence History Register
5	Co-Sequence History Register
6	Unprogrammed Transfer Register
7	Mask Index Register
8–15	Index Registers 0 through 7
16–27	General Purpose and Indirect Addressing Registers

These 28 locations are repeated 8 times so that each of the eight programs has a unique set of these registers.

In addition, there are eight each of the following registers which are associated with input-output. These registers are not uniquely associated with any program, but are available for convenient assignment to any program.

Address	*Description*
28	Read Address Counter
29	Distributed Read Address Counter
30	Write Address Counter
31	Distributed Write Address Counter

When the computer designer initially considers the specifications of the arithmetic unit of a digital computer, one of the prime considerations is the method of performing addition. A good design will be capable of meeting the speed specifications with a minimum of hardware. The format of the bits in the arithmetic unit is an important factor in fulfilling this objective. Various formats for a 48 bit word are discussed below.

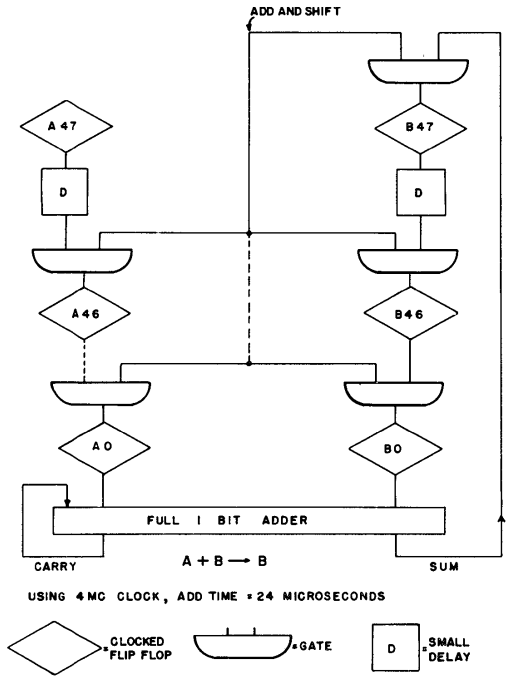

Fig. 2—48-bit serial accumulator.

Serial

A pictorial representation is shown in Fig. 2. In this arrangement, as well as all others to follow, for the sake of simplicity it is assumed that the A and B operand each reside in a 48 bit flip-flop register, each stage of which is capable of shifting. At the completion of the addition, the sum will be located in the B register. Since the addition is taking place only one bit at a time, a minimum of equipment is required. However, in order to achieve a reasonably fast add time, relatively high speed shifting flip-flops would be required. For instance, with 4 MC flip-flops, 24 microseconds would be required for a complete addition with end around carry.

Parallel-Serial

A pictorial representation of a 48-bit parallel-serial accumulator with 4 bits in parallel and 12 digits in serial is shown in Fig. 3. Other geometries could be used here, but this is a very important one, inasmuch as 4 bits in parallel can be used for a binary coded decimal digit. Since the adder is now a 4-bit adder instead of a 1-bit adder, more time will be required to propagate carries, thus resulting in a slower information shifting rate than in the serial adder. Using a 125 millimicrosecond carry propagation per stage and a 1.33 MC shifting rate, the add time will be 18 microseconds, again including end around carry prop-

agation. The addition time is not too much faster than the example given in the serial adder, but the speed requirement of the flip-flops is reduced.

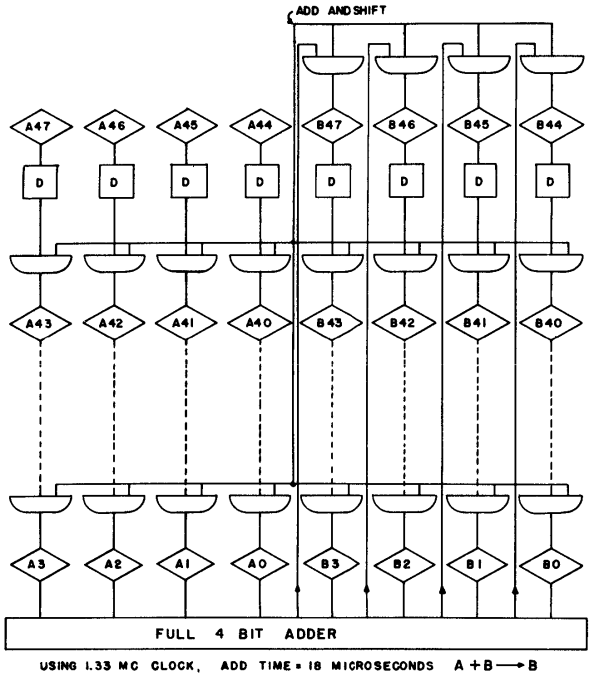

Fig. 3—48-bit parallel serial accumulator. 4 bits = 1 character in parallel, 12 characters in serial.

