A Hardware Fixed-Point Multiplier/Divider.

Nicholas Arpad Balatoni
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/6655

This online database contains the full-text of PhD dissertations and Masters’ theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000 ext. 3208.
A HARDWARE FIXED-POINT MULTIPLIER/DIVIDER

by

NICHOLAS ARPÁD BALATONI

A Thesis

Submitted to the Faculty of Graduate Studies through the Department of Electrical Engineering in Partial Fulfillment of the Requirement for the Degree of Master of Applied Science at the University of Windsor

Windsor, Ontario
1971
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.
ABSTRACT

This work describes the design and implementation of a low-cost Hardware Fixed-Point Multiplier/Divider to be used primarily with the PDP-8/S computer.

It was deemed advantageous to have the use of such a device in situations where neither an Extended Arithmetic Element nor the Data Break Facility were available and where at the same time fast Multiplication and Division operations were required - such applications may occur in on-line control and in graphics. The device constructed has a maximum Multiplication time of 90 microseconds, a minimum Multiplication time of 18 microseconds, and a Division time of 155 microseconds. The speed thus obtained is at least 15 times faster than the speed obtained using subroutine Multiplication and Division.

The device consists of three 12-bit registers which make up the main working areas, a serial Adder/Subtractor, two 4-bit counters and the appropriate Multiplier and Divider Controllers at an approximate cost of $2000.

To achieve Multiplication the Leech Algorithm was used in which the last three digits of the Multiplier are compared and accordingly the appropriate operation is carried out. For Division, the Booth Algorithm was adopted. In this method the first bit of the Divisor and the first bit of the Partial Remainder are compared and accordingly an Addition or Subtraction takes place during each cycle.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
between the Partial Remainder and the Divisor.

A workable unit was constructed with the desired performance features and the Multiplier/Divider proved to be a valuable addition to the available PDP-8/S installation.
ACKNOWLEDGEMENTS

The author wishes to express his appreciation to Dr. P.A.V. Thomas for his supervision and continuous assistance during the course of this research. Furthermore, the author's thanks go to the National Research Council of Canada for its generous financial support in making this project possible.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>v</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF ILLUSTRATIONS</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
</tbody>
</table>

**Chapter**

**I  INTRODUCTION**

1.1 Preliminary Considerations 1
1.2 General Outline 5

**II PHILOSOPHY OF DESIGN**

2.1 Algorithms Used 9
   2.1.1 Multiplication Algorithms 9
   2.1.2 Division Algorithms 17
2.2 Basic Timing Requirements: Multiplier 20
2.3 Basic Timing Requirements: Divider 22

**III SYSTEM LOGIC**

3.1 "Multiplicand/Divisor" Register (AR) 26
3.2 "Accumulator/Dividend/Most Significant Result" Register (BR) 28
3.3 "Multiplier/Quotient/Least Significant Result" Register (DR) 31
3.4 Adder/Subtracter 36
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>General Arrangement of Registers, Adder/Subtractor, Counters and Controllers</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Flow Diagram for Multiplication</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Flow Diagram for Division</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>Timing Pulses for Multiplier</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>Timing Pulses for Divider</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>&quot;Multiplicand/Divisor&quot; Register (AR)</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>&quot;Accumulator/Dividend/Most Significant Result&quot; Register (BR)</td>
<td>29</td>
</tr>
<tr>
<td>8</td>
<td>&quot;Multiplier/Quotient/Least Significant Result&quot; Register (DR)</td>
<td>32</td>
</tr>
<tr>
<td>9</td>
<td>Adder/Subtractor</td>
<td>42</td>
</tr>
<tr>
<td>10</td>
<td>Pulse Counter</td>
<td>44</td>
</tr>
<tr>
<td>11</td>
<td>Cycle Counter</td>
<td>50</td>
</tr>
<tr>
<td>12</td>
<td>Left/Right Shift Pulse and Level Control</td>
<td>54</td>
</tr>
<tr>
<td>13</td>
<td>Multiplier Controller</td>
<td>56</td>
</tr>
<tr>
<td>14</td>
<td>Divider Controller</td>
<td>60</td>
</tr>
<tr>
<td>15</td>
<td>I/O, Clock, Flag Circuits</td>
<td>63</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.a</td>
<td>Booth Algorithm for Multiplication</td>
<td>11</td>
</tr>
<tr>
<td>1.b</td>
<td>Leech Algorithm for Multiplication</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>Booth Algorithm for Division</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>Half-Adder for (X+Y)</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>Full-Adder for (X+Y)</td>
<td>37</td>
</tr>
<tr>
<td>5</td>
<td>Half-Subtractor for (X-Y)</td>
<td>38</td>
</tr>
<tr>
<td>6</td>
<td>Full-Subtractor for (X-Y)</td>
<td>39</td>
</tr>
<tr>
<td>7</td>
<td>Instruction Set</td>
<td>75</td>
</tr>
</tbody>
</table>
1.1 Preliminary Considerations.

When thought is given to the design of a Multiplier/Divider, some preliminary specifications must first be outlined. This includes a review of already existing requirements with thought being given to the environment in which the device will be used and to existing feasible equipment and components that are available to the designer.

The need that gave rise to the initial idea of a Hardware Fixed-Point Multiplier/Divider came about because of the availability of an installation that consists of a PDP-8/S Digital Computer and its associated peripheral devices. This computer is one of the earlier low-cost mini-computers of the Digital Equipment Company. The PDP-8/S computer is equipped with a bus-type I/O channel making it suitable for the addition of 64 peripheral devices. These devices are external to the computer proper and are readily programmable through the regular I/O channels with the aid of the I/O skip facility.

The arithmetic unit provided in the PDP-8/S is a one-bit full serial adder, and, using it, subtraction may be performed by first making one number negative and then performing an addition. With the higher line of PDP computers there is an Extended Arithmetic Element (EAE) available which can also be used for Multiplication and for Division. The EAE performs parallel arithmetic operations on positive binary numbers only. Because of its parallel nature, it performs unsigned multiplication in a length of time that varies

1
from 9.0 to 21.0 microseconds and a division in a time of 36.5 microseconds. However, on these higher lines of PDP computers, the subroutine for signed multiplication requires 40.5 to 66.0 microseconds and the subroutine for signed division requires 65.0 microseconds. By comparison, it is possible to perform signed multiplication and signed division on the PDP-8/S using subroutines, but the multiplication subroutine, as observed in an actual test programme, required up to 6 milliseconds; this is of course very slow. It must be considered further that memory and processor instruction times on the higher lines of computers range from 1.5 to 4.0 microseconds, compared with times of 20.0 to 40.0 microseconds per instruction on the PDP-8/S. The EAE furthermore requires the presence of the Data Break Facility available on the PDP-8 but not available on the PDP-8/S. Thus, times of up to 100 microseconds for multiplication and 160 microseconds for division, excluding storage times, would be deemed quite reasonable and advantageous for the proposed device.

It can be easily concluded then that subroutine multiplication and division are not feasible to use in applications where logical decisions involving these two operations, or where frequent use of these operations are required to take place rapidly. Such applications may easily occur in on-line control and in graphics.

Undeniably, the speed of the device and its cost must be of prime interest. To gain speed two things may be considered, namely, the type of components that will be used and the type of arithmetic that will be performed, parallel or serial. To make the external
device compatible with the PDP-8/S, DEC components provide the logical types of modules to be used. The two types of negative logic modules available are the 2MHZ and the 10 MHZ modules. Thus, to gain at best five times the speed, the high speed components should be used which would practically double the cost of the proposed device.

Similarly, to gain speed in the arithmetic, parallel operations should be adopted. The difference in cost is however prohibitive due to the fact that instead of having one Adder/Subtracter as for serial arithmetic, in parallel arithmetic one Adder/Subtracter is required for each bit of the 12 bit registers used.

The answer to the requirement problem must lie then in the time criteria placed on a single multiplication and on a single division. To gain an idea of these times, an examination of the capabilities and requirements of the algorithms used is called for. The algorithms, found in Section 2.1, suggest that six cycles are required for Multiplication, and twelve cycles for Division. Considering a parallel Adder/Subtracter and 4 pulse times per cycle, Multiplication would require 24 pulse times or less due to the existence of the Fast Operation which can occur under certain conditions to be discussed later. Considering 3 pulse times per cycle and 11 full cycles plus one short cycle, division would require 34 pulse times. If a serial Adder/Subtracter is considered with twelve pulse times per Addition or Subtraction, a Multiplication would require 90 pulse times or less, and division would require 155 pulse times.
If the most expensive proposal were adopted (10MHz modules and parallel operation) Multiplication and Division times of 5 and 7 microseconds respectively could be achieved. Since an I/O instruction time on the PDP-8/S is 38 microseconds, such speed is not justified for the added cost because of all the overhead, or unused, time.

If the serial Adder/Subtractor were used and 10MHz modules, 18 and 30 microseconds may be realized for the Multiplication and Division operations respectively. This would indeed confine the device operation time within one I/O instruction time. However, if the price difference is considered between the 10MHz and 2MHz modules, and the fact too that some interfacing is required between the two types if used in combination, the proposal is still an unjustifiably expensive one.

Let the proposal which was finally accepted be now considered, that is, using serial arithmetic and 2MHz components. It was found that the maximum speed obtainable, due to carry propagation in counters (4 stages) and in shift registers is 1.3MHz. However, considering 1MHz operation, each pulse time (time from the start of a pulse to the start of the next) is one microsecond. This then yields a Multiply time of 90 microseconds or less, and a Division time of 155 microseconds. Considering the presence of the Fast-Operation, the minimum time for a Multiplication is 18 microseconds (6 cycles of 3 pulse times each). An I/O Skip facility is available at no added cost, and could be used if a complete Multiplication or
Division operation should exceed one I/O instruction time. The Multiplication Operation, the main purpose of the device, never exceeds 3 I/O times and in most cases it can be completed within 2 I/O times, i.e. 76 microseconds. Since it is expected that Division will be much less used than Multiplication, the performance times thus acquired are quite acceptable. At this time it should be further pointed out that the device will be able to Multiply and Divide 12-bit signed binary numbers which gives the added advantage of not having to perform sign conversions by software. This would indicate an advantage even over the EAE, a unit which further requires the Data Break Facility. Furthermore, since the Multiplier/Divider is an external device, it is readily usable not only with the PDP-8/S but with other PDP 12-bit word computers.

1.2 General Outline.

It is possible to describe the Multiplier/Divider by discussing in general terms the major sections of its block diagram. Because one may multiply two 12-bit signed binary numbers, by the nature of Multiplication the result will yield in many cases a binary number that is more than 12-bits but never more than 24-bits long. Thus, it is necessary to provide a double length register from which the result can be extracted in two twelve bit segments through the regular programmed I/O channels. The unit then requires two single length registers which at the end of the operation will contain the most significant and least significant portions of the
result.

Furthermore, it is necessary to load in-through the I/O channels- the two numbers on which the arithmetic operation is to be performed. In order to retain the Multiplicand while the intermediate operations, addition and subtraction, take place a third register is necessary. It is found that during the Division operation this third register will contain the Divisor.

For identification purposes, let the register used to store the Multiplicand/Divisor be called the A-Register (AR). Also the register to hold the Most Significant Result or in Division, the Dividend, shall be called the B-Register (BR), and the third register, to hold the Least Significant Result/Multiplier/Quotient, shall be called the D-Register (DR).

These three registers provide the main working areas into which data is loaded, in which all shifting takes place, in which the intermediate results are held and from which the final answers are extracted. Figure 1 gives the Block Diagram form of the Multiplier/Divider.

Also noted in Figure 1, the fourth major block in the diagram is the Adder/Subtracter. An inspection of the algorithms in Section 2.1 shows that in some cases it is necessary to perform an Addition between the contents of AR and BR, and in other cases a Subtraction of the contents of AR from BR is required. In 2's complement arithmetic, a Subtraction may be performed by complementing and incrementing AR after which an Addition is performed. However,
FIGURE 1.

