Research in digital computers takes two paths: one uses the computer as a tool to reach certain objectives; on the other, the computer itself is the object of research. The NEBULA computer was built with the latter in mind. The system design and logical design are described here in detail. NEBULA is operational and is fulfilling its objective as a research tool.
The Logical Design of the NEBULA Computer

by

John Amos Boles

A THESIS

submitted to

Oregon State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

June 1968
APPROVED:

Redacted for privacy
Professor of Mathematics in charge of major

Redacted for privacy
Chairman of Department of Mathematics

Redacted for privacy
Dean of Graduate School

Date thesis is presented November 8, 1967
Typed by Clover Redfern for John Amos Boles
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. INSTRUCTION FORMAT</td>
<td>4</td>
</tr>
<tr>
<td>III. DATA FORMAT</td>
<td>8</td>
</tr>
<tr>
<td>IV. ADDRESS CALCULATION</td>
<td>12</td>
</tr>
<tr>
<td>V. MEMORY AND REGISTER ORGANIZATION</td>
<td>15</td>
</tr>
<tr>
<td>VI. INSTRUCTION FETCH AND EFFECTIVE ADDRESS CALCULATION LOGIC</td>
<td>29</td>
</tr>
<tr>
<td>Line Select and Register Select</td>
<td>29</td>
</tr>
<tr>
<td>Instruction Fetch - (F_0)</td>
<td>31</td>
</tr>
<tr>
<td>The Index State - (F_1)</td>
<td>35</td>
</tr>
<tr>
<td>Indirect Addressing - (F_2)</td>
<td>36</td>
</tr>
<tr>
<td>Execute States</td>
<td>38</td>
</tr>
<tr>
<td>VII. REGISTER SPECIFYING COMMANDS</td>
<td>40</td>
</tr>
<tr>
<td>Operand Fetch</td>
<td>41</td>
</tr>
<tr>
<td>Arithmetic OP3 Commands</td>
<td>43</td>
</tr>
<tr>
<td>Add to Register</td>
<td>43</td>
</tr>
<tr>
<td>Absolute Value Register</td>
<td>44</td>
</tr>
<tr>
<td>Negate Register</td>
<td>46</td>
</tr>
<tr>
<td>Subtract from Register</td>
<td>48</td>
</tr>
<tr>
<td>Non-Arithmetic OP3 Commands</td>
<td>49</td>
</tr>
<tr>
<td>AND to Register</td>
<td>49</td>
</tr>
<tr>
<td>Exclusive OR to Register</td>
<td>50</td>
</tr>
<tr>
<td>OR to Register</td>
<td>51</td>
</tr>
<tr>
<td>Load Register</td>
<td>52</td>
</tr>
<tr>
<td>Register Recirculation for OP3 Commands</td>
<td>53</td>
</tr>
<tr>
<td>Other Register Specifying Commands</td>
<td>54</td>
</tr>
<tr>
<td>Compare Register</td>
<td>55</td>
</tr>
<tr>
<td>Exchange Register</td>
<td>58</td>
</tr>
<tr>
<td>Increment Index Register and Skip on Zero</td>
<td>60</td>
</tr>
<tr>
<td>Shift Logically</td>
<td>63</td>
</tr>
<tr>
<td>Shift Modified</td>
<td>67</td>
</tr>
<tr>
<td>Store Register</td>
<td>70</td>
</tr>
<tr>
<td>VIII. REGISTER NON-SPECIFYING COMMANDS</td>
<td>73</td>
</tr>
<tr>
<td>Arithmetic Commands</td>
<td>74</td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Divide</td>
<td>74</td>
</tr>
<tr>
<td>Multiply</td>
<td>84</td>
</tr>
<tr>
<td>Shift Commands</td>
<td>90</td>
</tr>
<tr>
<td>Arithmetic Shift Double</td>
<td>90</td>
</tr>
<tr>
<td>Normalize</td>
<td>96</td>
</tr>
<tr>
<td>Shift Double Length</td>
<td>101</td>
</tr>
<tr>
<td>Partial and Multiple Word Transfer Commands</td>
<td>103</td>
</tr>
<tr>
<td>Load Character</td>
<td>103</td>
</tr>
<tr>
<td>Load Half Word</td>
<td>107</td>
</tr>
<tr>
<td>Store Character</td>
<td>110</td>
</tr>
<tr>
<td>Store Half Word</td>
<td>112</td>
</tr>
<tr>
<td>Reset Flags</td>
<td>114</td>
</tr>
<tr>
<td>Set Flags</td>
<td>115</td>
</tr>
<tr>
<td>Store Double Length</td>
<td>116</td>
</tr>
<tr>
<td>Transfer E Register</td>
<td>118</td>
</tr>
<tr>
<td>Sequence Control Commands</td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>121</td>
</tr>
<tr>
<td>Jump</td>
<td>122</td>
</tr>
<tr>
<td>Jump and Store C</td>
<td>123</td>
</tr>
<tr>
<td>Load Left Half of C</td>
<td>124</td>
</tr>
<tr>
<td>No Operation</td>
<td>125</td>
</tr>
<tr>
<td>Skip if True</td>
<td>126</td>
</tr>
<tr>
<td>Skip if False</td>
<td>127</td>
</tr>
<tr>
<td>Interrupt</td>
<td></td>
</tr>
<tr>
<td>Reset Interrupt Mask</td>
<td>130</td>
</tr>
<tr>
<td>Set Interrupt Mask</td>
<td>131</td>
</tr>
<tr>
<td>Set Interrupt Enable Toggle</td>
<td>132</td>
</tr>
</tbody>
</table>

IX. DISPLAY LOGIC 134

X. SOFTWARE VERSUS HARDWARE 141

XI. LOGIC IMPLEMENTATION 146

BIBLIOGRAPHY 150
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Y counter and S counter state sequence table.</td>
<td>19</td>
</tr>
<tr>
<td>2. Slow counter state sequence table.</td>
<td>22</td>
</tr>
<tr>
<td>3. Register specifying commands and their codes.</td>
<td>40</td>
</tr>
<tr>
<td>4. Register non-specifying commands.</td>
<td>73</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Instruction format.</td>
<td>5</td>
</tr>
<tr>
<td>2. Bit numbering of a word.</td>
<td>10</td>
</tr>
<tr>
<td>3. Floating point format.</td>
<td>10</td>
</tr>
<tr>
<td>4. Memory and register scheme.</td>
<td>16</td>
</tr>
<tr>
<td>5. Delay line bit arrangement and $Q_1$ and $Q_2$ diagram.</td>
<td>21</td>
</tr>
<tr>
<td>6. Processor bit timing.</td>
<td>27</td>
</tr>
<tr>
<td>7. PCK and UPK example.</td>
<td>93</td>
</tr>
<tr>
<td>8. Partial word addressing scheme.</td>
<td>104</td>
</tr>
<tr>
<td>9. I/O skip flow chart.</td>
<td>120</td>
</tr>
<tr>
<td>10. Register display console.</td>
<td>134</td>
</tr>
<tr>
<td>11. Double precision data formats.</td>
<td>144</td>
</tr>
<tr>
<td>12. NAND gate.</td>
<td>146</td>
</tr>
</tbody>
</table>
THE LOGICAL DESIGN OF THE NEBULA COMPUTER

I. INTRODUCTION

In 1957 the Department of Mathematics at Oregon State University bought the first computer on campus, an ALWAC III-E. It proved to be a very durable machine and is still in full-time operation today. One of the features of this computer is that it has room for new machine language commands. This allows the computer to grow along with the development of more sophisticated software techniques. A real interplay has gone on between software and hardware.

Two computer design projects that developed at Oregon State University have resulted in part from an association with the ALWAC III-E computer. The first was the design of the GALAXY computer. The second was the NEBULA computer.

NEBULA had its beginning in September, 1963, in the form of a graduate seminar given by Dr. F.H. Young, Professor of Mathematics. The original purpose of this seminar was to investigate the systems design of a low cost digital computer which could be built at the University to provide a modern complement to the ALWAC III-E. The author of this thesis was given the responsibility of the logical design.

Research in computers takes two paths: one uses the computer as a tool to reach certain objectives; on the other, the computer itself
is the object of research. NEBULA was built with the latter in mind. Of course, the main purpose of the second path is to provide a better tool for the first.

The first research topics were:

1) to study the use of glass delay lines as computer memory
2) to design a computer that will be relatively easy to modify
3) to design a computer that will be relatively inexpensive yet.
   have many of the features of a large and very fast computer.

The general design parameters of NEBULA developed from the above topics to be as follows:

1) 34-bit word
2) one instruction per word
3) serial hardware
4) memory organized to provide an effective zero latency time
5) two's complement arithmetic.

Glass delay lines provided a capability of circulating information at the rate of 22.735 MHz. This speed coupled with the processor speed of 350 kHz provided the effective zero latency time. All the arithmetic and control registers were implemented entirely in flip-flops rather than delay lines to provide complete access to any bit at any time. The computer was provided with an initial set of instructions that would be adequate but by no means exhaustive, providing room for expansion.
The following chapters provide some insight into the concepts designed into NEBULA. This paper will also provide a complete documentation of the logical design of the NEBULA computer. Underlying everything is the hope that NEBULA will aid others in the field of computer design. It has already provided the impetus at Oregon State University for the design of an associative memory and a last-in-first-out type (push down list) memory. Both of these designs are attractive because of the high speed of the glass delay lines and the ease with which NEBULA may be modified to make use of them.
II. INSTRUCTION FORMAT

The instruction set of the computer was designed so as to have the features of a large and very fast computer but still to retain some simplicity for ease in implementation. NEBULA has one instruction per word and one basic instruction format.

The tag bits at each end of the NEBULA word have no effect on the instructions at present; therefore every instruction is 32 bits in length. A 16-bit address field appears in the left half of every instruction. The right 16 bits are broken into two fields of 8 bits. Bits 1 to 8 are referred to as the operation code.

The operation code was placed to the extreme right to allow one word time fetch and execution of certain instructions. NEBULA is a serial machine with the rightmost bits appearing from memory first. Since the operation code is read during the first 8 bit times of the instruction fetch cycle, the remainder of the word time can be used for decoding and executing the command. At present, execution is actually completed during the instruction fetch cycle for three instructions: 1) no operation, 2) unconditional jump (modifier field zero), and 3) execute (modifier field zero).

The operation codes are broken into two classes: register specifying and register non-specifying operations (Figure 1). Register specifying commands can specify any one of eight registers as the
Figure 1. Instruction format.
implied operand. Register non-specifying operations are fixed as to the register(s) used as the implied operand(s), if any.

The modifier field appears after the operation code, allowing the processor to decide what action to take before the address appears. This is necessary for 2) and 3) above to be one word time instructions. If index registers are separate from memory, it is possible with this instruction format to do the first level of indexing during the last half of the instruction fetch. This last feature has not been implemented on NEBULA yet because the index registers reside in memory.

Indexing and the addressing mode are specified by the modifier field. Bits 9 to 11 are unused at present. Immediate addressing is specified by bit 12, indirect addressing by bit 13. One of seven index registers can be specified by bits 14 to 16.

Some instructions do not need the full 16-bit address field, and some do not require an address field at all. For these two classes of instructions, it is possible to use the address for different purposes. NEBULA in one instance uses bit 32 of a shift command to indicate the direction of the shift.

The following features of the NEBULA instruction format are considered to emphasize the goals NEBULA was designed to meet:

1) An address of 16 bits allowing the computer to directly address 65,536 words of memory.
2) Instruction format fields that are of a length compatible with the byte manipulation instructions (character and half word).

3) Operation and modifier field in the right half of the word to utilize the serial organization.

4) A single format for simplicity in design and use.

5) An instruction format that includes almost all of the features commonly found in other computers and also allows room for expansion.
III. DATA FORMAT

NEBULA is a 34 bit, fixed word length computer. Two of these bits, one at each end of the word, are reserved for special uses and are called tags. Single precision data are therefore represented by 32-bit words. Bits in a word are numbered from right to left. The tag bits are numbered 0 and 33 and data or instruction bits, 1 to 32 (Figure 2).

There are four specific formats for data that can be handled most conveniently. These are: 1) single precision fixed point numbers, 2) single precision floating point numbers, 3) half word bytes, and 4) quarter word bytes.

The binary point for fixed point numbers will be considered to be between bits 31 and 32. Positive numbers are represented by a 0 in bit 32 and the magnitude of the number to the right of the binary point. Negative numbers are in two's complement form \(2 - x\), where \(x \geq 0\). Two's complement notation has very obvious advantages over sign and magnitude representation for addition and subtraction but is slightly more difficult for multiply and division algorithms (2, p. 42-43). One's complement notation is not good for a serial machine and is not good for multi-precision arithmetic. The number with a 1 bit in bit 32 and zero bits elsewhere will be treated in all cases as minus one.

Multi-precision fixed point number format is to some extent
determined by the multiply, add and subtract commands. In any fixed word length computer, someone will want more precision than provided by the word length of the computer. NEBULA has some special features to facilitate multi-precision programming. These will be discussed in Chapter X.

Floating point arithmetic is not presently a hardware capability in NEBULA. Commands have been provided to normalize and to pack and unpack the exponent part. These commands assume a certain format for floating point numbers. Floating point numbers are 32 bits in length. The sign bit of the fraction is bit 32 with bits 1 to 23 representing the fraction (Figure 3). The binary point is considered to be immediately to the left of bit 23. To provide the proper scaling, the fraction is considered to be multiplied by a power of two. The power of two is the exponent part. The exponent is in the excess 128 notation. A negative floating point number is the two's complement of the positive form as if it were a 32-bit fixed point number. Floating point 0 is the same as fixed point 0 (all 32 bits 0). This floating point format has the advantage that the fixed point instructions--negate, absolute value, and compare--can be used for floating point numbers as well as fixed point numbers.

Binary normalization is fixed by the normalized command. The exponent part may be varied to any size provided it does not exceed 31 bits. A more generalized format might be desirable, but it would
Figure 2. Bit numbering of a word.

Figure 3. Floating point format.
be expensive. One possibility would be to provide other than base

two normalization (i.e., hexadecimal), but this would require special
logic to provide \( n \) bit shift increments, a special most significant
\( n \) bit non-zero check, etc.

There is always the need for manipulating data that is not a full
word in length. NEBULA provides the capability to address memory
as 8-bit bytes and 16-bit bytes for loading and storing. Also the word
length of 32 (or any computer with a power of two word length, i.e.,
8, 16, 32, 64) has a nice feature for addressing at the bit level. This
may be accomplished by shifting the address 5 bits to the right (in
effect dividing by 32) to find the proper word. The remainder indi-
cates the bit in the word. For any other word length a divide is re-
quired.
IV. ADDRESS CALCULATION

NEBULA's effective address calculation is applied to all instructions except the no operation (NOP) instruction, which never uses an effective address. This universality economizes implementation and aids the programmer by providing simplicity.

Effective address calculation takes place during the index word time \((F_1)\), the indirect fetch word time \((F_2)\) and the first execute word time \((F_3)\). The calculation takes place after the instruction fetch word time \((F_0)\).

Each instruction (except NOP) is capable of being modified by one of seven index registers and can use indirect addressing. These two options are always the first things done after \(F_0\). The scheme works as follows:

1) If the index field is non-zero, add the contents of the specified index register to the address of the instruction.

2) If the indirect bit is 1, go to Step 3. If is is 0, the address modification is completed.

3) Use the address now in the instruction register to obtain a new address and new index and indirect field and go to Step 1.

The modification scheme just described can be called pre-indexing with multi-level indirect addressing. Some computers \((3)\) use a modification of the above scheme called post-indexing in which
the indirect addressing takes place first followed by index modification. It has been suggested that a future modification to NEBULA allow both types of modification.

The immediate bit fetched with the instruction is preserved throughout the indirect addressing (as are bits 9-11 of the modifier field). The immediate bit specifies that the effective address be used as an operand rather than the address of the operand. Some instructions implicitly use the effective address as an operand (i.e., jump, shift). For these instructions the immediate bit has no effect.

Immediate addressing is fairly common but is rarely used to its fullest extent. NEBULA presently uses the immediate operand as a right justified positive number (the upper 16 bits 0). A common variation is to sign extend the most significant digit of the address to provide positive and negative immediate operands.

A modification that seems to be very useful is to allow two modes of immediate addressing. The second mode would left justify the effective address (supplying 0's for the lower 16 bits). This mode has the following applications:

1) Immediate floating point constants, such as 1, 2, \(-\frac{1}{2}\).

2) Immediate binary fractions for the common fixed point scaling (1, 31). This is the scaling assumed by the normalize command.

3) Immediate constants to provide left justified integer
arithmetic necessary for NEBULA's instruction format.

The first two applications can be applied to computers other than NEBULA. For a one's complement machine, the lower half should be ones if the most significant bit is a one. A floating point format similar to NEBULA's is also assumed.
V. MEMORY AND REGISTER ORGANIZATION

All flip-flop registers are treated as a part of memory with the exception of the instruction register. NEBULA memory addresses 0000\textsubscript{16} to 0007\textsubscript{16} refer to the hardware flip-flop registers. Addresses 0008\textsubscript{16} to ffff\textsubscript{16} refer to glass delay line memory (Figure 4). The index registers reside in memory at locations 0009\textsubscript{16} to 000f\textsubscript{16}. Location 0008\textsubscript{16} is not an index register but can be used as a counter with the increment command (IXS).

At present seven of the eight possible hardware registers are implemented in NEBULA. The following is a list of the registers and their locations:

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Register Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Z, the zeros register (34 bits)</td>
</tr>
<tr>
<td>1</td>
<td>U, the universal accumulator (34 bits)</td>
</tr>
<tr>
<td>2</td>
<td>X, the extension of U (34 bits)</td>
</tr>
<tr>
<td>3</td>
<td>Unused</td>
</tr>
<tr>
<td>4</td>
<td>C, the control counter and flags (32 bits)</td>
</tr>
<tr>
<td>5</td>
<td>E, the character register (8 bits)</td>
</tr>
<tr>
<td>6</td>
<td>V, the operand register (34 bits)</td>
</tr>
<tr>
<td>7</td>
<td>W, the ones register (34 bits)</td>
</tr>
</tbody>
</table>

When memory locations 0000\textsubscript{16} to 0007\textsubscript{16} are referred to as addresses for instruction fetch and indirect addressing, only U, X
<table>
<thead>
<tr>
<th>Register Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Z register</td>
</tr>
<tr>
<td>0001</td>
<td>U register</td>
</tr>
<tr>
<td>0002</td>
<td>X register</td>
</tr>
<tr>
<td>0003</td>
<td>Third register</td>
</tr>
<tr>
<td>0004</td>
<td>C register</td>
</tr>
<tr>
<td>0005</td>
<td>E register</td>
</tr>
<tr>
<td>0006</td>
<td>V register</td>
</tr>
<tr>
<td>0007</td>
<td>W register</td>
</tr>
<tr>
<td>0008</td>
<td>Index register zero</td>
</tr>
<tr>
<td>0009</td>
<td>Index registers one - seven</td>
</tr>
<tr>
<td>000f</td>
<td>General memory</td>
</tr>
</tbody>
</table>

Figure 4. Memory and register scheme.
and V are used with their actual content. The other locations are treated as if they contained all 0's.

Glass delay line memory logic consists basically of shift counters to interface the high speed of the delay lines with the relative slow speed of the processor. The initial 4096 word memory for NEBULA uses 64 glass delay lines purchased from the Corning Glass Works. Each line has a delay of 100 μs and operates at 22.735 MHz. In effect, each line stores 2275 bits. An individual delay line is selected by bits 7 to 12 of the L register. An individual word in the delay line may be selected by bits 1 to 6 of the D register.

The selection and organization of words in a glass delay line need to be described in some detail.

Each delay line contains 64 words of 34 bits each, along with timing bits. The 2275 bits are divided up into 35 blocks of 65 bits each. Each block of 65 bits is called a macrobit, and the individual bits in each macrobit are called microbits. The macrobits are labeled $T_i; i = 0, \ldots, 34$, where $T_0$ is the first macrobit to appear in the 100 μs period.

The bits of any word in a delay line are interlaced with the bits of the other 63 words. Macrobit $T_0$ contains the tag S bits of each word and a timing bit called $t_x$ which appears last in the macrobit. The next macrobit, $T_1$, contains bit 1 of all the 64 words and a timing bit. $T_{34}$ is necessary to allow time for the selection of a
different word for the next word time. Suppose that the current strobe occurs at \( t_{63} \) (just before \( t_x \) at the end of a macrobit) and the next word selected requires a strobe at \( t_0 \) (the first micro-bit in a macrobit). If macrobit \( T_{34} \) did not separate the word times, the first bit time of the next word time would be shortened by about 1.5 \( \mu s \). This would leave no time for information to propagate through gates and be written into the delay line.

There are two identical 65 counters that operate at 22.735 MHz, the Y counter and the S counter. The Y counter counts the microbits in a macrobit and is called the microbit counter. The S counter selects a word by providing a strobe at the proper time in each \( T_i \). The S counter is called the bit-select counter. The bit-select counter has loading logic for an address during \( T_{34} \) time, but otherwise both counters are the same design. Each counter consists of seven flip-flops.

The Y counter flip-flops are labeled \( Y_i, i = 1, \ldots, 7 \).

**Y counter logic:**

\[
1 \cdot Y_1 = Y'_1 \cdot Y'_1
\]

\[
0 \cdot Y_1 = Y'_1 \cdot Y'_1 + Y'_1 \cdot Y_6 \cdot Y_5 \cdot Y_4 \cdot Y_3 \cdot Y_2
\]

\[
Y_{i+1} \leftarrow Y_i; \; i = 1, \ldots, 5
\]

\[
1 \cdot Y_7 = Y_6 \cdot Y'_5 \cdot Y_4 \cdot Y_3 \cdot Y_2 \cdot Y_1
\]

\[
0 \cdot Y_7 = Y_7 \cdot Y_6
\]
The logic for the Y counter gives the counter the sequence of states as illustrated in Table 1. The term \( Y_6 Y_7 \) gives a simple means of detecting the \( t_x \) time in each macrobit.

Table 1. Y counter and S counter state sequence table.