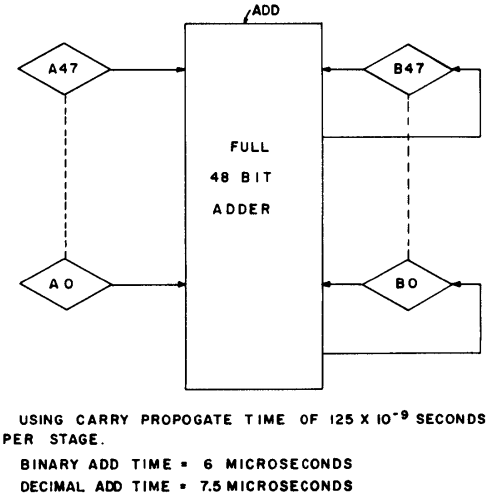

Fig. 4—48-bit parallel accumulator.

Parallel

When the ultimate in addition speed is required, a complete parallel accumulator, as shown in Fig. 4, is often used. To achieve the fullest speed advantages, the carry propagation time should be completely asynchronous. With no "carry hopping" or "end of carry" sensing, an equivalent add time with the same circuits and assumptions above would be 6 microseconds. If decimal add were included, another 1.5 microseconds would be required. Assuming the speed-up techniques suggested previously are used, average add times on the order of 1 to 2 microseconds are feasible, but the increase in the number of logical statements is substantial. The number of logical statements required without these speed-up techniques is about 12 times as many as the parallel serial adder since the full add logic is required for each of the 48 stages.

Upon examining the requirements of the response time of any adder stage in the parallel accumulator, it is noted that although any stage is required to propagate a carry in a short time, once that stage has responded, it rests for the remainder of the carry time, resulting in a very inefficient use of the inherent speed available. If good speed-up techniques are used, then this inefficiency is greatly reduced. This observation then suggests that a parallel accumulator without speed-up techniques is an extremely wasteful device. It was this observation that led to the invention of the parallel-serial-parallel accumulator. The parallel-serial-parallel accumulator is an efficient extension of the parallel-serial accumulator which results in speeds comparable with that of a parallel accumulator with no speed-up techniques, but with approximately one-fourth the number of logical inputs to the logical expressions for the adder.

Parallel-Serial-Parallel

The parallel-serial-parallel arrangement described here consists of three parallel 16-bit parallel-serial registers, with the bits of a 4-bit character in parallel, 4 characters in serial. Each of the three 16-bit groupings is referred to as a major character. Major character 1 contains bits 0–15, major character 2 contains bits 16–31, and major character 3 contains bits 32–47. In 4 pulse times the sum within each major character is computed. Carries generated as a result of these additions are then propagated and added into the next major character in the next 4 pulse times. At the end of these 8 pulse times, the probability that the carries will be finished propagating and the answer will be correct is $1 - 3 \times 2^{-17} = 0.999977$. Carry propagation completion can be sensed by means of a three-leg buffer, and as much additional time as necessary (8 pulse periods maximum) allowed to complete the carry propagation.

Fig. 5 shows a 48-bit binary PSP accumulator capable of either addition or subtraction. The equations for this are shown below.

S = Subtract; $\overline{S}$ = Add
A_n = Addend
B_n = Initial augend and final result
C_n = Carry functions
P_n = Final sum functions
CC_n = Character carry from each adder
T_{4n} = Timing function every 4th clock time such that $T_{4n}CC_n$ is the carry from the highest

order minor character in each major character.