General Arrangement of
Registers, Adder/Subtractor,
Counters and Controllers
when the Full Adder/Subtractor was developed from Boolean Functions (refer to Section 3.4), it was found that a Full Adder and a Full Subtractor were readily combined such that the Addition or Subtraction of two 12-bit signed numbers yielded the correctly signed 12-bit result, without the need for the complement and increment operation.

The block labelled "COUNTERS" serves the purpose of generating and distinguishing from each other the twelve OS pulses necessary for Addition or Subtraction, and the control pulses OA, OB, OX and OY used for shift purposes. There are two counters however, one being the Pulse Counter and the other being the Cycle Counter. These are dealt with in detail in Sections 3.5 and 3.6 respectively.

Not included in the block diagram are two major sections that control the Multiplication and Division processes which are detailed in Sections 3.8 and 3.9 respectively. These units compare the appropriate digits of AR, BR and DR, yielding decisions as to which pulses are applied to the registers and whether Addition, Subtraction or neither is to be carried out in each cycle of the particular operation.

For simplicity in the block diagram, the clock, flags and I/O control circuits are also omitted, but they are discussed at length in Section 3.10.
Chapter II
PHILOSOPHY OF DESIGN

2.1 Algorithms Used.

2.1.1 Multiplication Algorithms.

The most common forms of automatic multipliers are the serial and parallel type, as discussed by P.A.V. Thomas\(^1\). Their great limitation is that they are able to handle positive numbers only, such that a corrective subroutine must be introduced when negative numbers must also be handled. Their mode of operation is one of continued addition, one addition for each bit of the entire word length. This occurs in a serial multiplier where only one adder is utilized. In the parallel multiplier, one adder is required for each bit in the word length. To the above limitation must then be added one of slow speed or high cost respectively in the two cases mentioned.

Since it was decided to use the serial mode of operation, a scheme suggested by Drs. A.D. Booth and K.H.V. Booth\(^2\) was investigated. This scheme involves a serial multiplier which yields the correctly signed product of two signed input numbers. The method compares the least significant bit of the Multiplier with an extra bit at the end of the Multiplier Register and either adds or
subtracts the Multiplicand to the Partial Product in the accumulator. This occurs if these two digits that are compared differ from each other. If the two digits are the same, the Partial Product is left unaltered. The content of the Multiplier Register and Accumulator are shifted one place to the right in each cycle for the number of cycles as there are bits in a word length. On a 12-bit word on low speed modules and serial operation this would mean a Multiplication time of approximately 170 microseconds.

In quest for a faster method of Multiplication, the method proposed by J.W. Leech\(^{(1)}\) was adopted in which the last two significant digits of the Multiplier Register, along with the extra Multiplier Register bit, are compared and the Multiplier and Partial Product are right-shifted twice, in this manner almost halving the Multiplication time with only a slight increase in cost.

The Booth Algorithm for Multiplication may be simply stated in tabular form as shown in Table 1.a. In the table, the Partial Product is given the short form PP, the Multiplier, MLR, and the Multiplicand is shortened to MND.

Noting that only two digits are compared, the last Multiplier bit and the extra bit associated with the Multiplier Register, only four combinations of these two bits can occur. Accordingly, the logic for the sequence controller of such a Multiplier becomes
<table>
<thead>
<tr>
<th>Multiplier Digits</th>
<th>Operation Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_n ) ( b_{n+1} )</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>Shift PP and MLR one place right</td>
</tr>
<tr>
<td>0 1</td>
<td>Add MND to PP and shift PP and MLR one place to the right</td>
</tr>
<tr>
<td>1 0</td>
<td>Subtract MND from PP and shift PP and MLR one place to the right</td>
</tr>
<tr>
<td>1 1</td>
<td>Shift PP and MLR one place to the right</td>
</tr>
</tbody>
</table>

Note: On last cycle the right shift is omitted.
relatively simple. Booth and Booth further offer a recursive formula for the above process and this formula is given in Equation (1) below.

\[ a_{n-p} = (1/2).a_{n-p+1} + (b_{n-p+1} - b_{n-p}).M; \text{ for } p=1, \ldots, (n-1) \quad (1) \]

In this equation \( a_{n-p} \) is the value of the accumulator at the \( p^{th} \) stage of the process, and \( M \) is the value of the Multiplicand.

On the basis of this recursive formula, Booth and Booth give a constructive proof by means of a stage-wise substitution of the values of \( p \). After the substitutions, all the equations are multiplied by a power of 2, namely \( 2^{p-1} \), then the equations are summed and rearranged to form an equation of the type \( a_1 = b.M \) and in this way the Algorithm is proved.

As noted earlier, the recursive formula encountered by Booth and Booth is a relatively simple one due to the fact that only two digits are compare and only the four possible combinations must be accounted for. When attention is directed to the Leech Algorithm, it is found that three digits are compared. This gives rise to eight different possible patterns. To show this, the Leech Algorithm for Multiplication is now stated in tabular form as shown below in Table 1.b in which all abbreviations of Table 1.a hold and all shifting is to the right.

A recursive formula for the Leech Algorithm was not found in
<table>
<thead>
<tr>
<th>Multiplier Digits</th>
<th>Operation Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>b_{n-1}</td>
<td>b_n</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

TABLE 1.b
LEECH ALGORITHM FOR MULTIPLICATION

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
any literature, and as a consequence, it was decided to construct a recursive formula that would completely describe the Algorithm. A more complex formula was expected and the equation that was finally found is as shown in Equation (2).

\[
an_{-2p} = \frac{1}{8} a_{n-2p+2}. [2-(1/2).(|b_{n-2p} + b_{n-2p+1} - 2b_{n-2p-1}|). (1-|b_{n-2p} - b_{n-2p+1}|). [1 + (1/2).(|b_{n-2p+2} + b_{n-2p+3} - 2b_{n-2p+1}|). (1 - |b_{n-2p+2} - b_{n-2p+3}|)] + [(b_{n-2p} - b_{n-2p-1}) + (1 - b_{n-2p} - b_{n-2p-1}). |b_{n-2p} - b_{n-2p+1}|]. M]
\]

(2)

The notation used for Equation (1) also applies for Equation (2). The coefficient, 1/2, in Equation (1) appears because a right shift is performed during each cycle. This right shift is the digital equivalent of multiplying by 1/2. Thus it is not surprising to find the coefficient 1/8 in Equation (2) for, if two post-shifts and one pre-shift take place before the addition or subtraction, the coefficient multiplying the 1/8 is 1. If only two post-shifts have occurred before the addition or subtraction, this coefficient is 2. This is also the case if a post-shift and a pre-shift have taken place. If only one post-shift occurred before the addition or subtraction, the coefficient is 4. The multiplying coefficient before the accumulator term thus becomes 1/8, 1/4, 1/4 or 1/2 depending on whether 3, 2 or 1 shifts have occurred.

To obtain correct equations in the stage-wise substitution

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
in a constructive proof, the equation at each stage must be multiplied by the power of two that has occurred in the previous stage and to that power must also be added the number of post-shifts and pre-shift, if any, that have occurred just before the addition or subtraction that is about to take place during this stage.

Due to the complexity of the equations and the difficulties which arise from it, a constructive proof will not be offered here. However, a sample calculation showing the method is given in the Appendix. The mere purpose of the above discussion is to point out how the complexity of the recursive formula increases as more digits of the Multiplier are compared at each stage of the Multiplication process. In passing it may be further mentioned that, if an algorithm were designed such that four digits would be compared at a time, three right shifts would have to be carried out at each stage and the whole Multiplication could be carried out in four cycles. The difference in the time required for a Multiplication in this way would of course not justify the complexity that would arise in the control circuits of the process.

Thus, the Leech Algorithm was adopted for Multiplication because the time for one Multiplication could be halved in comparison to the Booth Method and because the difference in costs was found to be acceptable even when the increase in the cost of the more complex controller logic was considered.

The Flow Diagram for Multiplication is given in Figure 2 where in Multiplication is described with respect to the number of
FIGURE 2.

Flow Diagram for Multiplication

<table>
<thead>
<tr>
<th>CYCLES</th>
<th>PULSES</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>First 5 Cycles</td>
<td>OA</td>
<td>Advance CC; Set Add/Sub FF; Optional Pre-Shift of BR and DR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not (DR₁₀=DR₁₁=DR₄)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Perform Addition or Subtraction during all Long Operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DR₁₀=DR₁₁=DR₄ (Fast Op.)</td>
</tr>
<tr>
<td></td>
<td>OX</td>
<td>Optional Post-Shift of BR and DR</td>
</tr>
<tr>
<td></td>
<td>OY</td>
<td>Post-Shift of BR and DR; Clear Pulse Counter</td>
</tr>
<tr>
<td>Last Cycle</td>
<td>OA</td>
<td>Advance CC; Set Add/Sub FF; Optional Pre-Shift of BR and DR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Perform Addition or Subtraction during all Long Operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS (12)</td>
</tr>
<tr>
<td></td>
<td>OX</td>
<td>Optional Post-Shift of BR and DR</td>
</tr>
<tr>
<td></td>
<td>OY</td>
<td>Post-Shift of BR and DR; Clear Pulse Counter; Produce Shut-Off Pulse</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
cycles required and with respect to the number of pulses required per cycle.

2.1.2 Division Algorithms.

To achieve Division on a binary base computer, there are two alternative processes available, namely, the standard trial and error method and the non-restoring method.

In the first case, the Divisor is compared with the Partial Remainder at each sub-operation, and if the former quantity is the lesser, a subtraction is performed and a '1' is added to the least significant bit of the Quotient Register. If, however, the Divisor is greater than the Partial Remainder, then nothing is done except a zero is added to the least significant end of the Quotient Register. The Quotient and the Partial Remainder are then doubled (i.e. shifted left one place) and the unit is ready for the next sub-operation in the Division process.

To achieve the above process in hardware, the easiest method to use is to subtract the Quotient from the Partial Remainder, test the sign and, if negative, add the Divisor back to the Partial Remainder. The limitations of this method are that only positive numbers can be handled and the method is slow considering the number of re-additions that must take place.

The second method, which was found to be preferable and which was adopted for this work, is the non-restoring method. As described by Booth and Booth, the method is able to handle both
positive and negative numbers and requires no re-additions to take place. The time of all Division operations is thus fixed at as many word lengths as there are bits in one word. In this case a word is twelve bits long.

Let it be assumed that the Divisor \( m \) is stored in the A-Register (AR), the Dividend \( a \) in B-Register (BR), and the Quotient \( b \) will appear in the D-Register (DR) at the completion of the Division process. The Booth Algorithm as stated by Booth and Booth is given in Table 2.

It should be noted that in the Division process integers are not obtained in the Quotient. The form of the answer is such that a binary point exists after the sign bit. As an example the number 0.10 000 000 000 represents the result \( 1/2 \). This occurs because by the nature of the Algorithm the Quotient, \( a/m = b \), is generated only in cases where \( m > a \). It is thus required to scale the value of \( m \) in cases where \( m < a \) until \( m > a \). The Division is then carried out and the Quotient may be re-scaled to obtain the true value of the result.

Booth and Booth give a constructive proof in detail based on the following recursive formula:

\[
r_k = 2r_{k-1} + (1-2b_k).m
\]

where \( r_k \) is the remainder after the \( k^{\text{th}} \) operation and \( b_k \) is the \( k^{\text{th}} \) digit of the Partial Remainder \( b \). A stage-wise substitution is performed and the equation in each stage is multiplied by the
<table>
<thead>
<tr>
<th>Sub-Operation</th>
<th>Result of Comparison</th>
<th>Operation Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to (n-1)</td>
<td>$A_{R_0} = B_{R_0}$</td>
<td>a) Add $2^{-(n-1)}$ to DR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b) Shift $B_R$ and $D_R$ left</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c) Subtract $A_R$ from $B_R$</td>
</tr>
<tr>
<td>1 to (n-1)</td>
<td>$A_{R_0} \neq B_{R_0}$</td>
<td>a) Add zero to DR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b) Shift $B_R$ and $D_R$ left</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c) Add $A_R$ to $B_R$</td>
</tr>
<tr>
<td>$n$</td>
<td>does not matter</td>
<td>Add $2^0 + 2^{-(n-1)}$ to $D_R$</td>
</tr>
</tbody>
</table>
appropriate power of two. The equations thus obtained are then added and the resulting equation is rearranged until an expression is obtained in the form $a/m = \text{Quotient}$. The result generated is correct in all respects except for the sign. To correct the sign unity is added to the result. The above scheme generates the true Quotient, $a/m$, regardless of the signs of 'a' and 'm'.

