### TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>NAME</th>
<th>MNEMONIC</th>
<th>OCTAL</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I. Transmittive</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A. Load</strong></td>
<td>LD-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Address</td>
<td>LDN</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>Constant</td>
<td>LDC</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>Direct</td>
<td>LDD</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>Indirect</td>
<td>LDI</td>
<td>40</td>
<td>5</td>
</tr>
<tr>
<td>Memory</td>
<td>LDM</td>
<td>50</td>
<td>6</td>
</tr>
<tr>
<td><strong>B. Load Complement</strong></td>
<td>LC-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Address</td>
<td>LCN</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td><strong>C. Store</strong></td>
<td>ST-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct</td>
<td>STD</td>
<td>34</td>
<td>7</td>
</tr>
<tr>
<td>Indirect</td>
<td>STI</td>
<td>44</td>
<td>8</td>
</tr>
<tr>
<td>Memory</td>
<td>STM</td>
<td>54</td>
<td>9</td>
</tr>
<tr>
<td><strong>II. Arithmetic</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>A. Shift</strong></td>
<td>SH-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Address</td>
<td>SHN</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td><strong>B. Add</strong></td>
<td>AD-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Address</td>
<td>ADN</td>
<td>16</td>
<td>11</td>
</tr>
<tr>
<td>Constant</td>
<td>ADC</td>
<td>21</td>
<td>13</td>
</tr>
<tr>
<td>Direct</td>
<td>ADD</td>
<td>31</td>
<td>14</td>
</tr>
<tr>
<td>Indirect</td>
<td>ADI</td>
<td>41</td>
<td>16</td>
</tr>
<tr>
<td>Memory</td>
<td>ADM</td>
<td>51</td>
<td>18</td>
</tr>
<tr>
<td><strong>C. Replace Add</strong></td>
<td>RA-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct</td>
<td>RAD</td>
<td>35</td>
<td>38</td>
</tr>
<tr>
<td>Indirect</td>
<td>RAI</td>
<td>45</td>
<td>41</td>
</tr>
<tr>
<td>Memory</td>
<td>RAM</td>
<td>55</td>
<td>44</td>
</tr>
<tr>
<td><strong>D. Replace Add One</strong></td>
<td>AØ-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct</td>
<td>AØD</td>
<td>36</td>
<td>39</td>
</tr>
<tr>
<td>Indirect</td>
<td>AØI</td>
<td>46</td>
<td>42</td>
</tr>
<tr>
<td>Memory</td>
<td>AØM</td>
<td>56</td>
<td>45</td>
</tr>
<tr>
<td><strong>E. Subtract</strong></td>
<td>SB-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Address</td>
<td>SBN</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>Direct</td>
<td>SBD</td>
<td>32</td>
<td>15</td>
</tr>
<tr>
<td>Indirect</td>
<td>SBI</td>
<td>42</td>
<td>17</td>
</tr>
<tr>
<td>Memory</td>
<td>SBM</td>
<td>52</td>
<td>19</td>
</tr>
<tr>
<td><strong>F. Replace Subtract One</strong></td>
<td>SØ-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct</td>
<td>SØD</td>
<td>37</td>
<td>40</td>
</tr>
<tr>
<td>Indirect</td>
<td>SØI</td>
<td>47</td>
<td>43</td>
</tr>
<tr>
<td>Memory</td>
<td>SØM</td>
<td>57</td>
<td>46</td>
</tr>
</tbody>
</table>
### III. Logical

<table>
<thead>
<tr>
<th>NAME</th>
<th>MNEMONIC</th>
<th>OCTAL</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A. Logical Product</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Address</td>
<td>LP-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>LPN</td>
<td>12</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>LPC</td>
<td>22</td>
<td>33</td>
</tr>
<tr>
<td><strong>B. Logical Difference</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Address</td>
<td>LM-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>LMN</td>
<td>11</td>
<td>30</td>
</tr>
<tr>
<td>Direct</td>
<td>LMC</td>
<td>23</td>
<td>34</td>
</tr>
<tr>
<td>Indirect</td>
<td>LMD</td>
<td>33</td>
<td>35</td>
</tr>
<tr>
<td>Memory</td>
<td>LMI</td>
<td>43</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>LMM</td>
<td>53</td>
<td>37</td>
</tr>
<tr>
<td><strong>C. Selective Clear</strong></td>
<td>SC-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Address</td>
<td>SCN</td>
<td>13</td>
<td>32</td>
</tr>
</tbody>
</table>

### IV. Jumps

<table>
<thead>
<tr>
<th>NAME</th>
<th>MNEMONIC</th>
<th>OCTAL</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A. Conditional</strong></td>
<td>-JN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zero</td>
<td>ZJN</td>
<td>04</td>
<td>26</td>
</tr>
<tr>
<td>Non Zero</td>
<td>NJN</td>
<td>05</td>
<td>27</td>
</tr>
<tr>
<td>Positive</td>
<td>PJN</td>
<td>06</td>
<td>28</td>
</tr>
<tr>
<td>Minus</td>
<td>MJN</td>
<td>07</td>
<td>29</td>
</tr>
<tr>
<td><strong>B. Unconditional</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Address</td>
<td>UJN</td>
<td>03</td>
<td>25</td>
</tr>
<tr>
<td>Memory</td>
<td>LJN</td>
<td>01</td>
<td>23</td>
</tr>
<tr>
<td>Return</td>
<td>RJN</td>
<td>02</td>
<td>24</td>
</tr>
</tbody>
</table>

### V. Pass

<table>
<thead>
<tr>
<th>NAME</th>
<th>MNEMONIC</th>
<th>OCTAL</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSN</td>
<td>PSN</td>
<td>00</td>
<td>20</td>
</tr>
<tr>
<td>PSN</td>
<td>24</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>PSN</td>
<td>25</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>

### VI. Input/Output

<table>
<thead>
<tr>
<th>NAME</th>
<th>MNEMONIC</th>
<th>OCTAL</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A. Tests</strong></td>
<td>-JM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Active</td>
<td>AJM</td>
<td>64</td>
<td>53</td>
</tr>
<tr>
<td>Inactive</td>
<td>IJM</td>
<td>65</td>
<td>54</td>
</tr>
<tr>
<td>Full</td>
<td>FJM</td>
<td>66</td>
<td>55</td>
</tr>
<tr>
<td>Empty</td>
<td>EJM</td>
<td>67</td>
<td>56</td>
</tr>
<tr>
<td><strong>B. Input</strong></td>
<td>IA-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To A</td>
<td>IAN</td>
<td>70</td>
<td>57</td>
</tr>
<tr>
<td>To Memory</td>
<td>IAM</td>
<td>71</td>
<td>58</td>
</tr>
<tr>
<td><strong>C. Output</strong></td>
<td>OA-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>From A</td>
<td>OAN</td>
<td>72</td>
<td>59</td>
</tr>
<tr>
<td>From Memory</td>
<td>OAM</td>
<td>73</td>
<td>60</td>
</tr>
<tr>
<td>NAME</td>
<td>MNEMONIC</td>
<td>OCTAL</td>
<td>PAGE</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>D. Activate</td>
<td>ACN</td>
<td>74</td>
<td>61</td>
</tr>
<tr>
<td>E. Disconnect</td>
<td>DCN</td>
<td>75</td>
<td>62</td>
</tr>
<tr>
<td>F. Function</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Address</td>
<td>FAN</td>
<td>76</td>
<td>63</td>
</tr>
<tr>
<td>Constant</td>
<td>FNC</td>
<td>77</td>
<td>64</td>
</tr>
</tbody>
</table>

**VII. Central**

| A. Exchange Jump    | EXN      | 26    | 47   |
| B. Read P           | RPN      | 27    | 48   |
| C. Read Memory      | CR-      |       |      |
| To A                | CRD      | 60    | 49   |
| To Memory           | CRM      | 61    | 50   |
| D. Write Memory     | CW-      |       |      |
| From A              | CWD      | 62    | 51   |
| From Memory         | CWM      | 63    | 52   |
LOAD A NO ADDRESS

(P)

MACHINE

11 5 0

14 d

LOCATION  OPERATION  ADDRESS FIELD

ASPER

LDN d

Constant
Symbol + Constant
Symbol - Symbol

The above values must result in an octal value in the range of 00-77. The decimal equivalents are 00-63.

Mnemonic Operation Code

DESCRIPTION

This instruction clears the A-Register and loads d into the lower 6 bits of A. The upper 12 bits of A are set to zero. RNI @ P+1

REFERENCES:

DATA TRANSMISSION  June 1, '66
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → $\text{OPERAND \ 00000d} \rightarrow A \rightarrow P+1 \rightarrow P \rightarrow \text{EXIT RNI}$

6400/6600

1 USEC

TIMING

1 USEC → 475 NSEC → RNI → SLOT → d → A

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction to clear the A-Register

ANSWER:

Exercise #2 - Code an instruction to set the lower 6-bits of the A-Register to ones.

ANSWER:

Exercise #3 - Code an instruction that will set the A-Register to 31 decimal, so it could be stored and counted down later.

\[ (31_{10} = 37_{8}) \]

ANSWER:

Exercise #4 - Code an instruction that will set the A-Register to the constant 12, octal.

ANSWER:
This instruction clears the A-Register and loads the complement of \( d \) into the lower 6-bits of A. The upper 12-bits are set to ones.

RNI @ P+1
FLOW DIAGRAM

ENTER -> FETCH INSTRUCTION -> \(\text{\$OPERAND 7777\text{dd} \rightarrow A}\) -> \(\text{P+1 \rightarrow P}\) -> EXIT RNI

TIMING

6400/6600 1 USEC

\[\text{475 NSEC} \quad \text{1 USEC} \quad \text{1 USEC}\]

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction that will set the A-Register to a negative 60 decimal.

Exercise #2 - Code an instruction that will give us a mask, with the lower 6-bits cleared and the upper 12-bits ones.

Exercise #3 - Code an instruction to set the A-Register to all ones.
Load a constant

This instruction clears the A-Register and loads the 18-bit quantity consisting of $d$ as the upper 6 bits and $m$ as the lower 12 bits. RNI 3 P+2
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → OPERAND \( d \rightarrow A \) → \( P+2 \rightarrow P \) → EXIT RNI

TIMING

6400/6600  2 USEC

\[ \text{SLOT} \quad \text{RNI} \quad \text{475 NSEC} \quad \text{READ OPERAND} \quad \text{dm} \rightarrow A \quad \text{475 NSEC} \quad \text{SLOT} \quad 1 \text{ USEC} \quad 1 \text{ USEC} \quad \text{SLOT} \]

\[ \text{SLOT TIME} = 100 \text{ NSEC} \]
Exercise #1 - Code an instruction that will set the upper 6-bits of the A-Register to ones, in order that two 12-bit quantities may be added later (with end around carries).