Equation for Major Character 1 Adder

$$C_0 = T_{4n}\, CC_3 + \overline{T}_{4n}\, CC_1$$

$$C_1 = C_0 B_0 + \bar{S} A_0 B_0 + \bar{S} A_0 C_0 + S \bar{A}_0 B_0 + S \bar{A}_0 C_0$$

$$C_2 = C_1 B_1 + \bar{S} A_1 B_1 + \bar{S} A_1 C_1 + S \bar{A}_1 B_1 + S \bar{A}_1 C_1$$

$$C_3 = C_2 B_2 + \bar{S} A_2 B_2 + \bar{S} A_2 C_2 + S \bar{A}_2 B_2 + S \bar{A}_2 C_2$$

$$CC_1 = C_3 B_3 + \bar{S} A_3 B_3 + \bar{S} A_3 C_3 + S \bar{A}_3 B_3 + S \bar{A}_3 C_3$$

$$B_{12} = P_0 = A_0 B_0 C_0 + A_0 \bar{B}_0 \bar{C}_0 + \bar{A}_0 B_0 \bar{C}_0 + \bar{A}_0 \bar{B}_0 C_0$$

$$B_{13} = P_1 = A_1 B_1 C_1 + A_1 \bar{B}_1 \bar{C}_1 + \bar{A}_1 B_1 \bar{C}_1 + \bar{A}_1 \bar{B}_1 C_1$$

$$B_{14} = P_2 = A_2 B_2 C_2 + A_2 \bar{B}_2 \bar{C}_2 + \bar{A}_2 B_2 \bar{C}_2 + \bar{A}_2 \bar{B}_2 C_2$$

$$B_{15} = P_3 = A_3 B_3 C_3 + A_3 \bar{B}_3 \bar{C}_3 + \bar{A}_3 B_3 \bar{C}_3 + \bar{A}_3 \bar{B}_3 C_3$$

B_i shifted into B_{i+4} where $O \leq i \leq 11$

Equations for the major character 2 adder are the same with subscripts on A_n, B_n, C_n increased by 16, CC_1 substituted for CC_3, and CC_2 substituted for CC_1.

Equations for the major character 3 adder are the same with subscripts on A_n, B_n, C_n increased by 32, CC_2 substituted for CC_3, and CC_3 substituted for CC_1.

The average add time with 125 millimicrosecond carry propagate time and 1.33 MC flip-flops is approximately 6 microseconds.

This accumulator has been organized in such a manner that decimal arithmetic using binary coded decimal representation can be easily incorporated, since each minor character is a binary coded decimal digit. To include decimal arithmetic two areas need to be changed. The first is involved with rectification of the binary sum where either the binary coded decimal sum is greater than nine, or a major character carry was generated. This can easily be done by inserting the logic between B_{13} and B_9, B_{14} and B_{10}, and B_{11} below.

D = Decimal $\quad \overline{D}$ = Binary

$$B_8 = B_{12}$$

$$B_9 = \overline{D} B_{13} + \overline{CC_1} \overline{B}_{15} B_{13} + D B_{15} B_{14} \overline{B}_{13} + D CC_1 \overline{B}_{13}$$

$$B_{10} = \overline{D} B_{14} + \overline{CC_1} \overline{B}_{15} B_{14} + \overline{CC_1} B_{14} B_{13} + D CC_1 \overline{B}_{15} \overline{B}_{13} + D CC_1 B_{15} \overline{B}_{14} B_{13} + CC_1 B_{14} \overline{B}_{13}$$

$$B_{11} = \overline{D} B_{15} + \overline{CC_1} B_{15} \overline{B}_{14} \overline{B}_{13} + D CC_1 \overline{B}_{15} \overline{B}_{14} B_{13} + CC_1 B_{15} B_{14} B_{13}$$

The above can be verified with a simple truth chart.

The second area that needs change is the generation of inter-digit carries. This is accomplished by adding a few terms to C_0 to take care of those cases where the decimal sum or difference is between 10 and 15.

$$C_0 = T_{4n}\, CC_3 + \overline{T}_{4n}\, CC_1 + \overline{T}_{4n}\, D\, B_{15} B_{14} + \overline{T}_{4n}\, D\, B_{15} B_{13} + T_{4n}\, D\, B_{37} B_{36} + + T_{4n}\, D\, B_{37} B_{35}$$