In Table 2, it is shown that $2^{-(n-1)}$ is added to the Quotient in the twelfth or last cycle. This is optional since its purpose is to give the least biased round-off error as will be discussed in Section 4.3. A sample calculation is given in the Appendix.

The Flow Diagram for Division is given in Figure 3, in which a Division is described with respect to the number of cycles required and with respect to the number of pulses required per cycle.

2.2 Basic Timing Requirements: Multiplier.

According to the Multiplication Algorithm, six basic cycles are required to accomplish the Multiplication operation. The pulse requirements in each cycle are not necessarily the same because 15 pulses are required in a cycle involving a Long Operation and only 3 pulses are necessary in a cycle involving a Fast Operation.

In both types of cycles, the first pulse is the OA pulse. This pulse is necessary in carrying out a pre-shift, an initial right shift of BR and DR, for setting a flip-flop to block out the optional OX pulse, for generating the Jam Pulse in the Fast Operation, and for setting the Add/Subtract Flip-Flop to its
FIGURE 3.

Flow Diagram for Division

<table>
<thead>
<tr>
<th>CYCLES</th>
<th>PULSES</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>First 10 Cycles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OA</td>
<td></td>
<td>Advance CC; Set Add/Sub FF; Add '1' or '0' to DR</td>
</tr>
<tr>
<td>OB</td>
<td></td>
<td>Left Shift BR and DR</td>
</tr>
<tr>
<td>OS (12)</td>
<td></td>
<td>Perform Addition or Subtraction</td>
</tr>
<tr>
<td>Last Cycle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OA</td>
<td></td>
<td>Advance CC; Set Add/Sub FF; Add '1' or '0' to DR</td>
</tr>
<tr>
<td>OB</td>
<td></td>
<td>Left Shift BR and DR</td>
</tr>
<tr>
<td>OS (12)</td>
<td></td>
<td>Perform Addition or Subtraction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>On OS₁₂ Complement DR₀ and Produce Shut-Off Pulse</td>
</tr>
</tbody>
</table>
desired state.

In the Fast Operation, the Jam Pulse advances the Pulse Counter by 12. In the Long Operation, after the OA pulse, 12 pulses are produced to carry out the addition or subtraction. These twelve OS pulses are channeled to AR and BR, and also to the Adder/Subtractor.

After the OA pulse in the Fast Operation, and after OS₁₂ in the Long Operation, two more pulses are produced, OX and OY. Each operation requires two right shifts due to the number of bits being compared. When a pre-shift has taken place on pulse OA, the right shift of BR and DR are blocked out on the OX pulse, and the mandatory second right shift occurs on the OY pulse. When no pre-shift has taken place, the required two right shifts are performed using the OX and OY pulses.

In Figure 4 are shown diagrams of the timing pulses for the different types of cycles encountered along with the total sequence of pulses for a complete Multiplication.

2.3 Basic Timing Requirements: Divider.

No algorithmic shortcuts have been incorporated into the Division process, making the first eleven cycles identical as all pulses are mandatory.

The process begins with generating an OA pulse. On this pulse the Add/Subtract Flip-Flop is set to the desired state and also a '1' or '0' is added into the least significant bit of DR, namely DR₁₁. An additional function of the OA pulse is to advance
FIGURE 4.

Multiplier Timing

Long Operation, 2 Post-Shifts

OA
OS
OX
OY

Long Operation with Pre-Shift

OA
OS
OX
OY

Fast Operation

OA
OS
OX
OY

Complete Typical Multiplication

OA
OS
OX
OY
the Cycle Counter by one.

The next pulse that is generated is the OB pulse, the sole purpose of which is to perform a single left shift on BR and DR. This shift could not be performed on the OA pulse since DR$_{11}$ requires at least 400 nanoseconds to settle when initially pulsed by OA.

After OB, the twelve OS pulses are generated. These pulses perform the addition or subtraction as dictated by the Add/Subtract Flip-Flop.

Once the eleven identical cycles have been completed, the Division is essentially finished. However, as stated in the discussion of the algorithm, at the completion of the Division operation the sign bit is incorrect. Thus a twelfth cycle is necessary to correct this condition. It consists of a single pulse which changes the sign of the Quotient by complementing DR$_0$. Actually it was found that since the D-Register is pulsed only on the OB pulses, the OS$_{12}$ pulse of the eleventh cycle can be used to correct the sign.

In Figure 5 are shown an example of the eleven identical cycles and an example of the twelfth cycle pulse timing, which occurs in reality on the OS$_{12}$ pulse and is used for shut-off and for the complementation of DR$_0$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
FIGURE 5. Divider Timing

First Eleven Identical Cycles

OA

QB

OS

Eleventh Cycle Shut-Off Timing

OA

OB

OS

Shut-Off Pulse

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER III
SYSTEM LOGIC

3.1 Multiplicand/Divisor Register (AR).

This register, as its name implies, is mainly used for the storage of the Multiplicand and the Divisor. By necessity, the storage must be non-destructive since these two numbers are necessary in each stage of the Multiplication or Division operations. The circuit diagram of AR is shown in FIGURE 6.

The facilities necessary are the direct clear, load in, and the right shift operations, in addition to which it must be a circulating shift register in order to retain the 12-bit number stored in it. It was found that the R-205 Flip-Flop was versatile enough to meet the above requirements. The bussed direct clear terminals achieve the clear operation. The third DCD gate in each Flip-Flop achieves the load-in operation by bussing the pulse input. Also the level input of each DCD gate is the corresponding output line from the Accumulator of the PDP-8/S.

The DCD gates at the gated Set and gated Clear terminals are used for shifting by cross-connecting the output of the previous Flip-Flop to the level input of the DCD gates of the following Flip-Flop. Thus when the pulse inputs of these DCD gates are pulsed, the shifting of the contents of one Flip-Flop to the next is achieved. To create the circulating register, the output of the last Flip-Flop of AR is cross-connected to the DCD gate level inputs of the first
Flip-Flop of AR.

Furthermore, it is necessary to make the contents of $AR_{11}$ available to the Adder/Subtractor at each instance of shifting. AR thus becomes the required circulating storage register.

3.2 Accumulator/Dividend/Most Significant Result Register (BR).

Beside providing a facility for loading in the Divisor, this register is also used as an accumulator to hold the Partial Product during the intermediate steps of Multiplication operation. The circuit diagram of BR is shown in FIGURE 7.

Due to the fact that Division was also included in the hardware, it became necessary to provide not only the right shift but also the left shift facility. Furthermore, the direct clear, load in and unload facilities were also found necessary. Thus, the DEC R201 modules, which provide five DCD input gates were selected. Two of these gates are used for right shifting as necessitated in Multiplication, two more are used for left shifting as required in Division, and the fifth DCD gate is used for loading in data.

To achieve the unload facility, one output gate had to be provided for each Flip-Flop in the register. The R123 NAND-gate module was selected for this purpose so that, when these gates are pulsed, the contents of BR are transferred to the PDP-8/S accumulator through the I/O cables. The outputs of the first bit, ie. $BR_0$, are compared in Division with the outputs of $AR_0$ to determine the operation that is to be performed. Thus the outputs of $BR_0$ are fed to the Divider Controller. The outputs of $BR_{11}$ (the last bit of BR)
FIGURE 7. "Accumulator/Dividend/Most Significant Result" Register (BR)
must be made available to the Adder/Subtractor in Multiplication and in Division. Furthermore the outputs of BR_{11} must also be supplied to DR_0 in Multiplication via the right shift level control, for, effectively in the process BR and DR form a double length register for the 24-bit result of Multiplication.

Again, right shift is achieved by the cross-connection of the output of the lower bit to the level input of the next higher bit. Left shift on the other hand is accomplished by the cross-connection of the output of the higher bit to the level input of the DCD gate of the next lower bit. In both types of shifting the respective pairs of DCD gates are pulsed at the same time when required via the pulse inputs of these gates.

During the intermediate operations of Addition or Subtraction, the DCD gate level inputs of BR_0 (on right shifts) are obtained from the output of the Adder/Subtractor. Furthermore, on the right shifts possible on OA, OX and OY during Multiplication, the first bit, namely BR_0, must remain unchanged. Thus BR_0 is right shifted only on the twelve OS pulses, whereas BR_1 to BR_{11} are right shifted on all compulsory right shifts namely on OA and OY, or on OX and OY. The right shift DCD gate pulse inputs are thus bussed on BR_1 to BR_{11} whereas the right shift pulse inputs of BR_0 are obtained directly from the OS pulse source. In Division, the left shift pulse inputs are obtained for all BR bits from the OB pulse source (with the aid of Pulse Amplifiers to satisfy power requirements).
3.3 Multiplier/Quotient/Least Significant Result Register (DR).

The operating characteristics of DR are indeed similar to those of BR, because DR is effectively the lower section of a double length register. The D-Register provides a place of storage for the Multiplicand and for the Least Significant Product in Multiplication and also for the Quotient in Division. The circuit diagram of DR is given in FIGURE 8.

Both right and left shift capabilities are included in DR for Multiplication and Division respectively, as well as the direct clear, load in and unload facilities. The DEC R201 Flip-Flops were used and two of its five DCD gates were used for the right shift, two for the left shift and the remaining DCD gate was used for loading in the Multiplicand directly from the PDP-8/S Accumulator.

Identical to BR, the R123 Nand gate module was used for unloading data back to the PDP-8/S Accumulator, thus utilizing one gate at the '1' output of each Flip-Flop of DR, DR₀ to DR₁₁.

At this point however DR's similarity to BR ends. There is a thirteenth Flip-Flop included in DR, namely DRᵇ, making DR a thirteen bit register. The purpose of this extra bit is to provide a facility for comparing the last three bits of DR as required in the Multiplication Algorithm. Thus bits DR₁₀, DR₁₁ and DRᵇ are examined during each cycle, with the understanding that DRᵇ is zero at the first stage of the Multiplication process. Because a total of twelve right shifts of DR are used for a complete Multiplication, two right shifts per cycle, the initial content of DR₀ at the first
FIGURE 8. "Multiplier/Quotient/Least Significant Result" Register (DR)
FIGURE 8. continued
stage will be found in $DR_b$ after the completion of the sixth or final cycle. Shifting in $DR$ is achieved by the same cross-connection and pulsing method as described for $BR$ in Section 3.2.

During Multiplication the '1' and '0' outputs of $DR_1^0$, $DR_1^1$, and $DR_b$ are routed to the Multiplier Controller for comparison purposes, and during the OA, OX, and OY pulse times the contents of $BR_1^1$ are routed to $DR_o$ by means of the right shift level control. This Right Shift Level Control is described in Section 3.7.

During the Division operation $DR_b$ is not used. Consideration must nevertheless be given to its contents because $DR$ is left shifted in each cycle. It is found that $DR_b$ must be set to zero at all times in Division in order to have $DR_1^1$ in the zero state such that it will be possible to add a 1 or 0 to it on the OA pulse without overflow. When the left shift occurs on OB, $DR_1^1$ will again become zero ready to accept a 1 or 0 on the OA pulse of the next cycle. It may be mentioned that the 'Add 1 to $DR_1^1$' pulse from the Divider Controller is applied to the set terminal of $DR_1^1$. Thus when the pulse is present a '1' is added to the D-Register and when the pulse input is absent nothing is done, that is, effectively a '0' is added to $DR_1^1$.