Exercise #2 - Code an instruction to load the A-Register with the constant \(4096_{10}\), which could next be used for an input/output instruction. 
\((4096_{10} = 10000_8)\)

Exercise #3 - Code an instruction that will load the A-Register with all ones into the A-Register.
LOAD A DIRECT

MACHINE

LOCATION  OPERATION  ADDRESS FIELD

11  5  0

30  d

Constant
Symbol + Constant
Symbol ÷ Symbol

The above values must result
in an octal value in the range
of 00-77. The decimal equivalents
are 00-63.

Mnemonic Operation Code

This instruction clears the A-Register and loads the contents of
location d into the lower 12 bits of A. The upper 6 bits of A are
set to zero.

RNI @ P+1.

REFERENCES:

DATA TRANSMISSION       June 1, '66

4-0
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → OPERAND (00dd) → A → P+1 → P → EXIT RNI

6400/6600

T I M I N G

475 NSEC 1 USEC 475 NSEC 1 USEC

RNI SLOT READ OPERAND (d) → A

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction to load 7654 into the A-Register.
7654 is in M.L. 0077 (Octal).

**ANSWER**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDD 77B</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction to load the contents of Tempo.
Tempo = 0070, but could have been any value 00-778.
(Tempo) = 0010, but could have been an octal value in the range of \(2^{12} - 1\).

**ANSWER**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDD TEMPO</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #3 - Code an instruction to load 1234 into the A-Register.
The operand 1234 is in a location called ABLE, which happens to equal 0000.
ABLE = 0000, but could have been any octal value 00-77.
(ABLE) = 1234, but could have been an octal value in the range of \(2^{12} - 1\).

**ANSWER**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDD ABLE</td>
<td></td>
</tr>
</tbody>
</table>
LOAD A INDIRECT

MACHINE

LOCATION OPERATION ADDRESS FIELD

40 d

LDI a

Constant
Symbol
Symbol \( \div \) Constant
Symbol \( \div \) Symbol

The above values must result in an octal value in the range of 00-77. The decimal equivalents are 00-63.

Mnemonic Operation Code

This instruction clears the A-Register and loads into A the 12-bit quantity obtained by indirect addressing, \((d)\) \(\rightarrow\) A. The upper six bits of A are zero. Location \(d\) is read out of memory, and the word obtained is used as the operand address. RNI @ P+1

REFERENCES:

DATA TRANSMISSION       June 1, '66  5-0
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH OPERAND ADDRESS (00dd) → OPERAND ((00dd)) → P+1 → P

EXIT RNI

6400/6600

3 USEC

1 USEC → 475 NSEC → 1 USEC → 475 NSEC → 475 NSEC

RNI

READ OPERAND ADDRESS (d)

READ OPERAND ((d)) A

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction that loads the A-Register, and uses INDADDR as a 1st reference, and ADDR as a 2nd reference to obtain the operand 4311.

INDADDR = 50, but could have been any octal value 00-77.

(INDADDR) = ADDR = 3024, but could have any value 0000-7777.8

(ADDR) = 4311, and again could have been any value 0000-7777.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LDI</td>
<td>INADDR</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36</td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction that loads the A-Register, and uses location 0001 as its 1st reference to obtain the operand address. The operand address is equal to 63518, and the operand equals 01008.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LDI</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36</td>
</tr>
</tbody>
</table>

(0001) = 63518

(6351) = 01008
This instruction clears the A-Register and loads a 12 bit operand obtained by indexed addressing into the lower 12 bits of A. The upper 6 bits of A are set to zero.

Note: If $d=0$, the operand address is simply $m$.

If $d \neq 0$, then $m + (d)$ is the operand address. Thus the contents of $d$ may be used as an index quantity to modify operand addresses. RNI @ P+2

REFERENCES:

DATA TRANSMISSION

June 1, '66

6-0
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → d = 0?
No → FORM OPERAND ADDRESS m+(d) → A
Yes → FETCH OPERAND ADDRESS m → A

A → P+2 → P → EXIT RNI

TYPING

6400/6600

3-4 USEC

1 USEC → 1 USEC → 1 USEC → 1 USEC →
1 USEC

475 NSEC → 475 NSEC → 475 NSEC → 475 NSEC →

NO INDEX d = 0

INDEX d ≠ 0

READ INDEX (d) → READ OPERAND ADDRESS m+ (d) → A
READ OPERAND ADDRESS m → A

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction that will load the A-Register with an operand 0601, whose address is called SEL. Use the memory index instruction without indexing. SEL = 6322, but could have been equal to any value in the range \(2^{12} - 1\).

\[(SEL) = 0601\]

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDM SEL</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction that will load the A-Register with the first a series of numbers from a table. Using the memory index instruction, the basic address will be called TABLE, and the index address called INDEX.

INDEX = 70, but could have been any octal value 01-77.

TABLE = 5000, but could have been any value in the range of \(2^{12} - 1\).

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LDM TABLE, INDEX</td>
<td></td>
</tr>
</tbody>
</table>

Note: The contents of 70 should be zero, the 1st time a load is made, and then updated by one for each new reference.
This instruction stores the lower 12 bits of the A-Register into location \( d \). The contents of \( A \) are not altered.

RNI @ P+1
FLOW DIAGRAM

1. ENTER
2. FETCH INSTRUCTION
3. (A) → d
4. P + 1 → P
5. EXIT RNI

TIMING

6400/6600

2 USEC

1 USEC

475 NSEC

RNI

1 USEC

475 NSEC

STORE CYCLE

(A) → d

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction to store the contents of the A-Register, which is 765432, in M.L. 0000.

Note: At the completion of the instruction, M.L. 0000 will contain 5432.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STD 0</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction to store the contents of the A-Register which is 000000, in M.L. 0077.

Note: At the completion of the instruction, M.L. 0077 will contain 0000. This effectively clears that M.L.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STD 77B</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #3 - Code an instruction that will save the contents of the A-Register, and store it in some location 00-77. In this example the location will be called LOWCORE.

LOWCORE = 10

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STD LOWCORE</td>
<td></td>
</tr>
</tbody>
</table>

Note: Only the lower 12-bits of the A-Register will be stored in memory.

The A-Register remains unaltered.
STØRE A INDIRECT

MACHINE

LOCATION | OPERATION | ADDRESS FIELD
-----------|-----------|-------------------
           | STI d     |                   

(P)

11  5  0

44 d

Constant
Symbol
Symbol + Constant
Symbol = Symbol

The above values must result in an octal value in the range of 00-77. The decimal equivalents are 00-63.

Mnemonic Operation Code

DESCRIPTION

This instruction stores the lower 12 bits of the A-Register into the location specified by the contents of location d, (A) \( \rightarrow \) ((d)). The contents of A are not altered. RNI @ P+1.
Exercise #1 - Code an instruction that stores the contents of the A/Register, and uses INDREF as a 1st reference and STORE as a 2nd reference.

A-Register contains 514233.

INDREF = 00, but could have been any octal value 00-77.

(INDREF) = STORE = 6071, but could have been any octal value in the range \(2^{12} - 1\).

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>STI</td>
<td>INDREF</td>
<td></td>
</tr>
</tbody>
</table>

Note: Only the 4233 of A will be stored in location 6071.

Exercise #2 - Code an instruction that stores the contents of the A-Register, and uses M.L. 0077 as its 1st reference to obtain the operand address.

A-Register = 123456

(0077) = 3347

(3347) = 1000

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>STI</td>
<td>77B</td>
<td></td>
</tr>
</tbody>
</table>

Note: At the completion of this instruction, memory location 3347 will contain 3456, and 1000 will be destroyed.
This instruction stores the lower 12 bits of the A-Register in the location determined by indexed addressing. The contents of A are not altered.

Note: If \( d = 0 \), the operand address is simply \( m \). If \( d \neq 0 \), the \( m \) plus the contents of location \( d \), \( m + (d) \), is the operand address. Thus the contents of \( d \) may be used as an index quantity to modify operand addresses. RNI @ P+2
**FLOW DIAGRAM**

1. **ENTER**
2. **FETCH INSTRUCTION**
   - d = 0?
     - Yes: **FETCH OPERAND ADDRESS** m
     - No: **FORM OPERAND ADDRESS** m+(d)
   - B
   - (A) **OPERAND ADDRESS**
   - P+2 → P
   - EXIT RNI

---

**TIMING**

*6400/6600*  
*3-4 USEC*

- **INDEX d = 0**
  - READ OPERAND ADDRESS m
  - STORE (A → m)
  - 475 NSEC

- **INDEX d ≠ 0**
  - READ INDEX (d)
  - READ ADDRESS m and form m + (d)
  - STORE (A → m+(d))
  - 475 NSEC

**SLOT TIME = 100 NSEC**
Exercise #1 - Using memory index, with no indexing, code an instruction to store the contents of the A-Register in some location called STADDR.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STM STADDR</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - A series of numbers may be coming into the computer. Using Memory Index, with indexing, code an instruction to store the numbers in sequential locations beginning the location called STAN. Use the location called UPDATE, as an index.

Assume UPDATE = 70 and contains 0000 at 1st STAN = any value (see P. 9-0)

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STM STAN,UPDATE</td>
<td></td>
</tr>
</tbody>
</table>
This instruction shifts the contents of the A-Register (which is 18-bits) right or left \( d \) places. If \( d \) is positive (00-37\(_8\)), the shift is \textbf{left} and circular; if \( d \) is negative (40-77\(_8\)), A is shifted \textbf{right}, end off with no sign extension. \textit{RNI @ P+1}.
Exercise #1 - Code an instruction to shift the contents of the A-Register left $12_{10}$ places. This will place the lower 6-bits of A in the upper 6-bits of A.

Note: A left shift of more than $17_{10}$ would be impractical. Also, note each left shift would be a multiplication by 2.

Exercise #2 - There is an 18-bit quantity in the A-Register we wish to store. The lower 12-bits may be stored without any trouble. However, we must do some manipulation with the A-Register before the upper 6-bits can be stored. Code an instruction to shift the contents of the A-Register right 12 places.

Note: A right shift of more than $18_{10}$ would be impractical also, note each right shift would be a division by 2.
This instruction adds the 6-bit positive quantity \( d \) (0000dd) to the contents of the A-Register (which is 18-bits).

RNI @ P+1.

The above values must result in an octal value in the range of 00-77. The decimal equivalents are 00-63.

Mnemonic Operation Code
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → (A) +00dd → A → P+1 → P → EXIT RNI

TIMING

6400/6600 1 USEC

→ 475 NSEC

RNI

SLOT TIME = 100 NSEC

11-1
Exercise #1 - Code an instruction to add the constant $77_{8}$ to the A-Register.