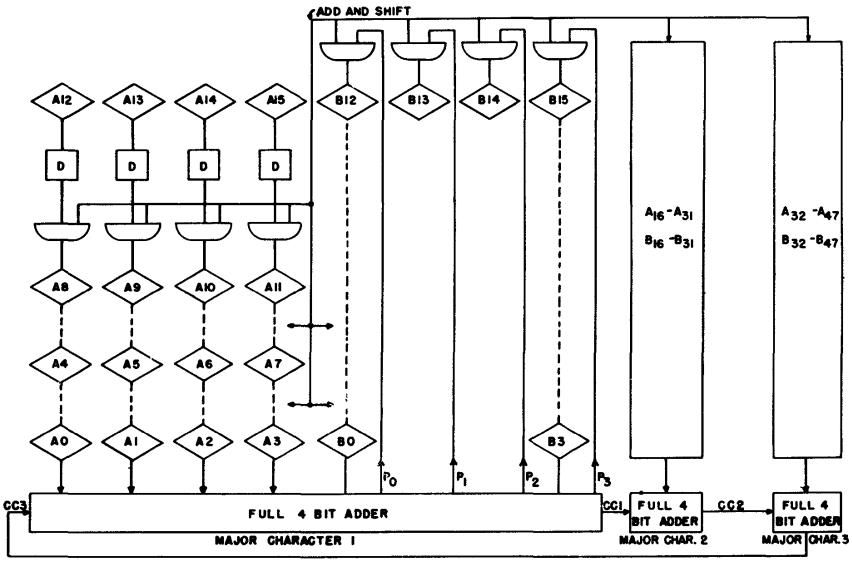

Fig. 5—48-bit binary parallel serial parallel accumulator.

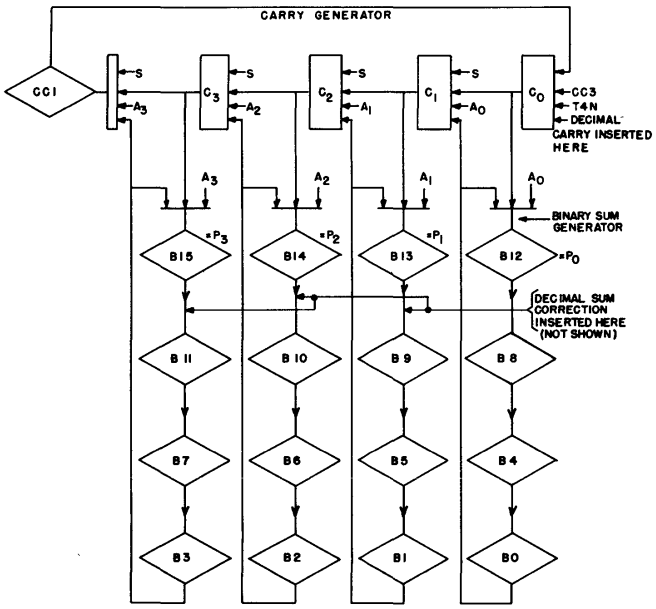

Fig. 6—Logical organization for major character 1 of parallel serial parallel accumulator.

The two corrections required the addition of only 230 diodes to the accumulator. No amplifiers were added. The probability that the answer will be correct after 8 pulse times is $1 - 3/2000 = 0.9985$. For these cases, up to 9 more pulse times may be required for the correct answer.

In addition to allowing a very economical method of arithmetic, the PSP format allows other machine simplifications over a parallel format. In particular, it is possible to transmit a 48-bit word to remote portions of the machine by means of time sharing 12 lines, resulting in a decreased number of cable drivers. It also allows the use of one flip-flop and 3 pulses of delay line to store 4 bits of information in those cases where the other 3 flip-flops are not required for manipulation reasons. A block diagram of type of storage is shown in Fig. 7.

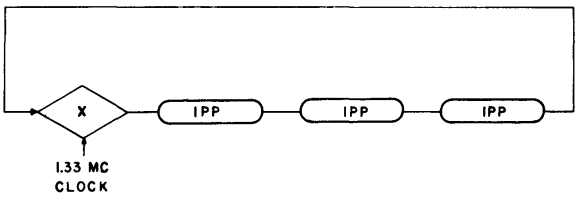

Fig. 7—Use of one flip-flop and 3 pulses of delay to store 4 bits.

Parallel-Serial-Parallel Arithmetic Example

Fig. 8 is an example illustrating two numbers being added together in a parallel-serial-parallel adder. The zeros immediately beneath the operands indicate the initial state of the carry circuits at the beginning of the addition. All of the numbers on a line with *tl* indicate the computation being performed during the first pulse period. The 11 indicates that the "4" and "7" in the low order position of major character 3 is added to form a 1 sum and a 1 carry, and, simultaneously, the "5" and "4" in the low order portion of major character 2 is added to form a "9" with 0 carry indicated by the 09, and, similarly, 1 and 6 are being added to form 07.