Because the Division Algorithm yields the correct magnitude but the wrong sign of the Quotient, the twelfth cycle operation, changing the sign, must be performed on $DR_0$. Instead of allowing the clock to run any further and having to find it necessary to block out all further pulses to or from the Pulse Counter, in order to
allow the twelfth cycle to take place, it was at first decided to merely delay the DIV. BUSY Shut-Off pulse long enough to allow the pulsing of the complement circuitry of \( DR_0 \). Upon further thought however, it was noted that the last pulse generated by the Clock and the Pulse Counter is the \( OS_{12} \) pulse. This last Add or Subtract pulse of the eleventh cycle merely changes the content of \( AR \) and \( BR \) only, and leaves the content of \( DR \) unaltered. In effect no reason could be found as to why \( DR_0 \) could not be complemented by the \( OS_{12} \) pulse; thus the eleventh and twelfth cycles are combined into one basic cycle.

The Multiplication and Division Shut-Off pulses are generated by the Pulse Counter and are OR-ed together to clear the Flags and to shut off the Clock. But since this complementation must be performed by the last pulse in Division only, the Pulse Counter was altered such that the DIV. BUSY Shut-Off pulse is generated by two separate parallel circuits. The output of one of these parallel circuits is OR-ed with the MUL. BUSY Shut-Off pulse and the output of the other is used to attain the complementation of \( DR_0 \).

Since all input facilities of the \( DR_0 \) Flip-Flop have already been used, it was found necessary to perform the complementation by means of collector triggering. Once the sufficient 100 nanosecond complement pulse was produced, the '1' and '0' level outputs of \( DR_0 \) were AND-ed with the complementing pulse by the DCD gate inputs of two Pulse Amplifiers. The output pulses of these Pulse Amplifiers were in turn connected back to '1' and '0' level outputs of the \( DR_0 \) Flip-Flop as shown in FIGURE 8. A very stable mode of operation of
the D-Register was thus achieved.

3.4 Adder/Subtractor.

The design of the Adder/Subtractor was achieved by consideration of the logical steps involved, expressing them in Boolean Algebra, and converting the final Boolean Function to a form suitable for implementation into NAND Logic.

Suppose that it is required to Add the two single-bit binary numbers X and Y. The result of \((X+Y)\) is '1' when \((X=0 \text{ and } Y=1)\) or when \((X=1 \text{ and } Y=0)\), otherwise, the sum equals 0 with the possibility of a carry. The above process can be represented in table form as shown below and it should be noted that the results obtained as, S and C, constitute a Half-Adder.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>S</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**TABLE 3. Half Adder for \((X+Y)\)**

Thus, calling the sum S,

\[
S = \overline{X}.Y + X.\overline{Y}
\]  \hspace{1cm} (1)

Also, if \((X=Y=1)\), the carry C is produced. Thus:

\[
C = X.Y
\]  \hspace{1cm} (2)
Considering now a binary number of length 'n' and recalling that serial operation is desired, the possibility that a carry was generated in the last previous stage has to be taken into account. This problem requires the storage or availability of the carry for the next stage of the addition process. The table below indicates the results, Sum (S) and Carry (C) from the addition of X and Y, and the previous carry (C_i) is also considered.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>C_i</th>
<th>S</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

TABLE 4. Full Adder for (X+Y)

The above process constitutes a Full-Adder and from the table the Boolean Equivalent may be written by representing the '1' value of each variable by the variable itself and the '0' value by the variable-bar. Thus for the Full Adder:

\[ S = \overline{XYC_i} + X\overline{YC_i} + \overline{XYC_i} + XYC_i \quad (3) \]

and

\[ C = XYC_i + X\overline{YC_i} + \overline{XYC_i} + XYC_i \quad (4) \]
Equations (3) and (4) were used in the implementation of the Adder.

In a similar manner a Half-Subtractor may be devised. Combining two Half-Subtracters the Full-Subtractor is obtained.

Using the Half-Subtractor, the binary number \( Y \) may be subtracted from the binary number \( X \). The results of the operation are shown in TABLE 5, in which the possibility of a Borrow (\( B \)) and Difference (\( D \)) are included.

\[
\begin{array}{|c|c|c|}
  \hline
  X & Y & D \quad B \\
  \hline
  0 & 0 & 0 \quad 0 \\
  0 & 1 & 1 \quad 1 \\
  1 & 0 & 1 \quad 0 \\
  1 & 1 & 0 \quad 0 \\
  \hline
\end{array}
\]

**TABLE 5. Half-Subtractor for \((X-Y)\)**

The Boolean Functions for the Half-Subtractor obtained from the table are:

\[
D = \overline{XY} + XY \quad \text{(5)}
\]

and

\[
B = \overline{X}Y \quad \text{(6)}
\]

At this point it may be worthwhile to note that the right side of the Boolean Function for \( S \) of the Half-Adder and for \( D \) of the Half-Subtractor are identical, but \( C \) and \( B \) are different.

Considering now the serial subtraction between binary numbers of longer length, it is noted that the new expressions for \( D \) and \( B \)
will be affected by the previous Borrow \( (B_i) \). TABLE 6. below shows this and gives the new values of \( B \) and \( D \).

\[
\begin{array}{cccc}
X & Y & B_1 & B \ D \\
0 & 0 & 0 & 0 0 \\
0 & 1 & 0 & 1 1 \\
1 & 0 & 0 & 1 0 \\
1 & 1 & 0 & 0 0 \\
0 & 0 & 1 & 1 1 \\
0 & 1 & 1 & 0 1 \\
1 & 0 & 1 & 0 0 \\
1 & 1 & 1 & 1 1 \\
\end{array}
\]

TABLE 6. Full-Subtractor for \((X-Y)\)

From TABLE 6., the Boolean Functions for the Full-Subtractor may now be written. They are:

\[
D = \overline{X}Y\overline{B_1} + X\overline{Y}\overline{B_1} + \overline{X}YB_1 + XYB_1 \quad (7)
\]

and \[
B = \overline{X}Y\overline{B_1} + \overline{X}YB_1 + \overline{X}YB_1 + XYB_1 \quad (8)
\]

Having thus obtained the Boolean expressions for the Adder-Equations (3) and (4), and for the Subtractor-Equations (7) and (8), it would be possible to construct the Adder and the Subtractor directly from these equations in AND/OR logic, noting the fact that an Add level and Subtract level are available for blocking out that section of the Adder/Subtractor which is not in use at a particular operation.
However, upon inspection of Equations (3) and (7), it is seen that they are identical in all respects except for the third variable used, namely $C_i$ and $B_i$. Since it is possible to OR $B_i$ and $C_i$ together, it is indeed possible to use the same circuitry to obtain the Sum ($S$) and the Difference ($D$). If Equations (3) and (7) were thus combined the resulting Boolean Function would be:

$$ (S \lor D) = \overline{XY} (C_i \lor \overline{B_i}) + \overline{XY} (C_i \lor \overline{B_i}) + \overline{XY} (C_i \lor \overline{B_i}) + XY (C_i \lor \overline{B_i}) \quad (9) $$

where $\lor$ indicates alternate inputs.

Having thus decreased the circuitry required for the Adder/Subtractor, it can further be seen, in the Carry and Borrow Expressions—Equations (4) and (8), that the terms $\overline{XY}C_i$ and $\overline{XY}B_i$ may also be similarly combined to form $XY (C_i \lor \overline{B_i})$.

Since NAND logic is used to implement the circuitry, the conversion of the Adder/Subtractor equations from AND/OR logic to purely NAND logic must now be performed. This is done by means of the STROKE method. Thus, all product terms are bracketed and all product and sum symbols are replaced by the Stroke symbol, $\parallel$.

Rewriting Equation (9) such that $(S \lor D) = R$ and $(C_i \lor B_i) = H$:

$$ R = \overline{XYH} + \overline{XYH} + \overline{XYH} + XYH \quad (10) $$

The conversion is now performed on Equation (10):

$$ R = (\overline{XYH}) + (\overline{XYH}) + (\overline{XYH}) + (XYH) $$

Thus

$$ R = ((\overline{X}/Y/\overline{H})/(X/\overline{Y}/\overline{H})/(\overline{X}/\overline{Y}/H)/(X/Y/H)) \quad (11) $$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
In the above equations, each bracketed term constitutes one NAND gate such that the bracketed variables are the inputs to the gate. The outputs are again NAND-ed together to obtain R.

By means of some Boolean Theorems, Equation (4) may be reduced to:

\[ C = XC_i + YC_i + XY \]  \hspace{1cm} (12)

which when converted to NAND form becomes:

\[ C = (X/C_i)/(Y/C_i)/(X/Y) \]  \hspace{1cm} (13)

Similarly Equation (8) may be reduced to:

\[ B = \overline{XY} + \overline{XB_i} + YB_i \]  \hspace{1cm} (14)

The conversion of Equation (14) to NAND form becomes:

\[ B = (\overline{X}/Y)/(\overline{X}/B_i)/(Y/B_i) \]  \hspace{1cm} (15)

Figure 9, the Adder/Subtracter, indicates how Equations (11), (13) and (15) have been implemented. It should be noted that in order to have \(B_i\) and \(C_i\) at hand, the values of \(B\) and \(C\) must be stored. This requires a Flip-Flop, conveniently called the Carry/Borrow Flip-Flop, which is changed on each OS pulse as made necessary by the Carry and Borrow Circuits. The outputs of this Flip-Flop are \((C_i \lor B_i)\) and \((\overline{C_i} \lor \overline{B_i})\) which are fed back as inputs to the Carry and Borrow sections of the logic.

One final note is that because the Add and Subtract levels
FIGURE 9. Adder/Subtractor

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
are used to block out the Borrow and Carry circuits as needed, it is not possible to have a Borrow during Addition or a Carry during Subtraction. For this reason, the $C_i$ or $B_i$ inputs may be used interchangeably as $C_i \lor B_i$ and $\overline{C_i} \lor \overline{B_i}$.

3.5 Pulse Counter.

The Pulse Counter (PC) is one of the more crucial sections in the control of the Multiplier/Divider mainly because it governs the length of each cycle and provides the necessary pulses within each cycle to carry out shifting, comparison, and internal control.

Because fifteen pulses are required for a long cycle of Multiplication and fourteen for a cycle of Division, the PC must be a four-stage counter that can count from 0 to 15. FIGURE 10 shows the Pulse Counter with its associated circuitry.

In general, as the PC is counting, the special counts are read-off from the output of its Flip-Flops and are gated to obtain the count levels on these special counts. These special count levels are then pulsed to obtain the required pulse on each special count. Since the PC is not allowed to advance until it returns to zero by itself, special count levels are used to reset the PC to zero by applying these levels to the level inputs of the DCD gates on a Pulse Amplifier. The pulse inputs of these DCD gates are then pulsed and thereby the Pulse Amplifier produces a 400 nanosecond pulse to clear the counter.

It was noted that the stagewise switching within the counter...
FIGURE 10. Pulse Counter
is not simultaneous and may vary up to 30 nanoseconds. Thus the special count levels are not clear but some spikes of short duration may appear at the beginning of the switching levels, especially switching at the 2's, 4's, and 8's positions. If the levels were thus simultaneously pulsed at the beginning of a change in a count of PC, these spikes could produce some undesirable pulses as well as the correct ones. For this reason it was found necessary to pulse the special count levels, then delay the clock pulse such that the delayed clock would be used to advance the PC to the next count.

A 200 nanosecond delay was produced for this purpose by means of two inverters and two Pulse Amplifiers. The real clock output was used to pulse the special count levels and to initiate the delay. The delayed clock pulses were then used to change the value of the PC. In this manner it became possible to read off the zero count which was used to generate the OA pulse.

The count of 0000 was detected via a NAND-gate and its associated level was pulsed by the real clock to obtain the OA pulse. All gates requiring the OA pulse were supplied from this point, i.e. the OA pulse source.

In Multiplication, the next twelve pulses required are the twelve OS pulses. In order to generate these pulses two special counts were read off, namely the first and the twelfth. These two counts were used to set and clear the OS Flip-Flop by means of some gating and a Pulse Amplifier. The output of the OS-FF was then pulsed by the real clock. Since the OS-FF is set for only twelve
pulse times, the result is twelve positive pulses and this point in
the gating thus becomes the positive source of the twelve OS pulses.