Exercise #2 - Code an instruction to add the constant $20_{10}$ to the A-Register.
This instruction subtracts the 6-bit positive quantity d (0000dd) from the contents of the A-Register (which is 18-bits).

RNI @ P+1.
FLOW DIAGRAM

ENTER ➔ FETCH INSTRUCTION ➔ (A) -00dd➔A ➔ P+1➔P ➔ EXIT RNI

TIMING

6400/6600 1 USEC

|槽时间 = 100 nsec |
Exercise #1 - Code an instruction to test if a program has gone through a loop the required number of times (4).

Note: This instruction assumes the A-Register contains the loop count, and the next instruction will finish the test.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SBN 4</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction to help find a number which is 20B. Assume one number from a list of numbers is already in the A-Register, and another instruction will finish the test.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SBN 20B</td>
<td></td>
</tr>
</tbody>
</table>
ADD CONSTANT

\[(P) \quad (P+1)\]

MACHINE

\[
\begin{array}{c}
11 \\
5 \\
011 \\
0 \\
\end{array}
\begin{array}{c}
21 \\
d \\
m \\
\end{array}
\]

LOCATION OPERATION ADDRESS FIELD

ASPER

\[ADC \ dm\]

Constant
Symbol \(+\) Constant
Symbol \(-\) Symbol

The above values may result in an octal value in the range 00-2^{18}-1, or the decimal equivalents.

Mnemonic Operation Code

This instruction adds to the A-Register (which is 18-bits) the 18 bit quantity \(dm\) (consisting of \(d\) as the upper 6-bits and \(m\) as the lower 12-bits). RNI @ P+2.
FLOW DIAGRAM

1. Enter
2. Fetch instruction
3. Fetch operand
4. $(A) + ddmmnnn \rightarrow A$
5. $P+1 \rightarrow P$
6. Exit

6400/6800

2 USEC

TIMING

$\leftarrow 475 \text{ NSEC} \rightarrow$

1 USEC

RNI

$\leftarrow 475 \text{ NSEC} \rightarrow$

1 USEC

SLOT

READ OPERAND and form
$(A) + dm \rightarrow A$

SLOT TIME = 100 NSEC

13-1
Exercise #1 - Code an instruction to add the bias of 741114 to the contents of A.

Exercise #2 - Code an instruction to add the value "symvalue" to the A-Register.
ADD DIRECT

(M)

MACHINE

LOCATION

OPERATION  ADD d

ADDRESS FIELD

11  5  0

13  d

Constant
Symbol + Constant
Symbol - Symbol

The above values must result in an octal value in the range of 00-77. The decimal equivalents are 00-63.

Mnemonic Operation Code

This instruction adds to the A-Register (which is 18-bits) the 12-bit positive quantity contained in memory location d. RNI @ P+1.
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH OPERAND → (A)+d → A → P+1 → P → EXIT RNI

TIMING

6400/6600  2 USEC

1 USEC  475 NSEC  475 NSEC  1 USEC

SLOT  RNI  READ OPERAND and form (A) +d → A

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction to add the contents of low-core location 01, which might be a 12-bit constant to the A-Register.

Exercise #2 - Code an instruction to add the contents of location TEMP7, which is a temporary value, to the A-Register.

TEMP7, = 77
This instruction subtracts from the A-Register (which is 18 bits) the 12 bit positive quantity contained in location d. RNI @ P+1.
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH OPERAND → (A) - OPERAND → A → P+1 → P

EXIT

RING

6400/6600

2 USEC

TIMING

475 NSEC

1 USEC

475 NSEC

1 USEC

RNI

READ OPERAND and form (A) - (d) → A

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction to decrement the A-Register by a value of \(120_{10}\). The value, \(120_{10}\), is in location 70, called TEMPO.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBD TEMPO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction to subtract a bias of 3333 from a value in the A-Register. The bias value is in location \(12_{10}\).

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBD 14B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ADD INDIRECT

MACHINE

LOCATION OPERATION ADDRESS FIELD

Asper

ADI \( d \)

Constant
Symbol
Symbol \( + \) Constant
Symbol \( / \) Symbol

The above values must result in an octal value in the range of 00-77. The decimal equivalents are 00-63.

Mnemonic Operation Code

DESCRIPTION

This instruction adds to the contents of the A-Register (which is 18-bits), a 12-bit positive operand obtained by indirect addressing. Location \( d \) is read out of memory, and the word obtained is used as the operand address. \( \text{RNI} @ \text{P+1} \)

REFERENCES:

ARITHMETIC

June 1, '66

16-0
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH OPERAND ADDRESS → FETCH OPERAND → C

C → (A) + OPER → A → P+1 → P → EXIT RNI

6400/6600 3 USEC

TIMING

475 NSEC 475 NSEC 475 NSEC

1 USEC 1 USEC 1 USEC

RNI SLOT READ OPERAND ADDRESS (d) READ OPERAND and form (A) + ((d)) → A

SLOT TIME = 100 NSEC
Exercise #1 - Various instructions throughout a program need to refer to the same location. Code an instruction that needs the contents of location 6000, and 6000 is also found in location 76.

Assume FIRSTADD = 0076

\[(0076) = \text{SECONADD} = 6000\]

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0016</td>
<td>ADI</td>
<td>FIRST ADD</td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction which uses location \(10_8\) as a first reference and location \(7000_8\) as a second reference to pick up a flag, to add to A.

\[(0010) = 7000\]

\[(7000) = \text{Flag}\]

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0008</td>
<td>ADI</td>
<td>10B</td>
</tr>
</tbody>
</table>
This instruction subtracts from the A-Register (which is 18-bits) a 12-bit positive operand obtained by indirect addressing. Location \(d\) is read out of memory, and the word obtained is used as the operand address. RNI @ P+1
Exercise #1 - Code an instruction to subtract from the A-Register the value found in location 7021, called RESULT. The address "RESULT" is also found in location 2 called "REFERI".

Assume \((D002) = (REFERI) = 7021 = \text{RESULT}\)

\((\text{RESULT}) = \Phi\text{Operand}\)

**Answer**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SBI REFERI</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction to decrement the contents of the A-Register by a constant, whose address is found in location \(50_8\).

**Answer**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SBI 50B</td>
<td></td>
</tr>
</tbody>
</table>
ADD MEMORY INDEX

\[ (P) \quad (P+1) \]

MACHINE

\[ \begin{array}{cccc}
11 & 5 & 011 & 0 \\
51 & d & m & 0 \\
\end{array} \]

LOCATION \quad OPERATION \quad ADDRESS FIELD

\[ \text{ADM} \ m \ a \]

- Separator space or comma
- Constant Symbol
- Symbol \(+\) Constant
- Symbol \(-\) Symbol

The value of \( d \) must result in an octal value in the range of 00-77.
The value of \( m \) must result in an octal value in the range of \( 2^{12} - 1 \).

Mnemonic Operation Code

This instruction adds to the contents of the A-Register (which is 18-bits) a 12-bit positive operand obtained by indexed addressing.

Note: If \( d = 0 \), the operand address is simply \( m \).

If \( d \neq 0 \), then \( m \) plus the contents of location \( d \), \( m + (d) \) is the operand address. Thus the contents of \( d \) may be used as an index quantity to modify operand addresses. RNI @ P+2.
**ADM**

**FLOW DIAGRAM**

1. ENTER → FETCH INSTRUCTION
2. d = 0?
   - Yes → FETCH OPERAND ADDRESS m → E
   - No → E → FETCH OPERAND ADDRESS m + (d)
3. FETCH OPERAND → (A) + OPER
4. P+1 → P → EXIT RNI

**6400/6600**

**TIMING**

- **INDEX d = 0**
  - NO INDEX
  - SLOT
  - READ OPERAND ADDRESS m → (A) + (m) → A
  - 475 NSEC

- **INDEX d ≠ 0**
  - SLOT
  - READ INDEX
  - READ ADDRESS (A) + (m+(d)) → A
  - 475 NSEC

**SLOT TIME = 100 NSEC**

3-4 USEC
Exercise #1 - Code an instruction that is used to add a constant from a series of constants to the A-Register. Let the base address equal WORD, which might happen to equal 1244, and the index address equal INCREMEN which may equal 01.

Exercise #2 - Code an instruction to add a constant to the A-Register. The constant has a 12-bit address and needs no indexing. The address of the constant is called INSTRUCT and may happen to equal 5670.
This instruction subtracts from the 18-bit, A-Register a 12-bit positive
operand obtained by indexed addressing.

Note: If \( d = 0 \), the operand address is simply \( m \).

If \( d \neq 0 \), then \( m \) plus the contents of location \( d \), \( m + (d) \) is the
operand address. Thus the contents of \( d \) may be used as an index
quantity to modify operand addresses.

RNI @ P+2
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → d = 0? → yes → FETCH OPERAND ADDRESS m → F

No

F ← FETCH OPERAND ADDRESS m + (d)

FETCH OPERAND ↔ A → P+1 → P → EXIT RNI

---

INDEX d = 0

NO

INDEX d ≠ 0

READ OPERAND ADDRESS m → (A) → (m) ↔ A

1 USEC → 1 USEC → 1 USEC → 1 USEC →

475 NSEC → 475 NSEC → 475 NSEC → 475 NSEC

SLOT

SLOT

SLOT

SLOT

READ INDEX

and form

m + (d)

SLOT TIME = 100 NSEC

---

3-4 USEC

---

1 USEC

475 NSEC

READ ADDRESS

(A) → (m + (d)) ↔ A

---

1 USEC

475 NSEC

---

1 USEC

475 NSEC

---

475 NSEC

---

1 USEC

475 NSEC

---

1 USEC

475 NSEC

---
Exercise #1 - Code an instruction to subtract the number found in location WALLY (in which WALLY equals 0245₈), from the A-Register. No indexing is needed on this instruction.

WALLY = 0245

(WALLY) = Operand

Exercise #2 - Code an instruction using memory index to subtract a number found in a table from the A-Register. The index address equals NEXT, which is equal to the arbitrary value 07. The base address of the table is called RANDUM, which might equal 1030.

NEXT = 07

(RANDUM + (NEXT)) = Operand

(NEXT) - some index value
This is a no operation instruction and is a convenience for padding out a program. RNI @ P+1.
FLOW
DIAGRAM

ENTER -> FETCH INSTRUCTION -> P + 1 = P -> EXIT RNI

TIMING

6400/8800 1 USEC

475 NSEC 1 USEC

RNI SLOT

SLÖT TIME = 100 NSEC
Exercise #1 - Code an instruction that takes up space in a program, so that instructions may be generated, and stored there during execution.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
</table>

**ANSWER**
This is a no operation instruction, and is a convenience for padding out a program. RNI + 1 (See pages 20-0 - 20-2)
This is a no operation instruction, and is a convenience for padding out a program. RNI + 1 (See pages 20-0 -20-2)
This instruction jumps to the sequence of instructions beginning at the address given by \( m + (d) \).