<table>
<thead>
<tr>
<th>State Number</th>
<th>Flip-Flop Number</th>
<th>State Number</th>
<th>Flip-Flop Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1</td>
<td>32</td>
<td>7 6 5 4 3 2 1</td>
<td>0 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1</td>
<td>33</td>
<td>0 1 0 1 1 0 0</td>
<td>0 0 1 1 0 0 0</td>
</tr>
<tr>
<td>1 0 1 1 1 1 0</td>
<td>34</td>
<td>0 1 1 0 0 0 0</td>
<td>35</td>
</tr>
<tr>
<td>2 0 1 1 1 0 0</td>
<td>35</td>
<td>0 1 0 0 0 1 1</td>
<td></td>
</tr>
<tr>
<td>3 0 1 1 0 0 0</td>
<td>36</td>
<td>0 1 0 0 1 1 1</td>
<td></td>
</tr>
<tr>
<td>4 0 1 0 0 0 0</td>
<td>37</td>
<td>0 0 0 0 1 1 1</td>
<td></td>
</tr>
<tr>
<td>5 0 0 0 0 0 0</td>
<td>38</td>
<td>0 0 1 1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>6 0 0 0 0 0 0</td>
<td>39</td>
<td>0 0 1 1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>7 0 0 0 0 0 0</td>
<td>40</td>
<td>0 1 1 1 0 1 0</td>
<td></td>
</tr>
<tr>
<td>8 0 0 0 0 0 1</td>
<td>41</td>
<td>0 1 1 0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>9 0 0 0 0 1 0</td>
<td>42</td>
<td>0 1 0 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>10 0 0 0 1 0 1</td>
<td>43</td>
<td>0 0 1 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>11 0 0 1 0 1 0</td>
<td>44</td>
<td>0 1 0 0 0 0 1</td>
<td></td>
</tr>
<tr>
<td>12 0 1 0 1 0 1</td>
<td>45</td>
<td>0 0 0 0 0 1 1</td>
<td></td>
</tr>
<tr>
<td>13 0 0 1 0 1 0</td>
<td>46</td>
<td>0 0 0 0 1 1 0</td>
<td></td>
</tr>
<tr>
<td>14 0 1 0 1 0 0</td>
<td>47</td>
<td>0 0 1 1 0 1 0</td>
<td></td>
</tr>
<tr>
<td>15 0 0 1 0 0 1</td>
<td>48</td>
<td>0 0 1 1 0 1 0</td>
<td></td>
</tr>
<tr>
<td>16 0 1 0 0 1 1</td>
<td>49</td>
<td>0 1 1 0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>17 0 0 0 1 1 0</td>
<td>50</td>
<td>0 1 0 1 0 1 1</td>
<td></td>
</tr>
<tr>
<td>18 0 0 1 1 0 0</td>
<td>51</td>
<td>0 0 1 0 1 1 1</td>
<td></td>
</tr>
<tr>
<td>19 0 1 1 0 0 1</td>
<td>52</td>
<td>0 1 0 1 1 1 0</td>
<td></td>
</tr>
<tr>
<td>20 0 1 0 0 1 0</td>
<td>53</td>
<td>0 0 1 1 1 0 0</td>
<td></td>
</tr>
<tr>
<td>21 0 0 0 1 0 0</td>
<td>54</td>
<td>0 1 1 1 0 0 1</td>
<td></td>
</tr>
<tr>
<td>22 0 0 1 0 0 0</td>
<td>55</td>
<td>0 1 1 0 0 1 1</td>
<td></td>
</tr>
<tr>
<td>23 0 1 0 0 0 1</td>
<td>56</td>
<td>0 1 0 0 1 1 1</td>
<td></td>
</tr>
<tr>
<td>24 0 0 0 0 1 0</td>
<td>57</td>
<td>0 0 0 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>25 0 0 0 1 0 0</td>
<td>58</td>
<td>0 0 1 1 1 1 0</td>
<td></td>
</tr>
<tr>
<td>26 0 0 1 0 0 1</td>
<td>59</td>
<td>0 1 1 1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>27 0 1 0 0 1 0</td>
<td>60</td>
<td>0 1 1 1 0 1 1</td>
<td></td>
</tr>
<tr>
<td>28 0 0 0 1 0 1</td>
<td>61</td>
<td>0 1 1 0 1 1 1</td>
<td></td>
</tr>
<tr>
<td>29 0 0 1 0 1 1</td>
<td>62</td>
<td>0 1 0 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>30 0 1 0 1 1 0</td>
<td>63</td>
<td>1 0 1 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>31 0 0 1 1 0 1</td>
<td>64</td>
<td>1 1 1 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>
The Y counter continually counts for every macrobit. This is not true for the bit-select counter. Instead of counting during macrobit time $T_{34}$, it is loaded with the address of the word to be selected. At the start of $T_0$, the S counter is free to count. The S counter will always be out of phase with the Y counter. The term $S_6S_7$ provides the timing or strobe that is necessary to pick the proper microbit out of each macrobit.

In addition to the S and Y counters, there is a 34-counter and two control flip-flops, $Q_1$ and $Q_2$. The 34-counter (also called the slow counter) runs at $1/65$ the speed of the S and Y counters. Its purpose as far as memory is concerned is to count macrobit times. The clock of the slow counter and the processor is the term $S_6S_7Q_2$ delayed 1.53 µs. At the start of $T_0$ both $Q_2$ and $Q_1$ are on. When the slow counter reaches $T_{33}$, both $Q_1$ and $Q_2$ are turned off when $S_6S_7 = 1$. $Q_1$ is turned on by the next $t_x$ time $(Y_6Y_7)$. The $t_x$ time provides at least one microbit time for $Q_1$ to be off, removing the possibility of a true input to both sides of $Q_1$. During the next $Y_6Y_7Q_1$ time, $Q_2$ is turned on, starting the next $T_0$ time (Figure 5).

The state sequence of the slow counter appears in Table 2. By building the proper term from the flip-flops of the slow counter, it is possible to indicate any macrobit time except $T_{34}$. The term $Q_1Q_2'$ indicates $T_{34}$. 
All $t_i$ from each macrobit form one word for the processor.

Figure 5. Delay line bit arrangement and $Q_1$ and $Q_2$ diagram.
### Table 2. Slow counter state sequence table.

<table>
<thead>
<tr>
<th>$T_0$</th>
<th>$P_5$</th>
<th>$P_4$</th>
<th>$P_3$</th>
<th>$P_2$</th>
<th>$P_1$</th>
<th>$T_s$</th>
<th>$T_p$</th>
<th>tag $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 1</td>
<td>operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 1 1 0 0</td>
<td>type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 0 0 0 1 1 0 0</td>
<td>register or operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 0 0 0 1 1 0 0</td>
<td>unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 0 0 1 0 1 1 0 0</td>
<td>immediate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 0 0 1 1 1 0 0</td>
<td>indirect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 0 0 1 1 1 0 0</td>
<td>index register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 0 0 1 1 0 0 0 0</td>
<td>address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 0 0 0 0 0 0 0 0</td>
<td>sign</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 0 1 0 0 1 1 0 0</td>
<td>tag $P$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 0 1 1 0 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 0 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 0 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 0 1 1 1 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 0 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 0 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 0 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 1 1 0 1 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 1 1 0 1 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19 1 1 0 1 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 1 1 1 0 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21 1 1 1 1 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22 1 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23 1 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24 1 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25 1 1 1 1 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26 1 1 1 1 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27 1 1 1 1 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28 1 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29 1 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30 1 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31 1 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 1 1 1 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33 0 0 0 0 0 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

During the strobe time, $S_6S_7Q_2$, the selected microbit is gated into a $2 \mu$s monostable multivibrator called $R$. It is the output of $R$ that provides the bits of the selected word to the processor.
The processor supplies bits to memory through the W line (the write bus). The strobe gates W into the delay line loop at the proper time.

The term \( T_s T_p \) is from the slow counter and is used to indicate \( T_{33} \). \( Q_1 \) and \( Q_2 \) are direct set and reset R-S flip-flops.

Logic for \( Q_1 \) and \( Q_2 \):

\[
\begin{align*}
1q_1 &= Y_6 Y_7 \\
0q_1 &= T_s T_s S_7 S_6 \\
1q_2 &= Q_1 Y_6 Y_7 \\
0q_2 &= T_s T_s S_7 S_7.
\end{align*}
\]

The slow counter flip-flops are labeled \( P_i \); \( i = 1, \ldots, 5 \) and \( T_s \) and \( T_p \). Terms such as \( (P'_3 P'_2 P'_1) \) appear in parentheses to indicate that they appear more than once in the slow counter logic.

Slow counter logic:

\[
\begin{align*}
1P_5 &= P_4 (P'_3 P'_2 P'_1) \\
0P_5 &= T_s T'_s P \\
1P_4 &= P'_4 (P'_3 P'_2 P'_1) T'_s T'_s \\
0P_4 &= P_5 P_4 (P'_3 P'_2 P'_1) \\
1P_3 &= P'_3 P'_2 P'_1 \\
0P_3 &= (P'_3 P'_2 P'_1) \\
1P_2 &= P_1 \\
0P_2 &= P'_1.
\end{align*}
\]
The S counter has a state sequence that is identical to the Y counter. The difference in logic comes from the loading of the S counter during \( Q' \). Loading of the S counter occurs from three different registers in the processor. Most of the time \( S_1 \) to \( S_6 \) are loaded from \( D_1 \) to \( D_6 \) of the address portion of the instruction register. The exceptions occur at 1) before \( F_1 \), \( S_1 \) to \( S_3 \) are loaded from \( M_6 \) to \( M_8 \), \( S_4 \) is set to a one, and \( S_5 \) to \( S_7 \) are zeroed; 2) before \( F_3 \) or \( F_4' \) during the execution of IXS (increment index and skip), \( S_1 \) to \( S_3 \) are loaded from \( O_6 \) to \( O_8 \), \( S_4 \) is set to a one and \( S_5 \) to \( S_7 \) are zeroed. For the special configuration 011111 in \( D_6 \) to \( D_1 \), \( S_7 \) is set if exceptions 1 and 2 above are not valid.
S counter logic:

\[ 1_s^1 = S'^1 S' Q_2 + (D_1 (F_1 + (F_3 + F_4))(IXS)) + M_6 F_1 + O_6 (F_3 + F_4)(IXS))Q_1^2 \]

\[ 0_s^1 = S'^1 S' S Q_2 + S S S S S S Q_2 + (D_1 (F_1 + (F_3 + F_4))(IXS)) \]

\[ + M_1 F_1 + O_1 (F_3 + F_4)(IXS))Q_1^2 \]

\[ 1_s^2 = S Q_2 + (D_2 (F_1 + (F_3 + F_4))(IXS)) + M_7 F_1 + O_7 (F_3 + F_4)(IXS))Q_1^2 \]

\[ 0_s^2 = S Q_2 + (D_2 (F_1 + (F_3 + F_4))(IXS)) + M_7 F_1 + O_7 (F_3 + F_4)(IXS))Q_1^2 \]

\[ 1_s^3 = S Q_2 + (D_2 (F_1 + (F_3 + F_4))(IXS)) + M_8 F_1 + O_8 (F_3 + F_4)(IXS))Q_1^2 \]

\[ 0_s^3 = S Q_2 + (D_2 (F_1 + (F_3 + F_4))(IXS)) + M_8 F_1 + O_8 (F_3 + F_4)(IXS))Q_1^2 \]

\[ 1_s^4 = S Q_2 + (D_4 (F_1 + (F_3 + F_4))(IXS)) + F_1 + (F_3 + F_4)(IXS))Q_1^2 \]

\[ 0_s^4 = S Q_2 + (D_4 (F_1 + (F_3 + F_4))(IXS)) + F_1 + (F_3 + F_4)(IXS))Q_1^2 \]

\[ 1_s^5 = S Q_2 + (D_5 (F_1 + (F_3 + F_4))(IXS))Q_1^2 \]

\[ 0_s^5 = S Q_2 + (D_5 (F_1 + (F_3 + F_4))(IXS))Q_1^2 \]

\[ 1_s^6 = S Q_2 + (D_6 (F_1 + (F_3 + F_4))(IXS))Q_1^2 \]

\[ 0_s^6 = S Q_2 + (D_6 (F_1 + (F_3 + F_4))(IXS))Q_1^2 \]

\[ 1_s^7 = S Q_2 + (D_1 (F_1 + (F_3 + F_4))(IXS))Q_1^2 \]

\[ 0_s^7 = S Q_2 + Q_1^2 \]

An individual line is selected by the 10-bit L register. Each delay line loop has a NAND gate that does a binary decode on the outputs from the L register.
As mentioned in the discussion of \( Q_1 \) and \( Q_2 \), the processor clock is delayed 1.53 \( \mu s \) after \( S_6 S_7 Q_2 \). In other words, the processor clock falls about halfway between two successive strobos of memory. This means that with a gate delay of 100 \( \mu s \), only ten levels of gating are allowed on the \( W \) input and ten levels of gates on the \( R \) output (Figure 6).

The system design and the logic for the counters and \( Q_1 \) and \( Q_2 \) of the delay line organization were done in the Mathematics seminar during the school year 1963-1964 conducted by F. H. Young. During the summer of 1964, Peter Rux did some initial electronics design and bread-boarding of the delay line loop and the \( S \) and \( Y \) counters. These were ordered and received in 1965. After receiving the delay lines, a prototype delay line was built and tested. In the summer of 1965, parts for the processor logic were ordered and gate and flip-flop card construction started. It was at this time that the slow counter was completed and tested. This was the first piece of processor logic built. Late in 1966 there were more than eight delay lines completed, and the processor was working well enough to test them. Many people had expressed doubts about the possibility of synchronizing two or more lines. This was not a problem, but the problem of shielding one line electronically from another did turn up at this time, along with a worst case bit configuration. These problems were soon solved by Peter Rux, and by April 1967 more
Figure 6. Processor bit timing.
than 16 lines were operating together. It should be remembered in the brief historical sketches that appear here that the work on both the design and the construction was done by graduate and undergraduate students, and because of this most of the work was done in the summers of 1964, 1965 and 1966.
VI. INSTRUCTION FETCH AND EFFECTIVE ADDRESS CALCULATION LOGIC

NEBULA has specific states that occur during the fetching and execution of an instruction. Each word time starts at the \( T' \) \( T' \) and ends at \( T' \) \( T' \) time. One or more word times comprise a state. The states are labeled \( F_i \), \( i = 0, \ldots, 9 \). The order in which the \( F \) states are executed is determined by each individual instruction. Instruction fetch occurs during \( F_0 \), indexing occurs during \( F_1 \) and indirect addressing occurs during \( F_2 \). For NEBULA, then, instruction fetch and effective address calculation take place during the word times labeled \( F_0 \), \( F_1 \) and \( F_2 \).

The following pages in this chapter describe the logic for these states in detail. The first logic that is described is for selecting the location of the instruction to be fetched.

Line Select and Register Select

NEBULA's 8 hardware registers are considered to be located in the first 8 locations in memory. For instruction fetch, operand fetch and store operations it is necessary to have logic to do the selection between glass delay line memory and hardware registers. When memory is selected it is necessary to select the proper glass delay line. The \( L \) register always either selects the proper line or provides the information to select the proper hardware register. The
H flip-flop determines whether to select memory or hardware registers.

The L register is 10 bits in length. Starting with the least significant end the numbering ranges from 7 to 16 corresponding to bits 7-16 of the D register (address field of the instruction register).

L register logic:

\[
l_i = (D_i H' + D_{i-6} H)(F_1' (F_3 + F_4')(IXS))Q'_2; \quad i = 7, 8, 9
\]

\[
l_i = (D_i H' + D_{i-6} H)(F_1' (F_3 + F_4')(IXS))Q'_2 + (F_1 + (F_3 + F_4)(IXS))Q_2
\]

\[i = 7, 8, 9\]

\[
L_i \leftarrow D_i; \quad i = 10, \ldots, 16 \text{ during } (F_1' (F_3 + F_4')(IXS))Q'_2
\]

For \(F_1\) (index word time) and \((F_3 + F_4')(IXS)\) the entire L register is set to 0, selecting delay line 0. The term \((IXS)\) is the increment index register and skip on 0 command. The index registers are memory locations \(9_{16}\) to \(f_{16}\). The states \(F_3\) and \(F_4\) are the execute states for IXS. If \(H\) is on, \(L_7\) to \(L_9\) copy the register address contained in \(D_1\) to \(D_3\). With \(H\) on, \(L_{10}\) to \(L_{16}\) have no significance. If \(H\) is off, bits \(L_7\) to \(L_{16}\) copy bits \(D_7\) to \(D_{16}\). The loading occurs between word times \((Q'_2)\) because the D register is static at this time. During \((T_s T_p)\), it is possible that the D register will be changing. This is why it is necessary to use the L register.
H logic:

\[ h = \left( \prod_{i=1}^{16} D_i \right) T_s T_p (IXS)_F 3 \]

\[ 0^h = \left( \sum_{i=1}^{16} D_i \right) T_s T_p (IXS)_F 3 + T'_s T_p F_3 (IXS) \]

If the contents of the D register are less than 000816, then \( H \) is turned on during the last bit time \( T_s T_p \) AND NOT (IXS)\( F_3 \).

Under any other conditions during \( T_s T_p \), \( H \) is turned off. For (IXS) AND \( F_3 \) it is necessary to insure that \( H \) is off so that line 0 is selected for the index registers.

**Instruction Fetch - F_0**

The instruction fetch state \( F_0 \) loads the instruction register with the next instruction. The instruction register is composed of three registers, the 16-bit D register (address field), the 8-bit M register (modifier field) and the 8-bit O register (operation code field). It should be noted that the O in the O register is the letter Oh.

During the first half of \( F_0 \) the control counter, bits 17 to 32 of the C register, is counted up by one if (EXT) is in the O register.

There are at present three commands that can be executed in \( F_0 \): 1) no operation (NOP); 2) execute (EXT) and 3) jump (JMP).

EXT and JMP must have their modifier fields \( (M_5 \text{ to } M_8) \) zero in order to be executed in \( F_0 \).

During the first quarter word time the O register is filled,
during the second quarter word time the M register is filled, and during
the last half word time the D register is filled. Depending on the
contents of the M and O registers, the next state may be \( F_0, F_1, \)
\( F_2, \) or \( F_3 \). The instruction will be coming from \( R \) if \( H \) is off
or from \( B \) if \( H \) is on. \( R \) contains serial by bit the word select-
ed from glass delay line memory. \( B \) contains serial by bit the word
selected from the registers.

B logic:

\[
B = (V_0(L_9L_8L_7') + U_0(L_9’L_8’L_7) + X_0(L_9’L_8L_7')F_0 \]
\[
V_{33} \leftarrow V_{0'}; V_{i+1} \leftarrow V_i; i = 0, \ldots, 32 \text{ during } F_0
\]
\[
U_{33} \leftarrow U_0'; U_{i+1} \leftarrow U_i; i = 0, \ldots, 32 \text{ during } F_0
\]
\[
X_{33} \leftarrow X_0'; X_{i+1} \leftarrow X_i; i = 0, \ldots, 32 \text{ during } F_0
\]

The \( V, U, \) and \( X \) registers completely recirculate during \( F_0 \).
The term \( B \) is a gate. Only the \( V, U, \) and \( X \) registers are gated
through \( B \). The other registers (\( C, E, Z, W \)) are treated as if they
contained zero.

O register load logic:

\[
O_8 \leftarrow (RH_1+BH); O_{i+1} \leftarrow O_i; i = 1, \ldots, 7 \text{ during } (P_5' P'_4 T'_1) F_0
\]

The \( O \) register copies \( R \) if \( H \) is off or \( B \) if \( H \) is on
for the first quarter word time \( (P_5' P'_4 T'_1) \).
M register logic:

\[ M_8 \leftarrow (RH' + BH); \quad M_i \leftarrow M_{i+1}; \quad i = 1, \ldots, 7 \text{ during } (P'_5 P_4)F_0 \]

The M register copies \( R \) is \( H \) is off or \( B \) if \( H \) is on for the second quarter word time \( (P'_5 P_4) \).

D register logic:

\[ D_{16} \leftarrow [(RH' + BH)(\overline{NOP}) + C_{17}(NOP)] \text{ during } P_5 F_0 \]

\[ D_i \leftarrow D_{i+1}; \quad i = 1, \ldots, 15 \text{ during } P_5 F_0 \]

The D register logic contains part of the execute logic for NOP as well as load logic. If the NOP command is in the O register during the last half \( (P_5) \) of \( F_0 \), the D register copies the upper half of the C register (bits 17 to 32, the control counter). This sets up the address for the next command, allowing NOP to be a one word time command. The contents of the M register have no effect on the execution of NOP. If NOP is not in the O register, the D register copies \( R \) if \( H \) is off or \( B \) if \( H \) is on.

K logic:

\[ k = T'_i T F_0 (\overline{EXT}) \]

\[ k = C'_i P'_i T F_0 + T_s T F_0 \]

The carry logic for the half add to the upper half of the C register is handled by \( K \). If NOT (EXT), \( K \) is turned on during the \( T_s \) bit time \( (T'_i T) \) and turned off during the first half word time \( P'_5 T'_p \) when the first zero in the control counter appears at \( C_{17'} \).
The upper half of the C register has the most varied logic that occurs in the \( F_0 \) state.

Control counter logic:

\[
C_{32} \leftarrow (C_{17}K' + C_{17}K); C_i \leftarrow C_{i+1}; i = 17, \ldots, 31 \text{ during } P'_5 T'_5 F_0
\]

For the first half word time \((P'_5 T'_5)\) of \( F_0'\), the control counter is counted up by one if \( K \) is on at the start of \((P'_5 T'_5)\).

The logic on \( C_{32} \) performs the required half add.

During the last half word time, the control counter either recirculates or copies the address from memory if an unmodified JMP is in the M and O registers.

Control counter logic:

\[
C_{32} \leftarrow [C_{17}((\text{JMP})+(M_8+M_7+M_6+M_5))+(\text{RH}'+\text{BH})(\text{JMP})(M'_8 M'_7 M'_6 M'_5)]
\]

during \( P_5 F_0 \)

\[
C_i \leftarrow C_{i+1}; i = 17, \ldots, 31 \text{ during } P_5 F_0
\]

The above logic for \( C_{17} \) to \( C_{32} \) provides the circulation of the control counter twice during \( F_0 \).

\( F_0 \) is turned off by the following logic:

\[
0^f_0 = (\text{NOP})(\text{JMP})(\text{EXT}) + (M_8 + M_7 + M_6 + M_5) T'_s T'_p F_0
\]

This logic turns \( F_0 \) off during the last bit time \((T'_s T'_p)\) only if the instruction fetched was not executed in \( F_0 \). NOP and unmodified JMP and EXT are executed during \( F_0 \). The logic to turn \( F_0 \) on occurs at the end of each instruction's logic section.
The contents of bits 5 to 7 of the M register determine the next state of NEBULA if $F_0$ is turned off. The possible new states are $F_1$, $F_2$ or $F_3$. For the logic that turns these states on, see their sections.

The Index State - $F_1$

The $F_1$ state follows $F_0$ if the contents of $M_6$ to $M_8$ are not zero and $F_0$ is turned off. The index state adds to the D register the address field of the word selected from memory locations $0009_{10}$ to $000f_{16}$. The word is selected by bits $M_6$ to $M_8$ of the M register. These words (locations $0009_{16}$ to $000f_{16}$) serve as index registers.

$F_1$ is turned on by the following logic:

$$f_1 = (M_8 + M_7 + M_6)[F_0(NO)P)((JMP)(EXT)+(M_6 + M_7 + M_6 + M_7)) + F_2]T_s T_p$$

If at the end of $F_0$, if $F_0$ is turned off and the contents of $M_6$ to $M_8$ are not zero, $F_1$ is turned on. If at the end of $F_2$ the contents of $M_6$ to $M_8$ are not zero, $F_1$ is also turned on.

Addition is performed by the adder $Q$. The inputs to the adder are $A$, $B$, and $K$. $K$ is the carry flip-flop. $A$, $B$, and $Q$ are gates.

A, B and Q logic:

$$A = D_1 P_{5} F_{1}$$
$$B = R P_{5} F_{1}$$
$$Q = A'B'K + A'BK' + AB'K' + ABK$$
D contains the address to be modified. The contents of the index register is coming from memory through \( R \). The output of \( Q \) provides the sum of \( A, B, \) and \( K \).

D register copies the sum from \( Q \):

\[
D_{16} \leftarrow Q; \quad D_i \leftarrow D_{i+1}; \quad i = 1, \ldots, 15 \text{ during } P_{5}F_{1}
\]

The carry logic is as follows:

\[
l_k = (ABG_1 + A'BG'_1)T'F_{p1}
\]

\[
o_k = (A'B'G_1 + AB'G'_1)T'F_{p1} + T_sT_pF_{p1}
\]

\[
l_{s1} = T'T_pF_{p1}
\]

Some of the logic for carry in \( F_1 \) is shared with the logic for the carry in other \( F \) states. For \( F_1 \) the flip-flop \( G_1 \) is turned on, enabling the add section of the carry. Since it is possible that the carry is on at the end of the add in \( F_1 \), it is necessary to turn it off at bit time \( T_sT_p \).

\[
F_1 \text{ is turned off at bit time } T_sT_p:
\]

\[
0'_{1} = T_sT_pF_{p1}
\]

**Indirect Addressing - \( F_2 \)**

The indirect addressing state \( F_2 \) is very similar to \( F_0 \) as far as the \( M \) and \( D \) registers are concerned. Bits \( M_5 \) to \( M_8 \) and all bits of the \( D \) register are replaced by the contents of the memory location specified by the \( D \) register at the start of \( F_2 \).
$F_2$ is turned on by the following logic:

$$f_2 = [(M_8'M_2'M_1'M_5)[F_0\overline{(NOP)} ((\overline{JMP})(EXT)+(M_8+M_7+M_6+M_5))]
+M_5']T_sT_p$$

$F_2$ follows $F_0$ if $F_0$ is turned off, the index field is zero, and the indirect bit is on. $F_2$ follows $F_1$ if the indirect bit is on.