The gating used to set and clear the OS-FF is as follows. The
OA level is positive at its source. When NAND-ed with the MUL. BUSY
Level, which is negative, the result is a positive going level
change at the trailing edge of the OA level. This level change
triggers the R602 Pulse Amplifier producing a 100 nanosecond pulse
used to SET the OS-FF. Due to the fact that the other DCD gate of the
R602 Pulse Amplifier is used to produce a SET pulse for the OS-FF
during Division on another special count, the SET pulses to the OS-FF
are effectively OR-ed together in Multiplication and in Division.
It may be noted that it would be possible to use the inverted OA
level to set the OS-FF directly; however, during a Fast Operation it
is desirable not to set the OS-FF to avoid producing any extraneous
pulses at the OS pulse source. This may be further explained by the
fact that in a Fast Operation the PC skips the counts from 1 to 12.
Accordingly, if the OS-FF were set by the trailing edge of the zero
count, there would be no means to clear it. Thus, the OX and OY
pulses would also appear at the OS pulse source, causing incorrect
results.

In order to clear the OS-FF, the special count 12 is read
from the counter. While the output of the OS-FF is pulsed by the
real clock to obtain the twelfth OS pulse, the twelve count level is
pulsed by the delayed clock to clear the OS-FF.

As the counter is advanced to 13, this count is again read
and pulsed to obtain the OX pulse in a similar fashion as was done to obtain the OA pulse.

Again the counter is advances and the special count, 14, is detected and is then pulsed to obtain the OY pulse. The special count level of 14, being positive for the duration of the count, is also sent to the DCD level input of the R602 Pulse Amplifier. The delayed clock pulse which arrives at the DCD pulse input of the Pulse Amplifier, triggers it to produce the pulse that is used to clear the PC. In this manner, the PC is reset to 0000, ready to start the next cycle. Furthermore the special count level, 14, is also inverted and is used to block the delayed clock pulse from advancing the counter while the PC is being cleared. To do this the negative 14 level is applied to the DCD level input of 'One' s Flip-Flop of the Pulse Counter.

In the case where the Multiplier Controller finds that a Fast Operation is required, the OA pulse is produced as before. However, the OS-FF is not set, and the Jam Pulse produced by the Multiplier Controller is used to jam the number 13 into the Pulse Counter. The jam transfer is accomplished by applying the Jam Pulse to the extra DCD gate at the 'set' input of the appropriate R205 Flip-Flops of the Pulse Counter. Thus, on the next real clock pulse the OX pulse is produced and the cycle is completed as before.

In Division, fourteen pulses are required during each cycle. The OA pulse, as produced for Multiplication, is also produced here without any change in the logic.
The next special count, namely 0001, is also read from the counter and is pulsed by the real clock to produce the OB pulse. The 0001 count level, ie. OB level, is inverted and applied to the DCD pulse input of the R602 Pulse Amplifier used to produce the set pulse of OS-FF. On the trailing edge of the OB level this set pulse is produced. The twelve OS pulses are then produced by pulsing the output of the OS-FF. Then, by reading the 13 count level from the counter, inverting it, and pulsing it by the delayed clock, the OS-FF is cleared.

Because on the twelfth OS pulse, this cycle in the Division process is finished, the 13 count level is also applied to the second DCD gate of the Pulse Amplifier that is used to clear the PC to zero. Furthermore, it is gated with the 14 count level of Multiplication to produce the level used to block the delayed clock from advancing the PC while it is being cleared.

In the above fashion the Pulse Counter produces all the pulses required in any single cycle within the Multiplication or Division processes.

3.6 Cycle Counter.

Of the two counters present, the Cycle Counter is the less complex one, and its primary purpose is to regulate the number of cycles required in the two types of processes. Since twelve cycles are required in Division, the Cycle Counter (CC) must be at least four bits in length, and because only six cycles are necessary in
Multiplication, the CC must be capable of counting by ones or twos. It must also be capable of detecting the final cycle in each case and shut off the unit such that the results are safely stored in the appropriate registers.

The DEC R202 Flip-Flops were found to be sufficient to use here since the CC is basically an up-counter and no direct transfer of data is required into it. The logic is arranged, as shown in FIGURE 11, such that the OA pulse, which is common to the Multiplier and to the Divider sections, advances the counter each time this pulse occurs. Thus the count 0000 is not used as the counter advances to 2,4,6,8,10,12 in Multiplication and to 1,2,3,4,5,6,7,8,9,10,11 in Division. Furthermore, before counting starts, the CC is direct cleared on the IOT1 pulse when the first data item is loaded into AR.

The OA pulse is routed to the DCD pulse inputs of the '1's Flip-Flop in Division by gating it with the DIV. BUSY Flag, and is routed to the pulse inputs of the DCD gates of the '2's Flip-Flop in Multiplication by gating OA with the MUL. BUSY Flag. One problem that had to be rectified was that if, as initially, the counter were in the zero state, then the output of the '1's Flip-Flop was at ground. Since the output level was then inverted and NAND-ed with the DIV. BUSY Flag and since the Flag was set before the first OA pulse arrived, for a short length of time the DCD pulse input of the '2's Flip-Flop was driven to ground. This set the counter to the '2' state before the first OA pulse arrived and thus counting was erroneous.

To remedy this situation, an R204 Flip-Flop was added to the
'1' output's circuitry. The R204 was set by the OA pulse and its output was then used in place of the DIV. BUSY level to gate the output of the '1's FF to the pulse inputs of the 2's FF. Thus, since up to the time of the first OA pulse the output of the '1's FF is blocked out, the first OA pulse on DIV. BUSY is capable of setting the CC to the '1' state. The fictitious DIV. BUSY Level comes on at this time but the inverted '1's output is now at ground again blocking out any change in the '2's FF. On the second and subsequent OA pulses the counter changes as in normal operation.

In the Multiplication operation, the OA pulse is NAND-ed with the MUL. BUSY Flag and is applied to the DCD inputs of the TWO's FF. At the same time, since the DIV. BUSY Flag is at ground, the output of the '1's FF is blocked out thus reducing the unit to a three-bit counter, and effectively causing the count to proceed upward by twos.

As Multiplication or Division is completed, the CC must generate a pulse that will stop the clock and indicate that the operation has been completed. For this purpose the level output of the count for the last cycle is detected and is pulsed by the appropriate last pulse within the final cycle. The results are effectively OR-ed together to achieve shut-off for both Multiplication and Division. The last cycle for Multiplication is the sixth and for Division the final cycle is the eleventh. It should be noted that the twelfth cycle for Division consists of a single pulse which is obtained directly from the Divider Shut-off Pulse Source. The number II is thus read by two gates from the CC and after further gating the results are
OR-ed together to produce the Shut-off level of the respective operation. It could be said that in reality the number six is detected in the CC in Multiplication because the CC at this time is considered as an effective three-bit counter. Really the '8's and '4's FF's of the four-bit counter are examined.

In Multiplication the last pulse in any cycle is the OY pulse which is gated at this point with the Shut-off level. In Division though, the last pulse of a cycle is the twelfth OS pulse. Since OS$_{12}$ is not easily detected or separated from the other OS pulses, the equivalent of OS$_{12}$ is generated by gating the real clock, the DIV. BUSY Level, and the Reset Level of the Pulse Counter. The result is that the OS$_{12}$ pulse is generated in each Division cycle. When this pulse is then gated with the Shut-off level produced in the CC the correct Shut-off pulse is produced in Division. Similarly, the OY pulse and the Shut-off level are gated to produce the correct Shut-off Pulse for Multiplication. The results of these two gates producing the separate Shut-off Pulses are then effectively OR-ed to bring about a source for a single Shut-off Pulse for both processes.

In Multiplication and Division the Shut-off Pulse may be directly used to clear the Busy Flag in order to shut off the clock. In Division however, the Divider Shut-off Pulse is also used as the complement pulse for DR$_0$ as previously discussed in Section 3.3.
3.7 Right Shift Pulse and Level Control.

The purpose of the logic shown in FIGURE 12, is to accumulate all the required right and left shift pulses, to combine them as necessary and to distribute them to the appropriate registers.

The requirements indicate that AR must be shifted right whenever OS pulses occur whether Multiplication or Division must take place. BR must be shifted right whenever OS pulses occur as well as when the Multiplier Controller generates right shift pulses on the OA or OX, and on the OY pulses. The right shift requirement of DR is that only the OA or OX, and the OY right shift are necessary as supplied by the Multiplier Controller.

Since OS pulses occur when Addition or Subtraction is required, and since 3 milliamps are required at each stage of the twelve bit AR, the OS pulses must be supplied from a Pulse Amplifier to provide sufficient power for the right shift requirements of AR. The OR-ing of the OS pulses with the right shift pulses from the Multiplier Controller supplies all the right shifts that must take place in BR again with the help of a Pulse Amplifier to supply the necessary current load. For DR it is merely necessary to provide the OA or OX, and OY shift pulse through a Pulse Amplifier to attain correct shifting.

This may be an appropriate point to mention also that in Division, where left shifting of BR and DR is necessary, the amplified OB pulse must be supplied to the two registers involved.

The Right Shift Level Control may be observed in FIGURE 12.
FIGURE 12. Left/Right Shift Pulse and Level Control
and its output is routed to the input circuitry of the \( \text{DR}_0 \) Flip-Flop in FIGURE 8. A set of four NAND gates are placed between the last Flip-Flop of \( \text{BR} \) and the first Flip-Flop of \( \text{DR} \) so that, during the OS pulse times, the output of \( \text{BR}_{11} \) is fed to the Adder/Subtracter, and during the OA, OX and OY pulse times the output of \( \text{BR}_{11} \) is applied to the DCD level inputs of \( \text{DR}_0 \). This controls the level inputs to the Adder/Subtracter and to \( \text{DR}_0 \) during the right shifts and has thus been called the Right Shift Level Control.

3.8 Multiplier Controller.

As shown in FIGURE 13, the logic diagram for the Multiplier Controller, the output levels of \( \text{DR}_{10} \), \( \text{DR}_{11} \) and \( \text{DR}_b \) are compared. Gates 1 and 2 compare the '0' and '1' outputs of \( \text{DR}_{11} \) and \( \text{DR}_b \). If the results of this comparison show that these bits are unlike, then neither gate is satisfied and their outputs are -3 volts. In this case, the inverted outputs yield a ground input to gates 4 and 5. As a result, the outputs of gates 4 and 5 are -3 volts causing a ground output from gate 6. If the output of gate 6 is ground a Long Operation will occur during this cycle, and, if the output is -3 volts a Fast Operation is indicated.

The Leech Algorithm states that if the three bits compared are alike, a Fast Operation is required, and if unlike a Long Operation must occur. Supposing then that \( \text{DR}_{11} = \text{DR}_b = 1 \), the output of gate 1 will be ground. In either of the above cases gates 4 and 5 are used to determine whether or not one of the Fast Operation patterns has
FIGURE 13. Multiplier Controller
occurred. Thus if \( DR_{10} = DR_{11} = DR_b = 1 \), the output of gate 4 is ground. This causes a -3 volt output from gate 6 indicating a Fast Operation. Similarly when the pattern is \( DR_{10} = DR_{11} = DR_b = 0 \), the output of gate 5 is ground causing -3 volts at the output of gate 6, again indicating a Fast Operation. This concludes the recognition of the 000 and 111 patterns.

If we still assume \( DR_{11} = DR_b \) whether 11 or 00, the output of gate 3 is -3 volts. If a Fast Operation does not occur, ie. output of gate 6 is ground, two types of operations may still be recognized. These are the cases 100 and 011 in which a pre-shift must take place on the OA pulse. The ground output of gate 6 is inverted and, together with the output of gate 3, is pulsed by the OA pulse. If the above conditions are thus satisfied, an OA pulse will be generated at the output of gate 7 for purposes of right shifting.

In the remaining four cases, where the patterns may be 010, 101, 110 or 001, no OA pulse will be generated at the Right-Shift Pulse source, namely at the output of gate 11.