If \( d = 0 \), then \( m \) is not modified.
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION

- d = 0?
  - Yes: FETCH JUMP ADDR m → JUMP ADDR P → EXIT RNI
  - No: FETCH JUMP ADDR m + (d)

TIMING

6400/6600

INDEX

- d = 0
  - No INDEX
  - READ JUMP ADDR m → P
  - 475 nsec
  - 1 usec

- INDEX
  - d ≠ 0
  - READ INDEX (d)
  - 475 nsec
  - 1 usec
  - READ JUMP ADDR m + (d) → P
  - 475 nsec
  - 1 usec

2-3 usec
Exercise #1 - Code an instruction that unconditionally transfers control to a new sequence of instructions, whose beginning address is called INITIAL. Use no indexing. INITIAL might happen to equal 4320.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LJM INITIAL</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction that unconditionally transfers control to a new sequence of instructions depending upon the contents of a switch. Let the address of the first program be called START and assume the index is generated during execution and adds to START to give the sequence of instructions. The index location is called MODIFY.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LJM START, MODIFY</td>
<td></td>
</tr>
</tbody>
</table>
RETURN JUMP

(P) (P+1)

MACHINE

11  5  0 11  0
02  d  m

LOACTION  OPERATION  ADDRESS FIELD

RJM  m, d

Separator
space or comma

\{ Constant
   Symbol
   Symbol + Constant
   Symbol + Symbol \}

The value of d must result in an octal value in the range of 00-77. The value of m must result in an octal value in the range of 212-1.

Mnemonic Operation Code

This instruction jumps to the sequence of instructions, beginning at the location given by \( m + (d) \). If \( d = 0 \), then \( m \) is not modified.

The current program address plus two \((P + 2)\) is stored at the jump address, and is used as the return address to the main routine when the new sequence is finished. The new sequence starts at the jump address plus one.
FLOW DIAGRAM

ENTER → FETCG INSTRUCTION

\( d = 0 ? \)

No → G

G → FETCH JUMP ADDR \( m \)

Yes → FETCG JUMP ADDR \( m + (d) \)

FORM RETURN ADDR \( P+2 \) → P+2 \( \rightarrow \) JUMP ADDR

JUMP ADDR + 1 \( \rightarrow \) P

EXIT RNI

6400/6600

3-4 USEC

TIMING

INDEX \( d = 0 \)

NO INDEX \( d = 0 \)

INDEX \( d \neq 0 \)

READ INDEX \( (d) \) → READ JUMP ADDRESS \( m + (d) \) → STORE RETURN ADDR \( P+2 \)

1 USEC → 1 USEC → 1 USEC → 1 USEC →

1 USEC 1 USEC 1 USEC 1 USEC

\( \begin{array}{c}
\text{NO INDEX } \\
\text{d = 0}
\end{array} \)

\( \begin{array}{c}
\text{INDEX} \\
\text{d }\neq \text{ 0}
\end{array} \)

\( \begin{array}{c}
\text{READ INDEX} \\
\text{(d)}
\end{array} \)

\( \begin{array}{c}
\text{READ JUMP} \\
\text{ADDRESS m+(d)}
\end{array} \)

\( \begin{array}{c}
\text{STORE RETURN} \\
\text{ADDR P+2}
\end{array} \)

\( \begin{array}{c}
\text{1 USEC} \\
\text{1 USEC}
\end{array} \)

\( \begin{array}{c}
\text{1 USEC} \\
\text{1 USEC}
\end{array} \)

\( \begin{array}{c}
\text{1 USEC} \\
\text{1 USEC}
\end{array} \)

\( \begin{array}{c}
\text{1 USEC} \\
\text{1 USEC}
\end{array} \)

SLOT TIME = 100 NSEC
Exercise #1 - Different places in a program it is desired to execute a series of instructions and return to the main program each time. Code an instruction that would allow us to execute a subroutine called SCANNER and return upon completion of the subroutine. No indexing is needed on this instruction.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RJM SCANNER</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - In various parts of a program it is desirable to execute one of two subroutines, and return to the main program. The base address of the two routines is called CONVERT and the index is called SWITCH.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RJM CONVERT, SWITCH</td>
<td></td>
</tr>
</tbody>
</table>
This instruction provides an unconditional jump of up to $3\times 10$ steps forward or backward from the current program address. The value of $d$ is added to the current program address. If $d$ is positive ($0_{10} - 37_{10}$) the jump is forward. If $d$ is negative ($40_{10} - 76_{10}$) the jump is backward.

**NOTE:** If $d = 00$ or $77$, the processor will hang-up (stop) indefinitely.
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → P+d→P → EXIT RNI

TIMING

6400/6600 1 USEC

475 NSEC ← SLΦT ← RNI ← 1 USEC

SLΦT TIME = 100 NSEC
Exercise #1 - Code an instruction that unconditionally transfers control forward to a new set of instructions. The new address is called ALPHA and must be in the range of 01-37 places forward.

Note: d must not equal 00 or 77, for this will hang-up the PP until dead start.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>UJN</td>
<td>ALPHA</td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction that unconditionally transfers control backward to a new set of instructions. The new address is called BACK and must be in the range of 01-37 places back.

Note: See above note.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>UJN</td>
<td>BACK</td>
</tr>
</tbody>
</table>
This instruction provides a conditional jump of up to 31 steps forward or backward from the current program address if the content of the A-Register is zero. If A is a nonzero, the next instruction @ P+1 is executed. Negative zero (777777) is treated as nonzero. If \( d \) is positive (01\(_8\)-37\(_8\)) the jump is forward. If \( d \) is negative (40\(_8\)-76\(_8\)) the jump is backward.

Note: If \( d = 00 \) or 77 and the test is met, the processor will hang-up (stop) indefinitely.
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION

A = 0?

No → P+1 → P → EXIT

Yes → P+d → P

6400/6600

1 USEC

TIMING

475 NSEC

1 USEC

RNI

SLØT TIME = 100 NSEC
Exercise #1 - The computer has decision making ability, depending upon the contents of the A-Register. Code an instruction that tests A for being zero. If it is a jump forward an address call DECIMAL, if the condition is not met read the next instruction at P+1. DECIMAL must be in the range of 01-37, places forward. Note: See note on page 25-2.

Exercise #2 - Code an instruction that tests the A-Register for zero, and if a zero is found jump back 20 locations, otherwise continue at P+1. Note: See note on page 25-2.
This instruction provides a conditional jump of up to 31 steps forward or backward from the current program address if the content of the A-Register is nonzero. If $A$ is zero, the next instruction @ $P+1$ is executed. Negative zero ($777777_8$) is treated as nonzero. If $d$ is positive ($01-37_8$) the jump is forward. If $d$ is negative ($40-76_8$) the jump is backward.

Note: If $d = 00$ or $77$, and the test is met, the processor will hang-up (stop) indefinitely.
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → A ≠ 0? → No → P+1 → P
Yes → P+d → P

EXIT

TIMING

6400/6600 1 USEC

<table>
<thead>
<tr>
<th>475 NSEC</th>
<th>1 USEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNI</td>
<td>SLOT</td>
</tr>
</tbody>
</table>

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction that tests the A-Register for a non-zero quantity and jumps forward 3 places if the condition is met.

Note: See note on page 25-2.

ANSWER

Exercise #2 - Code an instruction that loops back to a location called OCTAL, if the A-Register is a non-zero quantity. Otherwise RNI @P+1. OCTAL would be in the range of 01-37 places back.

Note: See note on page 25-2.
PLUS JUMP

MACHINE

LOCATION OPERATION ADDRESS FIELD

Constant Symbol
Symbol Constant
Symbol Symbol

The above values must result in an octal value in the range of 00-77. The decimal equivalents are 00-63.

Mnemonic Operation Code

This instruction provides a conditional jump of up to 31 steps forward or backward from the current program address if the contents of the A-Register is positive. If A is negative, the next instruction @ P+1 is executed. A is positive if the $^{27}$ is a zero. If d is positive (01-37$_8$) the jump is forward. If d is negative (40-76$_8$) the jump is backward.

Note: If d = 00 or 77 and the test is met, the processor will hang-up (stop) indefinitely.

REFERENCES:
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION

A = +?

No → P+1 → P

Yes → P + d → P

EXIT RNI

TIMING

6400/6600 1 USEC

475 NSEC 1 USEC

RNI

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction that tests the A-Register for a positive number, and if the condition is met, jump forward to location SYMBOL. SYMBOL must be in the range of 01-37 places forward.

Note: See note page 25-2.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PJN</td>
<td>SYMBOL</td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction that jumps backward if the contents of the A-Register is positive. Go back to location INSTFLAG. INSTFLAG must be back 01-37 places.

Note: See note page 28-2.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PJN</td>
<td>INSTFLAG</td>
</tr>
</tbody>
</table>
This instruction provides a conditional jump of up to 31 steps forward or backward from the current program address if the content of the A-Register is negative. If A is positive, the next instruction @ P+1 is executed. A is negative (if the $2^{17}$ bit is a one). If $d$ is positive (01-37g) the jump is forward. If $d$ is negative (40-76g) the jump is backward.

Note: If $d = 00$ or 77 and the test is met, the processor will hang-up (stop) indefinitely.

REFERENCES:
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION

A = - ?

Yes → P + d → P

No → P + 1 → P → EXIT RNI

TIMING

6400/6600

1 USEC

475 NSEC

1 USEC

RNI

SLAT TIME = NSEC
Exercise #1 - Code an instruction that tests the A-Register and if it finds it negative, jumps forward to a set of instructions beginning at location ERROR. ERROR must be in the range of 01-37 places forward.

Note: See note on page 25-2.

Exercise #2 - Code an instruction that tests the A-Register for a negative number and jumps back 4 locations.

Note: See note page 25-2.
The instruction forms in the A-Register the bit by bit logical difference of \( d \) and the lower 6 bits of A. This is equivalent to complementing the individual bits in A which correspond to bits in \( d \) equal to one. The upper 12-bits of A are not altered.

