THE DESIGN OF AN INTERDATA IMPLEMENTATION
OF THE U. S. NAVY MINI-COBOL

BY

TERRY WAYNE ANDERSON-ROVIA

B. S., LOYOLA UNIVERSITY (CHICAGO), 1964

A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1976

Approved by:

[Signature]

Major Professor
ACKNOWLEDGMENTS

I would like to express my deep gratitude and appreciation to Capt. Grace M. Hopper, USNR and the fine young ladies and men of the U. S. Navy Programming Languages Section without whose generosity Kansas State University would not have the MINI-COBOL compiler. I would also like to express my appreciation to Daniel Codespoti, Lee Allen, and Richard McBride for their assistance in technical matters.
ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE
THIS BOOK CONTAINS NUMEROUS PAGES WITH THE PAGE NUMBERS CUT OFF

THIS IS AS RECEIVED FROM THE CUSTOMER
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>List of Tables</th>
<th>vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. THE DESIGN OF AN INTERDATA IMPLEMENTATION OF MINI-COBOL</td>
<td>4</td>
</tr>
<tr>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>IMPLEMENTATION OVERVIEW – HIGH LEVEL</td>
<td>5</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>7</td>
</tr>
<tr>
<td>3. THE DETAIL DESIGN</td>
<td>8</td>
</tr>
<tr>
<td>Introduction</td>
<td>8</td>
</tr>
<tr>
<td>NOVA INSTRUCTION FORMATS</td>
<td>8</td>
</tr>
<tr>
<td>THE FETCH MODULE</td>
<td>10</td>
</tr>
<tr>
<td>THE DECODER MODULE</td>
<td>12</td>
</tr>
<tr>
<td>RULES OF PRECEDENCE</td>
<td>12</td>
</tr>
<tr>
<td>EMULATION ROUTINES</td>
<td>13</td>
</tr>
<tr>
<td>1. EMULATION OF THE JSR</td>
<td>14</td>
</tr>
<tr>
<td>2. EMULATION OF THE JMP</td>
<td>15</td>
</tr>
<tr>
<td>3. EMULATION OF THE DSZ</td>
<td>15</td>
</tr>
<tr>
<td>4. EMULATION OF THE ISZ</td>
<td>16</td>
</tr>
<tr>
<td>5. EMULATION OF THE LDA</td>
<td>17</td>
</tr>
<tr>
<td>6. EMULATION OF THE STA</td>
<td>17</td>
</tr>
<tr>
<td>7. EMULATION OF THE ADD</td>
<td>18</td>
</tr>
<tr>
<td>8. EMULATION OF THE SUB</td>
<td>19</td>
</tr>
</tbody>
</table>
9. EMULATION OF THE AND 20
10. EMULATION OF THE COM 20
11. EMULATION OF THE NEG 21
12. EMULATION OF THE ADC 22
13. EMULATION OF THE INC 23
14. EMULATION OF THE MOV 24
15. EMULATION OF THE Z, O, C FUNCTIONS 24
17. EMULATION OF THE LOAD/NO LOAD 27
18. EMULATION OF THE SKIP FUNCTION 28

SUMMARY

4. VERIFICATION 32
   INTRODUCTION 32
   ABBREVIATIONS IN SAMPLE ALGORITHM 32
   SAMPLE ALGORITHM 33
   ALGORITHM OUTPUT 43
   SUMMARY 45

5. CONCLUSION 47

BIBLIOGRAPHY 48a

APPENDIXED

A. MINI-COBOL LANGUAGE SPECIFICATIONS 49
   INTRODUCTION 50
   OVERVIEW 50
   PHASE I 50
   PHASE I TUPLES 52
   IDENTIFICATION DIVISION WORD 52
   ENVIRONMENT DIVISION WORD 53
COMPARE HALFWORD 274
EXCLUSIVE OR HALFWORD 274
AND HALFWORD 275
BRANCH AND LINK 275
EXTENDED MNEMONICS 276
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Emulation Register Assignments</td>
<td>14</td>
</tr>
<tr>
<td>4.1 Sample Algorithm Trace</td>
<td>39</td>
</tr>
<tr>
<td>4.2 Flow of Generated Code</td>
<td>45</td>
</tr>
<tr>
<td>B.1 MINI-COBOL Verb Formats</td>
<td>68</td>
</tr>
<tr>
<td>B.2 Coding Sheet</td>
<td>71</td>
</tr>
<tr>
<td>D.1 Operand Abbreviations</td>
<td>262</td>
</tr>
<tr>
<td>D.2 Abbreviated Instruction Word Fields</td>
<td>264</td>
</tr>
<tr>
<td>D.3 Memory Control Abbreviations</td>
<td>265</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>9</td>
</tr>
<tr>
<td>3.2</td>
<td>11</td>
</tr>
<tr>
<td>3.3</td>
<td>14</td>
</tr>
<tr>
<td>3.4</td>
<td>15</td>
</tr>
<tr>
<td>3.5</td>
<td>15</td>
</tr>
<tr>
<td>3.6</td>
<td>16</td>
</tr>
<tr>
<td>3.7</td>
<td>17</td>
</tr>
<tr>
<td>3.8</td>
<td>17</td>
</tr>
<tr>
<td>3.9</td>
<td>18</td>
</tr>
<tr>
<td>3.10</td>
<td>19</td>
</tr>
<tr>
<td>3.11</td>
<td>20</td>
</tr>
<tr>
<td>3.12</td>
<td>20</td>
</tr>
<tr>
<td>3.13</td>
<td>21</td>
</tr>
<tr>
<td>3.14</td>
<td>22</td>
</tr>
<tr>
<td>3.15</td>
<td>23</td>
</tr>
<tr>
<td>3.16</td>
<td>24</td>
</tr>
<tr>
<td>3.17</td>
<td>24</td>
</tr>
<tr>
<td>3.18</td>
<td>25</td>
</tr>
<tr>
<td>3.19</td>
<td>37</td>
</tr>
<tr>
<td>3.20</td>
<td>29</td>
</tr>
<tr>
<td>4.1</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>44</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

The use of mini-computers is constantly expanding in the business world. One of the major difficulties encountered when a mini-computer is chosen for an application is the language it uses. Should a company or institution decide that one manufacturer produces a machine suitable for one application and another manufacturer produces a machine suitable for another application, the company must be aware that more than likely the two machines will use different assemblers and will necessitate the training of the company programming staff in two languages. Writing in FORTRAN is a step in the direction of reducing training expense; however, the various manufacturers have implemented specialized "subsets" of FORTRAN which tend to make portability difficult. Additionally, the present trend is to use COBOL for business applications in preference to FORTRAN.

COBOL, however, suffers basically from the same "illness" which affects FORTRAN despite the best efforts of the American National Standards Institute. Each manufacturer has seen fit to add features to the basic subset as approved by the CODASYL committee generating in turn a myriad of hybrids which cause undesirable delays in the conversion effort when programs are transferred from one machine to another.

The U. S. Navy, being a large user of computers of various manufacture, long suffered from the ailments as described above until it decided to restrict the language options to only those officially sanctioned by ANSI. In their attempts to resolve the problem of portability,
the U. S. Navy conducted research and discovered that of the options offered as standard by ANSI, the majority of programmers really only utilized a relatively small portion. As a result of this study, the U. S. Navy constructed a COBOL compiler specifically designed for a mini-computer. While the special compiler is designed only for a specific mini-computer at this time, programs written in "MINI-COBOL" can be transferred with minor modifications to those statements considered machine dependent by ANSI to "MAXI" computers with a COBOL compiler of an equal or greater level. Appendix A contains a description of U. S. Navy MINI-COBOL.

While the U. S. Navy accomplished an admirable task, they did not make any attempt to design the MINI-COBOL compiler as a portable software item, i. e., the compiler itself can run on the specific machine of one manufacturer. The desire to use COBOL on other mini-computers gave birth to this project which will deal with the transfer of the U. S. Navy MINI-COBOL compiler designed for the NOVA mini-computer to the INTERDATA mini-computer. While the project may appear to be rather specific in nature, the same principles may be used as they are presented in this paper to transfer the MINI-COBOL compiler to any mini-computer which uses a micro-coded instruction set, and this feat can be effected with minor modification to the FORTRAN interpreter as designed by the Navy. Furthermore, the design presented allows for the use of a virtual system and for the use of other mini-computers as I/O device drivers. Also presented for discussion in this report is a description of the MODEL 80 INTERDATA micro-processor which when used in conjunction with the presentation of the micro-code instruction formats should give the
reader a basic framework of reference to understand the micro-code presented as emulated NOVA instructions. A description of each of the various mnemonics used in the micro-code is given to facilitate the viewing of the emulation code as presented. Formats of the NOVA instructions which will be emulated in this project are presented so that the viewer may inspect them and gain a more comprehensive view of the emulation technique. Specification of the Fetch and Decode modules found in the emulator will be presented along with the micro-code generated for each NOVA instruction is included in this report. A sample high-level algorithm along with an explanation of the abbreviations is given including a step-by-step trace table of two typical NOVA instructions with a condensed output listing in an INTERDATA-like assembly language and in the micro-code generated. A discussion on modification of the Navy supplied MINI-COBOL interpreter will be presented along with the specifications of the MINI-COBOL language, the MINI-COBOL compiler system, and the actual source code for all the modules used in the MINI-COBOL system.
Chapter 2

THE DESIGN OF AN INTERDATA IMPLEMENTATION
OF MINI-COBOL

Introduction

Two alternatives were seriously considered in designing an implementation of MINI-COBOL on the INTERDATA; the first alternative would have been to code the routines for the COBOL verbs in INTERDATA micro-instruction code; the other alternative was to write a micro-instruction emulation of the NOVA instructions. Should I have chosen the first alternative, every module in the MINI-COBOL compiler would have to be micro-coded. This approach would not utilize the work already accomplished by the U. S. Navy resulting in a genuine duplication of effort on my part. Furthermore, the net result of such an effort on my part would not have, in any way, facilitated the inter-connection and compatibility of the NOVA and INTERDATA mini-computers. The only advantage that presented itself with the first alternative was speed of execution due to the fact that the actual function of any given module would be taken into consideration when being coded. Micro-code of the function would be much faster than duplication of the NOVA architecture as would be the case with the second alternative.

The second alternative, emulation of NOVA instructions on the INTERDATA, would provide the following advantages:

1) MINI-COBOL could be utilized on the INTERDATA;
2) the NOVA instruction set could be utilized on the INTERDATA mini-computer;
3) because the second alternative was "general" in its approach and because micro-code was flexible, it would allow for future expansion into the virtual machine design that is currently being worked upon by others in our department.\textsuperscript{1,2}

There were some disadvantages to the second alternative, however, mainly that the INTERDATA, having a more powerful instruction set, could execute the same function in its own instruction set much faster than an emulated NOVA instruction set. However, I felt that the advantages of flexibility and possible growth far outweighed the disadvantage of speed in this case.

Implementation Overview - High Level

Trying to effect the second alternative, turned out to be an interesting task. Since the first two phases of the MINI-COBOL compiler are written in COBOL, it would be difficult to efficiently modify them. However, the interpreter phase seemed to lend itself naturally to modification so as to effect the second alternative. (see Appendix A - Interpreter). Basically, the scheme is as follows:

1) the user creates his COBOL source file;

2) he then invokes the COBOL compiler with the "START" instruction;

3) phase I and phase IA would create the tuples required for execution (see Appendix A - Phase I, Phase IA);

4) the user would then begin execution of his program by invoking the MINI-COBOL run time routines by using the "COBOL" "R" instructions;

5) the "COBOL" "R" instructions invoke the FORTRAN interpreter (see Appendix A - Interpreter);
6) after the interpreter allocates memory to the literals and values declared in the user program, it begins to read the execution tuples. It then examines the operation code and branches to a dispatch table (see Appendix A – figure A.4, step 5) which then gives the address of an entry point into a second dispatch table (see step 6) which contains the list of NOVA sub-routines in the order required to effect the COBOL verb. It is precisely at this point that the following modification would be inserted:

a) A "call" to the Fetch routine would be inserted, passing as parameters the list of NOVA sub-routines needed to effect the COBOL verb;

b) the Fetch routine would then read the instructions as written in NOVA assembler, one at a time, for each module listed, invoking for each instruction read, a Decoder which will translate the NOVA instruction into the Instruction Register format (see Appendix D – Instruction Register). The Decoder will then branch to the appropriate Micro-code routine for this instruction based upon the OP-CODE found in the Instruction Register.

c) the micro-coded routines will execute the specific function and return to the Decoder.

d) the Decoder will continue to construct the Instruction Register and branch to the appropriate micro sub-routine until it has completed all of the options requested by the original NOVA instruction. Upon completion, the Decoder will then return to the Fetch routine.

e) the Fetch routine will continue to read NOVA instructions until end-of-file condition is reached. It will then return to the interpreter.
f) the interpreter will then read the next execution tuple, thus closing the emulation loop.

To the user, the entire process is transparent. He is aware of only the fact that he has created a COBOL file and has invoked the execution of that file, thus anyone familiar with COBOL per se will be able to use the system with little or no training.

Summary

Thus far, I have presented the various conditions which influenced the emulator design and I have presented a high-level overview of the emulator design in light of those considerations. Succeeding chapters shall present in detail the specific design of the Fetch and Decoder modules.
Chapter 3

THE DETAILED DESIGN

Introduction

In order that the viewer gain a more complete understanding of the emulator, I will present a short section on the NOVA instruction formats. Throughout the rest of this report, the sixteen bits which contain any one of the various formats of the NOVA instruction shall be referred to as NI (0:15), wherein NI represents the abbreviation for NOVA Instruction and (0:15) represents the abbreviation for bits zero through fifteen inclusive. The Nova instructions should be considered as the input into the Fetch module which will be the second section in this chapter.

In the presentation of the Fetch module, I will use a high-level algorithmic language which will give the detailed function the module should accomplish upon invocation via the call statement from the FORTRAN interpreter (see Appendix A - Interpreter).

Since the next operation in the normal flow of the proposed emulation in the Decoder module, I will present the rules of precedence to be used in the module when constructing the Instruction Register (IR). I will also present the algorithm to be used to emulate the subset of NOVA instructions and functions I have chosen. The algorithms will be written in an INTERDATA-like assembler language for the viewer's benefit as micro-code is hardware directed and rather difficult to follow. (see Appendix E - Assembler Op-Codes, Micro-coded Emulation).

NOVA Instruction Formats

The NOVA instruction set is basically divided into three categories
with reference to format. I shall not present any discussion with regard to I/O instructions as they are treated as a special case as described later in this report. The formats of the classes of instructions which this emulator will receive as input are presented below.  

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP-CODE</td>
<td>A</td>
<td>I</td>
<td>X</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MOVE DATA INSTRUCTIONS**

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP-CODE</td>
<td>I</td>
<td>X</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MODIFY MEMORY/JUMP INSTRUCTIONS**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S</td>
<td>D</td>
<td>OP-CODE</td>
<td>SH</td>
<td>C</td>
<td>N</td>
<td>SK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ARITHMETIC AND LOGIC INSTRUCTIONS**

![Figure 3.1](image)

Selected NOVA Instruction Formats

An explanation of the abbreviations used in the above figure of selected NOVA formats is given below.

OP-CODE - this is a three bit field which specifies the function requested. An exception to this rule is in the Memory/Jump instruction which has a total of five bits. Bits three through four of the Memory/Jump instruction further delineate the specific function desired.

A - this represents one of four general purpose accumulators available for use by the application.

I - this field represents the address type, whether direct or indirect.
X - this field specifies an index register, if any.

D - these eight bits represent the displacement.

I - this is a constant indicating that the instruction is an
Arithmetic and Logic instruction.

S - this is an accumulator source address.

D - in the Arithmetic and Logic instruction, this represents the
accumulator destination address.

SH - this two bit field represents one of three possible shift
functions.

C - this two bit field represents one of three possible operations
that may be performed with respect to the Carry bit.

N - this is the No-Load function bit. If it is equal to '1'B, then
the loading of the Carry bit and accumulator is inhibited.

SK - this field represents one of eight possible functions with respect
to skipping the next sequential instruction.

Because of the rather limited mnemonics of the NOVA instruction set,
it is quite easy to jump to the conclusion that the assembler is under-
powered or limited. However, further study of the capabilities of the
instruction set will reveal the fact that up to six separate operations
may occur in any given Arithmetic and Logic instruction making the emu-
lation of this phenomenon, at best, a complicated process.

The Fetch Module

With the input to the Fetch module described above, the function of
the Fetch module can be presented. As was previously mentioned in
preceding chapters, the MINI-COBOL compiler utilizes a FORTRAN inter-
preter to read the execution tuples. The interpreter reads a tuple,
and then branches to an address which contains a list of addresses of NOVA assembler sub-routines in the order required to effect the operation requested by the COBOL verb. (see Appendix A for a complete discussion of this process).

At the point in the normal flow of operation described above wherein the FORTRAN interpreter branches to an address containing a list of addresses of NOVA sub-routines, I propose the following changes:

1) the branch to a list of address be replaced with a "call" statement to the Fetch module;

2) the list of addresses be passed as an input parameter to the Fetch module.

The effect of these proposed changes is that the Fetch will be allowed to read the first binary NOVA instruction of the routine as referenced by the parameter and it may then pass control to the Decoder, such that:

(PARM_1, PARM_2...PARM_n)---(ADR_1, ADR_2...ADR_n)

CALL "FETCH" USING PARM_1, PARM_2...PARM_n

RETURN TO READ-NEXT-TUPLE

.
.
.

FETCH: EQUATE (INPUT, PARM_1, PARM_2...PARM_n)

READ (INPUT) AT END RETURN

(NI_BUFFER)---NI (0:15)

CALL "DECODE" USING NI_BUFFER

BRANCH TO FETCH

Figure 3.2

Fetch Algorithm
The same techniques as presented above can be applied to facilitate
the virtual machine, in that, prior to using an instruction which uti-
lizes an address in main memory, a "call" can be made to the paging
routines which can resolve the address passed as a parameter and return
a real address as an argument so the instruction can be processed.²

I/O instructions can be expedited in a manner similar to the afore-
mentioned, i.e., upon recognition of an I/O instruction, the Decoder
(in this case) will construct an SVC passing as an argument the buffers
involved. The computer (of the SVC) can pass the contents of these
buffers as a message to the computer acting as the I/O device handler.²

The Decoder Module

As can be inferred from the proceeding section, the Decoder which
is the very heart of the proposed emulator, is the next topic of dis-
cussion in this report. Since the Decoder is the largest and most com-
plex module of the emulator, I will present it as a series of discussions
based upon the rules of precedence it must use. All algorithms presented
in this section will be written in an INTERDATA-like assembler language
for purposes of clarity. In the actual emulator, the algorithm will be
micro-coded instructions.

Rules of precedence.

When the Decoder is invoked by the Fetch module, it will receive an
input parameter, an entire NOVA instruction in binary format, to be
decoded. The Decoder, in turn, must examine the given instruction
according to the following rules of precedence:

1) Scan the instruction for the presence of indirection. If
indirection is present, then call the paging routine which will resolve
the address involved into an effective address. Since one or more levels of indirection may be involved, this particular sub-function may be invoked several times until the indirect bit (NI (5)) is found to be 'O'3.