As in $F_0'$, if $H$ is off, a word in the glass delay lines is selected; if $H$ is on, a word in the registers is selected and gated through $B$.

**B and register recirculation logic:**

$$B = (V_0(L_9L_8L_7') + U_0(L_9'L_8L_7) + X_0(L_9'L_8'L_7))F_2$$

$V_{33} \leftarrow V_0'; V_i \leftarrow V_{i+1}; i = 0, \ldots, 32$ during $F_2$

$U_{33} \leftarrow U_0'; U_i \leftarrow U_{i+1}; i = 0, \ldots, 32$ during $F_2$

$X_{33} \leftarrow X_0'; X_i \leftarrow X_{i+1}; i = 0, \ldots, 32$ during $F_2$

Only the $V$, $U$ and $X$ registers are gated through $B$. The other registers ($C$, $E$, $Z$, $W$) are treated as if they contained zero.

The $M$ register is partially refilled. Bits 1 to 4 stay the same.

Bits 5 to 8 copy the word selected.

**M register logic:**

$$M_8 \leftarrow (RH'+BH)$ during $P_5'P_4'(P_3P_2'+P_3P_1'+P_3P_1')F_2$

$$M_i \leftarrow M_{i+1}; i = 5, \ldots, 7$$ during $P_5'P_4'(P_3P_2'+P_3P_1'+P_3P_1')F_2$

The long term $P_5'P_4'(P_3P_2'+P_3P_1'+P_3P_1')$ is the timing for
the last half of the M register. This half copies R if H is off
or B if H is on.

D register logic:

\[ D_{16} \leftarrow (RHI + BH); \quad D_i \leftarrow D_{i+1}; \quad i = 1, \ldots, 15 \] during \( P_5 \)

The D register logic copies the word selected from memory.

Logic to turn \( F_2 \) off:

\[ 0_{F_2} = [(M_8 + M_7 + M_6) + (M_8' M_7' M_6' M_5')] T_s T_p F_2 \]

\( F_2 \) is turned off if the index field is not zero or if the index
field and the indirect bit are zero.

**Execute States**

All the presently implemented commands that are not executed
in \( F_0 \) enter \( F_3 \).

Logic to turn \( F_3 \) on:

\[ 1_{F_3} = [(M_8' M_7' M_6' M_5') \overline{(NOP)}(\overline{JMP}(\overline{EXT})+(M_8 + M_7 + M_6 + M_5))F_0 + F_2] \]

\[ + F_1' M_5' T_s T_p \]

\( F_3 \) is turned on at the end of \( F_0 \) if \( F_0 \) is turned off and
the contents of \( M_5 \) to \( M_8 \) are zero. \( F_3 \) is turned on at the end
of \( F_2 \) if the contents of \( M_5 \) to \( M_8 \) are zero. The turning on of
\( F_3 \) signifies the end of the effective address calculation.

For all commands, the states \( F_3 \) to \( F_9 \) are execute states.

Most commands like add and subtract use only \( F_3 \). The divide
command uses all the $F$ states from $F_3$ to $F_9$. If a command needs an operand from memory, it is fetched during $F_3$. Since the logic for the $F_3$ to $F_9$ states is usually specific for each command, the logic description appears in the section describing each command.
VII. REGISTER SPECIFYING COMMANDS

NEBULA's commands are divided into two groups: register specifying and register non-specifying commands. The register specifying commands can specify any one of eight possible registers as one of the implied operands.

If bit 4 or 5 of the O register is non-zero then the command is register specifying. Bits 6, 7 and 8 of the O register determine the register to be used. With the format just described there are 24 possible register specifying commands. There are 14 of these implemented at this time (Table 3). Each section describing logic that is specifically for one command will also describe the function of that command in two ways. The first way will be an informal algorithmic description and the second way will be by means of a paragraph.

Following these descriptions will be the detailed logic.

Table 3. Register specifying commands and their codes.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Command</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADR</td>
<td>add</td>
<td>13</td>
</tr>
<tr>
<td>ANR</td>
<td>and</td>
<td>17</td>
</tr>
<tr>
<td>AVR</td>
<td>absolute value</td>
<td>1d</td>
</tr>
<tr>
<td>CMR</td>
<td>compare</td>
<td>18</td>
</tr>
<tr>
<td>EOR</td>
<td>exclusive-or</td>
<td>16</td>
</tr>
<tr>
<td>EXR</td>
<td>exchange</td>
<td>1c</td>
</tr>
<tr>
<td>IOR</td>
<td>inclusive-or</td>
<td>15</td>
</tr>
<tr>
<td>IXS</td>
<td>increment index and skip</td>
<td>1f</td>
</tr>
<tr>
<td>LDR</td>
<td>load</td>
<td>11</td>
</tr>
<tr>
<td>NGR</td>
<td>negate</td>
<td>1e</td>
</tr>
<tr>
<td>SBR</td>
<td>subtract</td>
<td>14</td>
</tr>
<tr>
<td>SLR</td>
<td>shift logically</td>
<td>19</td>
</tr>
<tr>
<td>SMR</td>
<td>shift modified</td>
<td>1a</td>
</tr>
<tr>
<td>STR</td>
<td>store</td>
<td>12</td>
</tr>
</tbody>
</table>
Operand Fetch

The first logic that is described is for the most part independent of the command being executed.

B logic selects the operand from a memory location or register that is specified in the address field of the instruction register. This selection occurs regardless of the command being executed.

B logic for $F_3$:

$$B = \{ [U_0(L_9L_8L_7) + X_0(L_9L_8L_7) + C_1(L_9L_8L_7)$$

$$+ E_1(L_9L_8L_7)(P_5P_4T') + V_0(L_9L_8L_7) + (L_9L_8L_7)]H$$

$$+ RH]F_3M_4 + D_4M_4P_5T'F_3 \}

B is a gate. As described before, when H is on, the address of the register selected is contained in $L_7$ to $L_9$. Register Z and the unused third register do not appear and hence are treated as if they contained zero. Right shift logic for the registers must be provided by the appropriate command(s). The non-existent bits, 0 and 33 of the C register, will appear to have the contents of bits $C_1$ and $C_{32}$ respectively. The term $P_5P_4T'$ that is ANDed with $E_1$ causes the non-existent parts of the E register (bits 0 and 9 to 33) to appear as if they contained zeros.

If H is off, the operand comes from delay line memory through R. If $M_4$ is on, the operand comes from the D register. $M_4$ is the immediate bit and D is treated as bits 1 to 16 of a 34
bit operand. Bits 0, and 17 to 33 of the 34-bit immediate operand appear to contain zeros.

The logic on the \( A \) gate selects the register designated by bits 6 to 8 of the O register.

A logic for \( F_3 \):

\[
A = \left[ U_0(0'8'7'6') + X_0(0'8'7'6') + C_1(080706) \right. \\
\left. + E_1(080706)(P'0'180706') + V_0(080706') \right] F_3 (\text{IXS})
\]

IXS is the only command that uses \( A \) in a different manner during \( F_3 \). The register selection logic in \( A \) is identical to the register selection logic of \( B \) except that \( O_6 \) to \( O_8 \) take the place of \( L_7 \) to \( L_9 \).

D register recirculation:

\[
D_{16} \leftarrow D_1; D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \text{ during } P'T F_3 (\text{SMR})(\text{SLR})(\text{ASD})
\]

\[
(\text{SFD})(\text{TRE})
\]

The recirculation of \( D \) during the first half word time in \( F_3 \) provides the right justified immediate operand specified by \( M_4 \) on. For the commands \( \text{SMR}, \text{SLR}, \text{ASD}, \text{SFD}, \) and \( \text{TRE} \), the \( D \) register must not recirculate.

The following group of commands that are described have common recirculation logic for the registers. The recirculation logic will be described after the commands. These commands: \( \text{ADR}, \text{SBR}, \text{ANR}, \text{IOR}, \text{FOR}, \text{LDR}, \text{NGR}, \) and \( \text{AVR} \) are called OP3 commands.
Add to Register

\[ C(R) := C(R) + C(A) \]

if \( C(R) \geq 1 \) \( VC(R) < -1 \)

then \( C(C)_3 := 1 \)

The contents of the effective address are added to the specified register. Overflow will set \( C_3 \) and the register will contain the 32 least significant bits of the sum. Add to register \( Z \), three or \( W \) is treated as no operation.

Execution time: 100 \( \mu s \)

The add command uses the adder gate \( Q \) for the sum to be gated into the specified register.

\[ Q = A'B'K + A'BK' + AB'K' + ABK \]

The logic for \( A \) and \( B \) is given in the selection logic above.

Carry logic:

\[ k = ABG_1 F_3 (\overline{FOR})(\overline{TRE})(\overline{STD})(\overline{SKF})(\overline{SKT}) \]
\[ 0^k = A'B'G_1 F_3 (\overline{FOR})(\overline{TRE})(\overline{STD})(\overline{SKF})(\overline{SKT}) + T_s T_p F_3 \]
\[ 1^g_1 = T_s T_p (ADR)F_3 \]
\[ 0^g_1 = T_s T_p (DIV)(MUL) \]
G₁ is turned on at the bit time \( T_s T_p \) if ADR is on. G₁ on indicates that the logic on K will cause it to perform the function of carry.

Overflow:

\[ c_3 = (A\bar{B}Q' + A'B'O)T_s T_p (ADR)F_3 \]

During the sign bit time \( (T_s T_p') \) the sign of the result is checked against the sign of the operands; if there is disagreement in the case where the signs of the operands are alike, \( C_3 \) is turned on.

Sum gating:

\[ U_{33} \leftarrow Q \text{ during } (O' 0' 0') (ADR)F_3 \]
\[ X_{33} \leftarrow Q \text{ during } (O' 0' 0') (ADR)F_3 \]
\[ V_{33} \leftarrow Q \text{ during } (O' 0' 0') (ADR)F_3 \]
\[ C_{32} \leftarrow Q \text{ during } (O' 0' 0') T_p (ADR)F_3 \]
\[ E_8 \leftarrow Q \text{ during } (O' 0' 0') P_5 P_4 T_p (ADR)F_3 \]

The gating of the sum is determined by bits \( O_6 \) to \( O_8 \) of the O register. Since the C and E registers are not full 34-bit registers, they have 32-bit \( T_p' \) and 8-bit \( P_5 P_4 T_p' \) timing respectively.

**Absolute Value Register**

If \( C(R) = -1 \) then \( C(C)_3 := 1 \)

\[ C(R) := |C(R)| \]

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVU</td>
<td>3d</td>
</tr>
<tr>
<td>AVX</td>
<td>5d</td>
</tr>
<tr>
<td>AVV</td>
<td>dd</td>
</tr>
</tbody>
</table>
Overflow will set $C_3$. The contents of the specified register are replaced by the absolute value of the specified register. Immediate mode has no effect and the effective address is not used. If register $Z$, three, $C$, $E$ or $W$ is selected, the effect is no operation.

Execution time: 100 μs

$G_3$ logic:

\[ 1^g_3 = \left[ U_{32} \left( O_8' O_7' O_6 \right) + X_{32} \left( O_8' O_7 O_6' \right) + V_{32} \left( O_8 O_7' O_6' \right) \right] T_s T_p (AVR) F_3 \]

\[ 0^g_3 = T_s T_p F_3 \]

$G_3$ on indicates that a negative number is in the selected register and its complement must be taken for the AVR command.

$G_1$ logic:

\[ 1^g_1 = A(G_3 T'_p) (AVR) F_3 \]

\[ 0^g_1 = T_s T_p (DIV)(MUL) \]

The flip-flop $G_1$ indicates that the first 1 bit has been encountered and the remaining bits are to be reversed ($2'$s complement).

Absolute value gating:

\[ U_{33} \leftarrow (A'G_1 + AG_1') \text{ during } (O_8' O_7' O_6) (AVR) F_3 \]

\[ X_{33} \leftarrow (A'G_1 + AG_1') \text{ during } (O_8' O_7 O_6') (AVR) F_3 \]

\[ C_{32} \leftarrow C_1 \text{ during } (O_8 O_7' O_6') T'_p (AVR) F_3 \]

\[ E_8 \leftarrow E_1 \text{ during } (O_8 O_7' O_6') P'_5 P'_4 T'_p (AVR) F_3 \]

\[ V_{33} \leftarrow (A'G_1 + AG_1') \text{ during } (O_8 O_7 O_6') (AVR) F_3 \]
The gating of the absolute value is determined by bits $O_6$ to $O_8$ of the O register. Since the E register is always positive, no absolute value is taken. Both the E and C registers have 8 bit ($P'_5 P'_4 T'_3$) and 32 bit ($T'_p$) timing respectively.

Overflow logic:

$$c_3 = AG'T T'(AVR)F_3$$

Bit $c_3$ is turned on if at sign bit time $(T_s T'_p)$ the selected register contains a one and $G_1$ is still off, indicating a minus one in the register.

**Negate Register**

If $C(R) = -1$ then $C(C)_3 := 1$

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NGU</td>
<td>3e</td>
</tr>
<tr>
<td>NGX</td>
<td>5e</td>
</tr>
<tr>
<td>NGV</td>
<td>de</td>
</tr>
</tbody>
</table>

Overflow will set $c_3$. The contents of the specified register are replaced by the two's complement of the specified register. Immediate mode has no effect and the effective address is not used. If register Z, three, C, E or W is selected, the effect is no operation.

Execution time: $100 \mu s$

The logic of NGR is almost identical to the logic of AVR. The difference is that $G_3$ is not needed.
GI logic:

\[ 1^g_1 = A(T'_p F'_3)(NGR) \]

The first one bit that comes through \( A \) from the selected register turns \( G_1 \) on, indicating that the remaining bits are to be reversed (2's complement).

Two's complement gating:

\[
\begin{align*}
U_{33} &\leftarrow (A'G_1 + AG'_1) \text{ during } (O'_8 O'_7 O'_6)(NGR)F_3 \\
X_{33} &\leftarrow (A'G_1 + AG'_1) \text{ during } (O'_8 O'_7 O'_1)(NGR)F_3 \\
V_{33} &\leftarrow (A'G_1 + AG'_1) \text{ during } (O'_8 O'_7 O'_1)(NGR)F_3 \\
C_{32} &\leftarrow C_1 \text{ during } (O_8 O'_7 O'_6 T'_p)(NGR)F_3 \\
E_8 &\leftarrow E_1 \text{ during } (O_8 O'_7 O'_6 P'_5 P'_4 T'_p)(NGR)F_3
\end{align*}
\]

The two's complement gating is determined by bits \( O_6 \) to \( O_8 \) of the \( O \) register. The \( E \) register just recirculates since it is always positive. The \( E \) register has 8-bit timing \( (P'_5 P'_4 T'_p) \) and the \( C \) register has 32-bit timing \( (T'_p) \).

Overflow logic:

\[ 1^c_3 = AG'T T'_p (NGR)F_3 \]

If \( G_1 \) is off and \( A \) is on at sign bit time \( (T'_s T'_p) \) the selected register contains minus one, causing an overflow and turning \( C_3 \) on.
Subtract from Register

\[ C(R) := C(R) - C(A) \]

If \( C(R) \geq 1 \) or \( C(R) < -1 \)
then \( C(C)_3 := 1 \)

The contents of the effective address are subtracted from the specified register. Overflow will set \( C_3 \) and the register will contain the least significant bits of the difference. Subtraction from register 'Z', three, or 'W' is treated as no operation.

Execution time: 100 \( \mu s \)

The subtract command uses the adder \( Q \) for the difference to be gated into the specified register. The logic for the adder is given in the description of ADR. The logic that causes the adder to perform subtraction is in the carry, called the borrow here.

Borrow logic:

\[ 1^k = A'BG_1 F_3 (EOR)(TRE)(STD)(SKF)(SKT) \]

\[ 0^k = A'B'G_1 F_3 (EOR)(TRE)(STD)(SKF)(SKT) \]

\( G_1 \) is normally off in \( F_3 \) except for ADR. \( G_1 \) off causes \( K \) to act as a borrow. The term \( \overline{(EOR)} \) is necessary for the exclusive-or command uses the adder \( Q \) with the \( K \) term zero.

Overflow:

\[ 1^c_3 = (AB'Q' + A'BQ)T_3 P^T (SBR)F_3 \]
During the sign bit time \((T_s, T'_p)\) the sign of the result is checked against the sign of the operands; if the result has the wrong sign, \(C_3\) is turned on.

**Difference gating:**

\[
U_{33} \leftarrow Q \text{ during } (O_8' O_7' O_6')(SBR)F_3
\]

\[
X_{33} \leftarrow Q \text{ during } (O_8' O_7' O_6')(SBR)F_3
\]

\[
V_{33} \leftarrow Q \text{ during } (O_8 O_7' O_6')(SBR)F_3
\]

\[
C_{32} \leftarrow Q \text{ during } (O_8' O_7' O_6')T'_p(SBR)F_3
\]

\[
E_8 \leftarrow Q \text{ during } (O_8 O_7' O_6')P'_5 P'_4 T'_p(SBR)F_3
\]

The gating of the difference is determined by bits \(O_6\) to \(O_8\) of the \(O\) register. Since the \(C\) and \(E\) registers are not full 34-bit registers, they have 23-bit \((T'_p)\) and 8-bit \((P'_5 P'_4 T'_p)\) timing respectively.

**Non-Arithmetic OP3 Commands**

**AND to Register**

\[
C(R) := C(R) \land C(A)
\]

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANU</td>
<td>37</td>
</tr>
<tr>
<td>ANX</td>
<td>57</td>
</tr>
<tr>
<td>ANE</td>
<td>b7</td>
</tr>
<tr>
<td>ANV</td>
<td>d7</td>
</tr>
</tbody>
</table>

The logical product (bit by bit) of the contents of the effective address and the specified register is placed in the specified register.
And to register Z, three, C, or W is treated as no operation.

Execution time: 100 μs

The logical product is formed by gating the AND of operand gates A and B to the selected register.

Logical product gating:

\[ U_{33} \leftarrow AB \text{ during } (0'0'O'O')(ANR)F_3 \]

\[ X_{33} \leftarrow AB \text{ during } (0'O'O'O')(ANR)F_3 \]

\[ V_{33} \leftarrow AB \text{ during } (0'O'O'O')(ANR)F_3 \]

\[ C_{32} \leftarrow C_1 \text{ during } (0'O'O'O')T'(ANR)F_3 \]

\[ E_8 \leftarrow AB \text{ during } (0'O'O'O')P'P'T'(ANR)F_3 \]

**Exclusive OR to Register**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>C(R) := C(R) ( \oplus ) C(A)</td>
<td>Code Code</td>
</tr>
<tr>
<td>EOU</td>
<td>36</td>
</tr>
<tr>
<td>EOX</td>
<td>56</td>
</tr>
<tr>
<td>EOC</td>
<td>96</td>
</tr>
<tr>
<td>EOE</td>
<td>b6</td>
</tr>
<tr>
<td>EOV</td>
<td>d6</td>
</tr>
</tbody>
</table>

The modulo-two sum of the contents of the effective address and the specified register is placed into the specified register. EOR to register A, three or W is treated as no operation.

Execution time: 100 μs

The exclusive-or function is formed by gating the Q to the selected register with the K term (carry) turned off.
Exclusive OR gating:

\[ U_{33} \leftarrow Q \text{ during } (O'_{8}O'_{7}O'_{6})(EOR)F_{3} \]

\[ X_{33} \leftarrow Q \text{ during } (O'_{8}O_{7}O'_{6})(EOR)F_{3} \]

\[ V_{33} \leftarrow Q \text{ during } (O_{8}O'_{7}O'_{6})(EOR)F_{3} \]

\[ C_{32} \leftarrow Q \text{ during } (O_{8}O'_{7}O'_{6})T'(EOR)F_{3} \]

\[ E_{8} \leftarrow Q \text{ during } (O_{8}O'_{7}O'_{6})P'P'T'(EOR)F_{3} \]

**OR to Register**

\[ C(R) := C(R)VC(A) \]

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORU</td>
<td>35</td>
</tr>
<tr>
<td>ORX</td>
<td>55</td>
</tr>
<tr>
<td>ORE</td>
<td>b5</td>
</tr>
<tr>
<td>ORV</td>
<td>d5</td>
</tr>
</tbody>
</table>

The logical sum of the contents of the effective address and the specified register is placed in the specified register. Or to register \( Z \), three, \( C \) or \( W \) is treated as no operation.

Execution time: 100 µs

The logical sum is formed by gating the OR of operand gates \( A \) and \( B \) to the selected register.

**Logical sum gating:**

\[ U_{33} \leftarrow (A+B) \text{ during } (O'_{8}O'_{7}O'_{6})(IOR)F_{3} \]

\[ X_{33} \leftarrow (A+B) \text{ during } (O'_{8}O_{7}O'_{6})(IOR)F_{3} \]

\[ V_{33} \leftarrow (A+B) \text{ during } (O_{8}O'_{7}O'_{6})(IOR)F_{3} \]
\[ C_{32} \leftarrow C_1 \text{ during } (O_8 O_7 O'_6)T'_1(\text{IOR})F_3 \]

\[ E_8 \leftarrow (A+B) \text{ during } (O_8 O'_7 O'_6)P'_4 P'_4 T'_1(\text{IOR})F_3 \]

**Load Register**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDU</td>
<td>31</td>
</tr>
<tr>
<td>LDX</td>
<td>51</td>
</tr>
<tr>
<td>LDC</td>
<td>91</td>
</tr>
<tr>
<td>LDE</td>
<td>b1</td>
</tr>
<tr>
<td>LDV</td>
<td>d1</td>
</tr>
</tbody>
</table>

The contents of the effective address are loaded into the specified register. If register Z, three or W is selected, the effect is no operation.

**Execution time:** 100 μs

The operand specified by the effective address is gated through B into the register specified by bits 6 to 8 of the O register.

**Load register gating:**

\[ U_{33} \leftarrow B \text{ during } (O'_8 O'_7 O'_6)(\text{LDR})F_3 \]

\[ X_{33} \leftarrow B \text{ during } (O'_8 O'_7 O'_6)(\text{LDR})F_3 \]

\[ V_{33} \leftarrow B \text{ during } (O_8 O_7 O'_6)(\text{LDR})F_3 \]

\[ C_{32} \leftarrow B \text{ during } (O_8 O'_7 O'_6)T'_1(\text{LDR})F_3 \]

\[ E_8 \leftarrow B \text{ during } (O_8 O'_7 O'_6)P'_4 P'_4 T'_1(\text{LDR})F_3 \]
Register Recirculation for OP3 Commands

The logic in this section describes the recirculation logic for the registers not selected. This logic is for the OP3 commands only.