Example:  
\[
A = \begin{array}{c}
001110101101100101
\end{array}
\]  
\[
d = \begin{array}{c}
001010
\end{array}
\]  
\[
\overline{\begin{array}{c}
0011101011011000011
\end{array}}
\]

RNI @ P+1
FLOW DIAGRAM

ENTER \rightarrow FETC\text{H INSTRUCTION} \rightarrow A \rightarrow A^* \rightarrow P+1 \rightarrow P \rightarrow EXIT

6400/6600 \hspace{1cm} 1 \text{ USEC}

TIMING

\rightarrow 475 \text{ NSEC}  \hspace{1cm} 1 \text{ USEC} \hspace{1cm} \rightarrow

RNI \hspace{1cm} SL\text{OT}

SL\text{OT} \text{ TIME} = 100 \text{ NSEC}
Exercise #1 - Code an instruction that would complement the lower 6-bits of the A-Register, leaving the upper 12-bits unaltered.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMN 77B</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Assume the A-Register is zero, code an instruction that will set the $2^0$ bit position to a one for a flag indicator.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMN 1</td>
<td></td>
</tr>
</tbody>
</table>
This instruction forms in the A-Register the bit by bit logical product of \( d \) and the lower 6 bits of A. In the final result the upper 12 bits of A are zero.

Example: \[ A = 001110101011001001 \]
\[ d = 001010 \]
\[ \begin{array}{c}
000000000000001000 \\
\end{array} \]

RNI @ P41

REFERENCES:

LÔGICAL

July 15, 165

31-0
Exercise #1 - Code an instruction that saves just the lower 6-bits of the A-Register, setting the upper 12 bits to zero.

**Answer**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>LPN 77B</strong></td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction that saves the $2^3$ through the $2^5$ bits of the A-Register, setting the rest to zeros.

**Answer**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>LPN 70B</strong></td>
<td></td>
</tr>
</tbody>
</table>
SELECTIVE CLEAR
NO ADDRESS

MACHINE

LOCATION	OPERATION	ADDRESS FIELD

(p)

11 5 0

13 d

Constant
Symbol
Symbol \pm Constant
Symbol \mp Symbol

The above values must result in an octal value in the range of 00-77. The decimal equivalents are 00-63.

Mnemonic Operation Code

This instruction clears any of the lower 6 bits of the A-Register where corresponding bits of d are one. The upper 12 bits of A are not altered.

Example: \[ A = 001110101011001001 \]
\[ d = 001010 \]
\[ 001110101011000001 \]

RNI @ P+1

REFERENCES:

LOGICAL July 15, 1965 32-0
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → $A \rightarrow \overline{A}$ → $P + 1 \rightarrow P$ → EXIT RNI

6400/6600

1 USEC

TIMING

475 NSEC

1 USEC

RNI

SLOT

SLOT TIME = 100 NSEC

32-1
Exercise #1 - Code an instruction that clears any of the lower 6-bits of A that correspond to the $2^0$, $2^2$, $2^4$ bit positions.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>SCN 25B</strong></td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction that clears Flag Bit $2^0$ and leaves the other flag-bits unaltered.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>SCN 1</strong></td>
<td></td>
</tr>
</tbody>
</table>
This instruction forms in the A-Register the bit by bit logical product of the contents of A and the 18-bit quantity \( dm \), consisting of \( d \) as the upper 6 bits and \( m \) as the lower 12 bits.

Example: \( A = 001110101011001001 \)
\( dm = 001110000011001010 \)
\( 001110000011001000 \)

RNI @ P+2
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH OPERAND → A. dm → A

P + 1 → P

EXIT RNI

TIMING

6400/6600 2 USEC

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction that saves the upper 6-bits of the A-Register and sets the lower 12 bits to zeros.

**ANSWER**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LPC</td>
<td>770000B</td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction that clears the $2^3$ through $2^5$ and $2^9$ through $2^{11}$ bits in the A-Register, and leaves the other bits unaltered.

**ANSWER**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LPC</td>
<td>770707B</td>
</tr>
</tbody>
</table>
This instruction forms in the A-Register the bit by bit logical difference of the contents of A and the 18-bit quantity dm, consisting of d as the upper 6-bits and m as the lower 12 bits.

Example: \[ A = 00111010101101001 \]
\[ dm = 000010000000001010 \]
\[ 001100101011100011 \]

RNI @ P+2

REFERENCES:

LÔGICAL

July 15, '65
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH OPERAND → A >> dm → A* → P + 1 → P → EXIT RNI

* >> = Logical difference

TIMING

6400/6600 2 USEC

475 NSEC 1 USEC 475 NSEC 1 USEC

RNI READ OPERAND dm

SLÔT TIME = 100 NSEC
Exercise #1 - The $2^{12}$ bit in the A-Register may be set, the others will be zero if the results from previous were correct. Code the list of two instructions that could test to see if the bit is present.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMC 010000B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - In looking for a particular number in a list of numbers, code an instruction that might be the list of two that tests to see if the number is found. The number is 6162.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMC 6162B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This instruction forms in the A-Register the bit by bit logical difference of the lower 12 bits of A and the contents of location d_(00dd). This is equivalent to complementing individual bits of A which correspond to one bits in the contents of location d. The upper 6-bits of A are not altered.

Example: \[ A = 001110101011001001 \]
\[ d = \begin{array}{c}
01010001010 \\
00111011111100011
\end{array} \]
FLOW DIAGRAM

ENTER \rightarrow \text{FETCH INSTRUCTION} \rightarrow \text{FETCH OPERAND (d)} \rightarrow \text{A} \wedge \text{v(d)} \rightarrow \text{A}^* \rightarrow P + 1 \rightarrow P \rightarrow \text{EXIT RNI}

\text{\text{* \wedge = Logical difference}}

6400/6600 \hspace{2cm} 2 \text{ USEC}

TIMING

\text{\{RNI \rightarrow \text{SLOT} \rightarrow \text{475 NSEC} \rightarrow \text{SLOT} \rightarrow \text{READ OPERAND (d)} \rightarrow \text{SLOT} \rightarrow \text{475 NSEC} \rightarrow \text{SLOT} \rightarrow \text{RNI}\}}

\text{SLOT TIME = 100 NSEC}
Exercise #1 - The A-Register contains 770000. Code an instruction to set a 12-bit number in memory in A, which may have another 12-bit number added to it. Let $d$, which is some value 00-77, be called FORMADDR.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMD FORMADDR</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - In using the 6600 scope (display), A contains the X-coordinate digit (bits $2^9-2^{11}$). Code an instruction that will set the coordinate position in the A-Register along with the coordinate. The 1st reference is to COOR which is some value 00-77.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMD COOR</td>
<td></td>
</tr>
</tbody>
</table>

Note: Assume the operand is some value OXXX.
This instruction forms in the A-Register the bit by bit logical difference of the lower 12-bits of A and the 12-bit operand obtained by indirect addressing. This is equivalent to complementing individual bits of A which correspond to one bits in the operand. The upper 6 bits of A are not altered.

Example: $A = 001110101011001001$

$((d)) = 010100001010$

$00111011111000011$

RNI @ P+1
Exercise #1 - In building a word to send out to the display unit, which must be a number 0X0X, we find A contains 0X00. Code an instruction to set in the lower 6-bits. \( \delta \) is the 1st reference called BETA and the contents of BETA might be called GAMMA, which is the address of the operand.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMI BETA</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code a one word instruction that makes three memory references and forms the logical difference with the contents of A. Let \( d = 70_8 \), and the (70) of = LENGTH, which is the operand address.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMI 7OB</td>
<td></td>
</tr>
</tbody>
</table>
Logical Difference
Memory Index

\[(P) \quad (P+1)\]

\[
\begin{array}{cccc}
53 & d & m & 0 \\
11 & 5 & 0 & 11 & 0
\end{array}
\]

MACHINE

LOCATION OPERATION ADDRESS FIELD

\[
\text{LMM} \quad m, d
\]

Separator
space or comma
Constant
Symbol
Symbol $\pm$ Constant
Symbol $\pm$ Symbol

The value of $d$, must result in an octal value in the range of 00-77. The value of $m$, must result in an octal value in the range of $2^{12}-1$.

Mnemonic Operation Code

This instruction forms in the A-Register the bit by bit logical difference of the lower 12 bits of A and a 12 bit operand obtained by indexed addressing. This is equivalent to complementing individual bits of A which correspond to one bits in the operand. The upper 6 bits of A are not altered.

Note: If $d = 0$, the operand address is simply $m$.
If $d \neq 0$, then $m$ plus the contents of location $d$, $m+(d)$ is the operand address; thus the contents of $d$ may be used as an index quantity to modify operand addresses.

RNI @ P+2.

REFERENCES:

LOGICAL

June 1, '66

37-0
FLOW DIAGRAM

ENTER → Fetch Instruction

\[ d = 0 ; \] Yes → Fetch Operand \( m \)

No → Fetch Operand \( m + (d) \)

\( I \) → Fetch Operand \( A \) → \( P+2 \rightarrow P \) → Exit RNI

---

6400/6600

3-4 USEC

TIMING

\( \text{INDEX } d = 0 \)

\( \text{INDEX } d \neq 0 \)

\( \text{Slot Time} = 100 \text{ NSEC} \)
Exercise #1 - Code a two word instruction, which makes three memory references and no indexing to perform the logical difference with the contents of A.

Let $m = \text{CHECKSUM}$, and $d$ must $= 0$.

The contents of CHECKSUM would be the operand.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMM CHECKSUM</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code a two word instruction which makes four memory references, which includes indexing, to perform the logical difference with the contents of A.

Let $m = \text{LOCATION}$, which is the operand address, and $d = 71_8$ (the index location).

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LMM LOCATION, 71B</td>
<td></td>
</tr>
</tbody>
</table>
This instruction adds the 12 bit quantity in location d to the contents of the A-Register and stores the lower 12-bits of the result back in location d. The result is also left in the A-Register at the end of the operation.

RNI @ P+1.
FLOW DIAGRAM

ENTER → FETC

H INSTRUCTION

FEC

H TH CH OPCERAND

A+(d) → A

J

J → A → OPERATION

P+1 → P

EXIT RNI

TIMING

6400/6600

3 USEC

1 USEC 1 USEC 1 USEC

475 NSEC 475 NSEC 475 NSEC

RNI SLOT READ OPCERAND (d) STORE A

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction to add an operand to the contents of the A-Register and store the results.  Let d = RUNSUM, which is the operand address and has a value in the range of 00-77.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAD</td>
<td>RUNSUM</td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction to form the checksum of all the information on a 80 column card just read into memory. Let d = CARDSUM, which is the operand address, and takes on a value in the range of 00-77. The operand in location CARDSUM would start out = to zero.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAD</td>
<td>CARDSUM</td>
</tr>
</tbody>
</table>
This instruction adds one to the original value in location \( d \) and stores the result back in location \( d \). The result is also left in the A-Register at the end of the operation. The original contents of A are destroyed. RNI @ P+1.
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH OPERAND → OPER +1 → A → K

K → OPER ADDR → P + 1 → P → EXIT RNI

6400/6600

3 USEC

TIMING

---- 1 USEC ---- 1 USEC ---- 1 USEC ----

475 NSEC 475 NSEC 475 NSEC

RNI Slot READ OPERAND (d) Slot

Slot Time = 100 NSEC
Exercise #1 - Code an instruction to update an index location. Let $d$, the index location, be called INCREM, which is an address with a value in the range of 00-77.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0D</td>
<td>INCREM</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction to update a loop counter which would leave the result in the A-Register so it could next be tested. Let $d = LO\text{O}NT$.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0D</td>
<td>LO\text{O}NT</td>
<td></td>
</tr>
</tbody>
</table>
This instruction subtracts one from the original value in location \( d \) and stores the result back in location \( d \). The result is also left in the A Register at the end of the operation. The original contents of A are destroyed. RNI @ P+1.
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH OPERAND (d) → OPER → A → L

L → A → OPER ADDR → P + 1 → P → EXIT RNI

6400/6600
3 USEC

TIMING

1 USEC ← 475 NSEC ← 1 USEC ← 475 NSEC ← 1 USEC ← 475 NSEC

RNI
SLOT
SLOT
SLOT

READ OPERAND (d)
STORE A

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction that would decrement a loop counter (by one until it equals zero,) leaving the result in the A-Register each time to be tested.