2) Scan for the desired mnemonic and branch to the routine based on the specific mnemonic requested. All routines use temporary work areas identified in the corresponding code and R12 (the shifter).

3) Scan for the Carry function and based upon the particulars of the mnemonic involved from step two above, load the Carry bit into a work area identified as R15, as requested by the Carry function which is specified in R10 which represents NI (10:11).

4) Scan for the Shift function. If it is present, the Decoder will branch to the proper routine based on the specific function requested.

5) Scan for the Load function (NI 5)). If it is equal to 'O'3, then branch to the routine which will load R12 to the Accumulator as specified by Ni (3:4). Should the No-Load function be requested, the Accumulators as specified by NI (1:2) and NI (3:4) will be unaffected and will retain their original contents.

6) Scan for the SKIP function and branch to the routine which shall effect the function requested by NI (13:15).

Emulation routines.

Using the rules of precedence as I have just presented and the register assignments presented below, the following section will present an INTERDATA-like assembler language algorithm for each NOVA instruction function that will be used in this emulator (see Appendix E - Assembler Op Codes and Micro coded emulation). As I have previously mentioned, I have chosen assembler for purposes of clarity but, in the actual emulator, micro-coded instructions will be generated.
RO - R3 - used as Accumulator Source (NI (1:2)) and Accumulator Destination (NI (3:4)).

R4 - R5 - not used.
R6 - used as "Branch to" address.
R7 - used as "return" address.
R8 - used to hold number of shift positions and a general work register.
R9 - used as a general work register.
R10 - contains the Carry function (NI (10:11)).
R11 - used as a general work register.
R12 - used as the NOVA Shifter.
R13 - used as a general work register.
R14 - used as a general work register.
R15 - used as the Carry Bit of the Shifter.

Table 3.1
Emulation Register Assignments

1. Emulation of the JSR

Upon encounter of the JSR operation code (NI (0:4)), the NOVA will load the address of the next sequential instruction following the JSR instruction into Accumulator three. It will then branch to the effective address of the sub-routine as indicated by bits NI (6:15).

1  LH  R6, (NI 6:15)  LOAD "BRANCH TO" ADDR.
2  BALR R3, R6  BRANCH AND LINK REG.

Figure 3.3
Emulated JSR
NI (6:15) is the effective address of the routine to which the branch will occur. The Decoder must resolve the address as specified by bits 6:15 prior to loading Register 6. Register 3, emulating NOVA Accumulator 3, will have its previous contents destroyed by this instruction as it will contain the "return" address.

2. Emulation of the JMP

The NOVA will load the next sequential instructions' address into the Program Counter and branch unconditionally to the address specified by NI (6:15), when the JMP operation code is encountered in NI (0:4).

1   BAL R7,(NI 6:15)    BRANCH TO EFFECTIVE ADR.

Figure 3.4
Emulated JMP

The address of the next sequential instruction is loaded into Register 7. The Decoder must resolve the effective address as specified by NI (6:15).

3. Emulation of the DSZ

The NOVA will subtract 1 from the contents of the location as specified by NI (6:15) and will place the result back into the location as specified by NI (6:15). If the result is equal to zero, then the next instruction in sequence will be skipped.

1   LH R12,(NI 6:15)    GET AREA REFERENCED
2   SIS R12,X'01'      SUBTRACT ONE FROM AREA
3   BZ *+3            IF ZERO THEN SKIP NSI
4. Emulation of the ISZ

The NOVA will add 1 to the contents of the location as specified by NI (6:15) and will place the results back into the location as specified by NI (6:15). If the results of the add is equal to zero, then the next instruction in sequence will be skipped.

Figure 3.6
Emulated ISZ

The Decoder must resolve NI (6:15) into an effective address. Once this has been accomplished, 1 will be added to the area referenced and the results compared to zero. If the result is zero, the contents are
restored to the area referenced and the next sequential instruction is fetched. Otherwise, the results are restored and the flow continues.

5. Emulation of the LDA

The NOVA will load the area of memory as referenced by NI (6:15) into the Accumulator specified by NI (3:4).

1   LH   (NI 3:4), (NI 6:15)   LOAD REG. FROM MEM.

Figure 3.7
Emulated LDA

The Decoder must translate NI (3:4) into a hexadecimal digit 0 - 3, then it must resolve the address of the area of main memory as represented by NI (6:15). The contents of the area referenced will then be loaded into the Register specified by NI (3:4).

6. Emulation of the STA

The NOVA will load the contents of the Accumulator as referenced by NI (3:4) into the area of main memory as specified by NI (6:15).

1   STH  (NI 3:4), (NI 6:15)   STORE ACUM.

Figure 3.8
Emulated STA

The Decoder must translate NI (3:4) into a hexadecimal digit 0 - 3. Then the Decoder must resolve the address as specified by NI (6:15). The contents of the Register specified will be loaded into the area referenced.
7. Emulation of the ADD

The NOVA will add the contents of the Accumulator as specified by NI (1:2) to the contents of the Accumulator as specified by NI (3:4) and will place the result in the Shifter. If the sum is equal to or greater than 2E16, then the value specified by NI (10:11) will be complemented as the Carry bit; otherwise, the value as specified by NI (10:11) will be used as the Carry bit.

1 LHR R12,(NI 1:2) GET ACCUM. SOURCE
2 LHR R13,(NI 3:4) GET ACCUM. DESTINATION
3 AHR R12,R13 ADD THE ACCUMULATORS
4 CLHI R12,X'FF' GREATER OR EQUAL 2E16?
5 BNC *+3 YES
6 LH R10,(NI 10:11) NO, GET CARRY FUNCTION
7 B *+3 BRANCH OUT OF ROUTINE
8 LH R10,(NI 10:11) GET CARRY FUNCTION
9 XHR R10,X'03' COMPLEMENT CARRY FUNCTION

Figure 3.9
Emulated ADD

The contents of NI (1:2) and NI (3:4) must be translated into hexadecimal digits 0 – 3. The Decoder will then load the contents of these registers into R12 and R13, respectively. The registers are then added into R12 and the result checked to determine if it is equal or greater than 2E16. If it is, then the Carry function (NI (10:11)) is complemented into R10; otherwise, the Carry function is loaded into R10 unaltered.
8. Emulation of the SUB

The NOVA subtracts by adding the twos complement of the number from NI (1:2) to the number from NI (3:4) and will place the result in the Shifter. If the number in the Accumulator specified by NI (3:4) is greater or equal to the number in the Accumulator specified NI (1:2), then the complement of the value specified by Carry function (NI (10:11)) will be supplied; otherwise, the value of the Carry function as given will be supplied to the Carry bit.

1 LHR R12,(NI 1:2) GET ACS INTO R12
2 XHI R12,X'FF' COMPLEMENT IT
3 AHI R12,X'01' ADD ONE TO MAKE 2'S COMP.
4 LHR R14,(NI 3:4) GET ACD INTO R14
5 CHR R14,R12 ACD G.E. ACS?
6 BC *+4 NO, GO LOAD CARRY
7 LH R10,(NI 10:11) YES, GET CARRY FUNCTION
8 XHI R10,X'03' COMPLEMENT IT
9 $ *+2 GO ADD
10 LH R10,(NI 10:11) GET CARRY FUNCTION
11 AHR R12,R14 ADD ACS TO ACD

Figure 3.10

Emulated SUB

NI (1:2) will be translated by the Decoder into hexadecimal digit 0 - 3. The contents of the register specified will be loaded into Register 12 where it will be formed into the twos complement. Register 12 is then added to the contents of R14 which holds the number specified
by NI (3:4). If R14 is greater or equal to R12, the Carry function is complemented; otherwise it is loaded as specified by NI (10:11) into R10.

9. Emulation of the AND

The NOVA will logically "and" the contents of the Accumulator specified by NI (1:2) with the contents of the Accumulator specified by NI (3:4).

1. LHR R12,(NI 3:4)  GET ACD INTO R12
2. LHR R13,(NI 1:2)  GET ACS INTO R13
3. NHR R12,R13  AND INTO R12
4. LHR R10,(NI 10:11)  PUT CARRY FUNCTION IN R10

Figure 3.11

Emulated AND

The Decoder must translate NI (1:2) and NI (3:4) into hexadecimal digits 0 - 3. The contents of the Accumulators as specified by NI (1:2) and NI (3:4) are loaded into R13 and R12 respectively. The contents are "ANDed" into R12 and the Carry function is loaded into R10.

10. Emulation of the COM

The NOVA will logically complement the contents of the Accumulator as specified by NI (1:2) into the Shifter and place the value as specified by the Carry function into the Carry bit.

1. LHR R12,(NI 1:2)  LOAD ACS INTO R12
2. XHI R12,X'FF'  LOGICALLY COMPLEMENT R12
3. LH R10,(NI 10:11)  LOAD CARRY TO R10

Figure 3.12

Emulated COM
ACS as specified by NI (1:2) and translated into hexadecimal by the
Decoder is loaded and complemented into R12. The Carry function is loaded
into R10.

11. Emulation of the NEG

The NOVA will place the two's complement of the number contained in
the Accumulator specified by NI (1:2). If the number is zero, the comple-
ment of the value specified by the Carry (NI 10:11) will be placed in the
Carry bit; otherwise, the value as originally specified will be placed
in the Carry bit.

<table>
<thead>
<tr>
<th>Line</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LHR R12,(NI 1:2)</td>
<td>LOAD ACS INTO R12</td>
</tr>
<tr>
<td>2</td>
<td>LHR R11,R12</td>
<td>LOAD ACS INTO R11</td>
</tr>
<tr>
<td>3</td>
<td>XHI R12,X'FF'</td>
<td>COMPLEMENT R12</td>
</tr>
<tr>
<td>4</td>
<td>AH1 R12,X'01'</td>
<td>ADD ONE</td>
</tr>
<tr>
<td>5</td>
<td>LHI R13,X'00'</td>
<td>PUT ZERO IN R13</td>
</tr>
<tr>
<td>6</td>
<td>CHR R11,R13</td>
<td>IS ACS EQUAL TO ZERO?</td>
</tr>
<tr>
<td>7</td>
<td>BNE *+5</td>
<td>NO</td>
</tr>
<tr>
<td>8</td>
<td>LH R10,(NI 10:11)</td>
<td>PUT CARRY INTO R10</td>
</tr>
<tr>
<td>9</td>
<td>XHI R10,X'03'</td>
<td>COMPLEMENT CARRY</td>
</tr>
<tr>
<td>10</td>
<td>B *+2</td>
<td>BRANCH OUT</td>
</tr>
<tr>
<td>11</td>
<td>LH R10,(NI 10:11)</td>
<td>NOT ZERO, LOAD CARRY TO R10</td>
</tr>
</tbody>
</table>

Figure 3.13

Emulated NEG

ACS as specified by NI (1:2) is loaded into Register 12. Register
12 is complemented into the two's complement form. If ACS was zero, then
the Carry value is complemented; otherwise, the Carry function is loaded as it was originally into Register 10.

12. Emulation of the ADC.

The NOVA will add the logical complement of the number contained in the Accumulator as specified by NI (1:2) to the number contained in the Accumulator as specified by NI (3:4) and place the results in the Shifter. If the number contained in the Accumulator specified by NI (1:2) is less than the number contained in the Accumulator specified by NI (3:4), then the complement of the value as indicated by NI (10:11) will be loaded into the Carry bit; otherwise the value indicated will be loaded into the Carry bit.

```
1  LHR  R12,(NI 1:2)  LOAD ACS INTO R12
2  XHI  R12,X'FF'  COMPLEMENT IT
3  LHR  R14,(NI 3:4)  LOAD ACD INTO R14
4  AHR  R12,R14  ADD ACS TO ACD
5  LHR  R13,(NI 1:2)  GET ACS INTO R13
6  CLHR  R13,R14  IS ACD GREATER THAN ACS?
7  BC  *+3  YES, COMPLEMENT CARRY
8  LH  R10,(NI 10:11)  NO, LOAD CARRY
9  B  *+3  BRANCH OUT
10  LH  R10,(NI 10:11)  LOAD CARRY
11  XHI  R10,X'03'  COMPLEMENT IT
```

Figure 3.14

Emulated ADC
ACS as specified by NI (1:2) is complemented and added to ACD as specified by NI (3:4) and the results placed in the Shifter. The two Accumulators are compared and if ACD is greater than ACS the value specified by NI (10:11) is complemented into R10; otherwise, the value specified is placed into R10.

13. Emulation of the INC.

The NOVA will add one to the number specified in the Accumulator represented by NI (1:2) and will place the result in the Shifter. If ACS contained \((2E16) -1\), the Carry value is complemented; otherwise, the value supplied is loaded into the Carry bit.

```
1 LHR  R12,(NI 1:2)       GET ACS INTO R12
2 AHI  R12,X'01'          ADD ONE TO R12
3 LHR  R13,(NI 1:2)       GET ACS INTO R13
4 CLHI R13,X'FE'          ACS EQUAL \((2E16) -1\)?
5 BNE  **+4               NO, GO LOAD CARRY
6 LH   R10,(NI 10:11)     YES, GET CARRY
7 XHI  R10,X'03'          COMPLEMENT IT
8 B    **+2              BRANCH OUT
9 LH   R10,(NI 10:11)     LOAD CARRY
```

Figure 3.15

Emulated INC

One is added to the number contained in the Accumulator as represented by NI (1:2). If the number was equal to \((2E16) -1\) originally,
the Carry value is complemented; otherwise, the value specified by NI (10:11) is loaded into R10.

14. Emulation of the MOV

The NOVA will place the contents of ACS (NI (1:2)) in the shifter and place the Carry bit as specified by (NI (10:11)) in the Carry.

1  LHR R12,(NI 1:2)  GET ACS INTO R12
2  LH R10,(NI 10:11)  GET CARRY INTO R10

Figure 3.16

Emulated MOV

The contents of the register as specified by NI (1:2) are loaded into the Shifter (R12). The value of the Carry function as specified by NI (10:11) is loaded into R10.


The NOVA will examine NI (10:11) using the following rules:

1) if NI (10:11) equals '00'B, then the current state of the Carry bit will be used;

2) if NI (10:11) equals '01'B (Z), then a zero will be supplied to the Carry bit;

3) if NI (10:11) equals '10'B (O), then one will be supplied to the Carry bit;

4) if NI (10:11) equals '11'B (C), then the current state of the Carry bit will be complemented.

1  CLHI R10,X'01'  EQUAL ZERO?
2  BE *+6  YES
3  CLHI R10,X'10'  EQUAL ONE?
4  BE *+6  YES

Figure 3.17

Emulated Z, O, C
5  CLHI  R10,X'11'  EQUAL COMPLEMENT?
6  BE    *+6    YES
7  B     *+6    DO NOT ALTER CARRY
8  LHI   R15,X'00'  LOAD ZERO IN R15
9  B     *+4    BRANCH OUT
10 LHI   R15,X'01'  LOAD ONE IN R15
11 B     *+2    BRANCH OUT
12 XHI   R15,X'FF'  COMPLEMENT CARRY

Figure 3.17 (Cont.)
Emulated Z, O, C


The NOVA will examine NI (8:9) and will respond according to the following rules:
1) if NI (8:9) equals '00'B, then no shift will take place;
2) if NI (8:9) equals '01'B, then rotate left one place. Bit 0 is rotated into the Carry bit and the Carry bit into bit 15;
3) if NI (8:9) equals '10'B, then rotate right one place. Bit 15 is rotated into the Carry position and the Carry bit into bit 0.
4) if NI (8:9) equals '11'B, then swap half of the 16 bit result.
The Carry bit is not affected.

1  LHR   R13,(NI 8:9)   GET SHIFT FUNCTION
2  CLHI  R13,X'01'     IS IT "L"?
3  BE    *+6    YES
4  CLHI  R13,X'02'     IS IT "R"?

Figure 3.18
Emulated L, R, S
5 BE *+12
6 CLHI R13,X'03'
7 BE *+24
8 B *+40
9 SLLS R12,X'01'
10 BC *+4
11 AHR R12,R15
12 LHI R15,X'00'
13 B *+27
14 AHR R12,R15
15 LHI R15,X'01'
16 B *+24
17 SRLS R12,X'01'
18 BC *+7
19 CLHI R15,X'01'
20 BE *+2
21 B *+19
22 AHI R12,X'80'
23 LHI R15,X'00'
24 B *+16
25 CLHI R15,X'01'
26 BE *+3
27 LHI R15,X'01'
28 B *+12
29 AHI R12,X'80'
30 B *+10

YES
IS IT "S"?
YES
NO SHIFT
SHIFT LEFT 1
WAS THERE A CARRY?
NO, ADD CARRY BIT
PUT ZERO IN CARRY BIT
BRANCH OUT
YES, ADD CARRY BIT
PUT ONE IN CARRY BIT
BRANCH OUT
SHIFT RIGHT 1
WAS A ONE SHIFTED?
NO, IS THERE A 1 CARRY?
YES
NO, BRANCH OUT
ADD THE CARRY 1
ZERO THE CARRY
BRANCH OUT
YES, DOES CARRY EQUAL 1?
YES
NO, PUT 1 IN CARRY
BRANCH OUT
ADD CARRY
BRANCH OUT

Figure 3.18 (Cont.)
Emulated L, R, S
31 LHI R8,X'08'
32 SLLS R12,X'01'
33 BC *+3
34 LHI R9,X'00'
35 B *+3
36 LHI R9,X'01'
37 AHR R12,R9
38 SHI R8,X'01'
39 BNZ *-7

LOAD # OF SHIFTS
SHIFT LEFT 1
WAS A 1 SHIFTED
NO, PUT 0 IN R9
GO DECREMENT SHIFTS
YES, PUT 1 IN R9
ADD SHIFT BITS
DECREMENT SHIFT
GO SHIFT AGAIN

Figure 3.18 (Cont.)

Emulated L, R, S

Based on NI (8:9), a branch is made to the proper routine to expedite the requested function of Shift L, Shift R, or Swap halves.

17. Emulation of the Load/No Load.

The NOVA will examine NI (12) for a '1'B. Should '1'B be encountered, ACD (NI (3:4)) will not be loaded. If NI (12) contain '0'B, then ACD (NI (3:4)) will be loaded.

1 LH R9,(NI 12) GET LOAD/NO LOAD FUNCTION
2 LH R11,X'00' PUT ZERO IN R11
3 CLHR R9,R11 EQUAL ZERO?
4 BNE *+2 YES, BRANCH OUT (NO LOAD)
5 LHR R12,(NI 3:4) NO, LOAD R12 INTO ACD

Figure 3.19

Emulated Load/No Load
If NI (12) is equal to one then no loading of ACD as specified by NI (3:4) will occur; otherwise, ACD will be loaded with the contents of Register 12.