During the next pulse period, *t*2, (and through the same addition circuits which produced the "*tl*" addition) "2" plus "9" in major character 1, together with the previous carry, "0", forms the 11 on the *t*2 line. Similarly "1" plus "8" plus "0" carry forms 09 and "7" plus "1" carry forms 09.

This process is repeated until the *tl* line of word cycle 2. At this point, the carry from the high order position of major character 1 is added to the low order digit in the partial sum of major character 2. The carry is then propagated as shown until the 3699 is "corrected" to 3700. Similarly, the 9991 formed in major character 3 is corrected to 9992.

As seen in this example, the addition is completed after 8 pulse periods.

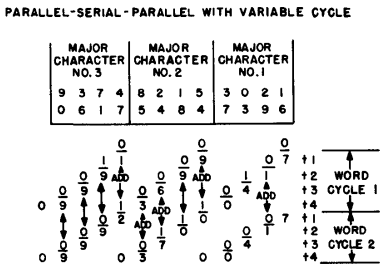

Fig. 8—Parallel-serial-parallel with variable cycle.

In summary, the parallel-serial-parallel format provides a fast arithmetic speed at a relatively economic cost of logical circuitry. The addition of the control memory to the system provides a wide range of flexibility in order to achieve an efficient usage of the computer.

The authors of this paper wish to extend credit to all those at the Datamatic Division of Minneapolis Honeywell Regulator Company, whose ideas contributed to the creation of this data processing machine.

Discussion

M. Rubinoff: A criterion often used to measure the efficiency of a computer design is computing per second per dollar. Can you comment on the efficiency of multiple operation for various types of problems?

Mr. Lourie: The multiple independent program operation feature costs remarkably little when integrated properly in the design of a data processor. The amount of work which can be accomplished per second — such work includes card reading, tape operations, etc., as well as actual computing — is, of course, increased tremendously in a multi-programmed design, since these various operations are being performed simultaneously rather than serially. This is especially so where the traffic control of the various programs is automatic so that maximum use is made of each memory cycle of the machine. I think, therefore, that it is obvious that the efficiency as measured by your criterion is exceptional in a machine of this nature.

J. Gosden (Leo Computers): What are the major uses of the co-sequence registers?

Mr. Lourie: The major uses of the co-sequence registers are to perform sub-routines without the use of "housekeeping" type orders.

M. S. Maxwell (US Naval Weapons Lab): What is the speed of the central computer memory and control memory?

Mr. Lourie: Six microseconds complete cycle time for both memories.

J. Daniels (ISI): Are there provisions to prevent one program from overwriting another in the main memory?

Mr. Lourie: An executive routine has been designed for use with all H-800 installations. This routine automatically solves all problems of interference with regard both to equipment and to memory space allocations. Automatic assignment of memory locations is arranged for by this routine on a non-conflicting basis.

Miss E. Berezin (Teleregister): How is the 256 bit control register loaded, particularly if one wished to start one program while others were in operation?

Mr. Lourie: The loading routine would use one of the unused eight sequence registers to load the control memory.

L. Clapp (Sylvania): Is there any possibility of losing information if several output/input trunks are working simultaneously?

Mr. Lourie: The machine speed is such that it can simultaneously handle eight input and eight output trunks, all of which occur to and from magnetic tape mechanisms. There would be 16 trunks acting simultaneously with approximately one-third of the computer time available for computation. The maximum input/output rate is such that no data would be lost.

J. H. Hughes (American Mutual Liability): What happens when you try to add, say, hexadecimal (15) plus (12)?

Mr. Lourie: If any generalized binary configuration is to be added to another binary configuration, the Binary Add order of the machine should be used and will lead to a correct answer. The normal Decimal Add orders are used only for operations involving the binary coded decimal code for digits 0–9.

B. Tasini (IBM): Can you comment on the debugging of problems on a multi-programmed computer?

Mr. Lourie: Debugging is not very much more difficult on a multi-program machine, since each program would be debugged independently, and then these programs would be run simultaneously.

P. Seaman (IBM): Is the program priority system and the I/O priority system fixed, or can it be specified by the programmer?

Mr. Lourie: There is no priority with regard to programs. Rather they are handled cyclicly in order. If one program requires the machine for a certain period of time, it can insert an order that will shut off all other programs and then, at some later time, turn these programs back on. The priority of peripheral equipment is fixed.

C. L. Foster (IBM): Is control memory addressable from main memory?

Mr. Lourie: Yes.