Let \( d = L\text{OPCNT} \)

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S0D LOPCNT</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction that would decrement a Y-coordinate position in Dot Mode using the display unit.

Let \( d = POSITION \)

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S0D POSITION</td>
<td></td>
</tr>
</tbody>
</table>
This instruction adds to the contents of the A-Register the operand obtained from indirect addressing. \( \text{OPERAND} = ((d)) \). The resultant sum is left in the A-Register at the end of the operation and the lower 12 bits of A replace the original operand in memory. RNI @ P+1
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH OPER ADDR (d) → FETCH OPERAND ((d)) → M

M → A + OPER → A → A → OPER ADDR → P + 1 → P → EXIT RNI

6400/6600  4 USEC

TIMING

1 USEC  1 USEC  1 USEC  1 USEC

475 NSEC  475 NSEC  475 NSEC  475 NSEC

RNI  READ OPER ADDR (d)  READ OPER ((d))  STORE A

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction to add the contents of A to an operand and replace the original operand with the results using the indirect method.

Let \( d = 60_8 \), and the contents of \( 60 = \text{MESSAGE} \), which is

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAI</td>
<td>60B</td>
</tr>
</tbody>
</table>

Exercise #2 - Code a replace add instruction which 1st references location \( 50_8 \), called LENGTH, and 2nd location \( 5000_8 \), called BUFFER.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAI</td>
<td>LENGTH</td>
</tr>
</tbody>
</table>
REPLACE ADD ONE INDIRECT

MACHINE

LOCATION OPERATION ADDRESS FIELD

11 5 0

46 d

Constant
Symbol
Symbol + Constant
Symbol ÷ Symbol

The above values must result in an octal value in the range of 00-77.
The decimal equivalents are 00-63.

Mnemonic Operation Code

DESCRIPTION

The instruction adds one to the operand obtained from indirect addressing. OPERAND = ((d)). The resultant sum is left in the A-Register at the end of the operation and the lower 12 bits of A replace the original operand in memory. The original contents of A are destroyed. RNI @ P+1

REFERENCES:

REPLACE

July 15, '65 42-0
FLOW DIAGRAM

ENTER

FETCH INSTRUCTION

FETCH \( \phi \text{PER ADDR} \)

(\(d\))

FETCH \( \phi \text{PERAND} \)

((\(d\)))

\(N\)

\(\phi \text{PER} + 1 \rightarrow A\)

A \(\rightarrow \phi \text{PER ADDR}\)

P + 1 \(\rightarrow P\)

EXIT RNI

6400/6600

4 USEC

TIMING

\[1 \text{ USEC} \rightarrow 1 \text{ USEC} \rightarrow 1 \text{ USEC} \rightarrow 1 \text{ USEC}\]

\[\frac{475}{\text{NSEC}} \rightarrow \frac{475}{\text{NSEC}} \rightarrow \frac{475}{\text{NSEC}} \rightarrow \frac{475}{\text{NSEC}}\]

\(\text{RNI} \rightarrow \text{READ } \phi \text{PER ADDR} \rightarrow \text{READ } \phi \text{PER} \rightarrow \text{STORE } A\)

(d) (\(\phi \text{PER})\)

SL\(\text{T TIME} = 100 \text{ NSEC}\)
Exercise #1 - Code an instruction to update an operand that is updated from various parts of a program. All 1st references are made to location BUMP and 2nd references to location LAYOUT.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AOI BUMP</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction which updates a clock count each time the RTC goes through its cycle. Let d = CNT and the 2nd reference = CLOCK.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AOI CNT</td>
<td></td>
</tr>
</tbody>
</table>
This instruction subtracts one from the operand obtained from indirect addressing, \( \text{OPERAND} = ((d)) \). The resultant difference is left in the A-Register at the end of the operation and the lower 12 bits of A replaces the original operand in memory. The original contents of A are destroyed. RNI @ P+1.
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH OPER ADDR(d) → FETCH OPERAND((d)) → φ

φ → OPER -1 → A → A → OPER ADDR → P + 1 → P → EXIT RNI

6400/6600 4 USEC

TIMING

1 USEC 1 USEC 1 USEC 1 USEC

475 NSEC 475 NSEC 475 NSEC 475 NSEC

RNI READ OPER ADDR READ OPERAND STORE A

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction which decrements a record count and makes a 1st reference to location RELOC and a 2nd reference to location RECORD.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 1 0 3 1 4 1 5 1 6 1 7 1 8</td>
<td>S01 RELOC</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction which subtracts one from an available table. The 1st reference is to TABLOC, and the 2nd reference to AVAILTAB.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 1 3 4 5 6 6 1 7 8</td>
<td>S01 TABLOC</td>
<td></td>
</tr>
</tbody>
</table>
This instruction adds the contents of the A-Register to the operand obtained from indexed addressing. The resultant sum is left in the A-Register at the end of the operation and the lower 12-bits of A replace the original operand in memory. Note: If \( d = 0 \), the operand address is simply \( m \). If \( d \neq 0 \), then \( m + (d) \) is the operand address. Thus the contents of \( d \) may be used as an index quantity to modify operand addresses. RNI @ P+2.
Exercise #1 - Code an instruction to add an operand and the A-Register and replace the result using the memory index instruction with no indexing. Let m = MESSAGE, and d must = 0.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAM MESSAGE</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Using the same exercise as above, code an instruction using memory index with index modification. Let d = MOD.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAM MESSAGE, MOD</td>
<td></td>
</tr>
</tbody>
</table>
This instruction adds one to the operand obtained from indexed addressing. The resultant sum is left in the A-Register at the end of the operation and the lower 12-bits of A replace the original operand in memory. The original contents of A are destroyed.

Note: If \( d = 0 \), the operand address is simply \( m \). If \( d \neq 0 \), then \( m \) plus the contents of location \( d \), \( m + (d) \) is the operand address. Thus the contents of \( d \) may be used as an index quantity to modify operand addresses. RNI @ P+2.
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → d = 0? → Yes → FETCH OPER ADDR → Q
               No → Q

Q → FETCH OPER ADDR → m + (d)

FETCH OPERAND → OPER +1 → A → OPER ADDR → P + 2 → P → EXIT RNI

6400/6600

- 5 USEC

TIMING

INDEX d = 0

RNI → READ OPER ADDR and Addr

INDEX d ≠ 0

READ INDEX

READ OPER ADDR

(m + (d)) and Addr

STORE A

SLOT TIME = 100 NSEC

45-1
Exercise #1 - Code an instruction to update an operand by one. Use a memory index with no index modification. Let \( m = \text{AREA}. \)

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A0M AREA</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Using the example above, code the instruction using the index modification, and let \( d = \text{IND1}. \)

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A0M AREA, IND1</td>
<td></td>
</tr>
</tbody>
</table>
This instruction subtracts one from the operand obtained from indexed addressing. The resultant difference is left in the A-Register at the end of the operation and the lower 12-bits of A replace the original operand in memory. The original contents of A are destroyed.

Note: If \( d = 0 \), the operand address is simply \( m \). If \( d \neq 0 \), then \( m \) plus the contents of location \( d \), \( m + (d) \) is the operand address. Thus the contents of \( d \) may be used as an index quantity to modify operand addresses. RNI @ P+2.
FLOW DIAGRAM

ENTER -> FETCH INSTRUCTION -> d = 0?
  Yes -> FETCH OPER-ADDR m -> R
  No -> FETCH OPER-ADDR m + (d)

R -> FETCH OPERAND -> OPER - 1 -> A

A -> OPER-ADDR -> P + 2 -> P
EXIT RNI

6400/6600 4-5 USEC

TIMING

INDEX d = 0
NO INDEX

INDEX d ≠ 0

READ INDEX (d) m + (d)
READ OPER-ADDR (m + (d))
READ OPER-ADDR and Subtract STORE A

1 USEC 1 USEC 1 USEC 1 USEC 1 USEC
NSEC NSEC NSEC NSEC NSEC

RNI

1 USEC 1 USEC 1 USEC 1 USEC 1 USEC
NSEC NSEC NSEC NSEC NSEC

READ OPER-ADDR and Subtract STORE A

475< m 475< m 475< m 475< m
NSEC NSDC NSEC NSDC

INDEX d ≠ 0

READ INDEX (d) m + (d)
READ OPER-ADDR (m + (d))
READ OPER-ADDR and Subtract STORE A

475< m 475< m 475< m 475< m
NSEC NSDC NSEC NSDC

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction to decrement an operand by one, using memory index with no index modification. Let \( m = \text{FLAG}. \)

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>SØM</td>
<td>FLAG</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Using the above exercise code the instruction using indexing. Let \( d = \text{TEMPO}. \)

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>SØM</td>
<td>FLAG, TEMPO</td>
<td></td>
</tr>
</tbody>
</table>
EXCHANGE JUMP

This instruction transmits an 18-bit address from the A-Register to the central processor with a signal which tells the central processor to perform an exchange jump, with the address in A as the starting location of a file of 16 words containing information about the CP Program to be executed. The 18 bit initial address must be entered in A before this instruction is executed. The central processor replaces the file with similar information from the interrupted CP Program. The PP Program is not interrupted. RNI @ P+1.
* - Central may be busy for an indefinite amount of time, depending upon the program in operation in the Central Processor.

** - When the PP sets the Central Busy condition, it will stay set for a minimum of 2 usec. However, the PP is free to continue with the next instruction.

6400/6600

Timing

Æ 475 NSEC

Slot Time = 100 NSEC
Exercise #1 - Code an instruction to initiate the CP on a new program.