18. Emulation of the SKIP function.

The NOVA will examine NI (13:15) and will respond according to the following rules;

1) if NI (13:15) equal zero, then it will not Skip;

2) if NI (13:15) equal one, then it will always skip the next sequential instruction;

3) if NI (13:15) equal two, then it will skip on the Carry bit being equal to zero;

4) if NI (13:15) equal three, then it will skip the next sequential instruction if the Carry bit is nonzero;

5) if NI (13:15) equal four, then it will examine the Shifter and skip the next sequential instruction if the result is zero;

6) if NI (13:15) equal five, then it will examine the Shifter and skip the next sequential instruction if the result is nonzero;

7) if the NI (13:15) equal six, then it will examine the Shifter and the Carry bit and skip the next sequential instruction if either is equal to zero;

8) if NI (13:15) equal seven, then it will examine the Shifter and the Carry bit and skip the next sequential instruction if both are equal to nonzero.
LH R9, (NI 13:15)          GET SKIP FUNCTION
CLHI R9, X'00'              EQUAL NO SKIP?
BE *+15                      YES
CLHI R9, X'01'              EQUAL SKIP?
BE *+14                      YES
CLHI R9, X'02'              EQUAL SKIP ZERO CARRY?
BE *+14                      YES
CLHI R9, X'03'              EQUAL SKIP NONZERO CARRY?
BE *+15                      YES
CLHI R9, X'04'              EQUAL SKIP ZERO RESULTS?
BE *+16                      YES
CLHI R9, X'05'              EQUAL SKIP NONZERO RES.?
BE *+17                      YES
CLHI R9, X'06'              EQUAL SKIP EITHER ZERO?
BE *+18                      YES
CLHI R9, X'07'              EQUAL SKIP BOTH NONZERO?
BE *+21                      YES
B FETCH                     GO FETCH NXT. SEQ. INST.
READ (INPUT)                GET NXT. SEQ. INST.
B FETCH                     GO FETCH NXT. SEQ. INST.
CLHI R15, X'00'             CARRY EQUAL ZERO?
BE *-3                      YES, GO READ, FETCH
B FETCH                     NO, GO FETCH NXT. SEQ. INST.
CLHI R15, X'01'             CARRY EQUAL 1?

Figure 3.20

Emulated SKIP
Based on NI (13:15), the Decoder will branch to the appropriate routine and test the results in the Shifter and Carry bit for zero/nonzero conditions and will skip or not skip the next sequential instruction based upon the conditions it was checking.
Summary

In the preceding pages, I have presented a detailed design of the emulator I propose to effect the transfer of the U.S. Navy MINI-COBOL compiler system to the INTERDATA 80 mini-computer. The input to the emulator, the NOVA instruction, was thoroughly discussed with the intent to help the reader understand the scope and overall function of the emulator. Specifications of the Fetch module were given and an algorithm for the implementation were given. It was also noted that the techniques for implementation of the Fetch module could be easily transferred to effect a virtual machine and the use of a mini-computer as an I/O device handler.

I presented the detail design of the Decoder module, explaining the rules of precedence to be used in the implementation and the functional description of each NOVA operand, including an INTERDATA-like assembler language algorithm and explanation of the algorithm.
Chapter 4

VERIFICATION

Introduction

The most difficult task in any design project is pre-implementation verification of the design itself. The existing tools to accommodate verification are few at most and awkward at best. However, one of the more trustworthy methods is the trace table and that is the method I chose to test my detailed design as presented in the preceding pages of this report.

Abbreviations In Sample Algorithm

The algorithm used to emulate is rather long and complex in nature; therefore, I decided to use a high level language to describe the various operations that were to be performed based upon the rules of precedence mentioned previously. I do use some rather standard assembler abbreviations to express myself in this algorithm which are as follows:

R1 - R15 - Represents General Registers 1 through 15 inclusive in the INTERDATA.

R0 - R3 - Represents ACO - AC3 of the NOVA as emulated in the INTERDATA.

R12 - Represents the Shifter of the NOVA as emulated in the INTERDATA.

Figure 4.1

Abbreviations
R10  - Represents the Carry function (NI 10:11).
R15  - Represents the Carry bit of the NOVA.
IR   - Represents the Instruction Register, i.e., the output buffer area.
PSW  - Represents the Program Status Word of the INTERDATA.
NI BUFFER - NOVA instruction read-in buffer consisting of bits (0:15) of the NOVA Instruction.
NI BUFFER BEGINNING - Represents the beginning address of the read-in buffer for the NOVA instruction.
NI BUFFER ENDING - Represents the ending address of the read-in buffer for the NOVA instruction.
C, X, B-Represents "constant" which can be either alphanumeric (C 'LHI'), Hexadecimal (X'01'), or binary ('01'B).
NSI ADDRESS - Represents the Next Sequential Instruction Address.

Figure 4.1 (Cont.)

Abbreviations

Sample Algorithm

The following algorithm was designed to test two examples of NOVA instructions; the first instruction is MOVOL 1,1 which will demonstrate a complex NOVA instruction; the second instruction is ADD 1,2 which will demonstrate a simple NOVA instruction.

1. FETCH: (R4)←-(NI BUFFER BEGINNING)
2. (R5)←-(NI BUFFER ENDING)
3. (NI BUFFER 0:15)←-(READ→(R4,R5))

Figure 4.2
Sample Algorithm
END BRANCH OUT

(R4) ←←(NI 5:7)

* EMULATED SCAN FOR MNEMONIC BRANCH TABLE

(R4) = ('010'B)

True; BRANCH MOV-RT

(R4) = ('110'B)

True; BRANCH ADD-RT

* EMULATED SCAN FOR CARRY BRANCH TABLE

CARRY:: (R10) = ('10'B)

True; BRANCH CARRY-RT

* EMULATED SCAN FOR SHIFT BRANCH TABLE

SHIFT:: (R13) ←←(NI 8:9)

(R13) = ('01'B)

True; BRANCH SHIFT-RT

LOAD:: (R9) ←←(NI 12)

(R9) = ('1'B)

True; BRANCH FETCH

False; BRANCH LOAD-RT

OUT:: END

* EMULATION OF MOV

MOV-RT:: (IR 0:7) ←←(C'LHR')

(IR 8:11) ←←(C'R12')

Figure 4.2 (Cont.)

Sample Algorithm
(IR 12:15)←-(NI 1:2)

WRITE IR

(IR 0:7)←-(C'LH')

(IR 8:11)←-(C'R10')

(IR 12:15)←-(NI 10:11)

WRITE IR

(R10)←-(NI 10:11)

BRANCH CARRY

* EMULATION OF CARRY (O FUNCTION)

CARRY-RT:: (IR 0:7)←-(C'LHI')

(IR 8:11)←-(C'R15')

(IR 12:15)←-(X'01')

WRITE IR

BRANCH SHIFT

* EMULATION OF SHIFT (L FUNCTION)

SHIFT-RT:: (IR 0:7)←-(C'SLLS')

(IR 8:11)←-(C'R12')

(IR 12:15)←-(X'01')

WRITE IR

(IR 0:7)←-(C'BC')

(IR 12:15)←-(C'*+4')

WRITE IR

(IR 0:7)←-(C'AHR')

(IR 8:11)←-(C'R12')

(IR 12:15)←-(C'R15')

Figure 4.2 (Cont.)

Sample Algorithm
WRITE IR

(IR 0:7)←-(C'LHI')

(IR 8:11)←-(C'R15')

(IR 12:15)←-(X'00')

WRITE IR

(IR 0:7)←-(C'B')

(IR 12:15)←-(C'*+3')

WRITE IR

(IR 0:7)←-(C'AHR')

(IR 8:11)←-(C'R12')

(IR 12:15)←-(C'R15')

WRITE IR

(IR 0:7)←-(C'LHI')

(IR 8:11)←-(C'R15')

(IR 12:15)←-(X'01')

WRITE IR

BRANCH LOAD

* EMULATION OF LOAD (NI 12 EQUALS '1'B)

LOAD-RT::

(IR 0:7)←-(C'LHR')

(IR 8:11)←-(NI 3:4)

(IR 12:15)←-(C'R12')

WRITE IR

BRANCH FETCH

* EMULATION OF ADD

ADD-RT::

(IR 0:7)←-(C'LHR')

Figure 4.2 (Cont.)

Sample Algorithm
68. (IR 8:11)←-(C'R12')

69. (IR 12:15)←-(NI 1:2)

70. WRITE IR

71. (IR 0:7)←-(C'LHR')

72. (IR 8:11)←-(C'R13')

73. (IR 12:15)←-(NI 3:4)

74. WRITE IR

75. (IR 0:7)←-(C'AHR')

76. (IR 8:11)←-(C'R12')

77. (IR 12:15)←-(C'R13')

78. WRITE IR

79. (IR 0:7)←-(C'CLHI')

80. (IR 8:11)←-(C'R12')

81. (IR 12:15)←-(X'FF')

82. WRITE IR

83. (IR 0:7)←-(C'BNC')

84. (IR 12:15)←-(C'*+3')

85. WRITE IR

86. (IR 0:7)←-(C'LH')

87. (IR 8:11)←-(C'R10')

88. (IR 12:15)←-(NI 10:11)

89. WRITE IR

90. (IR 0:7)←-(C'XHR')

91. (IR 8:11)←-(C'R10')

92. (IR 12:15)←-(X'03')

Figure 4.2 (Cont.)

Sample Algorithm
93. WRITE IR
94. (R10) ←→ (NI 10:11)
95. BRANCH CARRY

Figure 4.2 (Cont.)

Sample Algorithm
<table>
<thead>
<tr>
<th>ACTION/CONDITION</th>
<th>R4</th>
<th>R5</th>
<th>R9</th>
<th>R10</th>
<th>R13</th>
<th>STEP</th>
<th>IR OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>10101010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>01100000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>NOT END</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>010</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>IR 0:7-'LHR'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>21</td>
<td>IR 8:11-'R12'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>22</td>
<td>IR 12:15-'R1'</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>23</td>
<td>LHR R12,R1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24</td>
<td>IR 0:7-'LH'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25</td>
<td>IR 8:11-'R10'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>26</td>
<td>IR 12:15-TWO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>27</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LHR R10,TWO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>28</td>
</tr>
<tr>
<td>CARRY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
</tr>
</tbody>
</table>

Table 4.1
<table>
<thead>
<tr>
<th>ACTION/</th>
<th>BUFFER</th>
<th>CONDITION</th>
<th>R4</th>
<th>R5</th>
<th>R9</th>
<th>R10</th>
<th>R13</th>
<th>STEP #</th>
<th>IR OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFT-RT</td>
<td></td>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>35</td>
<td>IR 0:7-'SLLS'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>36</td>
<td>IR 8:11-'R12'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>37</td>
<td>IR 12:15-X'01'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td>38</td>
<td>SLLS R12,X'01'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>39</td>
<td>IR 0:71-'BC'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>40</td>
<td>IR 12:15-''+4''</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td>41</td>
<td>BC ++4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>42</td>
<td>IR 0:7-'AHR'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>43</td>
<td>IR 8:11-'R12'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>44</td>
<td>IR 12:15-'R15'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td>45</td>
<td>AHR R12,R15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>46</td>
<td>IR 0:7-'LHI'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>47</td>
<td>IR 8:11-'R15'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>48</td>
<td>IR 12:15-X'00'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td>49</td>
<td>LHI R15,X'00'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>50</td>
<td>IR 0:7-'B'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>51</td>
<td>IR 12:15-''+3''</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td>52</td>
<td>B ++3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>53</td>
<td>IR 0:7-'AHR'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>54</td>
<td>IR 8:11-'R12'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>55</td>
<td>IR 12:15-'R15'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td>56</td>
<td>AHR R12,R15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>57</td>
<td>IR 0:7-'LHI'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>58</td>
<td>IR 8:11-'R15'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>59</td>
<td>IR 12:15-X'01'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1 (cont.)

Sample Algorithm Trace
<table>
<thead>
<tr>
<th>ACTION/CONDITION</th>
<th>R4</th>
<th>R5</th>
<th>R9</th>
<th>R10</th>
<th>R13</th>
<th>STEP #</th>
<th>IR OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>60</td>
<td>LHI R15, X'01'</td>
</tr>
<tr>
<td>LOAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>NOT EQUAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>LOAD-RT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>FETCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>62</td>
<td>IR 0:7-'LHR'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>63</td>
<td>IR 8:11-'R1'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>64</td>
<td>IR 12:15-'R12'</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>65</td>
<td>LHR R1, R12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>66</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10110110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT END</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>NOT EQUAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>FALSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>EQUAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>TRUE:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>ADD-RT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>67</td>
<td>IR 0:7-'LHR'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>68</td>
<td>IR 8:11-'R12'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>69</td>
<td>IR 12:15-'R1'</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>70</td>
<td>LHR R12, R1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>71</td>
<td>IR 0:7-'LHR'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>72</td>
<td>IR 8:11-'R13'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>73</td>
<td>IR 12:15-'R2'</td>
</tr>
</tbody>
</table>

Table 4.1 (cont.)
Sample Algorithm Trace
<table>
<thead>
<tr>
<th>ACTION/COND.</th>
<th>R4</th>
<th>R5</th>
<th>R9</th>
<th>R10</th>
<th>R13</th>
<th>STEP #</th>
<th>IR OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>74</td>
<td>LHR R13,R2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>75</td>
<td>IR 0:7-'AHR'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>76</td>
<td>IR 8:11-'R12'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>77</td>
<td>IR 12:15-'R13'</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>78</td>
<td>AHR R12,R13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>79</td>
<td>IR 0:7-'CLHI'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>80</td>
<td>IR 8:11-'R12'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>81</td>
<td>IR 12:15-'X'FF'</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>82</td>
<td>CLHI R12,X'FF'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>83</td>
<td>IR 0:7-'BNC'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>84</td>
<td>IR 12:15-'*+3'</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>85</td>
<td>BNC *+3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>86</td>
<td>IR 0:7-'LH'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>87</td>
<td>IR 8:11-'R10'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>88</td>
<td>IR 12:15-ZERO</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>89</td>
<td>LH R10,ZERO</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>90</td>
<td>IR 0:7-'XHR'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>91</td>
<td>IR 8:11-'R10'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>92</td>
<td>IR 12:15-X'03'</td>
</tr>
<tr>
<td>WRITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>93</td>
<td>XHR R10,X'03'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>94</td>
<td>00</td>
</tr>
<tr>
<td>CARRY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>95</td>
<td>NOT EQUAL</td>
</tr>
<tr>
<td>NOT EQUAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>FALSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>00 12</td>
</tr>
<tr>
<td>NOT EQUAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1 (cont.)

Sample Algorithm Trace
Table 4.1 (Cont.)

Sample Algorithm Trace

Algorithm Output

The next step in verification was to examine the output of the above algorithm and trace its execution to insure correctness. For ease of the reader, I have condensed the output into figure 4.3 (Condensed Listing) and have provided a trace of its execution in Table 4.2 below.
1  LHR  R12,R1  PUT ACS IN R12
2  LH   R10,Two PUT '10'B IN R10
3  LHI  R15,X'01' PUT '01'B IN R15
4  SLLS R12,X'01' SHIFT R12 LEFT 1 BIT
5  BC   **+4 IS CARRY EQUAL TO 1?
6  AHR  R12,R15 NO, ADD R15 TO R12
7  LHI  R15,X'00' PUT '00'B IN R15
8  B    **+3 GO TO NEXT EMULATED INSTRUCTION
9  AHR  R12,R15 YES, ADD R15 TO R12
10 LHI  R15,X'01' PUT '01'B IN R15
11 LHR  R1,R12 PUT R12 INTO R1
12 LHR  R12,R1 PUT R1 INTO R12
13 LHR  R13,R2 PUT R2 INTO R13
14 AHR  R12,R13 ADD R13 INTO R12
15 CLHI R12,X'FF' GREATER/EQUAL 2E16?
16 BNC **+3 NO,
17 LH   R10,Zero YES, GET CARRY FUNCTION
18 XHR  R10,X'03' COMPLEMENT IT
19 LHR  R2,R12 PUT R12 INTO R2

Figure 4.3
Condensed Listing
Table 4.2

Flow of Generated Code

Summary

Since few algorithms, if any, are ever accepted on an unproven basis, I have presented a sample of the proposed algorithm for this emulator and
have presented a step-by-step trace table of the logic based upon the input of two instructions, the MOVOL 1,1 which is a sample of a multi-stepped instruction, and the ADD 1,2 which is an example of a single stepped instruction. The output of these instructions was presented so that the viewer might have a more concise view of the code generated from the two input instructions, and then a trace table was provided for the generated code, in order that it might be verified as correct.
Chapter 5

CONCLUSION

In this report I presented the design of an INTERDATA implementation of the U. S. Navy MINI-COBOL compiler system. The problem of portability with regards to the present system and the alternatives of a functional rewrite of the assembler routines to effect portability versus an emulation of the instruction set of the NOVA in INTERDATA micro-code was presented. The advantages of each of the alternatives were discussed and I decided that the emulation method, while not specifically designed for machine efficiency, would provide for a system which is entirely transparent to the user and would facilitate ease of operation, being in essence, no different than the job submission scheme for most modern computers.

A presentation of the general flow of the proposed system was given wherein I discussed the philosophy of the actual design to be used at a high level, feeling that the emulation overview would aid the viewer in the following sections which presented the emulator at a more particular level.

At the detailed design presentation of the emulator, I discussed the input (the NOVA instruction formats to be considered) in detail. I then presented each NOVA function, giving an INTERDATA-like assembler language algorithm for the emulation of the function discussed. I also presented an algorithm for the emulation of the Fetch and Decoder modules which would effect the emulation in the proposed system.

After having given the detailed specifications of the Fetch and Decoder modules, the input to be utilized in the emulator, and the algo-
rithms for the actual emulation, I presented a sample verification of the design by providing a step-by-step emulation of two NOVA instructions. I then provided a step-by-step trace table of the execution of the output of the emulation to determine its soundness.

In conclusion, then, I believe the results of the report indicate that the overall design of the proposed emulator is correct and desirable; correct because all instructions were verified in function and results and included in the report was a sample of this verification technique, and desirable because the extended capabilities of the mini-computers which would be presented with COBOL and of the "general" design which would allow for expansion in the areas of virtual memory systems and mini-computer I/O device handling systems.
BIBLIOGRAPHY


APPENDIX A

MINI-COBOL LANGUAGE SPECIFICATIONS
OVERVIEW OF U. S. NAVY COBOL COMPILING
SYSTEM FOR MINI-COMPUTERS

Introduction

This is an overview of the design of the U. S. Navy MINI-COBOL compiler. The language does not include, as you will see, the full ANS COBOL language set, but rather a limited subset known as MINI-COBOL.

This system was developed because of the increasing use of mini-computers within the Navy and the lack of program portability these systems caused. While MINI-COBOL is not completely portable, any program written in this subset, with minor modifications, will run on any machine with an equal or higher level compiler.

Overview

The Navy compiling system is essentially a two phase compiler. The first phase is subdivided into two passes, with the first pass examining the various MINI-COBOL verbs and building a tuple and the second pass resolving the references and completing the tuple. The second phase is subdivided into two separate functions. The first function will read tuples and allocate memory to declared variables, establish a constant/literal pool, and establish the various editing masks used in the application program in memory. The second function will read the tuples, examine them and then will execute them as required.