Copy logic:

\[
U_{33} \leftarrow U_0 \text{ during } (O_8 + O_7 + O') (OP3) F_3
\]

\[
X_{33} \leftarrow U_0 \text{ during } (O_8 + O_7 + O') (OP3) F_3
\]

\[
V_{33} \leftarrow V_0 \text{ during } (O' + O_7 + O') (OP3) F_3
\]

\[
C_{32} \leftarrow C_1 \text{ during } (O' + O_7 + O') T' (OP3) F_3
\]

\[
E_8 \leftarrow E_1 \text{ during } (O' + O_7 + O') (P'_5 P'_4 T') (OP3) F_3
\]

Shift logic:

\[
U_i \leftarrow U_{i+1}; \quad i = 0, \ldots, 32 \text{ (OP3) } F_3
\]

\[
X_i \leftarrow X_{i+1}; \quad i = 0, \ldots, 32 \text{ during (OP3) } F_3
\]

\[
V_i \leftarrow V_{i+1}; \quad i = 0, \ldots, 32 \text{ during (OP3) } F_3
\]

\[
C_i \leftarrow C_{i+1}; \quad i = 1, \ldots, 31 \text{ during (OP3) } T' F_3
\]

\[
E_i \leftarrow E_{i+1}; \quad i = 1, \ldots, 7 \text{ during (OP3) } P'_5 P'_4 T' F_3
\]

The registers must recirculate for OP3 commands so that they may be used as an operand for register to register operations.

OP3 gate logic:

\[
OP3 = [(ADR)+(SBR)+(ANR)+(IOR)+(EOR)+(LDR)+(NGR)+(AVR)] F_3
\]
Since much of the logic of the OP3 commands is identical, they were Ored together and given their special name.

**OP3 D register logic:**

\[ D_{16} \leftarrow C_{32}; \ D_i \leftarrow D_{i+1}; \ i = 1, \ldots, 15 \text{ during} \]

\[ [P_5(P_2 + P_3 + P_1' + P_4') + T_s](OP3)F_3 \]

The D register needs to copy \( C_{17} \) to \( C_{32} \) in \( F_3 \) in order to gate the next instruction in \( F_0 \). Since the C register may change during OP3 commands, it was necessary to copy from \( C_{32} \) for the last 16 bit times instead of the \( C_1 \) during \( P_5 \).

**F logic:**

\[ f_3 = T_s T_p(OP3)F_3 \]

\[ f_0 = T_s T_p(OP3)F_3 \]

All OP3 commands are finished in \( F_3 \); then control goes to \( F_0 \) to fetch the new instruction.

**Other Register Specifying Commands**

Since the remainder of the register specifying commands have no special groupings, recirculation for registers not selected is described with each command.
**Compare Register**

If $C(R) > C(A)$ then $C(C)_2 := 0 \land C(C)_1 := 1$

If $C(R) = C(A)$ then $C(C)_2 := 0 \land C(C)_1 := 0$

If $C(R) < C(A)$ then $C(C)_2 := 1 \land C(C)_1 := 0$

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMZ</td>
<td>18</td>
</tr>
<tr>
<td>CMU</td>
<td>38</td>
</tr>
<tr>
<td>CMX</td>
<td>58</td>
</tr>
<tr>
<td>CMC</td>
<td>98</td>
</tr>
<tr>
<td>CME</td>
<td>b8</td>
</tr>
<tr>
<td>CMV</td>
<td>d8</td>
</tr>
<tr>
<td>CMW</td>
<td>f8</td>
</tr>
</tbody>
</table>

The contents of the effective address and the specified register are compared. $C_1$ and $C_2$ indicate the result.

**Execution time:** 100 μs

The compare command uses the adder $Q$ with $K$ acting as a borrow to compare two words. Since the $A$ and $B$ gates are copying the selected operands, the difference, $A-B$, is coming through $Q$.

**G₁ logic:**

$$0^g_1 = T'_s T'_p (CMR)_3 F_3$$

To ensure that $K$ acts as a borrow, $G_1$ is turned off at the first bit time of $F_3$.

**J₁ logic:**

$$1^j_1 = Q(CMR)_3 T'_p$$

$$0^j_1 = T'_s T'_p (CMR)_3$$

$J_1$ is used to check for equality. During the first bit time $(T'_s T'_p)$, $J_1$ is turned off. If at any time during the next 32 bit times...
(T'_p), a one comes through Q, J_1 is turned on. Therefore, J_1 off at the sign bit time indicates that the operands are equal if the sign bits agree. C_1 and C_2 are used to indicate the ordering of the operands.

<table>
<thead>
<tr>
<th>C_2</th>
<th>C_1</th>
<th>Operand Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A = B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>A &gt; B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>A &lt; B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>not used</td>
</tr>
</tbody>
</table>

C_1 logic:

\[ 1^{c_1} = [(Q'A'B' + Q'AB)J_1 + A'B]T_s T'_p (CMR)F_3 \]

\[ 0^{c_1} = T'_p T_s (CMR)F_3 \]

C_2 logic:

\[ 1^{c_2} = [(QA'B' + QAB)J_1 + AB']T_s T'_p (CMR)F_3 \]

\[ 0^{c_2} = T'_p T_s (CMR)F_3 \]

If the signs of the operands are not equal at the sign bit time (T'_p T), then the terms A'B' and AB' are used to set C_1 and C_2. If A and B are equal and J_1 is off at the sign bit time, then both C_1 and C_2 should be zero.

Register recirculation (U, X, V, and E) logic:

\[ U_{33} \leftarrow U_0; U_i \leftarrow U_{i+1}; i = 0, 1, \ldots, 32 \text{ during } (CMR)F_3 \]

\[ X_{33} \leftarrow X_0; X_i \leftarrow X_{i+1}; i = 0, 1, \ldots, 32 \text{ during } (CMR)F_3 \]
\[ V_{33} \leftarrow V_0; \quad V_i \leftarrow V_{i+1}; \quad i = 0, 1, \ldots, 32 \text{ during } (CMR)F_3 \]

\[ E_8 \leftarrow E_1; \quad E_i \leftarrow E_{i+1}; \quad i = 1, \ldots, 7 \text{ during } P_5' P'T'(CMR)F_3 \]

All the registers have the usual recirculation logic during \( F_3 \) except the C register which has an exception to allow \( C_1 \) and \( C_2 \) to be set by the compare logic.

**C register recirculation:**

\[ C_{32} \leftarrow C_1; \quad C_i \leftarrow C_{i+1}; \quad i = 3, \ldots, 31 \text{ during } T'_p(CMR)F_3 \]

\[ C_1 \leftarrow C_2; \quad C_2 \leftarrow C_3 \text{ during } T'_s T'(CMR)F_3 \]

**D logic:**

\[ D_{16} \leftarrow C_1; \quad D_i \leftarrow D_{i+1}; \quad i = 1, \ldots, 15 \text{ during } P_5(CMR)F_3 \]

The control counter part of the C register is in the lower 16 bits at the start of \( P_5 \). The D register copies the control counter as it comes out of \( C_1 \).

**F logic:**

\[ f_3 = T_s T_p(CMR)F_3 \]

\[ f_0 = T_s T_p(CMR)F_3 \]

At the end of \( F_3 \) the computer returns to \( F_0 \) for the next instruction fetch.
Exchange Register

C(R) := C(A)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXU</td>
<td>3c</td>
</tr>
<tr>
<td>EXX</td>
<td>5c</td>
</tr>
<tr>
<td>EXE</td>
<td>bc</td>
</tr>
<tr>
<td>EXV</td>
<td>dc</td>
</tr>
</tbody>
</table>

The contents of the specified register and the effective address are exchanged. An exchange with the C register will not alter C.

Execution time: 100 μs

Exchange between registers or memory uses the A and B gates as the source of the two operands to be exchanged. The operand selected by the effective address comes through B and the operand selected by the command comes through A.

Exchange logic:

\[
U_{33} = A(L_9 L_8 L_7)H + B(O_8 O_7 O_6) \text{ during (EXR)F}_3
\]

\[
X_{33} = A(L_9 L_8 L_7)H + B(O_8 O_7 O_6') \text{ during (EXR)F}_3
\]

\[
V_{33} = A(L_9 L_8 L_7')H + B(O_8 O_7 O_6) \text{ during (EXR)F}_3
\]

\[
F_8 = A(L_9 L_8 L_7)H + B(O_8 O_7 O_6) \text{ during P}_5 P_4 P_1 (EXR)F_3
\]

\[
W = AH'(EXR)F_3
\]

\[
WE = H'(EXR)F_3
\]

If H is on, the register selected by L_{7-9} will copy A.

If H is off, the write line (W) copies A and the write enable line (WE) is turned on. If the command selects a register, the
register will copy B. It is possible for a register to exchange with itself.

Copy logic:

\[ U_{33} \leftarrow U_0 \text{ during } (L_9 + L_8 + L_7 + H')(O_8 + O_7 + O_6)(EXR)F_3 \]

\[ X_{33} \leftarrow X_0 \text{ during } (L_9 + L_8 + L_7 + H')(O_8 + O_7 + O_6)(EXR)F_3 \]

\[ V_{33} \leftarrow V_0 \text{ during } (L_9 + L_8 + L_7 + H')(O_8 + O_7 + O_6)(EXR)F_3 \]

\[ C_{33} \leftarrow C_1 \text{ during } T'_{p}(EXR)F_3 \]

\[ E_8 \leftarrow E_1 \text{ during } P_5 P_4 T'_{p}(L_9 + L_8 + L_7 + H')(O_8 + O_7 + O_6)(EXR)F_3 \]

If a register is not selected by either the effective address or the command, it will recirculate.

Shift logic:

\[ U_i \leftarrow U_{i+1}; \quad i = 0, 1, \ldots, 32 \text{ during } (EXR)F_3 \]

\[ X_i \leftarrow X_{i+1}; \quad i = 0, 1, \ldots, 32 \text{ during } (EXR)F_3 \]

\[ V_i \leftarrow V_{i+1}; \quad i = 0, 1, \ldots, 32 \text{ during } (EXR)F_3 \]

\[ C_i \leftarrow C_{i+1}; \quad i = 1, \ldots, 31 \text{ during } T'_{p}(EXR)F_3 \]

\[ E_i \leftarrow E_{i+1}; \quad i = 1, \ldots, 7 \text{ during } P_5 P_4 T'_{p}(EXR)F_3 \]

All registers shift in for EXR regardless of whether or not they are selected.

D register logic:

\[ D_{16} \leftarrow C_1; \quad D_i \leftarrow D_{i+1}; \quad i = 1, \ldots, 15 \text{ during } P_5(EXR)F_3 \]
The D register copies the control counter from \( C_1 \) during the last half of \( F_3 \).

F logic:

\[
0^{f_3} = T_s T_p (EXR)F_3
\]

\[
1^{f_0} = T_s T_p (EXR)F_3
\]

EXR is finished at the end of \( F_3 \) so the next state is \( F_0 \).

Increment Index Register and Skip on Zero

If \( C(M)_4 = 0 \) then \( C(I)_{17-32} := C(I)_{17-32} + A \)

If \( C(M)_4 = 1 \) then \( C(I)_{17-32} := A \)

If \( C(I)_{17-32} = 0 \)
then \( C(C)_{17-32} := C(C)_{17-32} + 1 \)

If the immediate mode is off, the effective address is added to bits 17 to 32 of the specified index register. If the immediate mode is on, the effective address is loaded into bits 17 to 32 of the specified index register. In either case, if the result is zero, the next instruction is skipped.

Execution time: 200 \( \mu \)s

The \( B \) gate is forced to copy memory by turning \( H \) off at the start of \( F_3 \).
H logic:

\[ h = T' T' (IXS)F' \]

The A gate has logic added to allow it to copy the D register during the last half word time \((P_5)\).

A gate logic:

\[ A = D_1 P_5 (IXS)F_3 \]

The D register shifts, copying the output of the adder \( Q \). The sum of the effective address and the contents of the index register comes from \( Q \).

D register logic:

\[ D_{16} \rightarrow Q; D_i \rightarrow D_{i+1}; i = 1, \ldots, 15 \text{ during } P_5 (IXS)F_3 \]

To insure that \( K \) acts as a carry for the adder, \( G_1 \) is turned on at the beginning of \( F_3 \).

G_1 logic:

\[ 1g_1 = T' T' (IXS)F_3 \]

At the end of \( F_3 \) the D register should contain the effective address plus the contents of the selected index register. The flip-flop \( J_1 \) is used to indicate whether or not the sum in the D register is zero.

J_1 logic:

\[ 1j_1 = T' T' (IXS)F_3 \]

\[ 0j_1 = QP_5 (IXS)F_3 \]
During the first bit time of $F_3$, $J_1$ is turned on. If at any time during the last half word time ($P_5$) a one comes through $Q$, $J_1$ is turned off. Therefore, at the last bit time of $F_3$ if $J_1$ is on, the contents of the D register equals zero.

F logic:

$$0^F_3 = T_s T_p (IXS)F_3$$

$$1^F_4 = T_s T_p (IXS)F_3$$

$F_4$ follows $F_3$ for the IXS command. It is during $F_4$ that the contents of the D register are put into the specified index register.

The control counter is counted up by one if $J_1$ is on.

Load index register logic:

$$W = D_1 (IXS)F_4$$

$$WE = P_5 (IXS)F_4$$

$$D_i \leftarrow D_{i+1}; \quad i = 1, \ldots, 15 \quad \text{during } P_5 (IXS)F_4$$

The D register shifts, allowing the write line ($W$) to copy it. The write enable line ($WE$) is turned on during the last half word time, causing the store to take place into the last half of the index word.

K logic:

$$1^k = J_1 P'_1 T'_5 (IXS)F_4$$

$$0^k = C'_1 P'_5 (IXS)F_4$$
If $J_1$ is on, $K$ is turned on and used as a carry for a half add to the control counter.

Half add logic:

$$D_{16} \leftarrow (C_{17}' + C_{17} K) \text{ during } P_5(IXS)F_4$$

$$C_{32} \leftarrow (C_{17}' + C_{17} K); C_i \leftarrow C_{i+1}; i = 17, \ldots, 31 \text{ during } P_5(IXS)F_4$$

The control register is counted up by one if $K$ is on at the start of $P_5$. As the D register shifts it copies the input to the control counter. At the end of $F_4$ both the D register and control contain the address of the next instruction.

F logic:

$$f_4^0 = T_s T_p(IXS)F_4$$

$$f_0^1 = T_s T_p(IXS)F_4$$

Control goes to $F_0$ at the end of $F_4$ for the IXS command.

**Shift Logically**

This command uses bit 16 of the effective address to determine the direction of the shift. Therefore, this command is explained in two parts below.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLR</td>
<td>19</td>
</tr>
</tbody>
</table>
Shift Left Register:

For \( i = 1 \) step 1 until \( A \)

\[
C(R)_k^{i+1} = \frac{C(R)_k}{k} = 1, \ldots, 31
\]

\( C(R)_1 := 0 \)

The contents of the specified register is shifted left the number of places specified by the effective address. Zeroes are shifted in from the right. Immediate mode has no effect. If register \( Z \), three, \( C \), \( E \) or \( W \) is specified, the effect is no operation.

Execution time: 200 \( \mu s \)

Shift Right Register:

For \( i = 1 \) step 1 until \( A \)

\[
C(R)_k := C(R)_{k+1}/k = 1, \ldots, 31
\]

\( C(R)_{32} := 0 \)

The contents of the specified register is shifted right the number of places specified by the effective address. Zeroes are shifted in from the left. Immediate mode has no effect. If register \( Z \), three, \( C \) or \( W \) is specified, the effect is no operation.

Execution time: 200 \( \mu s \)

For \( SLR \) bit 16 of the \( D \) register determines the direction of the shift. If \( D_{16} \) is on, the shift is left; if it is off, the direction is right. Bits 1 to 6 of the \( D \) register form a binary counter which counts down. If bits 1 to 8 of the \( D \) register are all zero, the shifting
will stop. This means that if bit 6, 7, or 8 of the D register is on, the specified register will be cleared to zero.

Left shift logic:

\[
\begin{align*}
U_1 & \leftarrow 0; \ U_{i+1} \leftarrow U_i; \ i = 1, \ldots, 31 \quad \text{during } (O'O'O') \quad \begin{array}{c}
8 \quad 8 \quad 8 \\
1 \quad 7 \quad 6
\end{array} \quad (\Sigma D_j) T'D' \quad (SLR) F_3 \\
X_1 & \leftarrow 0; \ X_{i+1} \leftarrow X_i; \ i = 1, \ldots, 31 \quad \text{during } (O'O'O') \quad \begin{array}{c}
8 \quad 8 \quad 8 \\
1 \quad 7 \quad 6
\end{array} \quad (\Sigma D_j) T'D' \quad (SLR) F_3 \\
V_1 & \leftarrow 0; \ V_{i+1} \leftarrow V_i; \ i = 1, \ldots, 31 \quad \text{during } (O'O'O') \quad \begin{array}{c}
8 \quad 8 \quad 8 \\
1 \quad 7 \quad 6
\end{array} \quad (\Sigma D_j) T'D' \quad (SLR) F_3 
\end{align*}
\]

Right shift logic:

\[
\begin{align*}
U_{32} & \leftarrow 0; \ U_i \leftarrow U_{i+1}; \ i = 1, \ldots, 31 \quad \text{during } (O'O'O') \quad \begin{array}{c}
8 \quad 8 \quad 8 \\
1 \quad 7 \quad 6
\end{array} \quad (\Sigma D_j) T'D' \quad (SLR) F_3 \\
X_{32} & \leftarrow 0; \ X_i \leftarrow X_{i+1}; \ i = 1, \ldots, 31 \quad \text{during } (O'O'O') \quad \begin{array}{c}
8 \quad 8 \quad 8 \\
1 \quad 7 \quad 6
\end{array} \quad (\Sigma D_j) T'D' \quad (SLR) F_3 \\
V_{32} & \leftarrow 0; \ V_i \leftarrow V_{i+1}; \ i = 1, \ldots, 31 \quad \text{during } (O'O'O') \quad \begin{array}{c}
8 \quad 8 \quad 8 \\
1 \quad 7 \quad 6
\end{array} \quad (\Sigma D_j) T'D' \quad (SLR) F_3 \\
E_8 & \leftarrow 0; E_i \leftarrow E_{i+1}; \ i = 1, \ldots, 7 \quad \text{during } (O'O'O') \quad \begin{array}{c}
8 \quad 8 \quad 8 \\
1 \quad 7 \quad 6
\end{array} \quad (\Sigma D_j) T'D' \quad (SLR) F_3 
\end{align*}
\]

D register count down logic:

\[
\begin{align*}
1^{d_1} & = D_1 (D_2 + D_3 + D_4 + D_5 + D_6) (EDC) \\
0^{d_1} & = D_1 (EDC) \\
1^{d_2} & = D_2 D_1 (D_2 + D_4 + D_5 + D_6) (EDC) \\
0^{d_2} & = D_2 D_1 (EDC) \\
1^{d_3} & = D_3 D_2 D_1 (D_4 + D_5 + D_6) (EDC) \\
0^{d_3} & = D_3 D_2 D_1 (EDC)
\end{align*}
\]
The Term EDC stands for enabled D register count. It is used here as a notational convenience. Other shift commands will add terms to EDC and refer to SLR for the D register count logic. The above counter is designed to stop counting when it reaches zero.

F logic:

\[ f_3 = T_s T_p (SLR)F_3 \]

\[ f_4 = T_s T_p (SLR)F_3 \]

The state after \( F_3 \) is \( F_4 \) for SLR. Since the D register is used in \( F_3 \) as a counter, \( F_4 \) is necessary to allow the control counter to be gated into D.

Control counter recirculation:

\[ C_{32} \rightarrow C_{17} \; ; \; C_i \rightarrow C_{i+1} \; ; \; i = 17, \ldots, 31 \] during \( P_5 (SLR)F_4 \)

D register copy:

\[ D_{16} \rightarrow C_{17} \; ; \; D_i \rightarrow D_{i+1} \; ; \; i = 1, \ldots, 15 \] during \( P_5 (SLR)F_4 \)
F logic:

\[
\begin{align*}
\mathcal{F}_0 &= T_s T_p (SLR) F_4 \\
\mathcal{F}_{10} &= T_s T_p (SLR) F_4
\end{align*}
\]

\( F_4 \) is the last state for SLR.

**Shift Modified**

This command uses bit 16 of the effective address to determine the type of shift. Therefore, this command is explained in two parts below.

**Shift Circular:**

For \( i = 1 \) step 1 until A

\[
C(R)_k := C(R)_{k+1}/k = 1, \ldots, 31
\]

\( C(R)_{32} := C(R)_1 \)

The contents of the specified register are rotated right the number of places specified by the effective address (maximum of \( 32 \times 10 \) places). Immediate mode has no effect. If register Z, three, C or W is specified, the effect is no operation.

**Execution time:** 200 \( \mu s \)

**Shift Arithmetic:**

\[
C(R) := C(R) \times 2^{-A}
\]

If \( C(R)_1 = 1 \) and \( C(R) < 1 \) during shift

then \( C(C)_7 := 1 \)
The contents of the specified register are shifted right the number of places specified by the effective address. The sign bit is spread. Immediate mode has no effect. If register Z, three, C or W is specified the effect is no operation.

Execution time: 200 µs

For SMR, bit 16 of the D register determines the type of shift. If D_{16} is on, the shift is circular; if it is off, the shift is arithmetic. Bits 1 to 6 of the D register form a binary counter which counts down. If bits 1 to 8 of the D register are all zero, the shifting will stop.

Circular shift logic:

\[ U_{32} \leftarrow U_1; U_{i} \leftarrow U_{i+1}; i = 1, \ldots, 31 \text{ during } (O'X'O')(\sum_{D})T'D_{16}(SMR)F_3 \]

\[ X_{32} \leftarrow X_1; X_{i} \leftarrow X_{i+1}; i = 1, \ldots, 31 \text{ during } (O'O'O')(\sum_{D})T'D_{16}(SMR)F_3 \]

\[ V_{32} \leftarrow V_1; V_{i} \leftarrow V_{i+1}; i = 1, \ldots, 31 \text{ during } (O'O'O')(\sum_{D})T'D_{16}(SMR)F_3 \]

\[ E_{8} \leftarrow E_1; E_{i} \leftarrow E_{i+1}; i = 1, \ldots, 7 \text{ during } (O'O'O')(\sum_{D})T'D_{16}(SMR)F_3 \]

Circular shift logic is similar to recirculation logic except the contents of D_{1-8} control the timing.

Arithmetic shift logic:

\[ U_0 \leftarrow U_1; U_{32} \leftarrow U_{32}; U_{i} \leftarrow U_{i+1}; i = 1, \ldots, 31 \text{ during } (O'O'O')(\sum_{D})T'D_{16}(SMR)F_3 \]
\[ x_0 = x_1; x_{32} \leftarrow x_{32}; x_i \leftarrow x_{i+1}; i = 1, \ldots, 31 \text{ during } \]

\[ (O' O' O'')(\sum D)T'D' (SMR)F_3 \]

\[ v_0 = v_1; v_{32} \leftarrow v_{32}; v_i \leftarrow v_{i+1}; i = 1, \ldots, 31 \text{ during } \]

\[ (O' O' O'')(\sum D)T'D' (SMR)F_3 \]

\[ E_8 \leftarrow 0; E_i \leftarrow E_{i+1}; i = 1, \ldots, 7 \text{ during } \]

\[ (O' O' O'')(\sum D)T'D' (SMR)F_3 \]

\[ U_0 \leftarrow 0; X_0 \leftarrow 0; V_0 \leftarrow 0 \text{ during } T'T (SMR)F_3 \]

The sign bit copies itself for the arithmetic shift to spread the sign. The E register is always considered positive. Since an arithmetic right shift of a two's complement negative number causes rounding in the negative direction (as compared to sign and magnitude representation), bits \( U_0 \), \( X_0 \), and \( V_0 \), are used to detect any rounding.