Note: The A-Register must already contain the 18-bit central memory address.
This instruction transfers the contents of the central processor program address register to the peripheral processor A-Register to allow the PP to determine whether the central processor is running.
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → CENT. PROC. P REG → A → P + 1 → P → EXIT RNI

TIMING

6400/6600

1 USEC

↓ 475 NSEC ↓

RNI

SLOT TIME = 100 NSEC
Exercise #1 - Code the instruction to read the contents of the CP P-Register (Program Address Counter)

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RPN</td>
<td></td>
</tr>
</tbody>
</table>

ANSWER
This instruction transfers a 60-bit central memory word to 5 consecutive PP memory locations. The A-Register must contain the 18-bit CM address before the instruction is executed. The 60-bit CM word is disassembled beginning at the left with the location specified by \( d \) receiving the left most 12-bit word; \( d+1 \), the next 12-bit word, and so on.

RNI @ P+1.
6400/6600

MINIMUM 6 usec

1 USEC ** 1 USEC * 1 USEC * 1 USEC ** 1 USEC

475 NSEC 475 NSEC 475 NSEC 475 NSEC 475 NSEC 475 NSEC

SLOT SLOT SLOT SLOT SLOT SLOT

RNI STORE IN STORE IN STORE IN STORE IN STORE IN

d d+1 d+2 d+3 d+4

** This cycle will loop until Cent. Busy FF is not being used by another PP, and until there is an empty stage in the Read Pyramid.

* Each one of these cycles will loop if the next stage in the Read Pyramid is also full, from another processor.

SLØT = 100 nsec.
Exercise #1 - Code the instruction to transfer a Cent. Memory Word (60-bits) to 5 PP Mem. locations, (12-Bit Words)

Let \( d = 76 \)

\[
\begin{array}{cccccc}
59 & 48 & 47 & 36 & 35 & 24 & 23 & 1211 & 0 \\
\end{array}
\]

\[
\begin{align*}
2^{48} - 59 & \rightarrow 0076 \\
2^{36} - 47 & \rightarrow 0077 \\
2^{24} - 35 & \rightarrow 0100 \\
2^{12} - 23 & \rightarrow 0101 \\
2^0 - 11 & \rightarrow 0102 \\
\end{align*}
\]

**ANSWER**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CRD</td>
<td>76B</td>
</tr>
</tbody>
</table>

Note: A initially contains the 18-bit Cent. Memory Address

Exercise #2 - Code the instruction to transfer a Cent. Memory Word starting at PP location STAN.

STAN may = any value 00-77g.

**ANSWER**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9</td>
<td>CRD STAN</td>
<td></td>
</tr>
</tbody>
</table>

Note: See above note
This instruction reads a block of 60-bit words from central memory into PP memory. The A-Register contains the 18-bit CM starting address and must be loaded prior to the execution of this instruction. The contents of A are increased by one as each 60-bit CM word is disassembled and stored. During execution of this instruction, the original contents of the P-Register are stored in PP location 0000 and the address of the first word to be stored into PP memory, m, goes to the P-Register. P is updated for each new address. The original contents of P are restored upon completion. RNT @ P+2. The block length or number of CM words to be read is contained in location d, CM addresses = (A) + (d)-1. The block length also goes to the Q Register where it is reduced by one as each CM word is processed. The transfer is completed when Q=0. The PP memory addresses = m+5(d)-1, where m is the starting address in the PP.
6400/6600

MIN. 5-PLUS 5/WORD usec

1 USEC 1 USEC 1 USEC 1 USEC

475 NSEC 475 NSEC 475 NSEC 475 NSEC

SLOT SLOT SLOT SLOT

RNI READ BLOCK P+1 0000 m P
LENGTH STORE CYCLES

SLOT - 100 nsec
* n = 0, on 1st 60-bit word, 5 on 2nd 60 bit word, etc.
Exercise #1 - Code an instruction to transfer a block of words from Cent. Memory to PP Memory.

Let \( d = \text{LEN} = 00-77_8 \)

\((\text{LEN}) = \# \text{ of Words}\)

Let \( m = \text{FWA}, \text{first word address}\)

Note: The \((A - \text{Register})\) is the starting address in Cent. Memory.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CRM, FWA, LEN</td>
<td></td>
</tr>
</tbody>
</table>

50-3
This instruction assembles five successive 12-bit words into a 60-bit word and stores the word in central memory. The 18-bit CM address must be in the A-Register prior to the execution of the instruction. The first word to be read out of PP memory is contained in location d. This word appears as the leftmost 12-bits of the 60-bit word. The remaining 12-bit groups are taken from successive addresses in PP memory. RNI @ P+1.
** Central will be busy minimum 600-700 nsec.

---

*Each one of these cycles will loop if the next stage in the write pyramid is also full, from another processor.*

\[
\text{SL0T} = 100 \text{ nsec}
\]
Exercise #1 - Code an instruction to transfer 5 PP words to Cent. Memory.

Let \( d = \text{FWWRITE} = 75_8 \)

\[
\begin{align*}
(0075) & \rightarrow 2^{48} - 59 \\
(0076) & \rightarrow 2^{36} - 47 \\
(0077) & \rightarrow 2^{24} - 35 \\
(0100) & \rightarrow 2^{12} - 23 \\
(0101) & \rightarrow 2^0 - 11
\end{align*}
\]

Note: \( a \) contains the 18-bit Cent. Memory Address

Exercise #2 - Code an instruction to transfer, from PP Mem, a Cent. Mem word to Cent. Mem.

Let \( d = \text{PPWRITE} = \text{a value } 00-77_8 \).

Note: See above note
CENTRAL WRITE BLOCK

MACHINE

LOCATION

OPERATION

ADDRESS FIELD

Separator
space or comma
Constant
Symbol
Symbol + Constant
Symbol - Symbol

The value of d, must result in an octal value in the range of 00-77. The value of m, must result in an octal value in the range of 2\(^12\)-1.

Mnemonic Operation Code

This instruction assembles a block of 60-bit words and writes them in central memory. The A-Register contains the beginning 18 bit CM address and must be loaded prior to the execution of this instruction. Then the address in A is increased by one after each 60-bit word is assembled to provide the next Cm address. The (d) specify the number of 60-bit words to write. This count also goes to the Q-Register where it is reduced by one as each Cm word is assembled. The transfer is completed when Q=0. The Cm Address = (A) + (d)-1. The PP memory addresses = m + 5(d)-1, where m is the starting address in the PP. During execution of this instruction the original contents of the P-Register are stored in PP location 0000 and the address of the first word to be fetched from PP memory, m, goes to the P-Register. P is updated for each new address. The original contents of P are restored upon completion. RNI @ P+2.

REFERENCES:

CENTRAL MEMORY July 15, '65 52-0
FLOW DIAGRAM

1. ENTER
2. FETCH INSTRUCTION
3. FETCH BLOCK LENGTH
4. P/I = 0000
5. P = PRG CNT
6. 7
7. m → P
   m = ADDR FIRST WORD
8. WRITE BLOCK
   Page 52-2
9. (0000) → P
10. 8
11. P + 1 → P
12. EXIT RNI

6400/6600

Min. 5-Plus 5/word usec

<table>
<thead>
<tr>
<th>1 USEC</th>
<th>1 USEC</th>
<th>1 USEC</th>
<th>1 USEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>475 NSEC</td>
<td>475 NSEC</td>
<td>475 NSEC</td>
<td>475 NSEC</td>
</tr>
</tbody>
</table>

RNI
READ BLOCK LENGTH
(0000) + 1 → P
5 CYCLES/WORD
FETCH CYCLES

SL&T = 100 nsec
Exercise #1 - Code an instruction to transfer a block of words from PP memory to Central Memory.

Let \( d = \text{LENGTH} \)

\( (\text{LENGTH}) = \# \text{ of words} \)

Let \( m = \text{FIRWORD} \)

Note: A contains the address of Cent. Mem. that is to receive the first word.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 0 0</td>
<td>CWM FIRWORD, LENGTH</td>
</tr>
</tbody>
</table>
ACTIVE JUMP

(P)    (P + 1)

11  5  0 11  0

MACHINE

64  d  m

LOCATION   OPERATION   ADDRESS FIELD

ASPER

AJM m, d

Separator
space or comma
Constant
Symbol
Symbol + Constant
Symbol + Symbol

The value of d, must result in an octal value in the range of 00-13. The value of m, must result in the range of $2^{12} - 1$.

Mnemonic Operation Code

DESCRIPTION

Jump to $m$ if channel $d$ active. This instruction provides a conditional jump to a new program sequence beginning at address $m$ if the channel specified by $d$ is active. If the channel is inactive, the current program sequence continues.

RNI @ P+2

REFERENCES:

INPUT/OUTPUT

July 15, '65
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH JUMP ADDR \( -m \) → CHAN ACTIVE

Yes → EXIT

No → \( P+2 \rightarrow P \)

\( \rightarrow m \rightarrow P \)

TIMING

6400/6600

2 USEC

<table>
<thead>
<tr>
<th>475 NSEC</th>
<th>1 USEC</th>
<th>475 NSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNI</td>
<td>SLOT</td>
<td>SLOT</td>
</tr>
<tr>
<td>READ JUMP ADDR ( m )</td>
<td>SLOT</td>
<td>SLOT</td>
</tr>
</tbody>
</table>

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction to test if a channel is active, and if it is jump to a new sequence of instructions.

Let \( d = 00 \) (channel #) and \( m = \text{NEWADDR} \)

Note: RNI @ P+2 if the condition is not met.

**Answer**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>AJM NEWADDR,0</code></td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Using the exercise above test channel 13 and loop (Wait) until the channel becomes free.

Note: See above note.

**Answer**

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>AJM *,13B</code></td>
<td></td>
</tr>
</tbody>
</table>
Jump to \( m \) if channel \( d \) inactive. This instruction provides a conditional jump to a new program sequence beginning at address \( m \) if the channel specified by \( d \) is inactive. If the channel is active, the current program sequence continues.

RNI @ P+2

REFERENCES:

INPUT/OUTPUT July 15, '65 54-0
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH JUMP ADDR → CHAN INACTIVE

Yes → m → P

No → P + 2 → P → EXIT RNI

6400/6600

2 USEC

TIMING

475 NSEC → 1 USEC → 475 NSEC

RNI SLOT READ JUMP ADDR

SLOT TIME = 100 NSEC
Exercise #1 - Test channel 1, and if it is inactive jump to a location called ACTIVATE, otherwise RNI @ P+2.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>IJM</td>
<td>ACTIVATE, 1</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Test channel 13, and if it is inactive jump to location STATUS, otherwise RNI @ P+2.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>IJM</td>
<td>STATUS, 13</td>
<td></td>
</tr>
</tbody>
</table>
Jump to m if channel d is full. This instruction provides a conditional jump to a new program sequence beginning at address m if the channel specified by d is full. If the channel is empty, the current program sequence continues. A channel is full: 1) Input operation and the input equipment sends a word to the channel register and sets the full flag. The channel goes empty when the PP accepts the word and clears the full flag. 2) Output operation and the PP places a word in the channel register and sets the full flag. The channel is empty when the external equipment accepts the word and clears the full flag.