Phase I

Phase I of the U. S. Navy MINI-COBOL compiler is that portion of the system which is written in COBOL. All of the machine independent portions of the compiler are found in these programs. The inputs to
Figure A.1

Overview Flowchart
Phase I is the user's MINI-COBOL source file and the output of Phase I is the user's program as translated into tuples.

![Diagram of Phase I File Flow]

Figure A.2

Phase I File Flow

Phase I Tuples

As was previously mentioned, the input into the system is the user's source input and the output a series of tuples. The following text is a description of those tuples as generated by Phase I.

Identification Division word.

The output for the IDENTIFICATION DIVISION from the MINI-COBOL compiler consists solely of the PROGRAM-ID; a five character name which will start in position number one of the output.
Environment Division word.

The ENVIRONMENT DIVISION word represents the SELECT statements and has the following format.

<table>
<thead>
<tr>
<th>POSITION</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 3</td>
<td>Always Zero</td>
</tr>
<tr>
<td>4 - 8</td>
<td>Reserved</td>
</tr>
<tr>
<td>9 - 13</td>
<td>File-name</td>
</tr>
<tr>
<td>14 - 18</td>
<td>Device-name</td>
</tr>
<tr>
<td>19 - 23</td>
<td>Device file-name</td>
</tr>
</tbody>
</table>

Environment Division Tuple

Data Division word.

The DATA DIVISION word has two different functions. It is used to represent the FD's (file descriptions) and it is used to represent all data items which have been declared, including all particular characteristics of the declared data-items. The DATA DIVISION word has the following format.
<table>
<thead>
<tr>
<th>POSITION</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 5</td>
<td>Data-name/file-name</td>
</tr>
<tr>
<td>6 - 10</td>
<td>Displacement from top-of-memory</td>
</tr>
<tr>
<td>11</td>
<td>Data type (X or 9)</td>
</tr>
<tr>
<td>12 - 15</td>
<td>Length of item</td>
</tr>
<tr>
<td>16 - 17</td>
<td>Decimal point location</td>
</tr>
<tr>
<td>18</td>
<td>BLANK ZERO flag</td>
</tr>
<tr>
<td>19</td>
<td>SIGN flag</td>
</tr>
<tr>
<td>20</td>
<td>USAGE flag</td>
</tr>
<tr>
<td>21</td>
<td>SYNC flag</td>
</tr>
<tr>
<td>22</td>
<td>OCCURS</td>
</tr>
<tr>
<td>23 - 24</td>
<td>Edit table reference</td>
</tr>
<tr>
<td>25 - 27</td>
<td>Value table reference</td>
</tr>
</tbody>
</table>

Figure A.5
Data Division Tuple

At the FD level, the first twenty-seven character word for that file is started by translating statements that follow, when present, into the total size of the file plus the number of records. The file-name is placed in the first five positions. If the BLOCK and the RECORD clauses are present, they are multiplied and the result will be placed in the twenty-seven character word as this is the length of the physical record which will be read. Should the BLOCK clause be omitted, then one record per block is assumed. Should the RECORD clause be omitted, a default of eighty characters per record will be assumed.
The record-name of the 01 level is placed in the data-name field. The top-of-memory address is passed along from the FD level (if in the FILE SECTION). If the record-name encountered is not within the FILE SECTION, then an augmented address is started for the record, i.e., an address relative to top-of-memory will be maintained on a sequential basis for each structure.

Processing continues through all the elementary data-items. The data-name for each elementary item is placed in the data-name field. The starting position (within the area of storage) is computed and placed in the displacement field. The length and data type, either X or 9, are placed into their respective fields, if present. One twenty-seven character word will be generated for each elementary data-item.

**Procedure Division word.**

For each PROCEDURE DIVISION entry, a twenty-three character word is generated. The PROCEDURE DIVISION word has the following format.

<table>
<thead>
<tr>
<th>POSITION</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 4</td>
<td>Sequence number</td>
</tr>
<tr>
<td>5</td>
<td>Always zero</td>
</tr>
<tr>
<td>6 - 8</td>
<td>Routine number</td>
</tr>
<tr>
<td>(6 - 7)</td>
<td>Verb</td>
</tr>
<tr>
<td>(8)</td>
<td>Variant of verb</td>
</tr>
<tr>
<td>9 - 13</td>
<td>First data-name</td>
</tr>
<tr>
<td>14 - 18</td>
<td>Second data-name</td>
</tr>
<tr>
<td>19 - 23</td>
<td>Third data-name</td>
</tr>
</tbody>
</table>

*Figure A.6*

**Procedure Division Tuple**

55
Each verb in the MINI-COBOL system has been assigned a unique verb code (see Appendix B) which is placed in position six and seven of the PROCEDURE DIVISION word. Since there are in some cases more than one variation of the verb, the system uses position eight to further delineate the operation requested.

The data-names are placed into the appropriate fields as they are encountered on the source statements. The Source sequence number is the number of the MINI-COBOL source statement from which the PROCEDURE DIVISION word is derived.

Phase IA

Although Phase IA is a separate step at this time, it is anticipated by the Navy Programming Languages Section, that this process will be incorporated into Phase I in the very near future providing a single program two step compiler.

This portion of the compiler accepts as input the tuples as generated by Phase I. These tuples are those of the PROCEDURE DIVISION statements and those generated which represent the literals/values of the DATA DIVISION and PROCEDURE DIVISION. The tuples will be manipulated and will be used to build a series of Data Tables and Procedure tuples which can be executed by the Phase II portion of the compiler.
Phase IA File Flow

Phase IA Tuples

As was previously mentioned, Phase IA produces a series of updated tuples and several tables the format of which is presented below.

I/O table.

\[
\begin{align*}
X(5) & \quad \text{Device Name (ASCII)} \\
X(5) & \quad \text{Channel Number}
\end{align*}
\]

The I/O table is utilized to link COBOL files to specific I/O channels. The Channel Number will be filled in by the Interpreter (Phase II) at run time.
File descriptor table.

9(5)  -  Raw Data Byte Pointer
9(4)  -  Data-name Table Entry Pointer
9(2)  -  I/O Table Pointer
9(4)  -  Total Length (Bytes)
9     -  Label
9(3)  -  File Access Method*
9(4)  -  Keylen*

Figure A.9

File Descriptor Table

The file descriptor table link COBOL file name to associated I/O channels and record descriptions.

Data-name table.

9(5)  -  Raw Data Byte Pointer
9     -  Sign Flag
9     -  Data Type
9(4)  -  Data Length
9     -  Occurs Flag
9     -  Sync Flag
9     -  Blank Zero Flag
9(3)  -  Value Table Pointer
9(3)  -  Data Name Edit Pointer
9(2)  -  Decimal Point Location

Figure A.10

Data-name Table

* File access method and keylen have not been implemented but are seen as additions for the very near future and space was reserved for them.
The Data-name table describes all data-names used in the specific COBOL program and provides all necessary information needed to utilize that data-name.

Data type may contain various values which are as listed below:

1) X - signifies the data type is alphanumeric and will be translated into the digit "2";

2) 9 - signifies numeric data and will be translated into the digit "3".

Value/literal table.

X(80) - Actual Value

Figure A.11
Value/Literal Table

The Value/Literal table (also called Memory Layout Map) sets up the initial values in memory that have been specified by the user in his COBOL program.

Edit table.

X(30) - String of ASCII Characters

Figure A.12
Edit Table

The Edit Table sets up masks to be used in editing as the user has requested in his COBOL program.
Procedure Division Quints.

9(5) - Quint Sequence Number
9(3) - Operation Code
9(4) - Operand one
9(4) - Operand two
9(4) - Operand three

Figure A.13
Procedure Division Quints

The Procedure Division Quintuplets are basically the same as those generated by Phase I with data-names replaced and sequence numbers added.

Phase II - Interpreter

Phase II of the MINI-COBOL compiler is written in NOVA FORTRAN and is stored in the machine in executable binary form. There are two distinct operations performed by Phase II, the first being DATA DIVISION memory allocation, and the second being COBOL quintuple examination and execution. Figure A.14 below presents the general file flow of the Interpreter.

Figure A.14
Phase II File Flow
Upon execution of Phase II, the first part of the Interpreter allocates the COBOL program DATA DIVISION to memory and proceeds to map, into the proper memory locations, any initial value or literals which have been specified by the programmer. Upon completion of this task, the second part of the Interpreter begins to examine each tuples' opcode and will then pass control based on the operation code to a dispatch table. Depending upon where entry occurred in the dispatch table, control is again passed to a subroutine dispatch table. The second dispatch table contains a list of "Calls" to NOVA assembler routines in the MINI-COBOL library which are in the correct order required to process the particular operation code with its options. Upon completion of the list, control is then passed back to the Interpreter which will read the next tuple, thereby closing the loop. (see Figure A.15).
Execution Summary

The first MINI-COBOL statement of importance to the compiler is the PROGRAM-ID source statement. Upon its encounter, the compiler will produce a Control card with the digit "1" in the first position. The Control Card will then be written out to the file MINIZ. Control Cards are used in Phase IA to identify the type of tuple which immediately follows. The compiler will then construct the Identification Division word as given in figure A.3 and write it to the file MINIZ.
The next source statement of significance to the compiler is the ENVIRONMENT DIVISION statement. The compiler will produce a Control Card with the digit "2" in the first position and write it out to the file MINIZ. The compiler will then construct the Environment Division word as described in figure A.4 and write it out to the file MINIZ.

When the source statement DATA DIVISION is encountered by the compiler, it will scan for the keyword FILE SECTION. When FILE SECTION is encountered it will produce a Control Card with the digit "3" in the first position and write it out to the file MINIZ. Data Division words will then be constructed as per figure A.5. When the keyword WORKING-STORAGE SECTION is encountered a Control Card with the digit "5" will be written out to the file MINIZ. Data division words will again be produced as per figure A.5 and written out to the file MINIZ. Upon completion of the construction of data division words, the compiler will produce a Control Card with the digit "5" in the first position and write it out to the file MINIZ. The edit tables and literal/value tables constructed in the program will be written out to MINIZ and the file LITF, respectively, and the compiler will then begin construction of the Procedure Division words as per figure A.6. Upon completion of construction of the last Procedure Division word, the compiler will produce a Control Card with the digits "9000" in the first four positions (sequence field) and the last sequence number will be entered into the third operand field. The compiler will then have completed Phase I after having written the above record to the file MINIZ.

Phase IA will then be invoked. The files MINIZ and LITF will be revised into a slightly different format to produce tables as per figure A.8 through A.13. When processing has been completed, Phase II will be invoked.
Phase II uses as input the save file created from Phase IA. This phase will read the save file, allocate memory and then read, examine and execute the various tuples as generated by Phase I and Phase IA.
APPENDIX B

MINI-COBOL VERBS
IDENTIFICATION DIVISION.
PROGRAM-ID. program-name.
AUTHOR. comment-entry.
INSTALLATION. comment-entry.
DATE-WRITTEN. comment-entry.
SECURITY. comment-entry.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. computer-name-model.
DEBUGGING MODE.
OBJECT-COMPUTER. computer-name-model.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

SELECT file-name ASSIGN implementor-name-1.
.
.
.

DATA DIVISION.
FILE SECTION.
FD file-name
   BLOCK integer-1 RECORDS
   RECORD integer-2
   LABEL RECORD STANDARD OMITTED
   record-description . . .

WORKING-STORAGE SECTION.

77 - Level-description-entries
record-description-entries

FORMAT:

level-number data-name-1

REDEFINES data-name-2

PIC character-string

Figure B.1
Language Skeleton
USAGE  COMP
        DISPLAY

SIGN    LEADING          SEPARATE
        TRAILING

OCCURS integer-1

SYNC    LEFT
        RIGHT

BLANK  ZERO

VALUE  literal-1

PROCEDURE DIVISION

paragraph-name.
    sentence ........
    .
    .
    .

Figure B.1 (Cont.)

Language Skeleton
ACCEPT  identifier-1

ADD  identifier-1  TO  identifier-2
    literal-1

ADD  identifier-1  identifier-2  TO  GIVING  identifier-3
    literal-1  literal-2

CLOSE  file-name-1  file-name-2  file-name-3

DISPLAY  identifier-1  identifier-2
    literal-1  literal-2

DIVIDE  identifier-1  INTO  identifier-2
    literal-1

DIVIDE  identifier-1  INTO  identifier-2  GIVING  identifier-3
    literal-1  literal-2

ENTER  language-name  routine-name

EXIT

GO TO  procedure-name-1

GO TO  procedure-name-1  procedure-name-2....
    procedure-name-10  DEPENDING  identifier-1

IF  condition

MOVE  identifier-1  TO  identifier-2
    literal-1

MULTIPLY  identifier-1  BY  identifier-2
    literal-1

MULTIPLY  identifier-1  BY  identifier-2  GIVING  identifier-3
    literal-1  literal-2

OPEN  INPUT  file-name-1  file-name-2  file-name-3
    I-0

Table B.1

MINI-COBOL Verb Formats
PERFORM procedure-name-1 THRU procedure-name-2

PERFORM procedure-name-1 THRU procedure-name-2
    identifier-1 TIMES
    literal-1
READ file-name-1 END GO TO procedure-name-1

REWRITE record-name-1

STOP     RUN
  literal-1

SUBTRACT identifier-1 FROM identifier-2
  literal-1
SUBTRACT identifier-1 FROM identifier-2 GIVING identifier-3
  literal-1
WRITE record-name BEFORE integer-1 LINES
  AFTER PAGE

All arithmetic verbs have the following optional clauses:

ROUNDED (only on the resultant identifier)

SIZE ERROR GO TO procedure-name-1

Table B.1 (Cont.)

MINI-COBOL Verb Formats
identifier-1 NOT GREATER THAN identifier-2
 NOT LESS THAN literal-1
 NOT EQUAL TO literal-2

Figure B.2
Relational Conditions

identifier NOT NUMERIC
 NOT ALPHABETIC

Figure B.3
Class Conditions

data-name (subscript)

Figure B.4
Subscripting/Identifiers
<table>
<thead>
<tr>
<th>CARD COLUMN</th>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 6</td>
<td>Sequence Number</td>
</tr>
<tr>
<td>7</td>
<td>Comment and DEBUG entries</td>
</tr>
<tr>
<td>8 to 12</td>
<td>Paragraph-name</td>
</tr>
<tr>
<td>12 to 23</td>
<td>Verb</td>
</tr>
<tr>
<td>24 to 29</td>
<td>Field 1</td>
</tr>
<tr>
<td>30 to 35</td>
<td>Field 2</td>
</tr>
<tr>
<td>36 to 41</td>
<td>Field 3</td>
</tr>
<tr>
<td>42 to 47</td>
<td>Field 4</td>
</tr>
<tr>
<td>48 to 53</td>
<td>Field 5</td>
</tr>
<tr>
<td>54 to 59</td>
<td>Field 6</td>
</tr>
<tr>
<td>60 to 65</td>
<td>Field 7</td>
</tr>
<tr>
<td>66 to 71</td>
<td>Field 8</td>
</tr>
<tr>
<td>72</td>
<td>Period</td>
</tr>
<tr>
<td>73 to 80</td>
<td>Identification</td>
</tr>
</tbody>
</table>

Table B.2
Coding Sheet

Because the U. S. Navy MINI-COBOL compiler expects its input to be entered in a fixed format, the coding sheet will be different than the usual coding sheet for COBOL. The above is a breakdown of the coding sheet used by this compiler.
**QUINT POSITIONS 6-8**  

<table>
<thead>
<tr>
<th>Verb and Option Requested</th>
</tr>
</thead>
<tbody>
<tr>
<td>010 ACCEPT ___________</td>
</tr>
<tr>
<td>020 ADD ____ GIVING ____</td>
</tr>
<tr>
<td>260 ADD ____ TO ____</td>
</tr>
<tr>
<td>030 CLOSE ____</td>
</tr>
<tr>
<td>040 DISPLAY ____</td>
</tr>
<tr>
<td>050 DIVIDE ____ INTO ____ GIVING ____</td>
</tr>
<tr>
<td>060 ENTER FORTRAN ____</td>
</tr>
<tr>
<td>061 ENTER ASSEMBLY ____</td>
</tr>
<tr>
<td>062 ENTER FORTRAN ____</td>
</tr>
<tr>
<td>063 ENTER ASSEMBLY ____</td>
</tr>
<tr>
<td>070 GO TO ____ *</td>
</tr>
<tr>
<td>080 WRITE ____ BEFORE ____</td>
</tr>
<tr>
<td>081 WRITE ____ AFTER ____</td>
</tr>
<tr>
<td>082 WRITE ____ BEFORE PAGE</td>
</tr>
<tr>
<td>083 WRITE ____ AFTER PAGE</td>
</tr>
<tr>
<td>090 MOVE ____ TO ____</td>
</tr>
<tr>
<td>100 MULTIPLY ____ BY ____ GIVING ____</td>
</tr>
<tr>
<td>110 OPEN INPUT ____</td>
</tr>
<tr>
<td>111 OPEN I-O ____</td>
</tr>
<tr>
<td>112 OPEN OUTPUT ____</td>
</tr>
<tr>
<td>120 PERFORM ____ THRU ____ (TIMES)</td>
</tr>
<tr>
<td>130 READ ____ END GO TO ____</td>
</tr>
<tr>
<td>140 REWRITE ____</td>
</tr>
<tr>
<td>150 STOP RUN ____</td>
</tr>
<tr>
<td>280 STOP ____</td>
</tr>
<tr>
<td>160 SUBTRACT ____ FROM ____ GIVING ____</td>
</tr>
<tr>
<td>270 SUBTRACT ____ FROM ____</td>
</tr>
<tr>
<td>170 EXIT ____</td>
</tr>
<tr>
<td>180 ROUNDED OPTION ____</td>
</tr>
<tr>
<td>190 SIZE ERROR OPTION ____</td>
</tr>
<tr>
<td>200 IF ____ EQUAL ____</td>
</tr>
<tr>
<td>201 IF ____ NOT EQUAL ____</td>
</tr>
<tr>
<td>210 IF ____ LESS ____</td>
</tr>
<tr>
<td>211 IF ____ NOT LESS ____</td>
</tr>
<tr>
<td>220 IF ____ GREATER ____</td>
</tr>
<tr>
<td>221 IF ____ NOT GREATER ____</td>
</tr>
<tr>
<td>230 IF ____ NUMERIC ____</td>
</tr>
<tr>
<td>231 IF ____ NOT NUMERIC ____</td>
</tr>
<tr>
<td>240 IF ____ ALPHABETIC ____</td>
</tr>
<tr>
<td>241 IF ____ NOT ALPHABETIC ____</td>
</tr>
<tr>
<td>250 DENOTES SUBSCRIPTS</td>
</tr>
</tbody>
</table>