Two right shifts must occur in each cycle of the Multiplication process. In all cycles, except where the observed pattern was 100 or 011, an OX and an OY shift pulse are produced at the output of gate 11 by means of gates 10 and 9 respectively. These pulses are of course obtained from the Pulse Counter and are gated at gates 9 and 10. If a pattern like 100 or 011 is observed and an OA pulse is produced at the output of gate 7, both OX and OY cannot be allowed to appear at the Right Shift Pulse Source. The R202 Flip-Flop was inserted for the
purpose of allowing only one of OX or OY through. Initially the FF is clear, and when the OA pulse occurs at the output of gate 7, the FF is set using the OA pulse. This causes the '0' output of the FF to become ground and blocks the OX pulse from passing through gate 10. The OX pulse is used in turn to clear this FF so that it is ready for the next cycle.

The outputs of gates 8, 9 and 10 are OR-ed together using gate 11 and the output of gate 11 is inverted and used as the Right Shift Pulse Source for the shift pulses OA, OX and OY.

One further consideration at this point is the generation of the Jam Pulse. When a Fast Operation occurs, the Fast Operation level is gated with the MUL. BUSY Flag and the OA pulse, as shown by gate 12, to produce a condition that will allow the Pulse Counter to be advanced by twelve, in other words, to skip the Addition or Subtraction operation in that particular cycle. The output of gate 12, used to set an R202 Flip-Flop, from here on is known as the Jam FF. The '1' output of the Jam FF is pulsed by the same clock that is used to advance the PC. In this manner a single pulse, the Jam Pulse, is produced. The Jam Pulse, as discussed in Section 3.5 is applied to the DCD load-in gates of the Pulse Counter. Thus, after the OA pulse, the OX pulse is produced by the PC. This OX pulse is further used to clear the Jam FF.

In the above manner, the Multiplier Controller first detects what kind of cycle will occur, Fast or Long, and then generates the OA, OX, OY and Jam pulses as necessary.
3.9 Divider Controller

It was due to the simple or compact nature of the Divider Controller that it was actually decided to include the Divider in this project. The Booth Algorithm for Division states that \( AR_o \) and \( BR_o \) are to be compared. On the basis of this comparison the Add/Subtract FF is set to the desired state, and also using the results of the comparison a '1' or '0' is added to DR. FIGURE 14 shows the logic for the Divider Controller, and, for the sake of compactness, the Add/Subtract FF is also included here even though the Add/Subtract FF is used both in Multiplication and in Division.

The '1' and '0' outputs of \( AR_o \) and \( BR_o \) are applied to gates 1 and 2 of the Divider Controller. The results are OR-ed together such that when \( AR_o = BR_o \) the output of gate 3 is -3 volts, and when \( AR_o \neq BR_o \) the output of gate 3 is ground. This output is applied to one pair of DCD gates of the Add/Subtract FF - the inverted output is applied to one DCD gate and the actual output to the other - and on the OA pulse the Add/Subtract FF is set for like results and cleared for unlike results. The '1' and '0' outputs of the Add/Subtract FF indicates Subtraction and Addition respectively.

The '1' and '0' outputs of \( DR_{10} \) are also applied to the second pair of clear and set DCD gates of the Add/Subtract FF to produce the proper Addition and Subtraction levels at the '0' and '1' output terminals of the Add/Subtract FF during Multiplication.

Furthermore, in Division the output of gate 3 is also gated with the OA pulse to produce the 'ADD 1 to DR' pulse if the result of
FIGURE 14. Divider Controller
the comparison at gate 3 is -3 volts, that is, if $A_{R_0} = B_{R_0}$.

3.10 I/O, Clock, and Flag Circuits.

From the regular I/O cable positions of the PDP-8/S Interface, six cables are connected to the Multiplier/Divider in order to accomplish data and instruction transfers to or from the accumulator of the PDP-8/S.

The PDP-8/S accumulator output lines are connected to the input gates of all three registers, AR, BR and DR. In turn the output lines of BR and DR of the Multiplier/Divider are connected in parallel to the input gates of the PDP-8/S accumulator. Furthermore, the last 8 bits of the Memory Buffer Register (MBR) of the PDP-8/S are parallel connected to the device coding inputs of the two Device Selector Modules (W103 Modules) of the Multiplier/Divider as necessitated by the code numbers assigned to the device, namely 40 and 41. The only remaining cable connections required are the three lines connecting the IOP Generator of the PDP-8/S to the IOP inputs of the two Device Selectors of the Multiplier/Divider. The I/O Skip line that is used to indicate the ready status of the external device is also required.

Figure 15 shows the Device Selectors, Flags, clock and Delay circuits. When an I/O instruction occurs, the PDP-8/S IOP Generator produces the corresponding IOP pulses, namely IOP1, IOP2, IOP4 or a combination of these depending on the particular instruction. The external device is selected using the MBR whose contents are decoded.
by means of the NAND gate in the particular Device Selector. When an external device is selected, the IOP pulses are allowed to enter that particular device by means of the three Pulse Amplifiers in the Device Selector. These Pulse Amplifiers generate the appropriate and corresponding IOT pulses, IOT1, IOT2 or IOT4, and these pulses are in turn used to initiate the execution of the various instructions.

The IOT1 pulse of Device Selector 40 is used to first clear all registers, counters and flags. It is then delayed 400 nanoseconds using an R302 delay to allow the cleared Flip-Flops to settle in the '0' state and at the end of the delay, using the delayed pulse, AR is parallel loaded with the value of the Multiplicand or Divisor from the accumulator of the PDP-8/S.

The IOT2 pulse generated by code 6402 is used to set the MUL. BUSY Flag which is an R204 Flip-Flop shown in FIGURE 15. The '0' outputs of the Flags are OR-ed together such that whenever one or the other of the two Flags is set, the resulting negative level starts the clock (the R401 module) after another 400 nanosecond delay.

At the same time that the IOT2 pulse sets the MUL. BUSY Flag it is also used to pulse the DCD input gates of DR to load the value of the Multiplier from the Accumulator of the PDP-8/S into DR. Similarly, at the same time that the IOT4 pulse sets the DIV. BUSY Flag, the IOT4 pulse is also used to load the value of the Dividend into BR from the PDP-8/S Accumulator.

The setting of the R204 Flip-Flops would ordinarily start the clock, but since the loading operation is performed by IOT2 as well as
FIGURE 15. I/O, Clock, Flag Circuits
by IOT4, if the starting of the clock were not delayed the contents of
the BR or DR could be destroyed. Thus, the delay circuit shown in
FIGURE 15 was inserted and the clock is started after the delay is
complete.

The Device Selector, coded 41, is used for unloading data and
for testing the DEVICE BUSY Flags. The negative IOT1 pulse generated
by 6411 is used to pulse the outputs of the MUL. BUSY and DIV. BUSY
Flags such that if either Flag indicates a Busy Device a pulse is
sent back to the PDP-8/S instructing the computer to test again. When
the Flags are cleared, no IOS pulse is returned to the PDP-8/S
indicating that the current operation has been completed.

The IOT2 pulse generated by Code 6412 is used to pulse the
Unload NAND gates (R123 Modules) of DR achieving a parallel transfer
of the contents of DR to the Accumulator of the PDP-8/S. Similarly,
the IOT4 pulse generated by Code 6414 is used to achieve an identical
unloading of the content of BR back to the PDP-8/S Accumulator.

3.11 Total Sequence of Operation.

All the logic used in the Multiplier/Divider has now been
described and an overall sequence shall now be described for a
Multiplication and for a Division Operation.

The Multiplication sequence begins with loading the Multipli-
cand into AR using the IOT1 pulse as generated by the Code 6401.
Then using the IOT2 pulse generated by the code 6402, the Multiplier
Quantity is loaded into DR. The MUL. BUSY Flag is then set and the
last three bits of DR are compared with the understanding that $DR_b$ is initially in the '0' state. The clock begins to advance the Pulse Counter and the Cycle Counter and from the Pulse Counter the OA pulse sets the Add/Subtract Flip-Flop, as warranted, and if necessary, a pre-shift is performed on BR and DR, or a Jam Pulse may be produced as a result of the comparison. If a Fast Operation is warranted, the OX and OY pulses are produced next; if however, the result of the comparison indicates a Long Operation, then the OS-FF is set to the '1' state and the twelve OS pulses follow. The OS pulses perform an Addition or Subtraction between AR and BR, i.e. $(BR \pm AR)$. Following this, the OX and OY pulses are produced which, along with the previous OA pulse, perform the necessary right shifts. The first cycle of Multiplication is completed, the PC is cleared and, on the OA pulse of the next cycle, the Cycle Counter is advanced by two again. In a similar manner the second to the sixth cycles are completed. On the sixth cycle, when the OY pulse occurs, the MUL.BUSY Flag is cleared and the Multiplication Operation has been completed. As the MUL. BUSY Flag is cleared, and the I0P1 pulse is generated by Code 6411, the skip pulse is not sent back to the PDP-8/S indicating that the Multiplication is complete. Following this, the 6412 and 6414 instruction can unload the most and least significant portion of the result and place them into the PDP-8/S Accumulator.

The Division sequence similarly begins by loading AR, with the Dividend in this case, using the IOT1 pulse generated by Code 6401. At first a general clearing is done on all registers, flags and counters,
following which the loading operation occurs. Using the IOT4 pulse generated by Code 6404, the Divisor is loaded into BR and the DIV. BUSY Flag is set. Following a delay the clock starts to pulse the PC producing the OA pulse which advances the Cycle Counter by one. The first bits of AR and BR are compared and on the OA pulse the Add/Subtract FF is set to the required state and at the same time a '1' or '0' is added to DR. On the next clock pulse, the PC produces the OB pulse which is used to left shift BR and DR by one place. The other purpose of the OA pulse is to set the OS-FF to the '1' state such that on the next twelve clock pulses the twelve OS pulses are produced. The twelve OS pulses accomplish the required Addition or Subtraction of AR and BR, i.e. (BR + AR) or (BR - AR). At this point the first Division Cycle is complete so that the PC is cleared making the device ready for the second cycle. On the upcoming OA pulse, the CC is again advanced by one and cycles two to eleven are performed in an identical manner.

When the eleventh cycle has been completed the twelfth OS pulse has been detected and used to complement DR0 thus correcting the sign of the Quotient. The twelfth OS pulse also shuts off the clock and clears the DIV. BUSY Flag. On the next IOT1 pulse generated by Code 6411 no IOS pulse is sent back to the PDP-8/S indicating that the Division has been completed. The Quotient is then unloaded from DR to the PDP-8/S using the IOT2 pulse generated by Code 6412. In closing it may be mentioned that the final remainder may also be unloaded by the IOT4 pulse generated by Code 6414 for examination and round-off correction by the programmer.
4.1 Cost Realization.

The overall cost of the Multiplier/Divider was estimated at $1500., however, the actual cost which also included the mounting panels, bussing strips, cables, and cable ends, just exceeded $2000.

It should be mentioned though that in some places the circuitry may not be the optimal one due to the fact that in some sections different logic may be used to accomplish the same function. As an example, a delay may be achieved using an R302 module or by the interconnection of several Pulse Amplifiers, or even by means of a counter. The method used was decided by the type of components that were available at the time. Another example may be the question of which method requires more circuitry: to have a counter count by ones and twos until the final count or to have it count by ones at all times and detect the different final counts?

Also, due to problems incurred in the debugging stage and due to the solution of these, in many places there are modules in which all the circuitry has not been utilized. In some spots where spare gates were available they were not used because of the lengths of wires which would have had to be attached for use in more distant locations. If long lengths of wire are used, wiring capacitance could become noticeable. If the project were to be carried into the production stage, many of these cost inefficiencies could easily be corrected.
With this in mind and keeping in mind that the main purpose was the realization of a functional unit, the intention of the project was indeed realized and well rewarded. Producing the device on a larger quantity scale would also reduce costs and the overshoot in the estimated cost could then be termed minimal. A low cost unit was produced with excellent performance results as will now be discussed.