REFERENCES:

INPUT/OUTPUT  July 15, '65  55-0
FLOW DIAGRAM

ENTRY

FETCH INSTRUCTION

FETCH JUMP ADDR

CHAN FULL?

No

P + 2 → P

Yes

m → P

EXIT RNI

6400/6600

T I M I N G

SLOT

RNI

READ JUMP ADDR

SLOT

475 NSEC

475 NSEC

1 USEC

1 USEC

SLOT TIME = 100 NSEC

2 USEC
Example #1 - Code an instruction to test a channel to see if an output operation is completed. If not wait and otherwise RNI @ P+2 and disconnect the channel (07).

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>FJM 9,7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Jump to m if channel d empty. This instruction provides a conditional jump to a new program sequence beginning at address m if the channel specified by d is empty. If the channel is full, the current program sequence continues. (See page 55-0 for full and empty explanation.)
FLOW DIAGRAM

ENTER → FETCH INSTRUCTION → FETCH JUMP ADDR → EMPTY CHAN?

No → No

P + 2 → P

Yes → m → P → EXIT RNI

6400/6600

2 USEC

TIMING

475 NSEC → 1 USEC → 475 NSEC

RNI

SLOT

READ JUMP ADDR

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction to test channel 100 for the empty condition. If it is empty jump to CHANSTAT, otherwise RNI @ P+2.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EJM</td>
<td>CHANSTAT, I0B</td>
</tr>
</tbody>
</table>
This instruction transfers a word from input channel \( d \) to the lower 12 bits of the A-Register. The upper 6-bits are zero. Before this instruction is executed the channel must have been set activate, and the external equipment previously selected for this operation.  

\[ \text{RNI} @ P+1 \]

*The real time clock is read by simply giving this instruction with channel \( 14_8 \).
The number of times it loops depends upon the speed of the external equipment. However, for status responses, the select and following activate bring the data word in the input register.

SLØT TIME = 100 NSEC
Exercise #1 - Code an instruction to bring the status in from channel 13 on a piece of I/O equipment that was properly selected for that operation, prior to this instruction. The upper 6-bits of A will contain zeros.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAN</td>
<td>13B</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction to make reading from the RTC (Real Time Clock).

No previous instructions are required.

Note: RTC is on channel 14 and used for input only.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAN</td>
<td>14B</td>
<td></td>
</tr>
</tbody>
</table>
Input (A) words from channel d to m. This instruction transfers a block of words from input channel d to PP memory, beginning at a location specified by m. The A-Register contains the block length and is reduced by one as each word is read. The input operation is completed when A=0. The equipment selection and channel activate must have been done prior to this instruction. During this instruction the current contents of the P-Register are stored in PP location 0000. The P-Register is used as an address to store the data, and is restored to its original contents upon completion of the operation. RNI @ P+2.
6400/6600 MINIMUM 4 PLUS 1/WORD usec

* The number of cycles depend upon the number of words, plus the speed of the input/output equipment.

SLOT = 100 nsec
Exercise #1 - Assume the external equipment has been properly selected and the channel is ready. Code an instruction to input a block of words from Channel 0, to starting address DISC (this would be m) the A-Register would contain the number of words to be transferred. All transfers will be 12-Bits.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>123457789</td>
<td>IAM DISC,0</td>
<td></td>
</tr>
</tbody>
</table>
Output (A) on channel d. This instruction transfers a word from the lower 12 bits of the A-Register to output channel d. The external equipment and channel activate must have been previously set up.
*The number of times it loops depends upon the speed of the equipment it is communicating with.

SLØT TIME = 100 NSEC
Exercise #1 - Code an instruction to output the lower 12-bits of the A-Register to a previously selected piece of equipment output to channel 128.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ØAN 12B</td>
<td></td>
</tr>
</tbody>
</table>
**OUTPUT A BLOCK**

(P)    (P+1)

```
           11    5   0   11    0
MACHINE  73    d    m
```

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>6AM m, d</td>
</tr>
</tbody>
</table>

Separator
space or comma
Constant
Symbol
Symbol + Constant
Symbol + Symbol

The value of d, must result in an octal value in the range of 00-13. The value of m, must result in an octal value in the range of 2^12-1.

Mnemonic Operation Code

**DESCRIPTION**

Output (A) words from m on channel d. This instruction transfers a block of words on output channel d from PP memory, beginning at the location specified by m. The contents of the A-Register specify the number of words to be sent and are reduced by one as each word is sent. The operation is completed when A = 0. The equipment selection and channel activate must have been done prior to this instruction. During this instruction the current contents of the P-Register are stored in location 0000. The P-Register is used as an address to store the data, and is restored to its original contents upon completion of the operation. RNI @ P+2.

**REFERENCES:**

INPUT/OUTPUT

July 15, '65
6400/6600 MINIMUM 4 PLUS 1/WORD usec

- 1 USEC - 1 USEC - 1 USEC - 475 NSEC
- 475 NSEC - 475 NSEC - 475 NSEC - 475 NSEC

**BLOCK OUTPUT**

- RNI
- STORE P+1
- READ FIRST WORD
- 1 CYCLE/WORD FETCH & TEST CYCLES

* The number of cycles depend upon the number of words, plus the speed of the input/output equipment.

**SLöt = 100 nsec**
Exercise #1 - Code an instruction to output a block of information to a previously selected piece of equipment. The equipment is on Channel 3, the first word address in memory is called PUNCH, and the A-Register contains the length of the block.

All word transfers will be 12-bits.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0A</td>
<td>PUNCH, 3</td>
</tr>
</tbody>
</table>
This instruction activates the channel specified by \( d \). This instruction must precede a 70-73 instruction. Activating a channel alerts and prepares the I/O equipment for the exchange of data.

Note: Trying to activate a channel that is already active would hang up that one PP, until a monitor, (or some other PP) would deactivate it.

RNI @ P+1
FILL INSTRUCTION

Yes

No

CHAN

INACTIVE

ACTIVATE

CHANNEL

P + 1 → P

EXIT

RNI

---

6400/6600

2 USEC

---

1 USEC

*1 USEC

475 NSEC

1 USEC

475 NSEC

---

RNI

SLOT

DUMMY REFERENCE

ACTIVATE CHANNEL

---

*Instruction will loop indefinitely if the channel is already active.

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction to activate channel 0.

Note: If the channel is already active the processor executing this instruction will loop indefinitely.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACN 0</td>
<td></td>
</tr>
</tbody>
</table>

Exercise 2 - Code an instruction to activate Channel 138.

Note: See above note

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACN 13B</td>
<td></td>
</tr>
</tbody>
</table>
DISCONNECT CHANNEL

(P)

MACHINE

11  5  0

75  d

LOCATION   OPERATION   ADDRESS FIELD

ASPER

DCN d

Constant
Symbol + Constant
Symbol ÷ Symbol

The above value of d, must result in an octal value in the range of 00-13. The decimal equivalents are 00-11.

Mnemonic Operation Code

This instruction deactivates the channel specified by d. This stops the I/O equipment and the buffer terminates.

Note: Trying to deactivate a channel that is already deactivate would hang up that one PP, until a monitor, (or some other PP) would activate it.

RNI @ P+1

REFERENCES:

INPUT/OUTPUT   July 15, '65   62-0
**Flow Diagram**

1. **ENTER** → **FETCH INSTRUCTION** → **CHAN ACTIVE**
   - **No**
   - **Yes** → **DISCONNECT CHANNEL** → **P + 1 → P**

2. **EXIT RNI**

**Timing**

- **6400/6600**
- **2 USEC**

- **475 NSEC** → **1 USEC** → **475 NSEC** → **1 USEC**

3. **RNI** → **SLOT** → **SLOT**

4. **DUMMY REFERENCE DISCONNECT CHANNEL**

*Instruction will loop indefinitely if the channel is already inactive.

**SLOT TIME = 100 NSEC**
Exercise #1 - Code an instruction to deactivate (or disconnect) Channel 0.

Note: If the channel is already inactive the processor will loop indefinitely.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DCN 0</td>
<td></td>
</tr>
</tbody>
</table>

Exercise #2 - Code an instruction to deactivate channel 13.

Note: See above note.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DCN 13B</td>
<td></td>
</tr>
</tbody>
</table>
Function (A) on channel \( d \). This instruction sends out on channel \( d \) the external function code, previously placed in the lower 12-bits of the A-Register.

Note: Trying to function a channel that is in the active state would hang that one PP indefinitely, until a monitor, or some other PP de-activated that channel.

RNI @ P+1

References:
The chan is also set active and remains active until the external equipment responds with an inactive; therefore, the PP could continue reading instructions, but the chan remains active until the equipment responds.

**The instruction will loop indefinitely if the channel is already active.**

**Slot Time = 100 nsec**
Exercise #1 - Code an instruction that will send as a function code the lower 12-bits of the A-Register. \( d \) specifies the channel number (01).

Note: The channel must be inactive prior to a function code instruction or the processor will loop indefinitely.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FAN 1</td>
<td></td>
</tr>
</tbody>
</table>
FUNCTION CODE m

(P) (P = 1)

MACHINE

11 5 0 11 0
77 d m

LOCATION OPERATION ADDRESS FIELD

Separator space or comma

Constant Symbol
Symbol + Constant
Symbol - Symbol

The value of d must result in an octal value in the range of 00-13.
The value of m must result in an octal value in the range of $2^{12} - 1$.

Mnemonic Operation Code

DESCRIPTION

Function m on Channel d. This instruction sends out on channel d the external function code specified by m.

Note: Trying to function on a channel that is already active would hang up that one PP indefinitely until a monitor, or some other PP deactivated the channel.

RNI @ P+2

REFERENCES:

INPUT/OUTPUT

July 15, '65

64-0
* The channel is also set active and remains active until the external equipment responds with an inactive. Therefore, the PP could continue reading instructions, but the channel remains active until the equipment responds.

**The instruction will loop indefinitely if the channel is already active.

SLOT TIME = 100 NSEC
Exercise #1 - Code an instruction that sends out to a piece of external equipment the select code (m) on channel 6.

Note: See note on page 63-2.

0704 = m = 405 - card reader status request.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OPERATION</th>
<th>ADDRESS FIELD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FNC 0704B,6</td>
<td></td>
</tr>
</tbody>
</table>