*GO TO ... DEPENDING IS TREATED AS A SERIES OF IF STATEMENTS.*

---

**Figure B.5**

Verb Code
APPENDIX C

MINI-COBOL SOURCE LISTING
.TITL  ACCEPT
.ENT  AP1
.EXTD  PUT
.NREL

APT1:
ISZ  RTST.2 ;GET NEXT RETURN POSITION
STA  3,AP1ST.2 ;SAVE RETURN ADDRESS
LDA  1,OP1DP.2
STA  1,OPPOP.2
LDA  1,OP1DL.2
STA  1,OPD1L.2
LDA  0,DMPK.2 ;LOAD QUESTION MARK IN AC0
.SYSTEM  1,WRITE QUESTION MARK OUT TO THE TELETYP
Jsr  0,RTER.3
MOV  3,2
LDA  0,BELL.2 ;LOAD BELL IN AC0
.SYSTEM  2,DRING THE BELL ON THE TELETYP
Jsr  0,RTER.3

FETCH:
.SYSTEM  3,GET A CHARACTER FROM THE
OCHAR  2,TELETYP
Jsr  0,RTER.3
.SYSTEM
OCHAR
Jsr  0,RTER.3
MOV  3,2 ;RESTORE AC2
LDA  3,C1.2 ;LOAD CARRIAGE RETURN
SUB#  3,0,SNR ;COMPARE
Jmp  FNISH ;EQUAL, JMP TO FINIS
MOV  0,1
LDA  3,OPPOP.2 ;LOAD AC2 WITH BYTE POINTER
STA  3, Rivptr.2 ;STORE IN BYTE POINTER
Jsr  0,PUT ;PERFORM THE STORE ROUTINE
Jsr  0,RTER.2
ISZ  OPDP.2 ;INCREMENT BYTE POINTER
DSZ  OPDL.2 ;DECREMENT LENGTH
Jmp  FETCH ;JUMP TO GET NEXT CHARACTER
LDA  0,C1.2

FNISH:
.SYSTEM
OCHAR
Jsr  0,RTER.3
MOV  3,2 ;RESTORE AC2
LDA  0,LF.2
.SYSTEM
OCHAR
Jsr  0,RTER.3
MOV  3,2 ;RESTORE AC2
Jmp  0,RTNL.2
THE CALL TO THIS ROUTINE IS (N, PAC1, PAC2, PAC3)

ADDPK: ISZ RTST.2 GET NEXT STORE POSITION
STA 3 @RTST.2 STORE RETURN ADDRESS
LDA 1 @PACK1.2 INITIALIZE ADDRESS
STA 1 @PACK1.2 FOR ADDPACK
LDA 1 @PACK2.2 ROUTINE
STA 1 @PACK2.2
LDA 1 @PACK3.2
STA 1 @PACK3.2
LDA 1 @TEN.2 GET AN EIGHT FOR EIGHT WORD ADD
STA 1 N.2 STORE IN N
JMP +3

ADDT: ISZ RTST.2 GET NEXT STORE POSITION
STA 3 @RTST.2 STORE RETURN ADDRESS
LDA 3 N.2 GET NUMBER OF WORDS TO BE ADDED
STA 3 I.2 STORE IN I
LDA 1 @PACK1.2 GET FIRST FIELD TO BE ADDED
ADD 3 I GET FIRST WORD
STA 1 @PACK1.2 STORE BACKOUT
LDA 1 @PACK2.2 DO THE SAME FOR PAC2
ADD 3 I
STA 1 @PACK2.2
LDA 1 @PACK3.2 DO THE SAME FOR PAC3
ADD 3 I
STA 1 @PACK3.2
DSZ PAC1.2 TO GET PROPER ALIGNMENT
DSZ PAC2.2 FOR ADDITION AND
DSZ PAC3.2 FOR STORING
LDA 0 @PACK1.2 GET FIRST ARGUMENT
LDA 0 @PACK2.2 GET SECOND ARGUMENT
ADDZ 1 0 ADD TOGETHER CHECK FOR
STA 0 @PACK3.2

LP:
DSZ 1 2
JMP +2
JMP ORTN.2
DSZ PAC1.2
DSZ PAC2.2
DSZ PAC3.2
SUBCL 0 0
LDA 0 @PACK1.2
ADDZ 1 0
LDA 0 @PACK2.2
ADD 1 0
STA 0 @PACK3.2
JMP LP
TITL   ADD
ENT    ADDTT, ADDPK
NREL

THE CALL TO THIS ROUTINE IS N, PAC1, PAC2, PAC3)

ADDPK:  ISZ   RTST. 2  , GET NEXT STORE POSITION
        STA   3, RTST. 2  , STORE RETURN ADDRESS
        LDA   1, PACK1, 2  , INITIALIZE ADDRESS
        STA   1, PAC1, 2  , FOR ADDPK
        LDA   1, PACK2, 2  , ROUTINE
        STA   1, PAC2, 2
        LDA   1, PAC3, 2
        STA   1, PAC3, 2
        LDA   1, TEN, 2  , GET AN EIGHT FOR EIGHT WORD ADD
        STA   1, N, 2  , STORE IN N
        JMP   +3

ADDIT:  ISZ   RTST. 2  , GET NEXT STORE POSITION
        STA   3, RTST. 2  , STORE RETURN ADDRESS
        LDA   3, N, 2  , GET NUMBER OF WORDS TO BE ADDED
        STA   3, I, 2  , STORE IN I
        LDA   1, PAC1, 2  , GET FIRST FIELD TO BE ADDED
        ADD   3, I  , GET FIRST WORD
        STA   1, PAC1, 2  , STORE PACKOUT
        LDA   1, PAC2, 2  , DO THE SAME FOR PAC2
        ADD   3, I  , DO THE SAME FOR PAC3
        STA   1, PAC2, 2
        LDA   1, PAC3, 2  , DO THE SAME FOR PAC3
        ADD   3, I  , DSZ PAC1, 2  , TOGET PROPER ALIGNMENT
        DSZ   PAC2, 2  , FOR ADDITION AND
        DSZ   PAC3, 2  , FOR STORING
        LDA   1, DPKAC1, 2  , GET FIRST ARGUMENT
        LDA   0, DPKAC2, 2  , GET SECOND ARGUMENT
        ADDZ   1, 0  , ADD TOGETHER CHECK FOR
        STA   0, DPKAC3, 2
        DSZ   1, 2
        JMP   +2
        JSR   @RTN. 2
        DSZ   PAC1, 2
        DSZ   PAC2, 2
        DSZ   PAC3, 2
        SUBCL   0, 0
        LDA   1, DPKAC1, 2
        ADDZ   1, 0
        LDA   1, DPKAC2, 2
        ADD   1, 0
        STA   0, DPKAC3, 2
        JMP   LP
READ TABLE PTRS AND ALLOCATE CORE

.TITLE ALCOR
.NREL
.TXTM 1
.ENT ALCOR
.EXTN DSKFL

ALCOR:  ISZ RTST, 2
         STA 3, OPTST, 2
         LDA 0, NEHF, 2
         MOV 0,0, SNR
         JMP ALC1
         SUBZ 0, 0
         STA 0, IYCNT, 2
         STA 0, FCNT, 2
         STA 0, ECNT, 2
         STA 0, NCNT, 2
         STA 0, PCNT, 2
         STA 0, NCNT, 2
         JMP ALC2
         LDA 3, IBUF
         STA 3, IUBF, 2
         SUBZL 0, 0
         STA 0, IFCN, 3
         READ
         STA 1, SVFCN, 2
         STA 1, IO, 3
         STA 0, STHD, 3
         LDA 1, 111
         STA 1, ICH, 3
         LDA 0, IOPTR, 2
         STA 0, IVEC, 3
         ADC 0, 0
         STA 0, -10, 3
         SET PSECTR TO MINUS 1
         JSR 0DSKFP
         READ PTRS
         SUBZ 0, 0

ALC1:  LDA 1, IYCNT, 2
         ADD 1, 0
         LDA 1, FCNT, 2
         ADD 1, 0
         LDA 1, ECNT, 2
         ADD 1, 0
         LDA 1, NCNT, 2
         ADD 1, 0
         LDA 1, RSCNT, 2
         ADD 1, 0
         LDA 1, NCNT, 2
         ADD 1, 0
         LDA 1, TADD
         ADD 1, 0
         STA 0, TEMP2, 2
         STORM 0, TEMP, 2

ALC2:  LDA 1, IYCNT, 2
         ADD 1, 0
         LDA 1, FCNT, 2
         ADD 1, 0
         LDA 1, ECNT, 2
         ADD 1, 0
         LDA 1, NCNT, 2
         ADD 1, 0
         LDA 1, TADD
         ADD 1, 0
         STA 0, TEMP2, 2
         TOTAL # WORDS IN SAVE FILE
         STA 0, TEMP, 2

.MEM

JMP .+1
MOV 3, 2
LDA 3, SNHAX, 2
MOV 3, S2R
JMP .+3
STA 1, SNHAX, 2
JMP ALC3
SUB 3.1
LDA 3. TEMP. 2
SUB 1.3
STA 3. TEMP2. 2
LDA 1. SNMAX. 2

ALC3:
SUB 1.0
INC 1.1
STA 1. IOST. 2
LDA 1. TEMP. 2
SUBZ 1.0
JSL RTER. 2
LDA 0. IOST. 2
LDA 1. IOCNT. 2
ADD 1.0
LDA 1. IOAD
ADD 1.0
STA 0. FDST. 2
LDA 1. FDCTN. 2
ADD 1.0
LDA 1. FDAD
ADD 1.0
STA 0. EDST. 2
LDA 1. EDCTN. 2
ADD 1.0
STA 0. DNST. 2
LDA 1. DNCNT. 2
ADD 1.0
LDA 1. DNAD
ADD 1.0
STA 0. RSST. 2
LDA 1. RSCNT. 2
ADD 1.0
STA 1. RMSX. 2
LDA 1. RSDAD
ADD 1.0
STA 0. ONSH. 2
LDA 0. TEMP2. 2

SYS TN
.MENI
JSR @RTER. 3
MOV 3.2
SUB 0.0
LDA 3. IOST. 2

ALC4:
STA 0.0.3
INC 3.3
DSZ TEMP. 2
JMP ALC4

JSL @RTN. 2

I11: 11.
TADD: 622.
IORD: 12.
FDAD: 10.
DNAD: 5A.
DNAD: 5A.
RSAD: 50A.
DSKFP: DSKFL
IBUF: IBUFG
Iblk: 22
IBUF: 256.
THESEROUTINESSETANDDELET20BREAKPOINTS

.TITLEBREAK
.NREL
.TXTM1
.ENTBREAK,DELET,BTL

BREAK:ISZPTST,2
STA3,PTST,2
JSRARDAEP,2READCHAR
JSR@TER,2
JMP.+2NOTCARRRET
JMPBR4CARRRET
JSR@MNCP,2READ#TOSET
JMP.+2
JMP.+3

BR1:ISZCERFL,2ILLCOMMAND
JMP@TER,2
MOV1,1,SNR
JSRBR1
LDA3,BTBL
LDA0,BTL
STA0,TEMP,2

BR2:LDA0,0,3
MOV0,0,SNRBREAKALREADYSET?
JMPBR3
INC3,3
DSZTEMP,2
JMPBR3
JSRBR1

BR3:STA1,0,3STOREQUINT#
JSR@RTN,2

BR4:LDA0,BTBLLISTALLCURRENTBREAKS
STA0,X3T,2BREAKTABLELENGTH
LDA0,BTBL
STA0,APGDE,2

BR5:LDA@APGDE,2QUINT#
MOV0,0,SNROCUPIED?
JMPBR6
INC3,3
JSR@PROCP,2PRINTQUINT#
JSR@TER,2
JSR@PROCP,2
JSR@TRFR,2

BR6:ISZAPGDE,2
DSZX3T,2
JMPBR5
JSR@RTN,2DONE

DELET:ISZPTST,2DELETEBREAKS
STA3,PTST,2
JSRARDAEP,2READCHAR
JSR@TER,2
JMP.+2NOTCARRRET
JMPBR1CARRRETURN
JSR@MNCP,2READ#TODELETE
JSRBR1ILLEGALCOMMAND
LDA3,BTBL
LDA0,BTBL

DE8:LDA0,0,3
SUD1,A,SNRISTHISBREAK?
JMPDE85
YES
.TITL BREAKDOWN
.ENT SETED
.EXTN EDIT1, EDITT
.NPEL

EDIT: EDIT1

SE TED: JSR @ EDIT
JSR @ RTFR.2
LDA 1, ECODE.2  ; GET EDIT CODE
LDA 0, MSK2.2   ; GET A FIFTEEN
SUB 0, 1, SZP ; CHECK TO SEE IF EQUAL
JMP AAD ; NOT EQUAL
JMP RETURN ; EQUAL

AAD: LDA 3, ECODE.2 ; GET EDIT CODE
LDA 0, ETAB ; GET EDIT CHAR TABLE
ADD 0, 3 ; GET EDIT CHAR DISPLACEMENT
LDA 3, 0, 3 ; GET THE CHAR
STA 3, ECHAR.2 ; STOR IN EDIT CHAR
JMP 0, EDIT

RETURN: JMP @ RTN.2

.EDIT: EDITT
.ETAB: TAX
.TAX: 0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

81
TITL  BYTE
ENT  F8.PU4.T

; THIS ROUTINE FETCHES BYTES AND RIGHT JUSTIFIED THEM IN AC1
; ROUTINE IS CALLED BY: JSR GET WITH AC2 CONTAINING BYTE POINTER
; ZREL  USES THESE ROUTINES REQUIRE 2 GLOBAL VECTORS

GET:  FETCH
     ; STORE
      .NREL

FETCH:  ISZ  RTST,2 ; GET NEXT STORE POSITION
        STA  3,RTST,2 ; SAVE RETURN ADDRESS
        LDA  3,BYTPR,2 ; GET BYTE POINTER
        MOVZR  3,3 ; CONVERT BYTE ADDRESS TO WORD ADDRESS
        LDA  1,0,3 ; FETCH WORD CONTAINING BYTE
        MOVW  0,0,SNC ; TEST LOW ORDER BIT OF BYTE ADDRESS
        MOVS  1,1 ; SNAP BYTES IF LOW ORDER BIT IS ZERO
        LDA  0,RMSK,2 ; FETCH BYTE RMSK,2
        AND  0,1 ; CLEAR LEFT BYTE OF AC1
        JMP  @RTN,2 ; EXIT ROUTINE

; THIS ROUTINE STORES BYTES FROM AC1 (RIGHT JUSTIFIED) INTO
; MEMORY LOCATION POINTED TO BY AC2. THE ADDRESS IN AC2 IS
; A BYTE POINTER.
; ROUTINE IS CALLED BY: JSR @PUT.

STORE:  ISZ  RTST,2 ; GET NEXT STORE POSITION
        STA  3,RTST,2 ; SAVE RETURN ADDRESS
        LDA  3,BYTPR,2 ; GET BYTE POINTER
        MOVZR  3,3 ; FORM WORD ADDRESS FROM BYTE ADDRESS
        LDA  0,0,3 ; GET WORD FROM BYTE STRING
        LDA  3,RMSK,2 ; FETCH BYTE MASK
        ANDS  3,1,SZC ; CLEAR LEFT BYTE OF STORE WORD.
        MOV  0,0 ; TEST EVEN/ODD OF TARGET BYTE ADDRESS
        AND  3,0 ; SNAP BYTES OF STORE WORD IF NEEDED
        COM  1,1 ; CLEAR REPR BYTE TO BE REPLACED (LEFT BYTE)
        AND  1,0 ; AND SNAP BYTE OF STORE WORD IF NEEDED
        ADC  1,0,SZC ; REORIENT BYTES FOR STORE BACK
        MOVS  0,0 ; ROTATE WORD BACK IF NECESSARY
        LDA  3,BYTPR,2 ; GET THE BYTE POINTER
        MOVZR  3,3 ; FORM WORD ADDRESS FROM BYTE ADDRESS
        STA  0,0,3 ; STORE WORD IN BYTE STRING
        JMP  @RTN,2 ; ROUTINE EXIT
THIS ROUTINE CALLS AND RETURNS FROM DEBUF

.TITLE CALDB
.NREL
.TXTM 1
.ENT CALDB, RTDBG
.EXTN DEBUG

CALDB:
ISZ RTST.2
STA 3, @RTST.2
ISZ RTST.2
LDA 3, DBGP
STA 3, @RTST.2
JMP @DBGP

RTDBG:
LDA 2, USP
SUB 0, 0
STA 0, @RTST.2
DSZ RTST.2
JSR @RTN.2

DBGP: DEBUG
INTEGER INFILE(6), INFIL(5,8), FND, FNT, FREF, FENIL(5,3), FNOIL
INTEGER FCT, STM, STC, STMCT, STMCT, STMIL(4,3), DMT, DUM, DENT, DENT
INTEGER DISP, DFLIL(8,7), VAD, VNCNT, VFLIL(4,3), DNE, DENT, DUD, DONT(5,4)
INTEGER SVNAM(7), TPNAME(7), CR, INPNM(7), STM, FNAME(200), SDFIL
COMMON /IDMT/ IDMT(254), IDMT(258), ICNVC(12), INFILE
COMMON /INFL/ OOT, CR, IPE, IEP, IOT, IOUT, INP, LLM, IULM, SDFIL
EQUIVALENCE (INFL, FDIL), (INFL, FDIL), (INFL, FDIL)
EQUIVALENCE (INFL, INFL), (INFL, INFL), (INFL, INFL)
EQUIVALENCE (ICNVC(1), ICOME), (ICNVC(2), ICOME), (ICNVC(3), ICOME),
     (ICNVC(4), ICOME), (ICNVC(5), ICOME), (ICNVC(6), ICOME), (ICNVC(7), ICOME),
     (ICNVC(8), ICOME), (ICNVC(9), ICOME), (ICNVC(10), ICOME), (ICNVC(11), ICOME),
DATA LLM/3/, IULM/7/, STM/12/, IBUF/257/, KBUFF/258/, I_BUF/258/,
DATA G0T/2/, CR/11/, LP/10/, IEP/4/, IOUT/5/, INP/3/
READ (CR, 4000) INPNM
READ (CR, 4000) SVNAME
READ (CR, 1000) ICNVC
1000 FORMAT (12I5)
   IF (ICNVC(1) EQ 0) GO TO 50
   IULM=ICNVC(1)
   CALL OPEN (INP, INPNM, IEP)
   IF (IER.NE.1) GO TO 996
   CALL OPEN (IOUT, SVNAME, IEP)
   IF (IER.NE.1) GO TO 996
   CALL OPEN (IOT, TPNAME, IEP)
   IF (IER.NE.1) GO TO 996
   CALL OPEN (IOUT, SVNAME, IEP)
   IF (IER.NE.1) GO TO 996
   CALL OPEN (INP, INPNM, IEP)
   IF (IER.NE.1) GO TO 996
   DO 800 IINDEX=LLM, IULM
   GO TO (600, 200, 300, 400, 500, 600, 700, 750), INDP
C
C 10 FILE PROCESSING SECTION
C
C INPUT IS 80 COL CARD FORMAT. THERE ARE 8 10 CHAR ALFA FIELDS PER CARD
C
C THE FILE TERMINATOR IS A CARD WITH
C
C COL 1-2 BLANK
C 11-15 # ENTRIES
C
C OUTPUT IS A 6 WORD ENTRY. THE FIRST FIVE ARE THE NAME & 6 IS CHANNEL
C
C 200 CONTINUE
ICNT=0
IIND=STEM
INFILE(6)=0
210 READ (CR, 2000) INFIL
2000 FORMAT (499Z)
   IF (INFIL(1,1), EQ, ' ') GO TO 230
   DO 220 I=1, 8
      IF (INFIL(1,1), EQ, ' ') GO TO 210
      DO 215 J=1, 5
      INFIL(1,1)=INFIL(1,1)
      CALL NSKIL (2, INFL, IOT, STM, 6, INFIL, IEP)
      IF (IPE. NE. 1, AN, IEP. NE. 9) GO TO 500
      STM=STM+6
      INPT=INPT+1
      220 CONTINUE
GO TO 210
210 CONTINUE
    IF (ICNT NE NUMB(INFIL(1,2),5,0)) GO TO 220
    GO TO 800