4.2 Speed Realization.

In theory as modules labelled '2 MHz' were used, it could be expected that 2 MHz operation should have been realized. NAND gating will respond to such a speed but DCD gating on Flip-Flops require a 400 nanosecond setup time on the level inputs before the gates can be pulsed. To this, carry propagation in counters may be added, 70 nanoseconds per stage, and the maximum realizable speed is cut to about 1.3 MHz. In an attempt to avoid overdriving and operation too close to the upper limit, 1 MHz operation was used and this speed does satisfy the initial requirements as outlined in CHAPTER I.

Accordingly the maximum Multiplication time became 90 microseconds and the minimum Multiplication time is 18 microseconds. This is of course exclusive of the time required by the PDP-8/S to generate the IOP pulses and exclusive of the times required to load in data and to unload results. The time required for a Division is furthermore fixed at 154 microseconds. These times also meet the speed and time requirements of CHAPTER I.

The speed of the device was also tested out with regards to
comparison with software methods. A programme was written to multiply together all possible combinations of positive and negative numbers. The range of positive numbers is 0 to \(3777\) and the range of negative numbers is \(-1\) to \(-3777\) (4000 to 7777). This gives the total number of possible different numbers as \(4096\). Thus 4096 different numbers were multiplied by 4096 different numbers. The total number of Multiplications thus performed was 16,777,216 Multiplications. The Multiplier/Divider performed the above task in 102 minutes. A little manual calculation would indicate that the average Multiplication time was 365 microseconds. This is a true average since the numbers multiplied during each operation were changing. The above average time includes the loading and unloading of the data as well as the incrementation of the counters used as the Multiplier and Multipli-cand.

It should be noted that since 18 and 90 microseconds are the lower and upper limits of the basic Multiplication, the average actual hardware Multiplication would require approximately 76 microseconds. Thus 289 microseconds of the average programme cycle time was used in the time required to programme the device on the PDP-8/S.

Subroutine Multiplication was also performed. If the pro-gramme used in this case were also required to perform 16,777,216 Multiplications, the expected run time would require not a matter of hours but a full day. For this reason, it was decided to perform a lesser number of Multiplications by software and, on the basis of
the run time and number of operations performed to calculate the average software Multiplication time. The average Multiplication time thus obtained was 5,375 microseconds, or approximately 5.4 milliseconds.

As a result and conclusion it was determined that hardware Multiplication is approximately 15 times faster than software Multiplication. This may vary from one Multiplication to another, but the above ratio is a fair average and would tend to prove the Multiplier/Divider to be a valuable asset. This value is very useful in graphics where to rotate an object displayed on a screen would require Multiplications performed in approximately 6 seconds by hardware and one and a half minutes by software. The image transformation rates are seemingly much faster by hardware.

Software Division was not performed since much longer times are automatically expected than for Multiplication due to the more complex nature of Division. But it can be pointed out that the 299 microsecond average programming time involved in Hardware Multiplication would be the same in Hardware Division. To this the constant 154 microseconds spent in the Hardware unit may be added obtaining a Hardware Division time of approximately 443 microseconds. It may be expected that the ratio of Software and Hardware programme operation times may even be higher than for Multiplication.

It may thus be concluded that the requirements outlined in CHAPTER I have been accomplished and that the Hardware unit is a valuable tool with impressive advantages over its Software counterpart.
4.3 Reliability and Accuracy with Error Discussion.

In the debugging stage, only one instability was found, this being the unstable collector triggering of DR₀ in correcting the sign of the Quotient. This condition was corrected with the replacement of the complementing NAND-gates by two Pulse Amplifiers having DCD gate inputs. Thus proper collector triggering of DR₀ was obtained.

For a reliability test, the unit was operated such that the Multiplicand Quantity was kept constant and the Multiplier Quantity was varied through the whole sequence of possible numbers, positive and negative. In effect this became a thorough test of the Multiplier Controller circuitry, the Adder/Subtractor circuitry and all the circuits involving shifting. As a consequence it was found that the results retrieved were all correct even to the last digit, leading to a conclusion that zero error is incurred in Multiplication.

A similar test programme for Division showed that whenever negative numbers were involved the least significant digit of the Quotient was always zero. The remainder in BR, is also retrievable; Thus, the user has the option to correct the result by adding a '1' to the final bit of DR. The results obtained are accurate to 2/1024 and the answer retrieved for -1/2 would be -1776. If the correction would be applied the result for -1/2 would be -1777 and the correct result for 1/2, which is ordinarily 2000, would be 2001. Thus the resulting error for each Division is +1/1024 or -1/1024. The average error is zero and the RMS error according to Booth(2) is 1/12 times
1/1024. This is very acceptable when it is considered that the numerical representation in Division is the fractional form.

The Multiplier/Divider is thus also reliable and yields accurate results.
CHAPTER V
USE AND LIMITATIONS

5.1 Instruction Set.

The two Device Selectors used make it possible to create six simple instructions. By means of storage and gating, it would be possible to realize these six instructions using a single Device Selector by the various combinations of the three IOP pulses, however, it was found less expensive to use a second Device Selector module and two device codes.

For convenience the Device Selector coded 40 is used for loading in data and for performing the actual Multiplication or Division and the Device Selector coded 41 is used for unloading the results and for the Skip instruction, testing for a busy device.

Since it is necessary to load in two pieces of data, the first load instruction is common to both the Multiplier and to the Divider. This instruction for which the actual code assigned in 6401 has also been assigned the mnemonic EMQL, meaning External Multiplier Quotient Load. For a Multiplication to take place, the second item of data, which must be located in the PDP-8/S Accumulator, is loaded into DR and the Multiplication is carried out by the instruction whose octal code is 6402 and to which the mnemonic EMUY, External Multiply, has been assigned. Similarly, for Division the Dividend is
loaded into BR and the Division is carried out by the instruction whose octal code is 6404. This instruction has been assigned the mnemonic EDVI, meaning External Divide.

Because a Multiplication or Division may exceed one computer I/O instruction time, the 6411 instruction is used to continuously test the two device Flags to detect when the particular operation has been completed. This instruction has the mnemonic DSKP, meaning Device Skip. It is not absolutely necessary to make use of this instruction. Knowing that the actual Multiplication time never exceeds 100 microseconds and that the actual Division time never exceeds 160 microseconds, if the results of the operation are not immediately necessary, the user can make certain that before unloading the results he performs numerous other instructions. He must be certain though that enough time elapses for the operation to be completed before unloading the results from the external device.

The unloading instructions have octal codes 6412 for the mnemonic ULDL and 6414 for the mnemonic ULDH. The former unloads the Least Significant Result, hence ULDL, and the latter unloads the Most Significant Result, hence the mnemonic ULDH.

A summary of the instruction set for the Multiplier/Divider appears in TABLE 7 where the Octal Code, Mnemonic, and Equivalent Meaning are given.
<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OP. CODE</th>
<th>EQUIVALENT MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMQL</td>
<td>6401</td>
<td>Clear all registers, counters, flags Load in Multiplicand or Divisor</td>
</tr>
<tr>
<td>EMUY</td>
<td>6402</td>
<td>Load in Multiplier Carry out Multiplication</td>
</tr>
<tr>
<td>EDVI</td>
<td>6404</td>
<td>Load Dividend Carry out Division</td>
</tr>
<tr>
<td>DSKP</td>
<td>6411</td>
<td>Test Device Busy Flags Skip next instruction when Flags are clear</td>
</tr>
<tr>
<td>ULDL</td>
<td>6412</td>
<td>Unload Least Significant Multiplier Results, or Quotient</td>
</tr>
<tr>
<td>ULDH</td>
<td>6414</td>
<td>Unload Most Significant Multiplier Results or Remainder</td>
</tr>
</tbody>
</table>
5.2 Multiplier Programme.

Unless the user changes the PAL III Assembler, the Mnemonic Set must be defined and placed into the symbol table at the beginning of each programme. The equal sign will accomplish this purpose. Depending on the preference of the user, he is at liberty to merely use the octal codes. For the Multiplication of two numbers the two programmes below will yield identical results which will be stored in locations labelled HIRES and LOWRES where MLTO and MLBY are the two numbers being multiplied together.

Programme Using Octal Codes

```
*200
CLA CLL
TAD MLTO
6401
CLA CLL
TAD MLBY
6402
6411
JMP.-1
CLA CLL
6414
DCA HIRES
6412
DCA LOWRES
HLT
MLTO,9
MLBY,4
LOWRES,0
HIRES,0
```

Programme Using Mnemonics

```
EMQL = 6401
EMQY = 6402
DSKP = 6411
ULDL = 6412
ULDH = 6414
*200
CLA CLL
TAD MLTO
EMQL
CLA CLL
TAD MLBY
EMQQY
DSKP
JMP.-1
CLA CLL
ULDH
DCA HIRES
ULDL
DCA LOWRES
HLT
MLTO,9
MLBY,4
LOWRES,0
HIRES,0
$`
5.3 Divider Programme.

In a fashion similar to Multiplication, simple programmes may be written to perform the Division operation. Below are two such programmes. In one the octal codes of the I/O instructions are used and in the other the mnemonics are merely defined and are used as the actual instructions. The Dividend and Divisor are stored and only the Quotient is extracted.

<table>
<thead>
<tr>
<th>Programme Using Octal Codes</th>
<th>Programme Using Mnemonics</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA CALL TAD DIVSOR 6401</td>
<td>EMQL = 6401</td>
</tr>
<tr>
<td>CLA CALL TAD DIVEND 6404</td>
<td>EDVI = 6404</td>
</tr>
<tr>
<td>CLA CALL 6411</td>
<td>DSKP = 6411</td>
</tr>
<tr>
<td>JMP.-1</td>
<td>ULDL = 6412</td>
</tr>
<tr>
<td>*200</td>
<td></td>
</tr>
</tbody>
</table>