**Negative rounding detection:**

\[ c_7 = (U_{32} U + X_{32} X + V_{32} V)T'TD' (SMR)F_3 \]

\( C_7 \) is turned on at the end of \( F_3 \) if any one bits were shifted off a specified register that contained a negative number.

**Enable D register count logic:**

\[ EDC = (SMR)F_3 T' \]

For the D register count logic see the SRL logic.
F logic:

\[ f_3 = T_s T_p (SMR) F_3 \]
\[ f_4 = T_s T_p (SMR) F_3 \]

State \( F_4 \) follows \( F_3 \) to allow time for the next instruction address to be loaded into the D register.

Control counter recirculation:

\[ C_{32} \leftarrow C_{17}; \ C_i \leftarrow C_{i+1}; \ i = 17, \ldots, 31 \] during \( P_5(SMR)F_4 \)

D register copy:

\[ D_{16} \leftarrow C_{17}; \ D_i \leftarrow D_{i+1}; \ i = 1, \ldots, 15 \] during \( P_3(SMR)F_4 \)

F logic:

\[ f_4 = T_s T_p (SMR) F_4 \]
\[ f_0 = T_s T_p (SMR) F_4 \]

\( F_4 \) is the last state for SMR.

Store Register

\[ C(A) := C(R) \]

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>STZ</td>
<td>12</td>
</tr>
<tr>
<td>STU</td>
<td>32</td>
</tr>
<tr>
<td>STX</td>
<td>52</td>
</tr>
<tr>
<td>STC</td>
<td>92</td>
</tr>
<tr>
<td>STE</td>
<td>b2</td>
</tr>
<tr>
<td>STU</td>
<td>d2</td>
</tr>
<tr>
<td>STW</td>
<td>f2</td>
</tr>
</tbody>
</table>

The contents of the specified register are stored into the
contents of the effective address.

Execution time: 100 µs

Storing the selected register into memory is relatively simple. The gate A always copies the selected register in F₃; thus all that is needed in the way of logic is to gate A into W (write line).

Write logic:

\[ W = AH'(STR)F_3 \]
\[ WE = H'(STR)F_3 \]

Storing the selected register into a register requires the following logic:

Store into register logic:

\[ U_{33} \leftarrow A \text{ during } (L_9' L_8' L_7')H(STR)F_3 \]
\[ X_{33} \leftarrow A \text{ during } (L_9' L_8 L_7')H(STR)F_3 \]
\[ V_{33} \leftarrow A \text{ during } (L_9, L_8, L_7')H(STR)F_3 \]
\[ C_{32} \leftarrow A \text{ during } (L_9' L_8' L_7')HT_p'(STR)F_3 \]
\[ E_8 \leftarrow A \text{ during } (L_9' L_8' L_7')HP_p'P_p'T_p'(STR)F_3 \]

Shift logic:

\[ U_i \leftarrow U_{i+1}; \ i = 0, \ldots, 32 \text{ during } (STR)F_3 \]
\[ X_i \leftarrow X_{i+1}; \ i = 0, \ldots, 32 \text{ during } (STR)F_3 \]
\[ V_i \leftarrow V_{i+1}; \ i = 0, \ldots, 32 \text{ during } (STR)F_3 \]
\( C_i \leftarrow C_{i+1}; \ i = 1, \ldots, 31 \) during \( T'(STR)F_3 \)

\( E_i \leftarrow E_{i+1}; \ i = 1, \ldots, 7 \) during \( P'_5P'_4T'(STR)F_3 \)

The registers shift in \( F_3 \) under all conditions for STR. For the last half of \( F_3 \) the D register copies the upper half of the C register to enable it to select the next command.

The D register copies the control counter:

\[ D_{16} \leftarrow C_{32}; D_i \leftarrow D_{i+1}; \ i = 1, \ldots, 15 \) during \[ [P_5(P_2 + P_3 + P'_1 + P'_4) + T_s](STR)F_3 \]

The D register must copy \( C_{32} \) because the C register may change during store into \( C \).

F logic:

\[ f_3 = T_s T_p (STR)F_3 \]

\[ f_0 = T_s T_p (STR)F_3 \]

At the end of \( F_3 \) the computer returns to \( F_0 \) for the next instruction fetch.
VIII. REGISTER NON-SPECIFYING COMMANDS

These commands, called register non-specifying, are fixed as to the registers they operate with, if any at all. Bits 4 and 5 of the O register must be zero. This means that there are 64 possible operation codes in this group. There are 23 of them implemented at this time (Table 4). Each of these commands will be described in the manner that the register specifying commands were described.

Table 4. Register non-specifying commands.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Command</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASD</td>
<td>arithmetic shift double</td>
<td>80</td>
</tr>
<tr>
<td>DIV</td>
<td>divide</td>
<td>26</td>
</tr>
<tr>
<td>EXT</td>
<td>execute</td>
<td>61</td>
</tr>
<tr>
<td>JMP</td>
<td>jump</td>
<td>60</td>
</tr>
<tr>
<td>JSC</td>
<td>jump and store C</td>
<td>66</td>
</tr>
<tr>
<td>LCH</td>
<td>load character</td>
<td>c0</td>
</tr>
<tr>
<td>LHW</td>
<td>load half word</td>
<td>c2</td>
</tr>
<tr>
<td>LLC</td>
<td>load left half of C</td>
<td>c7</td>
</tr>
<tr>
<td>MUL</td>
<td>multiply</td>
<td>25</td>
</tr>
<tr>
<td>NRM</td>
<td>normalize</td>
<td>83</td>
</tr>
<tr>
<td>NOP</td>
<td>no-operation</td>
<td>00</td>
</tr>
<tr>
<td>RIM</td>
<td>reset interrupt mask</td>
<td>c6</td>
</tr>
<tr>
<td>RST</td>
<td>reset flags</td>
<td>65</td>
</tr>
<tr>
<td>SCH</td>
<td>store characters</td>
<td>c1</td>
</tr>
<tr>
<td>SET</td>
<td>set flags</td>
<td>64</td>
</tr>
<tr>
<td>SFD</td>
<td>shift double length</td>
<td>81</td>
</tr>
<tr>
<td>SHW</td>
<td>store half word</td>
<td>c3</td>
</tr>
<tr>
<td>SIM</td>
<td>set interrupt mask</td>
<td>c4</td>
</tr>
<tr>
<td>SIT</td>
<td>set interrupt enable toggle</td>
<td>e1</td>
</tr>
<tr>
<td>SKF</td>
<td>skip if false</td>
<td>63</td>
</tr>
<tr>
<td>SKT</td>
<td>skip if true</td>
<td>62</td>
</tr>
<tr>
<td>STD</td>
<td>store double</td>
<td>84</td>
</tr>
<tr>
<td>TRE</td>
<td>transfer E register</td>
<td>a2</td>
</tr>
</tbody>
</table>
Arithmetic Commands

Divide

If divisor = 0 then C(C)₃:= 1

If |C(A)| ≥ |C(U)| then C(C)₃:= 1

If C(A) = -1 V C(U)C(X)₁₋₃₁ = -1,
then C(C)₃:= 1

C(X):= C(U)C(X)₁₋₃₁/C(A)

C(U):= Remainder (sign = sign of dividend)

C(V):= C(A)

The contents of the U register together with bits 1 to 31 of the X register are divided by the contents of the effective address. Overflow turns on C₃.

Execution time: 3.4 ms to 3.7 ms

During F₃ and divide, the contents of the effective address are loaded into the V register. All registers recirculate. The sign is spread in the U and V registers. The D register is set to zero.

U register recirculation:

U₁ ← Uᵢ₊₁; i = 0, ..., 32 during (DIV)F₃

U₃₃ ← U₀ during (T₁'+T₀')(DIV)F₃

X register recirculation:

X₃₃ ← X₁; Xᵢ ← Xᵢ₊₁; i = 1, ..., 32 during (T₁'+T₀')(DIV)F₃
V register copies divisor from B:

\[ V_i \leftarrow V_{i+1}; \ i = 0, \ldots, 32 \text{ during } (\text{DIV})F_3 \]

\[ V_{33} \leftarrow B \text{ during } (T'_s + T'_p)(\text{DIV})F_3 \]

The timing on the U and V registers cause the sign to spread during the last bit time.

Zero D register logic:

\[ D_{16} \leftarrow 0; \ D_i \leftarrow D_{i+1}; \ i = 1, \ldots, 15 \text{ during } P_5(\text{DIV})F_3 \]

\[ X_{33} \text{ and } X_{32} \text{ set to zero: } \]

\[ X_{33} \leftarrow 0; \ X_{32} \leftarrow 0 \text{ during } T_s T_p(\text{DIV})F_3 \]

The divide algorithm needs to have both operands positive. The divisor is complemented in \( F_4 \) and the double length dividend is complemented in \( F_5 \) and \( F_6 \). \( F_4 \) and \( F_5 \) are entered only if the proper operand is negative. If both operands are positive, \( F_7 \) follows \( F_3 \).

F logic:

\[ 0^f_3 = T_s T_p(\text{DIV})F_3 \]

\[ 1^f_4 = V_{33}s^T_p(\text{DIV})F_3 \]

\[ 1^f_5 = V'_33^T_U T_p(\text{DIV})F_3 \]

\[ 1^f_7 = U'_33^T_V T_p(\text{DIV})F_3 \]

\( J_1 \) and \( J_2 \) are used to indicate the signs of the operands. If \( J_1 \) is on in \( F_7 \) the divisor was negative, and if \( J_2 \) is on in \( F_7 \)
the dividend was negative.

\[ J_1 \text{ and } J_2 \text{ initialization:} \]

\[ 0^1_j = 0^2_j = T_s T_p (\text{DIV})_3 F \]

In \( F_4 \) the V register (containing the divisor) is negated. The V register recirculates until the first one bit appears in \( V_0 \). \( G_1 \) is turned on by \( V_0 \) on. After the first one bit appears in \( V_0 \), \( V_{33} \) copies the off side of \( V_0 \).

V register shift logic:

\[ V_i \leftarrow V_{i+1}; \ i = 0, \ldots, 32 \text{ during } (\text{DIV})_4 F \]

\( G_1 \) logic:

\[ 1g_1 = V_0 T'(\text{DIV})_3 F \]

\( V_{33} \) copy logic:

\[ V_{33} \leftarrow (V_1'G_1 + V_0 G') \text{ during } (\text{DIV})_4 F \]

If \( V_0 \) is on and \( G_1 \) is off at the sign bit time, the V register must contain a minus one. In this case \( C_3 \) is turned on indicating overflow.

\[ 1c_3 = V G_1 T_0 T'(\text{DIV})_4 F \]

\( J_1 \) logic:

\[ 1j_1 = T_s T_p (\text{DIV})_4 F \]

\( J_1 \) on indicates that the divisor was negative.
F logic:

\[ f'_4 = T_s T_p (DIV)F_4 \]

\[ f'_5 = U_{33} T_s T_p (DIV)F_4 \]

\[ f'_7 = U_{33} T_s T_p (DIV)F_4 \]

\( F_5 \) is entered if the dividend is negative; otherwise \( F_7 \) is entered. The X register part of the dividend is complemented in \( F_5 \).

\( G_1 \) serves as the first-one-bit indicator and as such must be left alone for the \( F_6 \) part of the complementation.

X register shift logic:

\[ X_i \leftarrow X_{i+1}; \, i = 0, \ldots, 32 \text{ during } (DIV)F_4 \]

G₁ logic:

\[ g_1 = X_0 T'(DIV)F_5 \]

X₃₃ copy logic:

\[ X_{33} \leftarrow (X_1 G_0 + X_0 G'_1) \text{ during } (DIV)F_5 \]

\( X_{32} \) and \( X_{33} \) were turned off in \( F_3 \) to allow the X register to be treated as a positive extension of the U register. The X register complementation follows the same scheme as the V register \( F_4 \) complementation.

J₂ logic:

\[ j_2 = T_s T_p (DIV)F_5 \]

\( J_2 \) is turned on at the end of \( F_5 \) to indicate that the dividend
was negative.

F logic:

\[ f_5 = T s p (DIV)F_5 \]

\[ f_6 = T s p (DIV)F_5 \]

During \( F_6 \) the last half of the dividend complementation takes place.

U register shift logic:

\[ U_i = U_{i+1}; \ i = 0, \ldots, 32 \text{ during } (DIV)F_6 \]

G₁ logic:

\[ g_1 = U_0 T' p (DIV)F_6 \]

U₃₃ copy logic:

\[ U_{33} = (U_0 G_1 + U_0 G'_1) \text{ during } (DIV)F_6 \]

\( U_0 \) on and \( G_1 \) off at sign bit time indicate that the dividend equals minus one. This condition turns \( C_3 \) on indicating overflow.

C₃ logic:

\[ c_3 = U_0 G_1 T s p (DIV)F_6 \]

F logic:

\[ f_6 = T s p (DIV)F_6 \]

\[ f_7 = T s p (DIV)F_6 \]

\( G_1 \) logic needs to be cleaned up at this stage. \( G_1 \) needs to be turned off at end of \( F_3 \) and \( F_4 \) if either operand is negative. \( G_1 \)
needs to be on at the start of $F_7$ so that recirculation of the U register takes place for the first word time of $F_7$.

$G_1$ logic:

$$1g_1 = [(U'_{33} V'_{33})F_3 + U'_{33} F_4 + F_6]T_s T_p (DIV)$$

$$0g_1 = (U_{33} + V_{33})(F_3 + F_4)T_s T_p (DIV)$$

In $F_7$ the divide of the positive dividend in $U$ and $X$ by the positive divisor in $V$ occurs. The method used is a comparison algorithm (1, p. 36)

The adder $Q$ is used in $F_7$ for the subtraction in divide.

The A gate copies the minuend and the B gate copies the subtrahend.

**A and B gate logic:**

$$A = U_1 (DIV)F_7$$

$$B = V_1 (DIV)F_7$$

The $V$ register recirculates in $F_7$ to provide the minuend for each word time.

**V register recirculation:**

$$V_{33} \rightarrow V_{i}; \ V_i \rightarrow V_{i+1}; \ i = 1, \ldots, 32 \text{ during } (T_s + T_p)(DIV)F_7$$

**U register shift logic:**

$$U_i \rightarrow U_{i+1}; \ i = 1, \ldots, 32 \text{ during } (T_s + T_p)(DIV)F_7$$

The $U$ register either copies the output of $Q$ or recirculates in $F_7$. If the flip-flop $G_1$ is on, the $U$ register recirculates; if it
is off, the U register copies Q.

\[ U_{33} \text{ copy logic:} \]
\[ U_{33} \leftarrow (AG_1 + QG_1') \text{ during } (T'_s + T'_p)(DIV)F_7 \]

During the last bit time in \( F_7' \), the U and X registers shift left one bit. The new quotient bit goes to \( X_1 \), \( X_{31} \) goes to \( U_1 \) and the extra sign bit is lost from \( U_{33} \). The U register does not shift left the last word time in \( F_7' \). The quotient bit that goes to \( X_1 \) is a one if a subtraction took place \( (G_1 \text{ off}) \) or a zero if it did not.

\( X \) register left shift:
\[ X_1 \leftarrow G_1'; \; X_{i+1} \leftarrow X_i; \; \text{i = 1, \ldots, 31 during } T_s T_p(DIV)F_7 \]

The \( X \) register shifts 32 bits in order to develop the full quotient. Originally the \( X \) register contains only 31 bits of the double length dividend.

\( U \) register left shift:
\[ U_1 \leftarrow X_{31}; \; U_{i+1} \leftarrow U_i; \; \text{i = 1, \ldots, 32 during } D'_s T_s T_p(DIV)F_7 \]

The \( U \) register shifts the full 33 bits. \( D_6 \) on indicates the 32\textsuperscript{nd} word time in \( F_7' \). For this last word time in \( F_7' \), the \( U \) register contains the true remainder and must not be shifted left.

The borrow for the adder is \( K \) and uses much the same logic as was developed for the \( K \) flip-flop in \( F_3 \).

\( K \) logic:
\[ k^1 = (AG_1 + A'BG_1')(T'_s + T'_p)(DIV)F_7 \]
\[ k = (A'B'G_1 + AB'G_1')(T'_s + T'_p)(\text{DIV})F_7 \]

\[ 0^k = T_s T_p (\text{DIV})F_7 \]

The flip-flop \( G_3 \) is used to make the comparison between the partial remainder being developed and the divisor. \( G_3 \) is off at the start of a word time and is turned on if the new partial remainder is greater than the divisor. Since the least significant bit of the new partial product is in \( X_{31} \), \( G_3 \) starts out by looking at \( X_{31} \) and from then on looks at the output of \( U_{33} \).

\( G_3 \) logic:

\[ 0^g_3 = T_s T_p (F_3 + F_7)(\text{DIV}) \]

\[ 1^g_3 = (X'_1 V_1)T'_s T_p (\text{DIV})F_7 \]

\[ 1^g_3 = (U_{33}'B)T'_p (\text{DIV})F_7 \]

\[ 0^g_3 = (U_{33}'B')T'_p (\text{DIV})F_7 \]

The flip-flop \( G_1 \) copies \( G_3 \) at the end of each of the first 31 word times in \( F_7 \).

\( G_1 \) logic:

\[ 1^g_1 = G_3 T_s T_p (\text{DIV})F_7 D'_6 \]

\[ 0^g_1 = G_1 T_s T_p (\text{DIV})F_7 \]

\[ 0^g_1 = T_s T_p D_6 (\text{DIV})F_7 \]

The flip-flop \( G_2 \) compares the contents of the \( U \) register with the contents of the \( V \) register. \( G_2 \) on at the end of the first word
time in \( F_7 \) indicates a divide overflow (dividend larger than divisor).

\[ G_2 \text{ logic:} \]
\[
1g_2 = A'B(T'_s + T'_p)(DIV)F_7 \\
0g_2 = AB'(T'_s + T'_p)(DIV)F_7 \\
0g_2 = T_s T_p (F'_3 + F'_7)(DIV)
\]

Overflow logic:
\[
1c_3 = G_2^1 \left[ \Pi D'_1 \right] D_1 (DIV)F_7
\]

If at the end of the first word time in \( F_7 \), \( G_2 \) is off, \( C_3 \) is turned on.

During \( F_7 \) the D register counts word times. The carry for the half add to the D register is performed by \( G_4 \).

\[ G_4 \text{ logic:} \]
\[
1g_4 = T'_s T_p (DIV)F_7 \\
0g_4 = D'_1 P_5 (DIV)F_7
\]

D register half add:
\[
D_{16} \leftarrow (D_1 G'_4 + D'_i G_i); D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \text{ during } P_5 (DIV)F_7
\]

The last word time in \( F_7 \) is indicated by \( D_6 \) on.

\[ F \text{ logic:} \]
\[
f_7 = D_6 T_s T_p (DIV)F_7 \\
f_8 = D_6 T_s T_p (DIV)F_7
\]
In \( F_8 \) the remainder is complemented if the dividend was negative \((J_2\) on). The D register copies the control counter.

Control counter recirculation:

\[
C_{32} \rightarrow C_{17}; C_i \rightarrow C_{i+1}; i = 17, \ldots, 31 \quad \text{during} \quad P_5(DIV)F_8
\]

D register copy logic:

\[
D_{16} \rightarrow C_{17}; D_i \rightarrow D_{i+1}; i = 1, \ldots, 15 \quad \text{during} \quad P_5(DIV)F_8
\]

\( G_1 \) is used to indicate the first one bit in the U register if complementation is to take place.

\( G_1 \) logic:

\[
1^{g_1} = U_0 J_2 T_p(DIV)F_8
\]

\[
0^{g_1} = T_s T_p(DIV)F_8
\]

U register complementation:

\[
U_{33} \leftarrow (U'_1 G_1 + U_0 G'_1); U_i \leftarrow U_{i+1}; i = 0, \ldots, 32 \quad \text{during} \quad (DIV)F_8
\]

J and G flip-flop zeroing logic:

\[
J_1 \leftarrow 0; J_2 \leftarrow 0; G_2 \leftarrow 0; G_3 \leftarrow 0; G_4 \leftarrow 0 \quad \text{during} \quad T_s T_p(DIV)F_8
\]

If the signs of the dividend and divisor were not equal, the next \( F \) state will be \( F_9 \). If the signs were equal, \( F_8 \) is the last state for DIV.

F logic:

\[
0^f_8 = T_s T_p(DIV)F_8
\]
\[ f_9 = (J_1J_2 + J_1'J_2')T_s T_p (DIV)F_9 \]
\[ f_0 = (J_1J_2 + J_1'J_2')T_s T_p (DIV)F_8 \]

The quotient in the X register is complemented in \( F_9 \). The flip-flop \( G_1 \) indicates the first one bit in the X register.

\( G_1 \) logic:

\[ 1g_1 = X_0T'_p (DIV)F_9 \]
\[ 0g_1 = T_s T_p (DIV)F_9 \]

X register complementation:

\[ X_{33} \leftarrow (X'_0G_1 + X_0G'_1); X_i \leftarrow X_{i+1}; i = 0, \ldots, 32 \text{ during } (DIV)F_9 \]

F state logic:

\[ 0f_9 = T_s T_p (DIV)F_9 \]
\[ 1f_0 = T_s T_p (DIV)F_9 \]

\( F_9 \) is the last state for DIV.

Multiply

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUL</td>
<td>25</td>
</tr>
</tbody>
</table>

\( C(U)C(X)_{1-31} := C(U) \times C(A) \)

\( C(U) := C(A) \)

\( C(X)_{32} := 0 \)

If \( C(U) \times C(A) = 1 \) then \( C(C)_3 := 1 \)

The contents of the U register and bits 1 to 31 of the X register are set equal to the product of the contents of the U register with the
contents of the effective address. Bit 32 of the X register is set to zero. Overflow sets \( C_3 \).

Execution time: 3.3 ms

During \( F_3 \) all registers recirculate or shift and the V register copies the multiplicand which is coming through the B gate. The X register copies the multiplier from \( U \) and \( U \) is set to zero.

\[
\begin{align*}
U_{33} &\to 0; U_i \leftarrow U_{i+1}; i = 0, \ldots, 32 \text{ during } (MUL)F_3 \\
X_{33} &\to U_0; X_i \leftarrow X_{i+1}; i = 0, \ldots, 32 \text{ during } (MUL)F_3 \\
V_{33} &\to B \text{ during } (T'_s + T'_p)(MUL)F_3 \\
V_i &\leftarrow V_{i+1}; i = 0, \ldots, 32 \text{ during } (MUL)F_3
\end{align*}
\]

The timing \( (T'_s + T'_p) \) causes the sign bit to be spread during the last bit time so that \( V_{33} \) has a copy of \( V_{32} \).

The D register will be used to count word times for the multiply and is set to the value 1.

\[
\begin{align*}
D_{16} &\to 0; D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \text{ during } P_5 (MUL)F_3 \\
D_1 &\leftarrow 1 \text{ during } T_s T_p (MUL)F_3
\end{align*}
\]

The flip-flop \( J_2 \) is used to indicate whether or not both
multiplicand and multiplier have zeros in bits 1-31.

\[ J_2 \text{ logic:} \]
\[ 1^1_j = T_1^j T_2^j (\text{MUL}) F_3 \]
\[ 0^0_j = (B+U) T_1^j T_2^j (\text{MUL}) F_3 \]

\[ J_2 \text{ on at } T_3^j T_4^j \text{ time together with both } X_{33} \text{ and } V_{33} \]
on indicate that both operands equal minus one.