FILE DESCRIPTOR PROCESSING SECTION

INPUT IS IN 6A COL CARD FORMAT. THERE ARE 5 16 COLUMN ENTRIES PER CARD.
THE FORMAT OF EACH ENTRY IS:
  COL 1-4  ENTRY # OF DATA NAME
  5-6    ENTRY # OF I0 TABLE
  7      LABEL  1-OMIT, 2-STD

THE FILE TERMINATOR IS A CARD WITH
  COL 1-5  BLANK
  17-21  # ENTRIES
  33-37  # BUFFER CHAR

OUTPUT IS 3 WORD ENTRIES IN ORDER OF THE INPUT

300 CONTINUE
    DO 305 I=1,300
305   FDNUM(I)=0
     FND=STD
     FCNT=0
     FBUF=0
310   READ (INP,30000) FDFIL
3000  FORMAT (3(I5,I4,I2,I4,I1,7X))
    IF (FDFIL(2,1).EQ.0) GO TO 330
    DO 320 I=1,3
    IF (FDFIL(2,1).EQ.0) GO TO 310
    CALL DSKFL (2, IBUF, IOUT, STD, 5, FDFIL(I,1), IEP)
    IF (IER NE 1 AND IEP NE 9) GO TO 900
    STD=STD+5
    FCNT=FCNT+1
    FDNUM(FCNT)=FDFIL(2,1)
320   CONTINUE
    GO TO 310
330   CONTINUE
    IF (FDFIL(1,2).NE.FCNT) GO TO 330
    IF (FDFIL(1,3).NE.FBUF) GO TO 330
    GO TO 800

EDIT FILE PROCESSING SECTION

INPUT IS IN 6A COL/card FORMAT WITH A CONTINUOUS CHARACTER STRING.
EACH EDIT PICTURE STRING IS TERMINATED WITH A '/' AND '' ENDS THE DATA ON A CARD
FILE TERMINATOR IS A CARD WITH
  COL 1-2  BLANK
  21-25  # STRINGS
  41-45  # CHARS

OUTPUT IS A PACKED EDIT PICTURE WITH 1 OR 2 BYTES FOR EACH CONSECUTIVE
STRING OF IDENTICAL CHARACTERS AND A TEMPORARY INDEX FILE WITH ONE
ENTRY PER STRING. THE FORMAT OF THE 'IOUT' FILE WORD IS
C
BIT 0       EDIT PACKING: 0 SHORT FORM, 1 LONG FORM
         1-4   EDIT CODE
         5-15  REPITITION COUNT FOR LONG FORM
         FOR SHORT FORM
         5-7   REPITITION COUNT
         8     1 IGNORE THIS BYTE, 0 2ND SHORT FORM
         9-12  EDIT CODE
         13-15 REPITITION COUNT

THE FORMAT FOR THE INDEX FILE ON CHANNEL IED IS ONE INTEGER
WORD PER STRING GIVING THE RELATIVE POINTER TO THE
START OF THE EDIT STRING IN THE OUTPUT FILE

C
400 CONTINUE
   END=STW
   STW=STW+1
   ITS=END
   NFLCH=0
   EDCT=1
   SVCH=-1
   BYTCT=0
   IT3=0
   IT4=0
   CALL DSKFL (2, ISBUF, IED, EDCT, 1, 0, IER)
   IF (IER NE 1 AND IER NE 9) GO TO 510
   410 READ (INP, 400A) EDFILE
       WRITE (1, 410A0) EDFILE
   410A FORMAT (4(4A2))
   IF (EDFILE(1, E0, -1) GO TO 450
   4000 FORMAT (4(4A2))
   DO 430 I=1,80
      BYTCT=BYTCT+1
      ITS=IGBYT(EDFILE, I-1)
      IT3=IT3+1
      IF (SVCH EQ 0) GO TO 420
      IF (IT1 EQ SVCH) GO TO 415
      IF (SVCH EQ -1) GO TO 415
      IEDC=IEDC(SVCH)
      IF (IEDC EQ -1) GO TO 990
      ISBYC=BYTCT-1
   412 CONTINUE
C 412   WRITE (8, 420A) IT4, ISBYC, IEDC, IT1, SVCH, STW, IT3
4200 FORMAT (1201A0)
       CALL IEDEP (IT2, IT4, ISBYC, IEDC, NFLCH)
       IF (IT4 EQ 1) GO TO 413
       CALL DSKFL (2, IINF, IOUT, STW, 1, IT2, IER)
       IF (IER NE 1 AND IER NE 9) GO TO 900
       STW=STW+1
       IF (IT4 EQ 2) GO TO 412
       IT4=0
       BYTCT=1
   413   SVCH=IT1
   415   GO TO 430
   420   IEDC=15
       ISBYC=1
   C
       WRITE (8, 420A) IT4, ISBYC, IEDC, IT1, SVCH, STW
       CALL IEDEP (IT2, IT4, ISBYC, IEDC, NFLCH)
       CALL DSKFL (2, IINF, IOUT, STW, 1, IT2, IER)
       IF (IER NE 1 AND IER NE 9) GO TO 900
       EDCT=EDCT+1
STWD=STWD+1
CALL DSKFL (2, IBUF1, IED, EDCT, 1, STWD-EDWD, IEP)
IF (IER NE 1) GO TO 910
IT4=0
CALL DSKFL (2, IREF, IOUT, IT5, 1, NFLCH, IEP)
IF (IER NE 1 AND IER NE 9) GO TO 900
IT5=STWD
STWD=STWD+1
NFLCH=0
IF (IT1 EQ 00T) GO TO 410
SVCT=IT1
BVTCT=1
430 CONTINUE
C 430 WRITE (0, 4200) IT3, BVTCT, IEPVC
GO TO 410
450 CONTINUE
STWD=STWD-1
EDCT=EDCT-1
IT1=NUMB(EDFIL(11), 5, 0)
IT2=NUMB(EDFIL(21), 5, 0)
IF (IT1+IT2 EQ 0) GO TO 800
IF (EDCT NE IT1) GO TO 940
IF (IT3 NE IT2) GO TO 940
EDCT=STWD+1-EDWD
GO TO 800

DATA NAME PROCESSING SECTION

INPUT IS IN 8A COL CARD FORMAT. THERE ARE 4 20 COLUMN ENTRIES PER CARD.

THE FORMAT FOR EACH ENTRY IS: COL 1-5 DATA STARTING LOC
6-7 DATA TYPE
8-11 DATA LENGTH
12 LEVEL
13 SYNC
14 BLANK
15-17 OCCURS
18-20 EDIT POINTER - IF DATA TYPE 4 OR 5 INPUT IS AN ENTRY # IN THE EDIT INDEX FILE (CHANNEL IED) WHICH CONTAINS THE ACTUAL POINTER
21-22 DIGITS TO RIGHT OF DECIMAL POINT FOR TYPES 8, 1

THE FILE TERMINATOR IS A CARD WITH COL 1-5 BLANK
23-27 # ENTRIES
45-49 # CHAR PAM STORAGE

OUTPUT IS A 5 WORD ENTRY OF THE FOLLOWING FORM:

WORD 1 DATA PTR
2 BLANK R4, SYNC B5, LEVEL B6-7, SIGN B11-12, TYPE B13-15
3 LENGTH
4 DECIMAL PTR B7, # OCCURS
5 EDIT POINTER

500 CONTINUE
X WRITE (12, 4300) (EDNAME(I), I=1, FCNT)
X4300 FORMAT (18X10)
DNW=STWD
DNCT=0

87
DST=0
DISP=0
SDFL1=1
IT4=0

510 READ (HUP, 9990) DFIL
C
WRITE (12, 51000) DFIL
IF (DFIL(1,1) EQ 9999) GO TO 530
DO 520 I=1,3
IF (DFIL(1, I) EQ 9999) GO TO 510
WRITE (12, 4200) (DFIL(I, J), J=1, 9)
C
IF (DFIL(4, I) NE 1) DFIL(1, I)=DFIL(1, I)+DISP
IT1=DFIL(2, I)/10
IT2=DFIL(2, I)-10+IT1
IF (IT2 LE 3 OR IT2 GE 6) GO TO 512
CALL DSKFL (1, IBD, IED, DFIL(8, I), 1, DFIL(8, I), IED)
C
WRITE (1, 4200) IED, DFIL(8, I), 1
IF (IED NE 3) GO TO 510
512 DFIL(2, I)=IT2*8*(32*(2+DFIL(6, I)+DFIL(5, I))+DFIL(4, I)+IT1)
C
WRITE (1, 4200) DFIL(2, I), DFIL(4, I), DFIL(5, I), DFIL(6, I), DFIL(2, I)
IF (DFIL(4, I) NE 2 AND IT2 NE 0) GO TO 514B
IT3=987
IF (IT2 EQ 0) GO TO 513
IF (SDFL1 EQ DFIL(1, I)) GO TO 514A
IF (DFIL=DST+DISP NE DFIL(1, I)) WRITE (12, 4200) IT3, DST+DISP, DFIL(1, I)
DST=DFIL+DISP
DFIL(1, I)=DFIL(1, I)+IT4
IT4=0
GO TO 514
513 DFIL(1, I)=DST+DISP
C
*****ALGORITHM
C
514 DISP=DISP+10
IF (MOD(DFIL(1, I), 2) EQ 0) GO TO 5142
DISP=DISP+1
DFIL(1, I)=DFIL(1, I)+1
5142 DO 5145 J=1,200
IF (FDNAM(J) EQ 0) GOTO 514A
IF (FDNAM(J) EQ DCNT+1) GO TO 5147
5145 CONTINUE
GO TO 5148
5147 IT4=1
5148 SDFL1=DFIL(1, I)
5149 DCNT=DCNT+1
515 DFIL(4, I)=DFIL(7, I)+DFIL(9, I)*2048
C
DFIL(5, I)=DFIL(8, I)
CALL DSKFL (2, IBD, IED, STD, 5, DFIL(1, I), IED)
C
IF (IED NE 1 AND IED NE 9) GO TO 900
STD=STD+5
520 CONTINUE
GO TO 518
530 CONTINUE
5000 FORMAT (3(I5, 12, I4, 3I1, 2I3, 12))
5100 FORMAT (3(I5, 12, I5, 3I5, 2I3, 12))
DST=DST+1
IF (DST NE DFIL(1, I)) GO TO 950
IF (DCNT NE DFIL(1, I)) GO TO 950
GO TO 808
C
C
VALUES PROCESSING SECTION
C
INPUT IS IN 80 COL CARD FORMAT WITH A CONTINUOUS CHARACTER STRING
EACH VALUE IS TERMINATED BY A ' ' AND '' ALSO ENDS A DATA CARD
THE FILE TERMINATOR IS A CARD WITH
COL 1-2 BLANK
21-25 # VALUES
41-45 # VALUES CHARS
THE FIRST 4 CHAR OF EACH VALUE THE ASSOC D-N FILE ENTRY #
OUTPUT IS THE TOTAL RAW STORAGE FILE ZERODED OUT WITH THE VALUES
INserted at the appropriate locations

600 CONTINUE
IT3=0
IT2='##'
CALL DSKFL (4, IIBU, IOUT, STWD, (DST+DISP+1)/2, IT2, IER)
IF (IER NE 1 AND IER NE 9) GO TO 600
IT2=0
IT1=0
WIN=STWD
WCNT=0
BYTCT=0
READ (INP, 4000) VFL
IT5=NUMB(VFL, 5, 10)
610 READ (INP, 4000) VFL
X WRITE (12, 4100) VFL
DO 690 I=1, 80
IF (IT1.EQ.-2) GO TO 630
IF (IT3.GT.0) GOTO 640
IT2=IT2+7
IF (I.LT.74) GO TO 620
J=I-74
K=7-J
IT3=0
L=0
IF (K.LE.4) GO TO 615
L=K-4
K=4
IT3=NUMB(VFL, L, 1+3)
615 IT4=NUMB(VFL, K, 1-1)
IG=101
X WRITE (12, 6000) IG, I, IT1, IT2, IT3, IT4, IT5, DNE, BYTCT, J, K, L
I=80
6000 FORMAT (13I10)
IT1=2
GO TO 650
620 CONTINUE
X WRITE (12, 4100) VFL
X IT4=NUMB(VFL,4,1-1)
X IT3=NUMB(VFL,3,1+3)+1
IG=02
X WRITE (12, 6000) IG, I, IT1, IT2, IT3, IT5, DNE, BYTCT
I=I+6
IT1=0
GO TO 670
630 I=J
IF (K.EQ.4) GO TO 633
J=J-3
IT3=NUMB(VFL,3,3)+1
IT4=IT4+10**3+NUMB(VFL, J, 0)
GO TO 626
633 IT3=IT3+10**3+NUMB(VFL, J, 0)+1
IG=103
IT1=0
WRITE (12, 6000) IG, IT1, IT2, IT3, IT4, IT5, DNE, BVTCT, J, K, L
GO TO 670
IT4=IT4+1
IG=106
WRITE (12, 6000) IG, IT1, IT2, IT3, IT4, IT5, DNE, BVTCT
CALL DSKFL (2, IBUF, IOUT, BVTCT, 1, IT4, IER)
IF (IER NE 1 AND IER NE 9) GO TO 910
BVTCT=BVTCT+1
GO TO 680
DNE=DNE+(IT4-1)*5
IG=107
WRITE (12, 6000) IG, I, IT1, IT2, IT3, IT4, ITK, DNE, J, K, L
CALL DSKFL (1, IBUF, IOUT, DNE, 1, IT4, IER)
IF (IER NE 1) GO TO 900
IG=104
WRITE (12, 6000) IG, I, IT1, IT2, IT3, IT4, IT5, DNE, BVTCT
BVTCT=IT4+VHD*2
IG=105
WRITE (12, 6000) IG, I, IT1, IT2, IT3, IT5, DNE, BVTCT
IT3=IT3+1
IF (IT3 GT 0) GO TO 690
IG=108
WRITE (12, 6000) IG
WRITE (12, 6000) IG, I, IT1, IT2, IT3, IT4, IT5, VCNT
WRITE (12, 6000) IG
VCNT=VCNT+1
IF (VCNT.EQ.10) GOTO 695
CONTINUE
GO TO 610
READ (INP, 4000) YFIL
IF (IT2+1 NE NUMA(YFIL(21),5,0)) GO TO 960
IF (VCNT NE NUMA(YFIL(11),5,0)) GO TO 960
VCNT=(DPSI+DISP+1)/2
STWD=VHD+VCNT
GO TO 800

C
C QUINT PROCESSING SECTION
C
C INPUT IS IN A00 COL CARD FORMAT THERE ARE 4 20 COLUMN QUNTS ON EACH CARD A QUINT IS OF THE FORM:
C COL 1-4 OPCODE
C 5-8 OPEPAND 1
C 9-12 OPEPAND 2
C 13-16 OPEPAND 3
C 17-20 SRCF SEQUENCE #
C THE FILE TERMINATOR IS A CARD WITH
C COL 1-4 BLANK
C 21-25 # QUNTS
C
C OUTPUT IS FIVE WORD ENTRY. THE STARTING WORD OF FILE IS WORD 1 OF THE DIRECTORY. THE NUMBER OF ENTRIES IS WORD 2 OF DIRECTORY. THE ENTRY FORMAT IS:
C WORD 1 OPCODE
C 2 OPEPAND 1
C 3 OPEPAND 2
C 4 OPEPAND 3
C 5 SEQ #
C

700 CONTINUE
CCNT=IP
END=IP
710 READ (INP,7000) QUINT
7000 FORMAT (2I14)
IF (QUINT(1,1).EQ.0) GO TO 730
DO 720 I=1,4
IF (QUINT(1, I).EQ.0) GO TO 710
IT1=QUINT(2, I)/10
IT2=QUINT(2, I)-IT1*10
QUINT(2, I)=IT1*IP+IT2
CALL DSKFL (I, IRUF, IOUT, STD, 5, QUINT(1, I), IER)
IF (IER NE 1 AND IER NE 9) GO TO 900
STD=STD+5
QCNT=QCNT+1
WRITE (1, 7590) QCNT, (QUINT(J, 1), J=1,5), QCNT, (QUINT(J, 1), J=1,5)
7500 FORMAT (618,15X,6018)
720 CONTINUE
GO TO 710
730 CONTINUE
IF (QUINT(1, 2)*10+QUINT(2, 2)/1000.0 NE QCNT) GO TO 970
GO TO 800
750 STD=1
760 CALL DSKFL (1, IRUF, IOUT, STD, 1, IT1, IER)
IF (IER NE 1 AND IER NE 9) GO TO 900
WRITE (1, 6500) IRUF, STD
6500 FORMAT (18/(1X,150I7))
IF (IER NE 0) GO TO 800
STD=STD+256
GO TO 760
800 CONTINUE
ICNT=ICNT+6
FCNT=FCNT+5
EDCT=EDCT-1
QCNT=QCNT+5
CALL DSKFL (2, IRUF, IOUT, 1, 11, ICNTC, IER)
IF (IER NE 1 AND IER NE 9) GO TO 900
CALL DSKFL (3, IRUF, IOUT, IT1, IT1, IT1, IER)
IF (IER NE 1 AND IER NE 9) GO TO 900
C WRITE (0,8000) IRUF
8000 FORMAT (18/(1X,150I7))
STOP
800 WRITE (LP,8999) IER
8999 FORMAT ('DISK ERROR ON INOUT FILE: ', I6)
STOP
900 WRITE (LP,9000) IER
9000 FORMAT ('DISK ERROR ON SAVE FILE: ', I6)
STOP
910 WRITE (LP,9100) IER
9100 FORMAT ('DISK ERROR ON TEMP FILE: ', I6)
STOP
920 WRITE (LP,9200) ICNT
9200 FORMAT ('FILE ERROR, COUNT: ', I6)
STOP
930 WRITE (LP,9300) FCNT, FAUF
9300 FORMAT ('FILE ERROR, COUNT: ', I6, ', BUFFER: ', I6)
STOP
940 WRITE (LP,9400) EDCT, IT3, IT1, IT2
9400 FORMAT ('EDIT FILE ERROR. COUNT: ', I6, ' CHARS: ', I6)
STOP
950 WRITE (LP, 9500) DNT, DFRST, DFIL(1,1), DFIL(1,2)
9500 FORMAT ('DN FILE ERROR. COUNT: ', I6, ' STORAGE: ', I6)
STOP
C 955 WRITE (LP, 9550) I, DFRST, DISP, DFIL(1,1)
C 9550 FORMAT ('DN ENTRY ', I6, ' BAD. LENGTH PARAMETERS: ', I6)
C STOP
960 WRITE (LP, 9600) VCNT, IT2
9600 FORMAT ('VALUES FILE ERROR. COUNT: ', I6, ' CHARS: ', I6)
STOP
970 WRITE (LP, 9700) DNT
9700 FORMAT ('OINT FILE ERROR. COUNT: ', I6)
STOP
980 WRITE (LP, 9800) SUCH
9800 FORMAT ('ILLEGAL EDIT CHARACTER: ', A2)
STOP
END
SUBROUTINE DSKFL (IFCN, IBUF, IO, STWD, ICT, IVEC, IER)
DIMENSION IBUF(256), IVEC(1)
INTEGER STWD, CNT, SECTR, WORD, DISP