```
*200
CLA CALL TAD DIVSOR
6401
CLA CALL TAD DIVEND
6404
6411
JMP.-1
CLA CALL
6412
DCA QUOTNT
HLT
DIVSOR,2
DIVEND,1
QUOTNT,0
$
```

5.4 Some Limitations.

No limitations or errors arise in Multiplications due to the fact that integer numbers are being multiplied and an integer value is the result. In this respect the Multiplier portion of this device
has become an accurate, reliable and useful tool in increasing the
speed of executing programmes involving Multiplication on the PDP-8/S.

As far as the Divider is concerned the greatest limitation
results from the fact that Division of numbers such as 9/3 or 32/8
is not directly possible. This occurs because the algorithm restricts
Division to those operations in which the Divisor is greater than the
Dividend. As a consequence in Divisions of the former type scaling is
required prior to the Division to insure a valid type of Division
after which a scaling is again required on the result to bring the
Quotient to the fixed point format. Such scaling will make it
possible to directly use the results of the Division in stages of
the programme after the Division operation.

Two types of scaling are available to the user, namely, where
the numerator is scaled down until it is smaller than the denominator,
and the alternative where the denominator is scaled up until it is
larger than the numerator. In the former case grave errors may be
introduced since, in scaling down, if the prime factors of the
numerator are such that the numerator can no longer be reduced by a
factor of two without incurring a fractional result, an error will be
introduced even before the Division can take place. Such would be the
case in the Division of numbers like 36/4. If 36 is scaled down by
factors of two until the numerator is less than the denominator, the
following scaling process will result: 36, 18, 9, 4, 2. Thus, the
Division will be 2/4 = 1/2 which when scaled up by a factor $2^4$ will
result in 36/4 = 8. This is of course incorrect since 36/4 = 9.
The second scaling method is much better in this respect. Suppose that the denominator is scaled up by factors of two until it is greater than 36. Thus 4, 8, 16, 32, 64 is the scaling sequence. The Division will be 36/64 which is allowed. By scaling the denominator up by $2^4$ the Quotient has to be scaled up by $2^4$ (i.e., 4 programmed left shifts are required). The Divider will yield the answer $0.10010000000$ which when scaled up will give the Quotient, 9.

This latter method must then be adopted by the user for scaling. Since the PDP-8/S operates on integer numbers in PAL III, it is felt that in most cases a Division of the type 36/4 will be performed as opposed to the type 4/36. The latter type may occur, but the effective future use or value of the result is zero. This is said because the result of such a Division in integer mode should be zero. At one time it was proposed to design an automatic scaling and rescaling section into the Hardware Divider. However, this was deemed costly due to the extra circuitry since a 4-bit up-down counter and an extra bi-directional storage register would be required. The overall Division time would also be increased because at each point in the scaling, a shift, a subtraction and a test of the sign of the result would be required. In effect it is conceivable that the Division time could be doubled.

It is not unreasonable to assume, however, that the user will organize his calculations such that the numbers he uses will actually contain a Binary Point. In such a situation scaling could be reduced or even become unnecessary. It is left totally to the user's
discretion which method of number handling he will utilize. Sample data on some actual Multiplications and Divisions performed on the Multiplier/Divider are also given in the Appendix.

5.5 Final Conclusions and Suggestions.

The basic concepts used have been well proved and most of the required features and expected performance times, have been realized. It is expected that Multiplication will be the major operation for which the device will be used, and for this purpose the Multiplier is well adapted. Division may also be performed keeping in mind the limitations previously outlined.

One suggestion for the rescaling technique to be used, if a Hardware Scale/Rescale unit is further implemented for the Divider, would be to merely use the decrementing counter to shift MR and DR to the left one longer count as the actual rescaling count. MR should be initially cleared of the remainder.

If it is desired to apply the design presented here for constructing a Multiplier/Divider for another computer, not a PDP-8/S the performance times of the device and of the host computer should be kept in mind and the logic modules should be chosen accordingly. It should further be mentioned that the Multiplier/Divider can be used on other models of the PDP-8 computer line without alteration, especially where the EAE is not available. However, some of the logic as well as the positions in which the particular modules are located may be further optimized.
A.

Sample Calculations.

To completely understand the operation of the Multiplier/Divider, the step by step calculations shown below indicate what in effect takes place in the registers during each step of the algorithms. The two processes shown below are sufficient examples since the algorithms yield correct results regardless of the sign of either number being operated on. The numbers selected here for Multiplication were selected to show several of the types of cycles than can occur.

**Multiply**: \((28) \cdot (-7248)\). The result should be \(-16508\).

- **DR** = \(-7248 = 111\ 000\ 101\ 100\)
- **AR** = \(28 = 000\ 000\ 000\ 010\)
- **BR** = \(0 = 000\ 000\ 000\ 000\) (initially)

<table>
<thead>
<tr>
<th><strong>BR</strong></th>
<th><strong>DR</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>000\ 000\ 000\ 000</td>
<td>111\ 000\ 101\ 100\ 0</td>
</tr>
</tbody>
</table>

**Cycle 1.** Comparison 000, Fast Operation, 2 Post shifts

<table>
<thead>
<tr>
<th><strong>BR</strong></th>
<th><strong>DR</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>000\ 000\ 000\ 000</td>
<td>001\ 110\ 001\ 011\ 0</td>
</tr>
</tbody>
</table>

**Cycle 2.** Comparison 110, Subtract AR, 2 Post Shifts

<table>
<thead>
<tr>
<th><strong>BR</strong></th>
<th><strong>DR</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>111\ 111\ 111\ 110</td>
<td>100\ 011\ 100\ 010\ 1</td>
</tr>
</tbody>
</table>

**Cycle 3.** Comparison 101, Subtract AR, 2 Post shifts

<table>
<thead>
<tr>
<th><strong>BR</strong></th>
<th><strong>DR</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>111\ 111\ 111\ 110</td>
<td>011\ 000\ 111\ 000\ 1</td>
</tr>
</tbody>
</table>
Cycle 4. Comparison = 001, Add AR, 2 Post shifts

\[
\begin{align*}
\text{Add} & : 000 000 000 001 \\
\text{Shifts} & : 000 000 000 000 010 110 001 110 0
\end{align*}
\]

Cycle 5. Comparison = 100, Pre-shift, Subtract AR, Post Shift

\[
\begin{align*}
\text{Pre-shift} & : 000 000 000 000 001 000 111 0 \\
\text{Subtract} & : 111 111 111 110 \\
\text{Post Shift} & : 111 111 111 111 000 101 100 011 1
\end{align*}
\]

Cycle 6. Comparison = 111, 2 Post shifts

\[
\begin{align*}
\text{Shifts} & : 111 111 111 111 110 001 011 000 1
\end{align*}
\]

Octal Result is \(7777_{8} 6130_{8}\) which is a negative number.
Its equivalent is \(-0000_{8} 1650_{8}\), which is the correct result.

Here six out of eight possible types of cycles have been shown. The double length result may be extracted in two sections and converted to octal or decimal format via software.

For Division only two types of cycles exist, namely, one where Addition takes place and the second where Subtraction takes place, both occurring after a left shift. The Division process below gives an indication of both types of cycles.

\[
\begin{align*}
\text{Divide: } & \frac{4_{10}}{36_{10}}. \\
\text{DR} & = 0_{10} = 000 000 000 000 \\
\text{AR} & = 36_{10} = 000 000 100 100 \quad \text{and} \quad -\text{AR} = -36_{10} = 111 111 011 100 \\
\text{BR} & = 4_{10} = 000 000 000 100
\end{align*}
\]
Cycle 1. Comparison 00, Add 1 to DR, Shift, Subtract AR.
Add 1 to DR
Shift
Subtract

Cycle 2. Comparison 01, Add 0 to DR, Shift, Add AR.
Add 0 to DR
Shift
Add

Cycle 3. Comparison, same as cycle 2.
Add 0 to DR
Shift
Add

Cycle 4. Comparison, same as cycle 2.
Add 0 to DR
Shift
Add

Cycle 5. Comparison, same as cycle 1.
Add 1 to DR
Shift
Subtract 111 111 011 100
       000 000 010 100

Cycle 6. Comparison, same as cycle 1.
Add 1 to DR
Shift 000 000 101 000
       000 000 000 100

Cycle 7. Comparison, same as cycle 1.
Add 1 to DR
Shift 000 000 001 000
Subtract 111 111 011 100
       111 111 100 100

Cycle 8. Comparison 01, same as cycle 2.
Add 0 to DR
Shift 111 111 011 000
Add 000 000 100 100
       111 111 111 100

Cycle 9. Comparison, same as cycle 2.
Add 0 to DR
Shift 111 111 011 000
Add 000 000 100 100
       111 111 111 100

Cycle 10. Comparison, same as cycle 2.
Add 0 to DR
Shift 111 111 111 000
Add 000 000 100 100
       000 000 011 100

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Cycle 11. Comparison, same as cycle 1.

Add 1 to DR 010 001 110 001
Shift 000 000 111 000 100 011 100 010
Subtract 111 111 011 100 000 000 010 100

Cycle 12. Change sign of DR. 000 011 100 010

Therefore the Quotient becomes 0.00 011 100 010

It should be noted that the Quotient has the correct sign and magnitude. If desired the contents of BR could be extracted for examination to decide whether or not a round-off correction should be made.
B.

SAMPLE OUTPUT

MULTIPLICATION

\[
\begin{array}{l}
0001 \times 0002 = 0000 \hspace{1cm} 0002 \\
0002 \times 0002 = 0000 \hspace{1cm} 0004 \\
0004 \times 0004 = 0000 \hspace{1cm} 0020 \\
0010 \times 0010 = 0000 \hspace{1cm} 0100 \\
0020 \times 0020 = 0000 \hspace{1cm} 0400 \\
0040 \times 0040 = 0000 \hspace{1cm} 2000 \\
0100 \times 0100 = 0001 \hspace{1cm} 0000 \\
0200 \times 0200 = 0004 \hspace{1cm} 0000 \\
0400 \times 0400 = 0020 \hspace{1cm} 0000 \\
1000 \times 1000 = 0100 \hspace{1cm} 0000 \\
2000 \times 2000 = 0400 \hspace{1cm} 0000 \\
3777 \times 3777 = 1777 \hspace{1cm} 0001 \\
\end{array}
\]

\[
\begin{array}{l}
-0001 \times -0001 = 0000 \hspace{1cm} 0001 \\
-0002 \times -0002 = 0000 \hspace{1cm} 0004 \\
-0004 \times -0004 = 0000 \hspace{1cm} 0020 \\
-0010 \times -0010 = 0000 \hspace{1cm} 0100 \\
-0020 \times -0020 = 0000 \hspace{1cm} 0400 \\
-0040 \times -0040 = 0000 \hspace{1cm} 2000 \\
-0100 \times -0100 = 0001 \hspace{1cm} 0000 \\
-0200 \times -0200 = 0004 \hspace{1cm} 0000 \\
-0400 \times -0400 = 0020 \hspace{1cm} 0000 \\
-1000 \times -1000 = 0100 \hspace{1cm} 0000 \\
-2000 \times -2000 = 0400 \hspace{1cm} 0000 \\
-3777 \times -3777 = 1777 \hspace{1cm} 0001 \\
\end{array}
\]

\[
\begin{array}{l}
0001 \times -0001 = -0000 \hspace{1cm} 0001 \\
0002 \times -0002 = -0000 \hspace{1cm} 0004 \\
0004 \times -0004 = -0000 \hspace{1cm} 0020 \\
1000 \times -1000 = -0100 \hspace{1cm} 0000 \\
2000 \times -2000 = -0400 \hspace{1cm} 0000 \\
3777 \times -3777 = -1777 \hspace{1cm} 0001 \\
\end{array}
\]

\[
\begin{array}{l}
-0001 \times 0001 = -0000 \hspace{1cm} 0001 \\
-0002 \times 0002 = -0000 \hspace{1cm} 0004 \\
-0004 \times 0004 = -0000 \hspace{1cm} 0020 \\
-1000 \times 1000 = -0100 \hspace{1cm} 0000 \\
-2000 \times 2000 = -0400 \hspace{1cm} 0000 \\
-3777 \times 3777 = -1777 \hspace{1cm} 0001 \\
\end{array}
\]
DIVISION

0001 / 0002 = 2000
0001 / 0004 = 1000
0001 / 0010 = 0400
0001 / 0020 = 0200
0001 / 0040 = 0100
0001 / 0100 = 0040
0001 / 0200 = 0020
0001 / 0400 = 0010
0001 / 1000 = 0004
0001 / 2000 = 0002
0001 / 3777 = 0000

-0001 / -0002 = 1776
-0001 / -0004 = 0776
-0001 / -0010 = 0376
-0001 / -0020 = 0176
-0001 / -0040 = 0076
-0001 / -0100 = 0036
-0001 / -0200 = 0016
-0001 / -0400 = 0006
-0001 / -1000 = 0002
-0001 / -2000 = 0000
-0001 / -3777 = 0000

0001 / -0002 = -2001
0001 / -0004 = -1001
0001 / -0010 = -0401
0001 / -0020 = -0201
0001 / -0040 = -0101
0001 / -0100 = -0041
0001 / -0200 = -0021
0001 / -0400 = -0011
0001 / -1000 = -0005
0001 / -2000 = -0003
0001 / -3777 = -0001

-0001 / 0002 = -1777
-0001 / 0004 = -0777
-0001 / 0010 = -0377
-0001 / 0020 = -0177
-0001 / 0040 = -0077
-0001 / 0100 = -0037
-0001 / 0200 = -0017
-0001 / 0400 = -0007
-0001 / 1000 = -0003
-0001 / 2000 = -0001
-0001 / 3777 = -0001

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
REFERENCES


VITA AUCTORIS

1945  Born on April 28th, in Gyöngyöspata, Heves Megye, Hungary.

1960  Completed Elementary School Education at De La Salle Separate School, Windsor, Ontario, Canada.


1969  Graduated from the University of Windsor, Windsor, Ontario, Canada, with the Degree of B.A.Sc. in Electrical Engineering.

1971  Candidate for the Degree of M.A.Sc. in Electrical Engineering at the University of Windsor, Windsor, Ontario, Canada.