**Overflow logic:**

\[ 1^1_3 = J_2 X_{33} V_{33} T_3^j T_4^j (\text{MUL}) F_3 \]

\[ C_3 \text{ is set to one if both operands are equal to minus one.} \]

\[ G_1 \text{ is used as the add control and is turned on in } F_3. \]

**G_1 logic:**

\[ 1^1_g_1 = (\text{MUL}) F_3 \]

**E recirculation:**

\[ E_8 \leftarrow E_1; E_i \leftarrow E_{i+1}; i = 1, \ldots, 7 \text{ during } P'_5 P'_4 T_1^j (\text{MUL}) F_3 \]

**C recirculation:**

\[ C_{32} \leftarrow C_1; C_i \leftarrow C_{i+1}; i = 1, \ldots, 31 \text{ during } T_1^j (\text{MUL}) F_3 \]

Both \( E \) and \( C \) recirculate to enable the B gate to copy them.

**F logic:**

\[ 0^0_f_3 = T_1^j T_2^j (\text{MUL}) F_3 \]
\[ 1^1_f_3 = T_1^j T_2^j (\text{MUL}) F_3 \]
\( F_6 \) is the state that performs the multiplication. The computer spends 32 word times in \( F_6 \) for multiplication, one word time for each bit in the multiplier (X register). A 33-bit partial product is constructed in the U register, and then shifted right double length one bit with \( X_{1-31} \). If the multiplier is negative, the last step is a subtract to correct the final product.

\( X_1 \) always contains the current multiplier bit and therefore is used to decide whether to add or not. The V register recirculates providing the multiplicand.

A and B gate logic:

\[
A = U_1 (T'_s T'_p) (MUL) F_6
\]

\[
B = V_1 (T'_s T'_p) (MUL) F_6
\]

A gates the augend and B gates the addend to the adder.

V register recirculation:

\[
V_{33} \leftarrow V_1; \quad V_i \leftarrow V_{i+1}; \quad i = 1, \ldots, 32 \text{ during } (T'_s T'_p) (MUL) F_6
\]

U register partial product logic:

\[
U_{33} \leftarrow (Q X_1 + U_1 X'_1) \text{ during } (T'_s T'_p) (MUL) F_6
\]

If \( X_1 \) is on, the U register copies the new partial product from the adder \( Q \); otherwise it just recirculates.

U register shift logic:

\[
U_i \leftarrow U_{i+1}; \quad i = 1, \ldots, 32 \text{ during } [(T'_s T'_p)' + G_1] (MUL) F_6
\]
The U register shifts for all of $F_6$ if $G_1$ is on. $G_1$ on indicates the first 31 word times in $F_6$. This shifting causes the sign to be spread the last bit time and the least significant bit to be shifted to $X_{31}$. When $G_1$ is off, U will only copy the final partial product and not perform a shift.

**X register shift logic:**

$$X_i \rightarrow X_{i+1}; \ i = 1, \ldots, 31 \text{ during } T_s T_p (MUL)F_6$$

The X register will always shift right one bit at $T_s T_p$ time in $F_6$. This brings the next multiplier bit into $X_1$ and allows the least significant end of the partial product to shift in from the upper end.

**X register copy logic:**

$$X_{32} \rightarrow (U_1 G_1) \text{ during } T_s T_p (MUL)F_6$$

The X register copies $U_1$ for the first 31 bit times in $F_6$ and then is set to zero for the last word time. ($G_1$ is on only for the first 31 word times.)

**K logic:**

$$1^k = (A B G_1 + A' B G_1')(T' + T')(MUL)F_6$$

$$0^k = (A' B' G_1 + A B' G_1'(T' + T')(MUL)F_6 + T_s T_p (MUL)F_6$$

K is used as the carry for the $Q$ adder. If $G_1$ is on, K is the carry; if $G_1'$ is off, K is the borrow.

The D register counts up by one each word time in $F_6$. The
flip-flop $G_3$ acts as the carry for a half add to the D register.

$G_3$ logic:

$$1_{G_3} = T_s T_p (MUL)F_6$$

$$0_{G_3} = D_1 P_5 (MUL)F_6$$

D register count logic:

$$D_{16} \leftarrow (D_1 G_3^{+} + D_1 G_3) \text{ during } P_5 G_1 (MUL)F_6$$

D register counts during $F_6$ and $G_1$ on. The D register contains 1 to start with. When the D register contains 32, the flip-flop $D_6$ will be on. $D_6$ on then indicates that 31 word times have gone past.

$G_1$ logic:

$$0_{G_1} = D_6 T_s T_p (MUL)F_6$$

The flip-flop $G_1$ controls the add and subtract functions of the carry $K$ as well as indicating when word time 32 is in progress. $G_1$ off at this time allows a subtraction if the multiplier was negative.

D register shift logic:

$$D_i \leftarrow D_{i+1}; \ i = 1, \ldots, 15 \text{ during } P_5 (MUL)F_6$$

During all of $F_6$ the D register shifts. During $G_1$ on it is counting; during $G_1$ off it copies the control counter.

Control counter recirculation:

$$C_{32} \leftarrow C_{17}; \ C_i \leftarrow C_{i+1}; \ i = 17, \ldots, 31 \text{ during } P_5 (MUL)F_6$$
D register copy:

\[ D_{16} = C_{17} \text{ during } P_5 G_1^\prime (MUL)F_6 \]

F logic:

\[ f_{6} = T s T p G_1^\prime (MUL)F_3 \]

\[ f_{0} = T s T p G_1^\prime (MUL)F_3 \]

\[ G_1 \text{ off in } F_6 \text{ is the last word time of } MUL. \]

Shift Commands

Arithmetic Shift Double

This command uses bits 15 and 16 of the effective address to determine the direction of the shift and whether to pack or unpack an exponent. This command is explained in four parts below.

Shift Left Arithmetic:

\[ C(U)C(X)_{1-31} := C(U)C(X)_{1-31} \times 2^A \]

If \( U_{32} \neq U_{31} \) during shift then \( C(C)_3 := 1 \)

\( C(X)_{32} := 0 \)

The contents of the U register and contents of bits 1 to 31 of the X register are shifted left, as a double length number, the number of places specified by the effective address (modulo \( 2^8 \)). Bit 32 of the
X register is set to zero. If the sign of the number changes during the shift, \( C_3 \) is set to one. Immediate mode has no effect.

**Execution time:** 200 to 300 \( \mu s \)

**Shift Right Arithmetic:**
\[
C(U)C(X)_{1-31} := C(U)C(X)_{1-31} \times 2^{-A}
\]

If \( C(X)_1 = 1 \) \& \( C(U) < 0 \) during shift
then \( C(C)_7 := 1 \)
\( C(X)_{32} := 0 \)

The contents of the U register and contents of bits 1 to 31 of the X register are shifted right as a double length number, the number of places specified by the effective address (modulo \( 2^8 \)). Bit 32 of the X register is set to zero. If the sign of the U register is negative and one bits are lost, \( C_7 \) is set to one. The sign bit is spread. Immediate mode has no effect.

**Execution time:** 200 to 300 \( \mu s \)

**Pack:**

For \( k = 1 \) step 1 until A
\[
C(U)_{32} := C(U)_{32}
\]
\[
C(U)_i := C(U)_{i+1}/_{i = 1, \ldots, 31}
\]

If \( C(X)_{32} = 0 \) then \( C(X)_{31} := C(U)_1 \)
else \( C(X)_{31} := C(U)_1 \)
\[
C(X)_i := C(X)_{i+1}/_{i = 1, \ldots, 30}
\]
This command packs an exponent from the U register into the X register between the sign bit, \( X_{32} \), and the rest of the fraction (Figure 7). If the quantity in X is negative, the one's complement of U is packed into X. The effective address (modulo \( 2^8 \)) determines the number of bits in U that are used as an exponent. Immediate mode has no effect.

Execution time: 200 to 300 \( \mu s \)

Unpack:

\[
\text{For } k = 1 \text{ step 1 until } A \\
\begin{align*}
C(X)_1 &:= 0 \\
C(X)_{i+1} &:= C(X)_i/i = 1, \ldots, 30 \\
\text{If } X_{32} = 0 \text{ then } C(U)_1 &:= C(X)_{31} \\
\text{else } C(U)_1 &:= C(\overline{X})_{31} \\
C(U)_{i+1} &:= C(U)_i/i = 1, \ldots, 31
\end{align*}
\]

This command unpacks an exponent from the X register into the U register. If the quantity in the X register is negative, the one's complement of the exponent is unpacked and put in the U register (Figure 7). The effective address (modulo \( 2^8 \)) determines the number of bits in U that are used as an exponent. Immediate mode has no effect.

Execution time: 200 to 300 \( \mu s \)
Figure 7. PCK and UPK example.
In \( F_3 \) and \( F_4 \) the \( U \) and \( X \) registers shift left if \( D_{16} \) is on or right if \( D_{16} \) is off. Bits 1 to 6 of the \( D \) register act as a down counter. The shifting occurs as long as there are any one bits in \( D_{1-8} \). If the shifting is completed in \( F_3 \), the \( F_4 \) state is skipped. \( D_{15} \) on causes the connection between the \( U \) and \( X \) registers to be complemented if the sign of \( U \) is negative.

U register right shift:

\[
U_{32} \leftarrow U_{32}; U_i \leftarrow U_{i+1}; \quad i = 1, \ldots, 31 \text{ during } D_{16} \cdot (\Sigma D_i) T' (ASD)(F_{3} + F_{4})
\]

\( X_0 \) logic:

\[
1 \times_0 = T_s T_p (ASD) F_3
\]

\( X_0 \) is set to zero so that it may look for truncated one bits.

X register right shift logic:

\[
1 \times_0 = X_1; X_i \leftarrow X_{i+1}; \quad i = 1, \ldots, 30 \text{ during } D_{16} \cdot (\Sigma D_i) T' (ASD)(F_{3} + F_{4})
\]

\( X_{31} \) copies \( U \) register logic:

\[
X_{31} \leftarrow [U_1 (X'_{32} + D'_{15}) + U'_1 (X_{32} D_{15})] \text{ during } D_{16} \cdot (\Sigma D_i) T' (ASD)(F_{3} + F_{4})
\]

If \( D_{15} \) is on indicating UPK and \( X \) is negative \((X_{32} \text{ on})\), \( X_{31} \) copies the complement of \( U_1 \); otherwise it is a straight copy.

The sign of \( U_{32} \) is spread.

X register left shift logic:

\[
X_1 \leftarrow 0; X_{i+1} \leftarrow X_i; \quad i = 1, \ldots, 30 \text{ during } D_{16} \cdot (\Sigma D_i) T' (ASD)(F_{3} + F_{4})
\]

U register left shift logic:

\[
U_{i+1} \leftarrow U_i; \quad i = 1, \ldots, 31 \text{ during } D_{16} \cdot (\Sigma D_i) T' (ASD)(F_{3} + F_{4})
\]
\(U_1\) copies \(X\) register logic:

\[
U_1 \leftarrow [X_{31}(X'_{31} + D'_{15}) + X_{31}'(X_{31} - D_{15})] \text{ during } D_{16} \sum_{i=1}^{8} (T'_{i}) (ASD) (F_3 + F_4)
\]

If \(D_{15}\) is on indicating PCK and \(X\) is negative \((X_{32} \text{ on})\)
\(U_1\) copies \(X_{31}\) complemented; otherwise it is a straight copy.

\(X_{32}\) set to zero logic:

\[0^*X_{32} = D_{15}' T_s T_p (ASD) F_3\]

The sign of the \(X\) register is set positive if \(D_{15}\) is off.

Enable \(D\) register count logic:

\[
EDC = T'_p (ASD) (F_3 + F_4)
\]

C register flag logic:

\[
1^c7 = (X_0 U_{32}) T_s T_p (D_{16}' - D_{15}')(ASD) F_5
\]

\[
1^c3 = (U_{32}' U_{31} + U_{32} U_{31}') D_{16} D_{15}' \sum_{i=1}^{8} (T'_{i}) (ASD) (F_3 + F_4)
\]

If the \(U\) register is negative and right arithmetic shifting spills
one bits \((X_0 \text{ on})\), \(C_7\) is turned on. \(C_3\) is set to a one if the sign
of \(U\) is lost during left arithmetic shifting.

\(F\) logic:

\[0^f3 = T_s T_p (ASD) F_3\]

\[1^f4 = \sum_{i=1}^{8} (T'_{i}) (ASD) F_3\]

\[0^f4 = T_s T_p (ASD) F_4\]

\[1^f5 = \sum_{i=1}^{8} (T'_{i}) (ASD) F_3 + T_s T_p F_4\]
ASD uses $F_4$ only if the shift count is not zero at the end of $F_3$. In $F_5$, the D register copies the control counter.

Control counter recirculation:

$$C_{32} \leftarrow C_{17}; C_i \leftarrow C_{i+1}; i = 17, \ldots, 31$$
during $P_5 \text{(ASD)} F_5$

D register copy logic:

$$D_{16} \leftarrow C_{17}; D_i \leftarrow D_{i+1}; i = 1, \ldots, 15$$
during $P_5 \text{(ASD)} F_5$

F logic:

$$0^f_5 = T_s T_p \text{(ASD)} F_5$$
$$1^f_0 = T_s T_p \text{(ASD)} F_5$$

$F_5$ is the last state for ASD.

Normalize

If $C(U)C(X)_{1-31} = 0$ no operation

$C(X)_{32} := 0$

$$C(U)C(X)_{1-31} = C(U)C(X)_{1-31} \cdot 2^t$$

where $t$ is such that the new value in $U$ and $X$ is

$$1/2 \leq \left| C(U)C(X)_{1-31} \right| < 1$$

$C(V) := -t$

The contents of the $U$ register and the contents of bits 1 to 31 of the $X$ register are shifted left (or right for the special case -1) arithmetically until the absolute value of the number is greater than or
equal to one-half and less than one. If the register contains zero, the effect is no operation. The two's complement of the number of shifts is put in the \( V \) register (i.e. if the number of left shifts is three, then a minus three is put in the \( V \) register). Immediate mode has no effect.

Execution time: 500 \( \mu s \)

In \( F_3 \) the \( D \) register is set to all ones and \( J_1 \) and \( J_2 \) are set to zero.

D register logic:

\[
D_{16} \leftarrow 1; \quad D_i \leftarrow D_{i+1}; \quad i = 1, \ldots, 15 \text{ during } P' T' (\text{NRM})F_3
\]

\( J_1 \) and \( J_2 \) logic:

\[
J_1 \leftarrow 0; \quad J_2 \leftarrow 0 \text{ during } (\text{NRM})F_3
\]

F logic:

\[
f_3 = T_s T_p (\text{NRM})F_3
\]

\[
f_4 = T_s T_p (\text{NRM})F_3
\]

During \( F_4 \) and \( F_5 \) the double length number in \( U \) and \( X \) is shifted left until the two most significant bits disagree or there have been 64 shifts. The \( D \) register counts the number of shifts.

U register shift:

\[
U_i \leftarrow X \quad ; \quad U_i+1 \leftarrow U_i; \quad i = 1, \ldots, 31 \text{ during } (U'_{32} U'_{31} + U_{32} U_{31}) T' (\text{NRM})(F_4 + F_5)\]
X register shift:

\[
X_1 \rightarrow 0; \ X_{i+1} \rightarrow X_i \quad ; \; i = 1, \ldots, 31 \text{ during } (U'_{32} U'_{31} + U_{32} U_{31}) T'(NRM)(F_4 + F_5)
\]

\[
X_{32} \rightarrow 0 \text{ during } (NRM)F_4
\]

Enable D register count:

\[
EDC = (U'_{32} U'_{31} + U_{32} U_{31}) T'(NRM)(F_4 + F_5)
\]

For the complete D register count logic see the SLR section.

F logic:

\[
0'_{4} = T_s T_p (NRM)F_4
\]
\[
1'_{5} = T_s T_p (NRM)F_5
\]
\[
0'_{5} = T_s T_p (NRM)F_5
\]
\[
1'_{6} = T_s T_p (NRM)F_5
\]

In \( F_6 \) the U and X registers are checked to see if they contain the number minus one. Both U and X are recirculated. \( J_1 \) checks for a zero X register and \( J_2 \) checks for a minus one in the U register.

U register recirculation:

\[
U_{32} \leftarrow U_1; \ U_i \leftarrow U_{i+1} \quad ; \; i = 1, \ldots, 31 \text{ during } T'(NRM)F_6
\]

X register recirculation:

\[
X_{32} \leftarrow X_1; \ X_i \leftarrow X_{i+1} \quad ; \; i = 1, \ldots, 31 \text{ during } T'(NRM)F_6
\]
J₁ and J₂ logic:

\[ j₁ = X_1 T'(NRM)F \]

\[ j₂ = U_1 T'(NRM)F \]

\[ j₂ = U'_1 T T(NRM)F \]

\[ s_4 = T T(NRM)F \]

\[ s_4 = U T'(NRM)F \]

The term \( J₁ J₂ \) indicates that a minus one is in \( U \) and \( X \) in \( F_7 \). \( G^1 J^1 \) indicates a double length zero.

F logic:

\[ f_6 = T T(NRM)F \]

\[ f_7 = T T(NRM)F \]

\( F_7 \) is the clean-up state for NRM. Since, under most conditions, the D register has a count of one more than necessary (it is initialized with a minus one in \( F_3 \)), a one is added to the count. This corrected count is gated into the V register. A two is added to the D count if a minus one is in \( U \) and \( X \). If a zero is in \( U \) and \( X \), a zero is gated to the V register.

K logic:

\[ k = (J₁ + J₂)T' T(NRM)F \]

\[ k = (J^1 J^1)P' P' P' P' P' T'(NRM)F \]

\[ k = D'_1 T' F_7(NRM) \]
During $F_7$, $K$ is used for the half add to the count in the D register.

**V register shift:**

$$V_i \leftarrow V_{i+1}; \, i = 1, \ldots, 31 \text{ during } T'_p(NRM)F_7$$

**V register copy:**

$$V_{32} \leftarrow (D_1K' + D'_1K) \text{ during } (G_4 + J_1)T'_p(NRM)F_7$$

**D register shift:**

$$D_i \leftarrow D_{i+1}; \, i = 1, \ldots, 15 \text{ during } T'_F(NRM)$$

$$D_{16} \leftarrow D_{16} \text{ during } P'_5T'_p(NRM)F_7$$

$$D_{16} \leftarrow C_{17} \text{ during } P'_5(NRM)F_7$$

During the first half word time, $D_{16}$ copies itself spreading the sign, and during the last half it copies the control counter.

**Control counter recirculation:**

$$C_{32} \leftarrow C_{17}; \, C_i \leftarrow C_{i+1}; \, i = 17, \ldots, 31 \text{ during } P'_5(NRM)F_7$$

**F logic:**

$$f_7^0 = T'_s T'_p(NRM)F_7$$

$$f_0^1 = T'_s T'_p(NRM)F_7$$

$F_7$ is the last state for NRM.
Shift Double Length

This command uses bit 16 of the effective address to determine the direction of the shift. The command is explained in two parts below.

Shift Left Double:

For  \( k = 1 \) step 1 until A

\[
\begin{align*}
C(U)_{i+1} &:= C(U)_i/i = 1, \ldots, 31 \\
C(U)_1 &:= C(X)_{32} \\
C(X)_{i+1} &:= C(X)_i/i = 1, \ldots, 31 \\
C(X)_1 &:= 0
\end{align*}
\]

The contents of the U register and contents of the X register are shifted left, as a double length number, the number of places specified by the effective address (modulo \( 2^8 \)). Zeros are shifted in from the right. Immediate mode has no effect.

Execution time: 200 to 300 \( \mu s \)

Shift Right Double:

For  \( k = 1 \) step 1 until A

\[
\begin{align*}
C(U)_{31} &:= 0 \\
C(U)_1 &:= C(U)_{i+1}/i = 1, \ldots, 31
\end{align*}
\]
\[ C(X)_{32} := C(U)_{1} \]
\[ C(X)_{i} := C(X)_{i+1} / i = 1, \ldots, 31 \]

The contents of the \( U \) register and the contents of the \( X \) register are shifted, as a double length number, right the number of places specified by the effective address (modulo \( 2^8 \)). Zeros are shifted in from the left. Immediate mode has no effect.

Execution time: 200 to 300 \( \mu \)s

In \( F_3 \) and \( F_4 \) the \( U \) and \( X \) registers shift left if \( D_{16} \) is on or right if \( D_{16} \) is off. Bits 1 to 6 of the \( D \) register act as a down counter. The shifting occurs as long as there are any one bits in \( D_{1-8} \). If the shifting is completed in \( F_3' \), the \( F_4 \) state is skipped.

**U register right shift:**
\[
U_{32} \leftarrow 0; \quad U_i \leftarrow U_{i+1}; \quad i = 1, \ldots, 31 \quad \text{during} \quad D'_{16} (\sum D)_{i} T'_p (SFD)(F_3 + F_4)
\]

**X register right shift:**
\[
X_{32} \leftarrow U_1; \quad X_i \leftarrow X_{i+1}; \quad i = 1, \ldots, 31 \quad \text{during} \quad D'_{16} (\sum D)_{i} T'_p (SFD)(F_3 + F_4)
\]

**X register left shift:**
\[
X_1 \leftarrow 0; \quad X_{i+1} \leftarrow X_i; \quad i = 1, \ldots, 31 \quad \text{during} \quad D'_{16} (\sum D)_{i} T'_p (SFD)(F_3 + F_4)
\]

**U register left shift:**
\[
U_1 \leftarrow X_{32}; \quad U_{i+1} \leftarrow U_i; \quad i = 1, \ldots, 31 \quad \text{during} \quad D'_{16} (\sum D)_{i} T'_p (SFD)(F_3 + F_4)
\]

If either \( D_8 \) or \( D_7 \) are on, the \( U \) and \( X \) registers are cleared to zero.
Enable D register count logic:

$$EDC = T_p'(SFD)(F_3 + F_4)$$

F logic:

$$0^f_3 = T_s T_p(SFD)F_3$$

$$1^f_4 = T_s T_p(SFD)F_3$$

$$1^f_5 = [(SFD)T_s F_3 + T_s F_4](SFD)$$

SDF uses $F_4$ only if the shift count is not zero at the end of $F_3$. In $F_5$ the control counter is copied into the D register.

Control counter recirculation:

$$C_{32} \leftarrow C_{17}; C_i \leftarrow C_{i+1}; i = 17, \ldots, 31 \text{ during } P_5(SFD)F_5$$

D register copy logic:

$$D_{16} \leftarrow C_{17}; D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \text{ during } P_5(SFD)F_3$$

F logic:

$$0^f_5 = T_s T_p(SFD)F_5$$

$$1^f_0 = T_s T_p(SFD)F_5$$

$F_5$ is the last state for SFD.

Partial and Multiple Word Transfer Commands

Load Character

$$C(E)_{1-8} := C(A) \ldots \text{memory}$$

addressed as 8-bit words (Figure 8).
## Figure 8. Partial word addressing scheme.

<table>
<thead>
<tr>
<th>Full word</th>
<th>Left half</th>
<th>Right half</th>
<th>4th quarter</th>
<th>3rd quarter</th>
<th>2nd quarter</th>
<th>1st quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0001</td>
<td>0000</td>
<td>0003</td>
<td>0002</td>
<td>0001</td>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
<td>0003</td>
<td>0002</td>
<td>0007</td>
<td>0006</td>
<td>0005</td>
<td>0004</td>
</tr>
<tr>
<td>0002</td>
<td>0005</td>
<td>0004</td>
<td>000b</td>
<td>000a</td>
<td>0009</td>
<td>0008</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3fff</td>
<td>7fff</td>
<td>7ffe</td>
<td>ffff</td>
<td>ffde</td>
<td>ffdf</td>
<td>ffdf</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7fff</td>
<td>ffff</td>
<td>ffde</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Memory not addressed as quarter words
- Memory not addressed as half words
The character addressed by the effective address is loaded into the E register. Immediate mode has no effect.