IF IFCN FUNCTION. 1 READ, 2 WRITE, 3 SAVE, 4 WRITE CONSTANT
IBUF DISK BUFFER, 256 DATA WORDS. WORD 257 IS SECTOR #. WORD 258 IS
WRITE FLAG -1 MEANS UPDATED BUT NOT ON DISK, 0 IS CLEAN
IO CHANNEL
STWD STARTING WORD OF DISK FILE
CNT # OF WORD TO PROCESS
IVEC DATA VECTOR TO PROCESS
IER ERROR FLAG

IF (IFCN EQ 2 OR IFCN EQ 4) WRITE (12, 6000) (IVEC(1), I=1, ICT)
CNT=ICT
SECTR=(STWD-1)/256
DISP=SECTR*256-STWD+1
WORD=1-DISP
IER=1
IGI=1
WRITE (0, 5000) IGI, IBUF(257), IBUF(258), LIM, WORD, CNT, DISP, IO, STWD, ICT, SEC
GO TO (100, 200, 300, 400), IFCN

READ

100 IF (SECTR EQ IBUF(257)) GO TO 120
IF (IBUF(258) GE 0) GO TO 110
CALL WRBLK (IO, IBUF(257), IBUF(1), IER)
IF (IER NE 1 AND IER NE 9) GO TO 500
IGI=1
WRITE (12, 6000) IBUF, SECTR, IGI, STWD, ICT, DISP, WORD, CNT
110 CALL RDRLK (IO, SECTR, IBUF, 1, IER)
IF (IER NE 1) GO TO 500
IF (LIM LE 256) GO TO 130
LIM=LIM-256
LIM=256
IGI=2
WRITE (0, 5000) IGI, IBUF(257), IBUF(258), LIM, WORD, CNT, DISP, IO, STWD, ICT, SEC
GO TO 140

130 CNT=0
IGI=3
WRITE (0, 5000) IGI, IBUF(257), IBUF(258), LIM, WORD, CNT, DISP, IO, STWD, ICT, SEC
DO 150 I=1, LIM
150 IVEC(I+DISP)=IBUF(I)
IF (CNT EQ 0) GO TO 180
IF (DISP GT 0) GO TO 170
DISP=LIM-WORD+1

160 WORD=1
SECTR=SECTR+1
CALL WRBLK (IO, SECTR, IBUF(1), IER)
IF (IER NE 1) GO TO 500
IF (CNT LE 256) GO TO 120
LIM=256
CNT=CNT-256
IGI=4
WRITE (0, 5000) IGI, IBUF(257), IBUF(258), LIM, WORD, CNT, DISP, IO, STWD, ICT, SEC
GO TO 140
170 DISP=DISP+256
GO TO 160
180  IBUF(257)=SECTR
     IBUF(258)=0
     GO TO 600
C
C

200  IF (SECTR .EQ. IBUF(257)) GO TO 220
     IF (IBUF(258) .GE. 0) GO TO 210
     CALL WRALK (IO, IBUF(257), IBUF(258), IER, NE, 1, IEP)
     IF (IER .NE. 1 AND IER .NE. 9) GO TO 500
     IGI=2
     X
     WRITE (12, 6000) IBUF, SECTR, IGI, STWD, ICT, DISP, WORD, CNT
     IGI=5
C
     WRITE (6) IGI, IBUF(257), IBUF(258), Lim, WORD, CNT, DISP, IO, STWD, ICT, SEC
     CALL RDALK (IO, SECTR, IBUF(258), IEP)
     IF (IER .NE. 1 AND IER .NE. 9) GO TO 500
     IGI=6
C
     WRITE (6) IGI, IBUF(257), IBUF(258), Lim, WORD, CNT, DISP, IO, STWD, ICT, SEC
     Lim=WORD+CNT-1
     IF (Lim .LE. 255) GO TO 230
     CNT=Lim-256
     Lim=256
     IGI=7
C
     WRITE (6) IGI, IBUF(257), IBUF(258), Lim, WORD, CNT, DISP, IO, STWD, ICT, SEC
     GO TO 240
     CNT=0
     IGI=8
C
     WRITE (6) IGI, IBUF(257), IBUF(258), Lim, WORD, CNT, DISP, IO, STWD, ICT, SEC
     GO TO 240
     IBUF(I)=IVEC(I+DISP)
     IF (CNT .EQ. 0) GO TO 280
     IF (DISP .GT. 0) GO TO 270
     DISP=Lim-WORD+1
     260  WORD=1
     CALL WRALK (IO, SECTR, IBUF(257), IEP)
     IF (IER .NE. 1 AND IER .NE. 9) GO TO 500
     IGI=3
     X
     WRITE (12, 6000) IBUF, SECTR, IGI, STWD, ICT, DISP, WORD, CNT
     IGI=9
C
     WRITE (6) IGI, IBUF(257), IBUF(258), Lim, WORD, CNT, DISP, IO, STWD, ICT, SEC
     SECTR=SECTR+1
     CALL RDALK (IO, SECTR, IBUF(258), IEP)
     IF (IER .NE. 1 AND IER .NE. 9) GO TO 500
     IF (CNT .LE. 255) GO TO 220
     Lim=256
     CNT=CNT-256
     IGI=10
C
     WRITE (6) IGI, IBUF(257), IBUF(258), Lim, WORD, CNT, DISP, IO, STWD, ICT, SEC
     GO TO 240
     DISP=DISP+256
     GO TO 260
     260  IBUF(257)=SECTR
          IBUF(258)=1
          GO TO 600
C
C
C

300  CALL WRALK (IO, IABS(IBUF(257)), IBUF(258), IEP)
     IF (IER .NE. 1 AND IER .NE. 9) GO TO 500
     IGI=11
C
     WRITE (6) IGI, IBUF(257), IBUF(258), Lim, WORD, CNT, DISP, IO, STWD, ICT, SEC
X
WRITE (12, 6000) IBUF, SECTR, IGI, STWD, ICT, DISP, WORD, CNT
IBUF(258) = 0
GO TO 600
C
C ZERO
C
400 IV = IVEC(1)
IF (SECTR .EQ. IBUF(257)) GO TO 420
IF (IBUF(258) .GE. 0) GO TO 410
CALL WRAK (10, IBUF, 257), IBUF, 1, IEP
IF (IER .NE. 1 .AND. IER .NE. 9) GO TO 500
IGI = 5
X
WRITE (12, 6000) IBUF, SECTR, IGI, STWD, ICT, DISP, WORD, CNT
CALL RDNLK (10, SECTR, IBUF, 1, IEP)
IF (IER .NE. 1 .AND. IER .NE. 9) GO TO 500
420 LIM = WORD + CNT - 1
IF (LIM .LE. 255) GO TO 430
CNT = LIM - 256
LIM = 256
IGI = 12
C
WRITE (0, 5000) IGI, IBUF(257), IBUF(258), LIM, WORD, CNT, DISP, IO, STWD, ICT, SEC
GO TO 440
430 CNT = 0
IGI = 13
C
WRITE (0, 5000) IGI, IBUF(257), IBUF(258), LIM, WORD, CNT, DISP, IO, STWD, ICT, SEC
440 DO 450 IMM = WORD, LIM
450 IBUF(1) = IV
IF (CNT .LE. 0) GO TO 460
IF (DISP .GT. 0) GO TO 470
DISP = LIM - WORD + 1
460 LIM = 1
CALL WRAK (10, SECTR, IBUF, 1, IEP)
IF (IER .NE. 1 .AND. IER .NE. 9) GO TO 500
IGI = 6
X
WRITE (12, 6000) IBUF, SECTR, IGI, STWD, ICT, DISP, WORD, CNT
SECTR = SECTR + 1
CALL RDNLK (10, SECTR, IBUF, 1, IEP)
IF (IER .NE. 1 .AND. IER .NE. 9) GO TO 500
IF (CNT .LE. 255) GO TO 420
LIM = 256
CNT = CNT - 256
IGI = 14
C
WRITE (0, 5000) IGI, IBUF(257), IBUF(258), LIM, WORD, CNT, DISP, IO, STWD, ICT, SEC
GO TO 440
470 DISP = DISP + 256
GO TO 460
480 IBUF(257) = SECTR
IBUF(258) = 1
IGI = 15
C
WRITE (0, 5000) IGI, IBUF(257), IBUF(258), LIM, WORD, CNT, DISP, IO, STWD, ICT, SEC
GO TO 600
C
C ERROR
C
500 IBUF(257) = 5555
IBUF(258) = 0
IGI = 16
C
WRITE (0, 5000) IGI, IBUF(257), IBUF(258), LIM, WORD, CNT, DISP, IO, STWD, ICT, SEC
C600 IF (IER .EQ. 1) WRITE (12, 6000) (IVEC(I), I=1, ICT)
600 CONTINUE
5000 FORMAT (i2,i10)
6000 FORMAT (i8,i1X,150I7)
RETURN
END
SUBROUTINE IEDPK (I, J, K, L, M)

C INPUT IS:
C J BYTE PTR FOR OUTPUT WORD
C K REPEITION COUNT
C L EDIT CODE
C M TOTAL # FILL CHAR (Z,9,X) -1 (-,0)
C
C OUTPUT IS PACKED EDIT CODE IN I IF J IS A OUTPUT IS IN LEFT BYTE
C IF J IS 1 LEFT BYTE HAS BEEN PACKED WITH SHORT FORM AND OUTPUT
C IS ADDED INTO RIGHT BYTE
C IF K > 0 OUTPUT IS A FULL WORD
C
C CONTINUE
A LDA 2,8T. +3,3
A ADDL 2,2
A MOVZL 2,2
A LDA 0,8T. +2,3
A ADC 1,1
A ADD 1,0
A DECRIMENT REPEITION COUNT
A MOVZR 0,1
A MOVZR 1,1
A MOVZR 1,1,5ZR
A LONG FORM?
A JMP IED2
A YES
A ADD 2,0
A COMBINE EDIT CODE & REPEITION COUNT
A LDA 1,8T. +1,3
A MOVZR 1,1,5ZR
A LEFT BYTE PACKED?
A JMP IED1
A YES
A ISZ @T. +1,3
A MOVLS 0,0
A JMP IED5
A IED1:
A LDA 1,8T. +0,3
A ADD 1,0
A ADD TO LAST CHAR
A JMP IED4
A IED2:
A LDA 1,8T. +1,3
A MOVZR 1,1,5NC
A LEFT BYTE PACKED?
A JMP IED3
A NO
A ISZ @T. +1,3
A SET FLAG TO 2 SO LEFT BYTE WILL BE STORED
A SURZR 1,1
A MOVSR 1,1
A LDA 0,8T. +0,3
A ADD 1,0
A SET BIT 8 TO 1 TO INDICATE LONG FORM NEXT
A JMP IED5
A IED3:
A MOVSR 2,2
A ADD 2,0
A ADD EDIT CODE TO REPEITION COUNT
A SURZR 1,1
A ADD 1,0
A SET BIT 0 TO 1 FOR LONG FORM
A JMP IED4
A IED4:
A SUR 1,1
A STA 1,8T. +1,3
A IED5:
A STA 0,8T. +0,3
A STORE PACKED WORD
C IF (L GT 12) RETURN
C IF (L LT 11 AND L NE 8) RETURN
C M=M+K
C IF (L EQ 8 OR L EQ 11) M=M-1
C RETURN
END
FUNCTION IGBVT (ARRAY, BYTPTR)

INPUT IS AN ARRAY AND POINTER TO THE BYTE WANTED (0 IS 1ST BYTE)

OUTPUT IS LEFT JUSTIFIED BYTE WITH RIGHT BLANK (A1 FORMAT)

CONTINUE

A LDA 2.T +1.3
A LDA 1.0T +2.3
A MOVZR 1.1
A ADD 1.2
A LDA 0.0.2
A MOV 0.0.5ZC
A MOV5 0.0
A LDA 1.0SK
A AND 0.1
A LDA 0.140
A ADD 1.0
A STA 0.0T +0.3
A JMP .+3

AMSK: 177400
AI40: 40

RETURN
END
FUNCTION NUMA (ARRAY, NCHAR, BYTPTR)

INPUT IS AN ARRAY, THE NUMBER OF ASCII CHAR TO BE CONVETED TO A
BINARY NUMBER AND A BYTE POINTER TO 1ST CHAR (0 = 1ST BYTE OF ARR)

OUTPUT IS BINARY #, OVER FLOW, VALIDITY OF DIGITS NOT CHECKED

CONTINUE

A LDA 0,0T.+2,3          J2ND ARG VAL
A STA 0,CTR
A LDA 0,0T.+3,3          J3RD ARG VAL
A LDA 1,T.+1,3           J1ST ARG ADDR
A MOVZL 1,1
A ADD 0,1               JCORE BYTE PTR 1ST CHAR
A STA 1,BYTPTR
A SUB 0,0
A STA 0,0T.+0,3          JZERO NUMB
A ANMB1: LDA 2,BYTPTR
A LDA 1,0T.+0,3
A MOVZL 1,0
A MOVZL 0,0
A ADDZL 0,1
A MOVZ 2,2               JNUMB * 10
A LDA 0,0,2
A LDA 2,MSK
A MOV 2,2,SNC
A MOVS 0,0               JSWAP LEFT BYTE TO RIGHT
A AND 0,2
A ADD 2,1               JADD NEW DIGIT
A STA 1,0T.+0,3          JSTORE IN NUMB
A ISZ BYTPTR
A DSZ CTR
A JMP NMB1
A JMP +4
A AMSK: 17
A ACTR: .BLK 1
A ABYTPTR: .BLK 1
C RETURN
C END
FUNCTION IEDCD (ICHR)

THE EDIT CODE FORMAT IS

CODE  CHAR
0  NULL
1  B
2  / 
3  .
4  D (OF DB)
5  C (OF CR)
6  7
8  $ 
9  - 
10  + 
11  =
12  Z
13  9 OR X
14  V
15  EOS (NOT IN INPUT STRING)

INTEGER ICOR(16)
COMMON/DTPN/ICOR
DATA ICOR/32,'A ','/ ', '0 ' ', 'D ','C ', ' $ ', ' - ', '+ ', ' = ', 'Z ', '9 ', 'X ', 'V ' 
DO 10 I=1,16
   IF (ICHR.EQ. ICOR(I)) GO TO 20
10 CONTINUE
   IEDCD=-1
   RETURN
20 IEDCD=I-1
   IF (IEDCD. GE 14) IEDCD=IEDCD-1
   RETURN
END
SUBROUTINE PUTBT (ARRAY, CHAR, BVTPTR)

INPUT IS A LEFT JUSTIFIED BYTE AND POINTER TO THE BYTE WANTED (A-1ST)

OUTPUT IS THE BYTE PACKED IN THE ARRAY/CHARACTER STRING AT BVTPTR

CONTINUE

A LDA 2. T. +0. 3
A LDA 1. @T. +2. 3
A MOVZR 1. 1
A ADD 1. 2
A LDA 1. @T. +1. 3
A LDA 0. MSK
A AND 0. 1
A STA 1. SVCHR
A MOV 1. 1. SZC
A JMP PUT1
A MOV 0. 8
A LDA 1. 0. 2
A AND 0. 1
A LDA 0. SVCHR
A ADD 0. 1
A JMP PUT2

APUT1: LDA 1. 0. 2
A AND 0. 1
A LDA 0. SVCHR
A MOVS 0. 0
A ADD 0. 1
APUT2: STA 1. 0. 2
A JMP +3

A MSK: 177400
A SVCHR: . BLK 1

C RETURN
C END
SUBROUTINE DSKRT (IFCN, IBUF, IO, BTPTR, BTCT, BYTE, IER)
DIMENSION IBUF(257)
INTEGER STDH, BTCT, BTPTR, BYTE
IF (BTCT .EQ. 1 AND IFCN .EQ. 2) GO TO 200
IER=-1
RETURN
200
STDH=(BTPTR)/2
X
WRITE (12,1000) BTPTR, BTCT, BYTE, STDH
CALL DSKFL (1, IBUF, IO, STDH, BTCT, IT1, IER)
IF (IER NE 1 AND IER NE 9) RETURN
IT2=MOD(BTPTR,2)
IG=100
X
WRITE(12,1000) IG, BTPTR, BYTE, IT1, IT2, STDH
1000
FORMAT (100110)
CALL PUTBT (IT1, BYTE, IT2)
IG=109
X
WRITE(12,1000) IG, BTPTR, BYTE, IT1, IT2, STDH
CALL DSKFL (2, IBUF, IO, STDH, BTCT, IT1, IER)
RETURN
END
.TITL  CLOSE
.ENT  CLOS

CLOS:
ISZ  RTST, 2  ; GET NEXT STORE POSITION
STA  3, @RTST, 2  ; SAVE RETURN ADDRESS
LDA  0, @DP, 2  ; LOAD THE BYTE POINTER
LDA  2, @CHNL, 2  ; LOAD THE CHANNEL NUMBER

.SYST
CLOSE  77

JMP  @RTER, 2  ; BAD CLOSE JUMP TO ERROR
MOV  3, 2

JMP  @RTN, 2  ; PROCESSING COMPLETE, RETURN
.TITLE COBOL
.EXTN 1
.EXTN DATA, CALOR, DELET

COBOL:
LDA 2, DATAP
STA 2, US3
LDA 0, TTOP, 2
JSR @OPAUP, 2
JSR ERROR
MOV 1, 2
LDA 0, CMSG

SVSTM
WRL 77
JWRITE COBOL
JRSR ERROR
LDA 2, CHNUM, 3

SVSTM
CLOSE 77
JCLOSE TTO
JRSR ERROR
MOV 3, 2

CB1:
LDA 0, ASTRK

SVSTM
PCCHAR
JRSR ERROR
MOV 3, 2
SUB 3, 3
STA 3, CERFL, 2
JSR CL1
JMP .+3
JMP CB2
JMP CB1
LDA 3, CERFL, 2
MOV 3, 3, SNR
JSR ERROR
JSR PREPR