Execution time: 200 µs

This command uses bits 3 to 16 of the D register to select a word (addresses 0 to 3fff instead of 0 to ffff). Bits 1 and 2 are used to select the proper quarter word. D2 off selects the right half word and D1 off selects the right quarter of the selected half word. For the complete selection see the E register copy logic below. To accomplish this selection the D register is shifted logically right by two in F3. The least significant bit of D is put into J1 and the next bit of D is put into J2. In F4 the D register can select the proper word; J1 and J2 can select the proper quarter word.

J1 and J2 logic:

\[ J_1 \leftarrow D_1 \text{ during } T'_s T_s (LCH)F_3 \]

\[ J_2 \leftarrow D_2 \text{ during } T'_s T_s (LCH)F_3 \]

D register shift:

\[ D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \text{ during } P_5(LCH)F_3 \]

D register recirculation:

\[ D_{16} \leftarrow D_3 \text{ during } P_5 T'_s (P_1 + P'_2 + P'_3 + P'_4)(LCH)F_3 \]

The above logic causes a right shift of two; D16 and D15 will be set to zero.
F logic:

\[ f_3^f = T_s T_p (LCH)F_3 \]
\[ f_4^f = T_s T_p (LCH)F_3 \]

\( F_4 \) is the state where the copy from memory takes place. All registers must recirculate in \( F_4 \) in order to be addressed as memory.

Register recirculation:

\[ U_{33} \leftarrow U_0; U_i \leftarrow U_{i+1}; i = 0, \ldots, 32 \text{ during } (LCH)F_4 \]
\[ X_{33} \leftarrow X_0; X_i \leftarrow X_{i+1}; i = 0, \ldots, 32 \text{ during } (LCH)F_4 \]
\[ V_{33} \leftarrow V_0; V_i \leftarrow V_{i+1}; i = 0, \ldots, 32 \text{ during } (LCH)F_4 \]
\[ C_{32} \leftarrow C_1; C_i \leftarrow C_{i+1}; i = 1, \ldots, 31 \text{ during } T'_p(LCH)F_4 \]

The B gate logic is the same as for \( F_3 \) except there is no immediate mode. B copies the contents of the effective address.

B gate:

\[ B = \{[U_0(L_9 L_8 L_7) + X_0(L_9 L_8 L_7) + C_1(L_9 L_8 L_7) + V_0(L_9 L_8 L_7)
\]
\[ + E_1(L_9 L_8 L_7)(P'_5 P'_4 T'_1)]H + RH'}(LCH)F_4 \]

E register copy:

\[ E_8 \leftarrow (BJ'_1 J'_1 + E_1(J_2 + J_1)) \text{ during } P'_5 P'_4 T'_p(LCH)F_4 \]
\[ E_8 \leftarrow (BJ'_1 J'_1 + E_1(J_2 + J'_1)) \text{ during } P'_5 P'_4 (LCH)F_4 \]
The E register either recirculates or copies B in each quarter word time depending on the contents of J2 and J1.

D register copy:

\[ D_{16} \leftarrow C_1; D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \] during \( P_5 \) \( (LCH)F_4 \)

The D register copies the control counter from \( C_1 \).

F logic:

\[ 0^f_4 = T_s T_p (LCH)F_4 \]
\[ 1^f_0 = T_s T_p (LCH)F_4 \]

\( F_4 \) is the last state for LCH.

Load Half Word

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>C(U)1-16 := C(A)...memory</td>
<td>LHW</td>
</tr>
</tbody>
</table>

addressed as 16-bit words (Figure 8).

\[ C(U)_{17-32} := 0 \]

Half word specified by the effective address is loaded to the right half of the U register; the left half is set to zero. Immediate mode has no effect.

Execution time: 200 μs

This command uses bits 2 to 16 of the D register to select a
word (addresses 0 to 7fff instead of 0 to ffff). Bit 1 is used to select the proper half word. If \( D_1 \) is off it selects the right half, and if \( D_1 \) is on it selects the left half. To accomplish this selection the \( D \) register is shifted logically right by one in \( F_3 \). The least significant bit of \( D \) is put into \( J_1 \). Thus, in effect, \( D \) can select the proper word in \( F_4 \), and \( J_1 \) selects the timing for the proper half word.

\[
\text{J}_1 \text{ logic:}
\]

\begin{align*}
\text{J}_1 &\rightarrow D_1 \text{ during } T' \text{ T_s' P} (\text{LHW}) \text{F}_3 \\
\text{D register shift:}
\end{align*}

\begin{align*}
D_i &\rightarrow D_{i+1} \text{ during } P_5 \text{ (LHW) F}_3 \\
\text{D register recirculation:}
\end{align*}

\begin{align*}
D_{16} &\rightarrow D_2 \text{ during } P_5 \text{ T_s' (LHW) F}_3 \\
\text{The above logic causes a right shift of one, and } D_{16} \text{ will be set to zero.}
\end{align*}

\[
\text{F logic:}
\]

\begin{align*}
f_{3} &= T \text{ T_s' P} (\text{LHW}) \text{F}_3 \\
f_{4} &= T \text{ T_s' P} (\text{LHW}) \text{F}_3 \\
\end{align*}

\( F_4 \) is the state where the copy from memory takes place. All registers must recirculate in \( F_4 \) in order to be addressed as memory.
Register recirculation:

\[ X_{33} ← X_0; X_i ← X_{i+1}; i = 0, \ldots, 32 \text{ during } (LHW)F_4 \]

\[ V_{33} ← V_0; V_i ← V_{i+1}; i = 0, \ldots, 32 \text{ during } (LHW)F_4 \]

\[ C_{32} ← C_1; C_i ← C_{i+1}; i = 1, \ldots, 31 \text{ during } T_p(LHW)F_4 \]

\[ E_8 ← E_1; E_i ← E_{i+1}; i = 1, \ldots, 7 \text{ during } P_5P' T'(LHW)F_4 \]

The B gate logic is the same as for \( F_3 \) except there is no immediate mode. B copies the contents of the effective address.

B gate:

\[ B = \{[U_0 (L^9 L^8 L^7) + X_0 (L^9 L^8 L^7) + C_1 (L^9 L^8 L^7) + V_0 (L^9 L^8 L^7) + \]

\[ E_1 (L^9 L^8 L^7) \} [P_5 P' T'(LHW)F_4] + RH'\} \]

Upper U register zeroing logic:

\[ U_{32} ← 0; U_i ← U_{i+1}; i = 17, \ldots, 31 \text{ during } P_5(LHW)F_4 \]

Lower U register copy:

\[ U_{16} ← (BJ'+U_{16} J_1) \text{ during } P_5 P'(LHW)F_4 \]

\[ U_{16} ← (BJ_1 + U_{16} J'_1) \text{ during } P_5(LHW)F_4 \]

\[ U_i ← U_{i+1}; i = 1, \ldots, 15 \text{ during } T_p(LHW)F_4 \]

Bits 1 to 16 of the U register either recirculate or shift in the proper half word depending on the value of \( J_1 \).

D register copy logic:

\[ D_{16} ← C_1; D_i ← D_{i+1}; i = 1, \ldots, 15 \text{ during } P_5(LWH)F_4 \]
The D register copies the control counter from \( C_1 \) in \( F_4 \).

\[
F \text{ logic:}
\]
\[
0^*_{F_4} = T_s T_p (LHW) F_4
\]
\[
1^*_{F_0} = T_s T_p (LHW) F_4
\]

\( F_4 \) is the last state for LHW.

**Store Character**

\[ C(A) := C(E) \ldots \text{memory} \]

addressed as 8-bit words (Figure 8).

The contents of the E register are stored into the character specified by the effective address. Immediate mode has no effect.

Registers cannot be addressed.

**Execution time:** 200 \( \mu s \)

Memory is addressed the same way as for LCH with one exception. The register may not be addressed.

\[
J_1 \text{ and } J_2 \text{ logic:}
\]
\[
J_1 \leftarrow D_1 \text{ during } T'_s T_p (SCH) F_3
\]
\[
J_2 \leftarrow D_2 \text{ during } T'_s T_p (SCH) F_3
\]

\[ D \text{ register shift:}
\]
\[
D_i \leftarrow D_{i+1}; \ i = 1, \ldots, 15 \text{ during } P_{5}(SCH) F_3
\]
D register recirculation:

\[ D_{16} \leftarrow D_3 \text{ during } P_5 T'(P_1 + P_2 + P_3 + P_4)(\text{SCH})F_4 \]

The above logic causes a right shift of two and sets \( D_{16} \) and \( D_{15} \) to zero. \( J_1 \) copies \( D_1 \) and \( J_2 \) copies \( D_2 \) which in turn select the specified quarter word.

F logic:

\[ f_3^0 = T_s T_p (\text{SCH})F_3 \]
\[ f_4^1 = T_s T_p (\text{SCH})F_3 \]

\( F_4 \) is the state where the store into memory takes place.

E register recirculation:

\[ E_8 \leftarrow E_1 \quad E_i \leftarrow E_{i+1} \; i = 1, \ldots, 7 \text{ during } T'_p (\text{SCH})F_4 \]

The E register recirculates four times during \( F_4 \). The write line \((W)\) copies \( E_1 \) and the write enable line \((WE)\) is set to one for the quarter word selected by \( J_1 \) and \( J_2 \).

Write logic:

\[ W = E_1 T'_p (\text{SCH})F_4 \]
\[ WE = [(J'_1 J'_1)'(P'_5 P'_4 T'_1) + (J'_2 J'_1')(P'_5 P'_4) + (J_2 J'_1)(P'_5 P'_4)](\text{SCH})F_4 \]

Control counter recirculation:

\[ C_{32} \leftarrow C_{17} \; C_1 \leftarrow C_{i+1} \; i = 17, \ldots, 31 \text{ during } P_5 (\text{SCH})F_4 \]
D register copy:

\[ D_{16} \leftarrow C_{17}; \ D_i \leftarrow D_{i+1}; \ i = 1, \ldots, 15 \text{ during } P_5(SCH)F_4 \]

F logic:

\[ f_4 = T_s T_p(SCH)F_4 \]
\[ f_0 = T_s T_p(SCH)F_4 \]

\( F_4 \) is the last state for SCH.

Store Half Word

\[ C(A) = C(U)_{1-16} \ldots \text{memory} \]
addressed as 16-bit words (Figure 8).

The contents of bits 1-16 of the U register are stored into the half word specified by the effective address. Immediate mode has no effect. Registers cannot be addressed.

Execution time: 200 µs

Memory is addressed the same way as for LHW with one exception. The registers may not be addressed.

J₁ logic:

\[ J_1 \leftarrow D_1 \text{ during } T'_s T_p(SHW)F_3 \]

D register shift:

\[ D_i \leftarrow D_{i+1}; \ i = 1, \ldots, 15 \text{ during } P_5(SHW)F_3 \]

D register recirculation:

\[ D_{16} \leftarrow D_2 \text{ during } P_5 T'_s(SHW)F_3 \]
The above logic causes a right shift of one and $D_{16}$ to be set to zero. $J_1$ copies $D_1$, which specifies left half if on and right half if off.

F logic:

\[
0^f_3 = T_s T_p (SHW)F_3
\]

\[
1^f_4 = T_s T_p (SHW)F_3
\]

$F_4$ is the state where the store into memory takes place.

Lower half of U register recirculation:

\[
U_{16} \leftarrow U_1; U_i \leftarrow U_{i+1}; i = 1, \ldots, 15 \text{ during } T'_p (SHW)F_4
\]

The lower half of the U register recirculates twice during $F_4$.

The write line $(W)$ copies $U_1$ and the write enable line $(WE)$ is set to a one for the half word selected by $J_1$.

Write logic:

\[
W = U_1 T'_p (SHW)F_4
\]

\[
WE = (J'_p 1^p 5 + J 1^p 5) (SHW)F_4
\]

Control counter recirculation:

\[
C_{32} \leftarrow C_{17}; C_i \leftarrow C_{i+1}; i = 17, \ldots, 31 \text{ during } P_5 (SHW)F_4
\]

D register copy:

\[
D_{16} \leftarrow C_{17}; D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \text{ during } P_5 (SHW)F_4
\]
F logic:

\[ f_4^T = T_s T_p (SHW) F_4 \]

\[ f_0^T = T_s T_p (SHW) F_4 \]

\[ F_4 \] is the last state for SHW.

Reset Flags

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>RST</td>
<td>65</td>
</tr>
</tbody>
</table>

If \( A_i = 1 \) then \( C(C)_i := 0 / i = 1, \ldots, 16 \)

The logical complement of each bit in the effective address is
ANDed with the corresponding bit in the C register (bits 1-16). Immediate mode has no effect.

Execution time: 100 \( \mu \)s

The lower half of the C register is shifted and \( C_{16} \) copies
the logical product of \( C_1 \) and \( D'_1 \). The D register will be recirculating at this time.

Reset flags logic:

\[ C_{16} \leftarrow (C_1 D'_1); C_i \leftarrow C_{i+1}; i = 1, \ldots, 15 \text{ during } P_5^T (RST) F_3 \]

Control counter recirculation:

\[ C_{32} \leftarrow C_{17}; C_i \leftarrow C_{i+1}; i = 1, \ldots, 15 \text{ during } P_5 (RST) F_3 \]

D register copy:

\[ D_{16} \leftarrow C_{17}; D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \text{ during } P_5 (RST) F_3 \]

The D register is loaded with the address of the next instruction.
F logic:

\[ 0^f_3 = T_s T_p (RST)F^3 \]
\[ 1^f_0 = T_s T_p (RST)F^3 \]

\[ F^3 \] is the last state for RST.

Set Flags

If \( A_i = 1 \) then \( C(C)_i := 1/i = 1, \ldots, 16 \)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>64</td>
</tr>
</tbody>
</table>

Each bit of the effective address is ORed with the corresponding bit in the C register (bits 1-16). Immediate mode has no effect.

Execution time: 100 \( \mu s \)

The lower half of the C register is shifted and \( C_{16} \) copies the logical sum of \( C_1 \) and \( D_1 \). The D register will be recirculating at this time.

Set flag logic:

\[ C_{16} \leftarrow (C_1 + D_1); C_i \leftarrow C_{i+1}; i = 1, \ldots, 15 \] during \( P^5 T^5 (SET)F^3 \)

Control counter recirculation:

\[ C_{32} \leftarrow C_{17}; C_i \leftarrow C_{i+1}; i = 17, \ldots, 31 \] during \( P^5 (SET)F^3 \)

D register copy:

\[ D_{16} \leftarrow C_{17}; D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \] during \( P^5 (SET)F^3 \)

The D register is loaded with the address of the next instruction.
F logic:

\[ f_3 = T_s T_p (SET) F_3 \]

\[ f_0 = T_s T_p (SET) F_3 \]

\( F_3 \) is the last state for SET.

Store Double Length

\[
\begin{array}{l}
C(A): = C(U) \\
C(A+1): = C(X)
\end{array}
\]

The contents of the U register are stored at the location specified by the effective address. The contents of the X register are stored at the location specified by the effective address plus one. Immediate mode has no effect. Command will not store in locations 0000\(_{16}\) to 0007\(_{16}\).

Execution time: 200 \( \mu s \)

During \( F_3 \) the U register recirculates and is put on the write line \( (W) \). The D register is counted up by one in preparation to store the X register.

U register recirculation:

\[ U_{33} \leftarrow U_0; U_{1} \leftarrow U_{i+1}; i = 0, \ldots, 32 \] during \( (STD)F_3 \)

Write logic:

\[ WE = (STD)F_3 \]

\[ W = U_0 (STD)F_3 \]
D register half-add:

\[ D_{16} \leftarrow (D_1 K' + D_1 K); \quad D_i \leftarrow D_{i+1}; \quad i = 1, \ldots, 15 \text{ during } P_5 (STD) F_3 \]

K logic:

\[ \begin{align*}
1^k &= P_5' T_5' (STD) F_3 \\
0^k &= D_1' P_5 (STD) F_3 
\end{align*} \]

The K flip-flop is used as the carry in the half-add to the D register.

F state logic:

\[ \begin{align*}
0^f_3 &= T_s T_p (STD) F_3 \\
1^f_4 &= T_s T_p (STD) F_3 
\end{align*} \]

During the \( F_4 \) state the X register part of the double length number is stored.

X register recirculation:

\[ \begin{align*}
X_{33} &\leftarrow X_0; \quad X_i \leftarrow X_{i+1}; \quad i = 0, \ldots, 32 \text{ during } (STD) F_4 
\end{align*} \]

Write logic:

\[ \begin{align*}
WE &= (STD) F_4 \\
W &= X_0 (STD) F_4 
\end{align*} \]

The X register is recirculated and stored in the address specified by the D register. The D register contains the effective address plus one.
Control counter recirculation:

\[ C_{32} \leftarrow C_{17}; \ C_i \leftarrow C_{i+1}; \ i = 17, \ldots, 31 \text{ during } P_5\text{(STD)}F_4 \]

D register copy:

\[ D_{16} \leftarrow C_{17}; \ D_i \leftarrow D_{i+1}; \ i = 1, \ldots, 15 \text{ during } P_5\text{(STD)}F_4 \]

F state logic:

\[ f_4 = T_s T_p \text{(STD)}F_4 \]

\[ f_0 = T_s T_p \text{(STD)}F_4 \]

\( F_4 \) is the last state for STD.

Transfer E Register

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRE</td>
<td>a2</td>
</tr>
</tbody>
</table>

Bits 1 to 12 of the effective address of the TRE command select 12 different I/O devices, one bit per device. Bits 13 to 16 of the effective address control the event pulses for the transfer of information or the control of the selected I/O device. The event pulses occur sequentially (EV\(_1\) then EV\(_2\), etc.).

Execution time: 100 \( \mu \)s

The logic for TRE performs three functions: 1) timing for the event pulses \( \text{(EV}_i\)\); 2) the device gates \( \text{(DV}_i\) and 3) the conditional skip logic.

The conditional skip comes from the I/O device selected. For
example, a wait loop could be programmed which will be skipped when
the device is ready to receive a character (Figure 9).

Event pulse logic:

\[ EV_1 = D_13 P^3 T^3 (TRE)F_3 \]
\[ EV_2 = D_14 P^4 P^3 P^2 P_1 (TRE)F_3 \]
\[ EV_3 = D_15 P^5 P^4 P^3 P_1 (TRE)F_3 \]
\[ EV_4 = D_16 P^6 P^5 P^4 P_1 (TRE)F_3 \]

The \( D \) register bits 13 to 16 control whether or not a given
event pulse will occur. \( EV_1 \) occurs at bit time 0, \( EV_2 \) occurs at
bit time 1, \ldots, \( EV_4 \) occurs at bit time 3.

Device gate logic:

\[ DV_i = D_i P^i (TRE)F_3; \ i = 1, \ldots, 12 \]

Skip logic:

\[ ^1_k = P^5 (SK)(TRE) \]
\[ ^0_k = C_1 P F_3 (TRE) \]

\( K \) is used as the carry for the half-add to the control counter.
The skip pulse \( (SK) \) comes from an external device.

Control counter half-add:

\[ C_32 \leftarrow (C_1 K^i + C_1 K); C_i \leftarrow C_{i+1}; i = 17, \ldots, 31 \text{ during } P_5 (TRE)F_3 \]

D register copy:

\[ D_16 \leftarrow (C_1 K^i + C_1 K); D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \text{ during } P_5 (TRE)F_3 \]
Device ready?

No

Jump back

Yes

Send character

Device sends skip pulse

Figure 9. I/O skip flow chart.
F logic:

\[ 0^f_3 = T_s T_p (TRE) F_3 \]
\[ 1^f_0 = T_s T_p (TRE) F_3 \]

\( F_3 \) is the last state for TRE.

Sequence Control Commands

**Execute**

<table>
<thead>
<tr>
<th>C(IR): = C(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mnemonic</strong></td>
</tr>
<tr>
<td>EXT</td>
</tr>
</tbody>
</table>

The instruction located at the effective address is executed.

Immediate mode has no effect. Normal sequencing will not occur if a successful branch instruction is executed.

Execution time: 0 to 100 µs

If bits 5 to 8 of the M register are zero, EXT will be executed in \( F_0 \). For the \( F_0 \) logic see the Instruction Fetch section. Since the control counter is not loaded into the D register, the address of EXT is used to fetch the next instruction. The contents of the control counter remains unchanged so that the sequencing of instructions is unaltered except for successful branch instructions.

F state logic:

\[ 0^f_3 = T_s T_p F_3 (EXT) \]
\[ 1^f_0 = T_s T_p F_3 (EXT) \]
All that happens in \( F_3 \) is a return to \( F_0 \).

**Jump**

\[ C(C)_{17-32} := A \]

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>60</td>
</tr>
</tbody>
</table>

Control is transferred to the location specified by the effective address.

Execution time: 0 to 100 \( \mu s \)

If bits 5 to 8 of the M register are zero, JMP will be executed in \( F_0 \). For the \( F_0 \) logic see the Instruction Fetch section. During \( F_3 \), the control counter is loaded with the effective address.

**D register recirculation:**

\[ D_{16} \rightarrow D_1; D_{i} \rightarrow D_{i+1}; i = 1, \ldots, 15 \] during \( P_5(JMP)F_3 \)

**Control counter copies \( D_1 \) logic:**

\[ C_{32} \leftarrow D_1; C_{i} \leftarrow C_{i+1}; i = 17, \ldots, 32 \] during \( P_5(JMP)F_3 \)

**F logic:**

\[ 0^f_3 = T_s T_p (JMP)F_3 \]

\[ 1^f_0 = T_s T_p (JMP)F_3 \]

The jump instruction will be finished in \( F_3 \) if it is not finished in \( F_0 \).
Jump and Store C

C(A) := C(C)

C(C)_{17-32} := A + 1

The contents of the C register are stored into the location specified by the effective address and control is transferred to the location specified by the effective address plus one.

Execution time: 200 \mu s

JSC does nothing in F_3 at present. When the interrupt scheme is implemented, F_3 will set the effective address for JSC caused by an interrupt (explained in the Interrupt section).

F logic:

\[ f'_3 = T_s T_p (JSC) F_3 \]

\[ f'_4 = T_s T_p (JSC) F_3 \]

In F_4 the C register recirculates during the first half word time and copies the D register plus one during the last half word time.

The carry is K.

K logic:

\[ k = T'_s T_p (JSC) F_4 \]

\[ k = D'_1 P_5 (JSC) F_4 \]

C register shift logic:

\[ C_i \leftarrow C_{i+1}; \quad i = 1, \ldots, 31 \text{ during } T'_p (JSC) F_4 \]
C<sub>32</sub> copy logic:
\[ C_{32} \leftarrow [C_1 P_1 T' + (D_1 K' + D_1' K) P_5] \text{ during } T_p (JSC) F_4 \]

D register shift logic:
\[ D_1 \leftarrow D_{i+1}; i = 1, \ldots, 15 \text{ during } P_5 (JSC) F_4 \]

D register half add logic:
\[ D_{16} \leftarrow (D_1 K' + D_1' K) \text{ during } P_5 (JSC) F_4 \]

During \( F_4 \) the write line \( (W) \) copies the C register.

Write logic:
\[ W = C_1 T'_p (JSC) F_4 \]
\[ WE = T'_p (JSC) F_4 \]

F logic:
\[ f'_4 = T_s T_p (JSC) F_4 \]
\[ f'_0 = T_s T_p (JSC) F_4 \]

\( F_4 \) is the last state for JSC.

Load Left Half of C
\[ C(C)_{17-32} = C(A)_{17-32} \]

The left half of the contents of the effective address are placed in the contents of the left half of the C register. This command acts as a one level indirect jump (Chapter X).
Execution time: 100 µs

The B gate has the left half of the contents of the effective address coming through during $P_5$ on. B is gated into both the control counter and the D register causing an effective jump.

Control register copy logic:

\[ C_{32} \leftarrow B; C_i \leftarrow C_{i+1}; i = 17, \ldots, 31 \] during $P_5(LLC)F_3$

\[ D_{16} \leftarrow B; D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \] during $P_5(LLC)F_3$

$F$ state logic:

\[ f_3^0 = T_s T_p (LLC)F_3 \]

\[ f_0^1 = T_s T_p (LLC)F_3 \]

$F_3$ is the last state for LLC.

No Operation

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>00</td>
</tr>
</tbody>
</table>

The modifier field has no effect. The instruction itself has no effect other than its use of an instruction fetch time.

Execution time: 0 µs

For the logic of this command see the Instruction Fetch section.

NOP never enters an F state above $F_0$. 
Skip if True

\[ \sum_{i=1}^{16} (C_i \land A_i) = 1 \]

then \( C(C)_{17-32} = C(C)_{17-32} + 1 \)

If any bits of \( C_{1-16} \) masked by the effective address are ones, the control counter is incremented by one. Immediate mode has no effect.

Execution time: 100 \( \mu \)s

Both the lower half of the C register and the D register recirculate during the first half word time in \( F_3 \).

Lower half recirculation of C:

\[ C_{i+1} \leftarrow C_i; \ C_i \leftarrow \ C_i+1; \ i = 1, \ldots, 15 \text{ during } \text{P'} \text{T'} (SKT)F_3 \]

The D register is recirculating in the first half of \( F_3 \) for the possibility of immediate mode operations.

Carry \( K \) is turned on if \( C_1 \) ANDed with \( D_1 \) results in a one at any time during the first half word time \( (\text{P'} \text{T'}) \). During the last half word time \( K \) is used for a half add to the control counter.

K logic:

\[ k = D_1 \ C_1 \ P' \ T' \ (SKT)F_3 \]

\[ 0^k = C'_{17} \ P_5 \ (SKT)F_3 \]
Control counter half add:

\[ C_{32} \leftarrow (C_{17}^{i} + C_{17}^{i}) : C_{i} \leftarrow C_{i+1} ; i = 1, \ldots, 15 \text{ during } P_5 (SKT) F_3 \]

D register copy:

\[ D_{16} \leftarrow (C_{17}^{i} + C_{17}^{i}) : D_{i} \leftarrow D_{i+1} ; i = 1, \ldots, 15 \text{ during } P_5 (SKT) F_3 \]

If the proper conditions are met in the first half of \( F_3 \), a one is added to the control counter causing a skip. If \( K \) is off there is no skip and normal sequencing proceeds.

F logic:

\[
\begin{align*}
0^f_3 &= TST (SKT) F_3 \\
1^f_0 &= TST (SKT) F_3
\end{align*}
\]

\( F_3 \) is the last state for SKT.

Skip if False

If \( \prod_{i=1}^{16} (C^{i}VA^{i}) = 1 \)

then \( C(C)_{17-32} := C(C)_{17-32} + 1 \)

If all the bits of \( C_{1-16} \) masked by the effective address are zero, then the control counter is incremented by one. Immediate mode has no effect.

Execution time: 100 \( \mu \)s

Both the lower half of the C register and the D register recirculate during the first half word time in \( F_3 \).
Lower half recirculation of C:

\[ C_{16} \leftarrow C_i; C_i \leftarrow C_{i+1}; i = 1, \ldots, 15 \text{ during } P'_5 T'_5 \text{ (SKF)} F_3 \]

The D register is recirculating in the first half of \( F_3 \) for the possibility of immediate mode operations.

Carry (K) is turned on at the start of \( F_3 \) and will be turned off if \( D_1 \) ANDed with \( C_1 \) results in a one during the first half word time \( (P'_5 T'_5) \). During the last half word time \( K \) is used for a half add to the control counter.

K logic:

\[ k^1 = T'_s T'_p \text{ (SKF)} F_3 \]

\[ k^0 = [(D_1 C_1 P'_5 T'_5 + C'_1 7 P_5)] \text{ (SKF)} F_3 \]

Control counter half add:

\[ C_{32} \leftarrow (C_{17} K'_i + C'_1 K); C_i \leftarrow C_{i+1}; i = 17, \ldots, 32 \text{ during } P_5 \text{ (SKF)} F_3 \]

D register copy:

\[ D_{16} \leftarrow (C_{17} K'_i + C'_1 K); D_i \leftarrow D_{i+1}; i = 1, \ldots, 15 \text{ during } P_5 \text{ (SKF)} F_3 \]

If the proper conditions are met in the first half of \( F_3 \), a one is added to the control counter causing a skip. If \( K \) is off, there is no skip and normal sequencing proceeds.

F logic:

\[ f^3 = T_s T_p \text{ (SKF)} F_3 \]

\[ f^0 = T_s T_p \text{ (SKF)} F_3 \]

\[ F_3 \] is the last stats for SKF.
Interrupt

The interrupt scheme presently implemented on NEBULA is about the simplest possible. An interrupt can be caused by an external device such as a switch. If the interrupt is recognized, it will cause a JSC #10, which stores the C register at location 10_{16} and starts executing instructions at location 11_{16}. Whether or not an interrupt will be recognized is determined by a mask register \((MK_i; i = 1, \ldots, 4)\) and an enable-interrupt flip-flop \((ENIT)\).

Recognize interrupt logic:

\[
\text{rux} = (ENINT)(\sum_i (MK_i)(INT_i))((EXT)'T'F)_{S \ p \ 0}
\]

\[
f_3 = f_0 = (ENINT)(\sum_i (MK_i)(INT_i))((EXT)'T'F)_{S \ p \ 0}
\]

There are four possible interrupt lines \((INT_i; i = 1, \ldots, 4)\). Only \(INT_2\) has been assigned, and this assignment is the manual interrupt switch \((MI)\).

JSC to O register logic:

\[
O_i \leftarrow 1; \ i = 2, 3, 6, 7 \text{ during } (ENINT)(\sum_i (MK_i)(INT_i))((EXT)'T'F)_{S \ p \ 0}
\]

\[
O_i \leftarrow 0; \ i = 1, 4, 5, 8 \text{ during } (ENINT)(\sum_i (MK_i)(INT_i))((EXT)'T'F)_{S \ p \ 0}
\]

The RUX flip-flop is turned on only if an interrupt is recognized. The command 66_{16} is loaded into the O register and the computer goes immediately to \(F_3\).
Enable interrupt off logic:

\[ 0_{\text{enint}} = T_s T_p (\text{RUX})(\text{JSC})F_3 \]

ENINT is turned off if an interrupt is recognized. In \( F_3 \) of JSC the D register is loaded with 10\(_{16}\).

D register interrupt load:

\[ D_{16} \leftarrow 0; \quad D_i \leftarrow D_{i+1}; \quad i = 1, \ldots, 15 \text{ during } P_5(RUX)(JSC)F_3 \]

\[ D_5 \leftarrow 1 \text{ during } T_s T_p (\text{RUX})(\text{JSC})F_3 \]

The \( F_4 \) state of JSC is the same as a programmed JSC except RUX is turned off.

RUX logic:

\[ 0_{\text{rux}} = T_s T_p (\text{JSC})F_4 \]

There are three commands which give the programmer control over the interrupt scheme: SIM, RIM, and SIT.

Reset Interrupt Mask

\[ C(MK)_i := C(MK)_i \land \overline{A_i}/i = 1, \ldots, 4 \]

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Hexadecimal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIM</td>
<td>c6</td>
</tr>
</tbody>
</table>

The first four bits of the effective address put zeros in the mask where ones appear in the effective address.

Execution time: 100 \( \mu \)s

Mask reset logic:

\[ 0_{\text{mk}}_i = D_i T_s T_p (\text{RIM})F_3; \quad i = 1, \ldots, 4 \]
Control counter logic:

\[ C_{32} \leftarrow C_{17}; \ C_i \leftarrow C_{i+1}; \ i = 17, \ldots, 31 \] during \( P_5(RIM)F_3 \)

\[ D_{16} \leftarrow C_{17}; \ D_i \leftarrow D_{i+1}; \ i = 1, \ldots, 15 \] during \( P_5(RIM)F_3 \)

F logic:

\[ f_3 = T_s T_p (RIM)F_3 \]

\[ f_0 = T_s T_p (RIM)F_3 \]

\( F_3 \) is the last state for RIM.

Set Interrupt Mask

\[ C(MK)_i := C(MK)_i \lor A_i/i = 1, \ldots, 4 \]

Mnemonic | Hexadecimal
---|---
SIM | c4

The first four bits of the effective address are ORed with the contents of the mask.

Execution time: 100 \( \mu s \)

Mask set logic:

\[ \text{mk}_i = D_i T_s T_p (RIM)F_3; \ i = 1, \ldots, 4 \]

Control counter logic:

\[ C_{32} \leftarrow C_{17}; \ C_i \leftarrow C_{i+1}; \ i = 17, \ldots, 31 \] during \( P_5(SIM)F_3 \)

\[ D_{16} \leftarrow C_{17}; \ D_i = D_{i+1}; \ i = 1, \ldots, 15 \] during \( P_5(SIM)F_3 \)
F logic:

$0^f_3 = T_s T_p (SIM) F_3$

$1^f_0 = T_s T_p (SIM) F_3$

$F_3$ is the last state for SIM.

Set Interrupt Enable Toggle

The enable interrupt flip-flop is loaded with the contents of the first bit of the effective address. There is a delay on the set side to allow the next command to be fetched before the set takes place.

Execution time: 100 µs

ENINT logic:

$0^{enint} = D_1 T_s T_p (SIT) F_3$

$1^{norv} = D_1 T_s T_p (SIT) F_3$

$0^{norv} = P_5 (NORV) F_0$

$1^{enint} = P_5 (NORV) F_0$

The NORV flip-flop provides the necessary delay for the set side of ENINT.

Control counter logic:

$C_{32} \leftarrow C_{17}; C_i \leftarrow C_{i+1}; i = 17, \ldots, 31$ during $P_5 (SIT) F_3$

$D_{16} \leftarrow C_{17}; D_i \leftarrow D_{i+1}; i = 1, \ldots, 15$ during $P_5 (SIT) F_3$
F logic:

\[ f'_3 = T_s T_p (SIT)F_3 \]

\[ f'_4 = T_s T_p (SIT)F_3 \]

\( F_3 \) is the last state for SIT.
IX. DISPLAY LOGIC

Every computer designed has some sort of console that will display registers. Since NEBULA will constantly have logic changes, the console display system needed to be all inclusive. NEBULA's display console is a compromise between an elaborate and a simple register display. Basically, all registers and memory can be displayed and changed from the console. When the clock to the processor is stopped, the display of flip-flops is a matter of attaching a lamp on the output. This type of display is called static display. All the registers and flip-flops have a static display hook-up. When the processor clock is running full speed other provisions need to be made.

To provide an active display a 34-bit display shift register (DSR) is used. The DSR is static except for the loading or storing action which takes one processor word time. Each bit in the DSR has an indicator light and a three position switch for turning DSR flip-flops on and off. There are seven push buttons which select a register or memory (Figure 10). When one of these buttons is pushed, the contents of the selected register are shifted into the DSR. The register stays selected until another one of the seven buttons is pushed again. There are two function buttons. One clears the DSR to zero (labeled CL in Figure 10); the second buttons stores the DSR into the
Figure 10. Register display console.
selected register (labeled ST in Figure 10).

The processor itself has a display state associated with every F state. NEBULA's display state causes the computer to recirculate all registers at the end of any F state and hold up program execution until released from the display state. The display state is indicated by the flip-flop DIS. The on side of DIS is ANDed with the output of the on side and the off side of all the F state flip-flops. For purposes of notation, the actual F state flip-flops will appear below (and only below) as \( \Phi_i \). The F's will be the output of the gates.

F state display logic:

\[
F_i = (DIS)(\Phi_i); \ i = 0, \ldots, 9
\]

The function switch called "store" turns a flip-flop called STORE on for one word time.

STORE logic:

\[
1^{sc} = (STORE)T_s T_p (DIS)
\]

\[
0^{sc} = (\text{store switch off})
\]

\[
1^{store} = (\text{store switch on})(\overline{SC})T'_s T_p (DIS)
\]

\[
0^{store} = (SC)T'_s T_p (DIS)
\]

The flip-flop SC turns STORE off after one word time. Starting with both STORE and SC off and DIS on, the store switch will turn STORE on at the beginning of a word time. Then at the end of this word time, SC will turn on which in turn turns STORE off. Both SC
and STORE are direct set flip-flops. The store switch off will turn SC off allowing the cycle to be repeated.

The logic scheme that controls the storing function of the DSR is very similar to the loading function logic. Any one of the seven select switches can act as a load switch.

LOAD logic:

\[
\begin{align*}
1_{\text{lc}} &= (\text{LOAD})T_{s}T_{p}(\text{DIS}) \\
0_{\text{lc}} &= (\text{load switch off})
\end{align*}
\]

\[
\begin{align*}
1_{\text{load}} &= (\text{load switch on}) (LC)'T_{s}T_{p}(\text{DIS}) \\
0_{\text{load}} &= (LC)'T_{s}T_{p}(\text{DIS})
\end{align*}
\]

The flip-flop LC mimics the action of SC, and LOAD the action of STORE.

The flip-flops of DSR will be labeled here as DSR\(_i\), where \(i = 0, \ldots, 33\).

DSR shift logic:

\[
\begin{align*}
\text{DSR}_{i} &\rightarrow \text{DSR}_{i+1}; \text{ for } i = 0, \ldots, 32 \text{ during } [(\text{LOAD})+(\text{STORE})](\text{DIS})
\end{align*}
\]

DSR copy logic:

\[
\begin{align*}
\text{DSR}_{33} &\leftarrow \left[ U_0(CU)+X_0(CX)+V_0(CV)+F_1(CE)P_5P_4T_p' \\
&\quad + C_1(CC)'T_p' + O_1(CIR)'T_p \right]; \text{ during } (\text{LOAD})(\text{DIS})
\end{align*}
\]

\[
\begin{align*}
\text{DSR}_{33} &\rightarrow \text{DSR}_{0}; \text{ during } (\text{STORE})(\text{DIS})
\end{align*}
\]
DIS logic:

\[ \text{disc}_1 = (\text{Control}_1) T_s T_p \]
\[ \text{disc}_0 = (\text{Control}_2) T_s T_p \]

The DIS flip-flop may be turned on and off only at the end of a word time. Control_1 is the input from console switches to turn DIS on and Control_2 is the input from the console switches to turn DIS off.

All seven display console selection switches are treated here as flip-flops. The names used are comprised of two letters; the first letter is C, the second letter may be one of the following: C, E, IR, M, U, V, or X, (i.e. CM stands for the switch that selects memory and CC stands for the switch that selects the C register).

Register shift logic:

\[ U_i \leftarrow U_{i+1} ; i = 0, \ldots , 32 \text{ during (DIS)} \]
\[ X_i \leftarrow X_{i+1} ; i = 0, \ldots , 32 \text{ during (DIS)} \]
\[ V_i \leftarrow V_{i+1} ; i = 0, \ldots , 32 \text{ during (DIS)} \]
\[ C_i \leftarrow C_{i+1} ; i = 1, \ldots , 31 \text{ during T'} (\text{DIS}) \]
\[ E_i \leftarrow E_{i+1} ; i = 1, \ldots , 7 \text{ during P'}_5 P'_4 T'_p (\text{DIS}) \]
\[ D_i \leftarrow D_{i+1} ; i = 1, \ldots , 15 \text{ during T'}_p (\text{DIS}) \]
\[ M_8 \leftarrow D_1 ; M_i \leftarrow M_{i+1} ; i = 1, \ldots , 7 \text{ during T'}_p (\text{DIS}) \]
\[ O_8 \leftarrow M_1 ; O_i \leftarrow O_{i+1} ; i = 1, \ldots , 7 \text{ during T'}_p (\text{DIS}) \]
The D, M, and O registers are connected in shift logic and treated as the IR register.

Register copy logic:

\[
U_{33} \leftarrow U_0 \text{ during } [(\text{STORE}) + (\text{CU})](\text{DIS})
\]

\[
X_{33} \leftarrow X_0 \text{ during } [(\text{STORE}) + (\text{CX})](\text{DIS})
\]

\[
V_{33} \leftarrow V_0 \text{ during } [(\text{STORE}) + (\text{CV})](\text{DIS})
\]

\[
C_{32} \leftarrow C_1 \text{ during } [(\text{STORE}) + (\text{CC})T_p'(\text{DIS})
\]

\[
E_8 \leftarrow E_1 \text{ during } [(\text{STORE}) + (\text{CE})P_5'P_4'T_p'(\text{DIS})
\]

\[
D_{16} \leftarrow O_1 \text{ during } [(\text{STORE}) + (\text{CIR})T_p'(\text{DIS})
\]

Register copy logic:

\[
U_{33} \leftarrow DSR_0 \text{ during } (\text{STORE})(\text{CU})(\text{DIS})
\]

\[
X_{33} \leftarrow DSR_0 \text{ during } (\text{STORE})(\text{CX})(\text{DIS})
\]

\[
V_{33} \leftarrow DSR_0 \text{ during } (\text{STORE})(\text{CV})(\text{DIS})
\]

\[
E_8 \leftarrow DSR_0 \text{ during } (\text{STORE})(\text{CE})P_5'P_4'T_p'(\text{DIS})
\]

\[
C_{32} \leftarrow DSR_0 \text{ during } (\text{STORE})(\text{CC})T_p'(\text{DIS})
\]

\[
D_{16} \leftarrow DSR_0 \text{ during } (\text{STORE})(\text{CIR})T_p'(\text{DIS})
\]

Memory display logic:

\[
DSR_{33} \leftarrow R \text{ during } (\text{LOAD})(\text{CM})(\text{DIS})
\]

\[
WE = (\text{STORE})(\text{CM})(\text{DIS})
\]

\[
W = DSR_0(\text{STORE})(\text{CM})(\text{DIS})
\]
DSR clear logic:

\[ d_{\text{sr}}^i = (\text{CLEAR}); \ i = 0, \ldots, 33 \]

Instruction one-step and word time one-step can be implemented by the appropriate inputs to the flip-flop DJS.
X. SOFTWARE VERSUS HARDWARE

All computers are designed with compromises made between software and hardware. An example of a compromise in NEBULA is the PCK, UPK, and NRM commands instead of floating point arithmetic commands. As software is developed on a new computer certain deficiencies in the command set may appear. For example a return from subroutine command may not (in general) do all that is needed.

The initial instruction set for NEBULA is a compromise between a small set and an all inclusive command set.

As already mentioned, the decision not to include floating point arithmetic commands was a major compromise. Software was developed for floating point arithmetic, and as a result it was decided to implement the PCK, UPK, and NRM instruction. An example of an unpack subroutine:

```
UNPACK: NOP: / enter with fl. pt. no. in X.
   LDU #0 / clear U
   (1) CMX #0 / compare X with zero
      (2) SKGF / skip on greater than zero
         NGX / set X positive
      SLA #8 / shift exponent in U and left justify
                  / the fraction
```
(3) SKGE / skip on greater than zero
   NGX / re-complement
(4) LLC UNPACK / return with the
   / exponent in U and the fraction in X.

The above subroutine does not make use of the UPK command.

If the UPK command is available a subroutine is not needed and the following sequence is used:

  .

  .

  . / fl. pt. no. is in X.

LDU #0 / zero the U register

UPK #8 / exponent is put in U, fraction in X.

  .

  .

  .

The subroutine above, besides pointing out the need for an UPK command, shows two features of the command set. The compare command sets flags (bits 1 and 2 of the C register). This means that until another compare is made the results of the first compare is available. In the subroutine above the results of the compare at (1) is used by the skip commands at (2) and (3).

The second feature of the command set in the subroutine is the subroutine entry and return. The entry command stores the entire
C register. There are several possible ways to return: 1) LLC UNPACK; 2) LDC UNPACK, 3) JMP I UNPACK. The first choice is the one that is used in the example at (4). The LLC does not alter the flags in the lower half of the C register allowing the main program to use the results of the compare command at (1). The second choice restores the flags to their original condition. The third choice used to be used instead of LLC. The JMP I UNPACK has the undesirable feature that if \( C_{13-16} \) is non-zero on entry the return will not work. Thus the LLC command was implemented, which acts as a one level indirect jump.

The first version of NEBULA's indirect addressing fetched the entire modifier field. This meant, among other things, that the final indirect modifier field determines the immediate mode. For programs like instruction interpretation, it is nice to simulate the effective address calculation using the zeroth level immediate bit. Since no strong evidence supported the original version of indirect addressing, it was changed so that only the index and indirect parts of the modifier field are fetched.

Other examples could be brought forth, but for the most part these are from the designer's own programming experience and the examples of other computers' command sets.

The decision on double word fixed point data format was made without any real programming evidence to support it. The
controversial point in the double precision data format is how to
treat the extra sign bit created by a multiply. A multiply of minus
one by minus one is the only exception (this product causes an over-
flow in NEBULA). There are at least three possible choices (Figure
11).

1) Use the extra sign as the most significant bit of the product.
2) Use the extra sign as the sign of the low order half of the
   product.
3) Throw away the extra sign and set the sign of the low order
   half positive.

Choice number three was made for NEBULA. This format is
also taken into account in the double length shift and the normalize
commands. Programming on NEBULA so far has brought forth no
comments pro or con.
1) Fraction sign bits agree

2) Fraction sign bits agree

3) Fraction sign bit (always positive)

Figure 11. Double precision data formats.
XI. LOGIC IMPLEMENTATION

In order to give some insight into the relationship between the Boolean equations and the actual hardware, an example of logic implementation will be given. Hardware details of NEBULA appear elsewhere (4, p. 31-47). Considerations such as fan-in and fan-out and levels of gates is as close as the implementer will get to hardware considerations.

First a Boolean equation is considered. The Boolean equation used will be one actually used, but the implementation will appear slightly different from that used because other Boolean terms will not be considered.

Logic to be implemented:

\[ f_9 = (J'_1 J'_2 + J_2 J'_1) T_s T_p (DIV) F_8 \]

\[ f'_9 = T_s T_p (DIV) F'_9 \]

The first step in implementation is to consider the building blocks available. All NEBULA gates are NAND gates. The basic gates available are three input and six input NAND gates. The truth table for a two input NAND is shown in Figure 12. The flip-flops are edge-triggered R-S flip-flops with input gating (4, p. 37). Since all flip-flops of this type are clocked, the clock term is assumed in the equation and drawing.

The logic drawing is now made. The standard symbols for
Figure 12. NAND gate.
NAND gates and flip-flops are rectangles. Since the command DIV is the only command in \(F_9\), DIV does not appear in the implementation of the off logic. The term \((J_1 J_2 + J_1 J_2)\) is drawn as follows:

The logic flow is from left to right. The inputs appear on the left side, the outputs appear on the right side. The connection (1) forms an AND of the two outputs. The output after that point is \((J_1 J_2)(J_1 J_2)\) or \((J_1 J_2 + J_1 J_2)\). The term \((J_1 J_2 + J_1 J_2)(DIV)F_8\) is drawn as follows:

The output of point (2) is \(J_1 J_2 J_1 J_2 (DIV)F_8\) and the output (3) is the complement of (2) or \((J_1 J_2 + J_1 J_2)(DIV)F_8\).

The timing term \((T_s T_p)\) that is common to the on and off side
is formed as follows:

The final form of the drawing:

The flip-flop input $Sh$ is ANDed with both the $S$ (set) and $R$ (reset) inputs.

For the final drawing above, pin assignments are made and wiring lists are filled out. There are two things the implementer must consult when completing an implementation: 1) the existing logic drawings and 2) the existing wiring lists.

This is by no means a complete explanation of all the details such as pin assignments, but these details may be found in the existing documents mentioned above (which would be available to the implementer).