PREPR:
LDA 0, TTOP, 2
JSR @OPAUP, 2
JOPEN TTO
JSR ERROR
MOV 1, 2
LDA 0, ILMG

SVSTM
WRL 77
JWRITE ILLEGAL COMMAND
JRSR ERROR
LDA 2, CHNUM, 3

SVSTM
CLOSE 77
JCLOSE TTO
JRSR ERROR
LDA 2, US3, 1

JRSR CB1

JUMP CB2

CL1:
ISZ PRTST, 2
STA 3, PRTST, 2
JSR @VALNP, 2
JMP ALFA
JSR @IMINP
JSR @RTLP, 2
ISZ @RTST.2
JSR @RTN.2
CMRG: +1*2
.TXT /COBOL VERSION 1.0 4-1-75<15>/
ILMSG: +1*2
.TXT /ILLEGAL COMMAND<15>/
ALFA: LDA 3,CDTA
ALFAI: LDA 1,0,3,COMMANDALFA:CHAR
       MOV 1,1,SNR;ENDTOTECL?JMP ALFA2
       SUB 0,1,SNR;THIS CHAR?JMP +3
       INC 3,3;NO
       JMP ALFAI
       JSR @DISPL.3 EXECUTE
       JSR @RTPT.2
       ISZ @RTST.2
       JSR @RTN.2
ALFA2: ISZ CERPFL.2
       JSR @RTPT.2
       TXTM 0
ASTRM: TXT */*
CDTA: +1
       TXTM 0
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXT /*/
       TXTM 0
       DISPL= 14
       CLG
       ENTER
       NEW
       RUN
       KILL
       BREAK
       DELET
       CONT
       LIST
       STFP
       CALDB
       IMINS
       IMINS
       DATA
       TXTM 1
       LDA 0,TTOP.2
       JSR @OPAVP.2
       JSR @RTPT.2
       MOV 1,2
       LDA 0,MESS
       SYSTEM
       .WRD 77
       JSR @RTPT.2
       MOV 3,2
       LDA 0,GC.2 GET QUINT COUNT
JSR @PROC, 2
JSR @STEP, 2 ; WRITE QUINT COUNT
LDA 0, FIVE, 2
STA 0, X4T, 2
AA:
LDA 3, OP, 2 ; GET QUINT POINTER
LDA 0, 0, 3 ; GET A WORD
JSR @PROC, 2 ; PUT THE WORD OUT TO TTI
JSR @STEP, 2
ISZ OP, 2
DSZ X4T, 2
JMP AA
JSR @PRCPRP, 2 ; PUT OUT A CARRAGE RETURN AND LINE FEED
JSR @STEP, 2
JSR PRERR
MESS: +1+2
.TXT "ERROR RETURN, ON QC# "
GDST
GRST
RDOE
PROCT
PRCPL
GTONT
FRONT
31
PCTM+2
SUBT
ZEROA
ADDIT
DIV10
MOVA
MUL10
CHAP
RTRN-1
177
DPDST
NNNCH
NNNCH
EXDNT
ERTRN
RETRN
TTOTP=2
VALDG
OPRVC
ALCOR
PRERR
PCTM
PCT
PCK3
PCK2
PCK1
17
71
177
11
377
100
"*/
3600
74000
7
170
200
147
000
","41104
51103
"-"
"*"
"*
12
10
11
40
6
DATA:
  .BLK  25
  RTRN-1
  .BLK  40
  NDPTI-1
  CNTI-1
  SAVEI-1
  RPTI-1
  .BLK  46
  +2
  12.
  .BLK  36
  RTRN:
  .BLK  20
  NDPTI:
  .BLK  20
  CNTI:
  .BLK  20
  SAVEI:
  .BLK  20
  RPTI:
  .BLK  20

IBUF= 0  ; POINTER TO INPUT BUFFER
DNPTI= 1  ; DATA NAME TABLE POINTERS
DNPT2= 2
DNPT3= 3
ECHT= 4  ; EDIT CHARACTER TEMPORARY
BTCNT= 5  ; BYTE COUNT IF ROUTINE
NFLAG= 6  ; NOT FLAG IF ROUTINE
CHKWD= 7  ; CHECK WORD
MM= 10  ; COUNTER MATH PACKAGE
BEFRL= 11  ; BEFORE LINE FLAG
SKPT= 12  ; FLAG WITHIN WRITE ROUTINE
ERCODE= 13  ; ERROR CODE SAVE AREA
SFLAG= 14  ; SIZE ERROR FLAG
CNT2= 15  ; TOTAL LENGTH COUNTER
SIGN= 16  ; SIGN CLAUSE
X9T= 17  ; TEMPORARY COUNTERS
X4T= 20
XIIT= 21
XLIT= 22
SAVE1= 23  ; SAVE AREAS FOR RETURN ADDRESS
BVPTP= 24
RTST= 25
SAV1= 26  ; COUNTER
SAV= 27  ; COUNTER
RTRN1= 30  ; COUNTER
SAV2= 31  ; COUNTER
SAV3= 32  ; COUNTER
STAV= 34
NUM= 35  ; COUNTER
I= 37  ; WORD COUNTER MATH PACKAGE
<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>NT=</td>
</tr>
<tr>
<td>41</td>
<td>PNT1=</td>
</tr>
<tr>
<td>42</td>
<td>PNT2=</td>
</tr>
<tr>
<td>43</td>
<td>ECCHAR=</td>
</tr>
<tr>
<td>44</td>
<td>ROCHK=</td>
</tr>
<tr>
<td>45</td>
<td>TADO=</td>
</tr>
<tr>
<td>47</td>
<td>POS=</td>
</tr>
<tr>
<td>50</td>
<td>INDIC=</td>
</tr>
<tr>
<td>51</td>
<td>RCNT=</td>
</tr>
<tr>
<td>52</td>
<td>ECNT=</td>
</tr>
<tr>
<td>53</td>
<td>CNT1=</td>
</tr>
<tr>
<td>54</td>
<td>EFLAG=</td>
</tr>
<tr>
<td>55</td>
<td>RDIG1T=</td>
</tr>
<tr>
<td>56</td>
<td>TYP=</td>
</tr>
<tr>
<td>57</td>
<td>ECODE=</td>
</tr>
<tr>
<td>60</td>
<td>DIGIT=</td>
</tr>
<tr>
<td>61</td>
<td>BEFPP=</td>
</tr>
<tr>
<td>62</td>
<td>CHANL=</td>
</tr>
<tr>
<td>63</td>
<td>QP=</td>
</tr>
<tr>
<td>64</td>
<td>QC=</td>
</tr>
<tr>
<td>65</td>
<td>SP=</td>
</tr>
<tr>
<td>66</td>
<td>NDPT=</td>
</tr>
<tr>
<td>67</td>
<td>CNT=</td>
</tr>
<tr>
<td>70</td>
<td>SAVE=</td>
</tr>
<tr>
<td>71</td>
<td>RTPT=</td>
</tr>
<tr>
<td>72</td>
<td>OP1DL=</td>
</tr>
<tr>
<td>73</td>
<td>OP1DP=</td>
</tr>
<tr>
<td>74</td>
<td>OP1DE=</td>
</tr>
<tr>
<td>75</td>
<td>OP1DX=</td>
</tr>
<tr>
<td>76</td>
<td>OP1DT=</td>
</tr>
<tr>
<td>77</td>
<td>DEC1=</td>
</tr>
<tr>
<td>100</td>
<td>OP2DL=</td>
</tr>
<tr>
<td>101</td>
<td>OP2DP=</td>
</tr>
<tr>
<td>102</td>
<td>OP2DE=</td>
</tr>
<tr>
<td>103</td>
<td>OP2DX=</td>
</tr>
<tr>
<td>104</td>
<td>OP2DT=</td>
</tr>
<tr>
<td>105</td>
<td>DEC2=</td>
</tr>
<tr>
<td>106</td>
<td>OP3DL=</td>
</tr>
<tr>
<td>107</td>
<td>OP3DP=</td>
</tr>
<tr>
<td>110</td>
<td>OP3DE=</td>
</tr>
<tr>
<td>111</td>
<td>OP3DX=</td>
</tr>
<tr>
<td>112</td>
<td>OP3DT=</td>
</tr>
<tr>
<td>113</td>
<td>DEC3=</td>
</tr>
<tr>
<td>114</td>
<td>PAC1=</td>
</tr>
<tr>
<td>115</td>
<td>PAC2=</td>
</tr>
<tr>
<td>116</td>
<td>PAC3=</td>
</tr>
<tr>
<td>117</td>
<td>PACT1=</td>
</tr>
<tr>
<td>120</td>
<td>PACT2=</td>
</tr>
<tr>
<td>121</td>
<td>PACT3=</td>
</tr>
<tr>
<td>122</td>
<td>M=</td>
</tr>
<tr>
<td>123</td>
<td>PAC2=</td>
</tr>
<tr>
<td>124</td>
<td>PAC3=</td>
</tr>
<tr>
<td>125</td>
<td>OPDL=</td>
</tr>
<tr>
<td>126</td>
<td>OPDP=</td>
</tr>
<tr>
<td>127</td>
<td>OPDE=</td>
</tr>
<tr>
<td>170</td>
<td>OPDT=</td>
</tr>
<tr>
<td>171</td>
<td>SHPT=</td>
</tr>
<tr>
<td>172</td>
<td>TOC=</td>
</tr>
<tr>
<td>133</td>
<td>NPFLG=</td>
</tr>
<tr>
<td>134</td>
<td>ARGDE=</td>
</tr>
</tbody>
</table>
CHAR1= 128  ; FIRST CHARACTER FROM ARGUMENT ONE
CHAR2= 136  ; FIRST CHARACTER FROM ARGUMENT TWO
TEMP= 137
IOPTR= 140
IOSTH= 141
IOCNT= 142
FDSTH= 143
FDCTH= 144
EDSTH= 145
EDCNT= 146
DNSTH= 147
DNCNT= 150
RSSTH= 151
RSCNT= 152
ONSTH= 153
ONCNT= 154
SVFCH= 155
CIFFL= 156
NEGNO= 160  ; NEGATIVE NUMBER FLAG
CHNUM= 161
NEWF= 162
SNMAX= 163
TEMP2= 164
OPFLG= 165
RSMX= 166
PNT0= 167  ; FLAG FOR ZERO OR BLANK FILL
CNT0= 170  ; NUMBER OF FILL CHARACTERS
IOST= 171
FDST= 172
EDST= 173
DNST= 174
RSST= 175
ONST= 176
STPFL= 177  ; JIBUF ARGUMENTS
IFCN= -1
IO= -2
STWDD= -3
ICT= -4
IVCE= -6
IER= -7
; END
.TITLE DECIMAL
.ENT FLDEC

FLDEC:  JSZ   RTST.2
       STA   3,RTST.2       ; SAVE RETURN ADDRESS
       LDA   1,DEC1.2       ; GET NUMBER OF DECIMAL POSITION
       MOV   1,SNR
       JSR   @RTN.2
       STA   1,SAV.2       ; SAVE IT
       LDA   1,PACK2.2
       STA   1,PACK1.2
       JSR   @MTEN.2
       JSR   @ORTER.2
       DSZ   SAV,2
       JMP   .-5
       JSR   @RTN.2
DSPLY:  
ISZ  PTST, 2 ;GET NEXT STORE POSITION  
STA  3, OP1DP, 2 ;STORE RETURN ADDRESS  
LDA  1, OP1DL, 2  
STA  1, OPDP, 2  
LDA  1, OPDL, 2  
STA  3, OPDP, 2 ;LOAD OPERAND BYTE POINTER  
STA  3, BVPTR, 2 ;SAVE BYTE POINTER  
JSR  @GET ;GET A CHARACTER  
JMP  @RTN, 2 ;MOVE CHARACTER TO AC0  
MOV  1, 0  

SYSTEM  
PCHAR  
JMP  @RTN, 2  
MOV  3, 2 ;RESTORE AC2  
ISZ  OPDP, 2 ;INCREMENT OPERAND BYTE POINTER  
DSZ  OPDL, 2 ;DECREMENT OPERAND LENGTH COUNTER  
JMP  DSPLY+6 ;RETURN TO PUT OUT NEXT CHARACTER  
LDA  0, C1, 2 ;LOAD CARRIAGE RETURN  

SYSTEM  
PCHAR  
JMP  @RTN, 2  
MOV  3, 2 ;RESTORE AC2  
LDA  0, LF, 2 ;LOAD LINE FEED CHARACTER  

SYSTEM  
PCHAR  
JMP  @RTN, 2 ;PROCESSING COMPLETED
CALL TO THIS ROUTINE IS (N, PACT1, PACT2, PACT3)

DIVDPK:
LDA 1, PACT1, 2 ;GET AN ADDRESS FOR 8 WORD DIVIDE
STA 1, N, 2 ;STORE IN N
LDA 1, PACT1, 2 ;INITIALIZE ADDRESS FOR
STA 1, PACT1, 2 ;DIVIDE ROUTINE
LDA 1, PACT2, 2
STA 1, PACT2, 2
LDA 1, PACT3, 2
STA 1, PACT3, 2

DIVDE:
ISZ RTST, 2 ;GET NEXT STORE POSITION
STA 1, PACT2, 2 ;SAVE RETURN ADDRESS
LDA 1, FOUND, 2 ;GET A FOUR
STA 1, X47, 2 ;INITIALIZE A COUNTER
LDA 1, N, 2 ;GET NUMBER OF WORDS LONG
MOVL 1, 1 ;MULTIPLY THE WORD
DSZ X47, 2 ;BY 16 TO GET PROPER
JMP .-2 ;VALUE FOR DIVISION
STA 1, M, 2 ;STORE FINAL VALUE IN M
LDA 1, PACT3, 2 ;ZERO OUT RECEIVING FIELD
STA 1, PACT2, 2
JSR @25A, 2 ;ZERO OUT RECEIVING FIELD
JSR @TER, 2
LDA 1, PACTM, 2 ;ZERO OUT TEMP FIELD
STA 1, PACT2, 2
JSR @25A, 2 ;GO TO ZERO-OUT ROUTINE
JSR @TER, 2
LDA 1, PACT2, 2 ;GET SECOND ARGUMENT
STA 1, PACT1, 2 ;FOR PROPER ADDRESS
STA 1, PACT2, 2 ;ALIGNMENT ON MOV LEFT ROUTINE
JSR @MOVL, 2 ;GO TO SHIFTL LEFT ROUTINE
JSR @TER, 2

DIVDE1:
LDA 1, PACT3, 2 ;INITIALIZE PACT3
STA 1, PACT1, 2 ;FOR PROPER SHIFT
STA 1, PACT2, 2 ;LEFT
JSR @MOVL, 2 ;JMP TO SHIFTL LEFT ROUTINE
JSR @TER, 2
SUBC 0, 0
MOVR 0, 0 ;GENERATE A ONE
STA 0, X47, 2 ;STORE IT
LDA 1, PACT1, 2 ;GET FIRST ARGUMENT
STA 1, PACT1, 2 ;FOR SUBTRACT ROUTINE
LDA 1, PACT3, 2 ;GET SECOND ARGUMENT
STA 1, PACT2, 2 ;FOR SUBTRACT ROUTINE
LDA 1, PACTM, 2
STA 1, PACT2, 2
JSR @SUBTL, 2 ;GO TO SUBTRACT ROUTINE
JSR @TER, 2
JSR @TER, 2
MOV 1, 1, 52C ;CHECK FOR OVERFLOW
JMP DIVDE3 ;CARRY
LDA 1, PACT1, 2 ;GET FIRST ARGUMENT
STA 1, PACT1, 2 ;FOR ADD ROUTINE
LDA 1, PACT3, 2 ;SECOND ARGUMENT
STA 1, PACT3, 2
LDA 1, PACTM, 2
STA 1, PACT3, 2

JSR @ADD.2  ; GO TO ADD ROUTINE
JSR @RETP.2
JSR AA
JSR @TER.2
LDA 1.X4T.2  ; GET FLAG FOR CARRY SET
MOVL 1.1  ; STORE IT SINCE IT SHOULD BE A SUB
DVDE3: LDA 1.PACT2.2  ; GET SECOND ARGUMENT
STA 1.PAC1.2  ; FOR PROPER ADDRESS
STA 1.PAC2.2  ; ALIGNMENT ON MOVE LEFT ROUTINE
JSR @MOVL.1  ; GO TO SHIFITLEFT ROUTINE
JSR @TER.2
DSZ M.2
JMP DVDE1
SUB 1.1
STA 1.SFLAG.2  ; EXIT ROUTINE
JMP @RTN.2
AA: LDA 1.PAC1.2  ; GET RECEIVING FIELD
STA 1.PAC1.2  ; SET ADDRESS FOR
LDA 1.PACT3.2  ; MATH-MOVE
STA 1.PAC2.2  ; ROUTINE
ISZ RTST.2  ; GET NEXT STORE POSITION
STA 3.RPST.2  ; SAVE RETURN ADDRESS
JSR @MPMOV.2  ; GO TO MATH-MOVE ROUTINE
JSR @TER.2
JMP @RTN.2  ; RETURN
MOVLL: MOZLF
MOVLL1: MOLF
.TITL DIVIDE10
.ENT DIV10
.NRFL

;THIS ROUTINE USES ONE PACK FIELD, ANSWER COMES BACK IN WORD SIX
;THE PACK FIELD TO BE USED BY THIS ROUTINE WILL BE PACK1.
DIV10:  JSR RTST.2  ;GET NEXT STUFF POSITION
        STA 3, @RTST.2  ;SAVE RETURN ADDRESS
        * (1/2***4)
        
        LDA 1, PAC1.2  ;GET FIELD TO BE DIVIDED
        LDA 0, N.2     ;GET N - 1
        NEG 0, 0
        CMP 0, 0
        STA 0, X4T.2
        ADD 0, 1
        STA 1, PAC3.2  ;STORE IT IN PAC3
        STA 1, X17T.2  ;PAC3+N-1
        STA 0, M.2     ;STORE IN COUNTER
        LDA 1, @PAC3.2  ;GET FIFTH WORD INFORMATION
        MOVZR 1, 1.1
        MOVZR 1, 1.1
        MOVZR 1, 1.1
        MOVZR 1, 1.1
        AA:
        DSZ PAC3.2     ;TO GET FOURTH WORD
        LDA 0, @PAC3.2  ;LOAD CONTENTS
        ISZ PAC3.2     ;PUT PAC3 TO FIFTH WORD
        SUB 3, 3       ;CLEAR AC1
        MOV R 0, 0     ;MOVE RIGHT FOUR TIMES
        MOV R 3, 3
        MOV R 0, 0     ;MOVE AC3 RIGHT FOUR TIMES
        MOV R 3, 3
        MOV R 0, 3     ;/ 16
        MOV R 3, 3     ;FRACTIONAL REMAINDER
        ADD 3, 1, S2C  ;"TWO WORD" / 16
        INC 0, 0
        STA 1, @PAC3.2  ;STORE IT BACKOUT
        DSZ PAC3.2     ;GET NEXT ADDRESS
        MOV 0, 1
        STORE AC0 INTO AC1
        DSZ M.2        ;DECREMENT COUNTER
        JMP AA          ;DO AGAIN
        STA 1, @PAC3.2
        * (1+1/2)
        
        LDA 1, X4T.2  ;GET N - 1
        LDA 0, X17T.2  ;GET PAC3+N - 1
        STA 0, PAC3.2
        STA 1, M.2
        LDA 1, @PAC3.2  ;GET DATA
        SUB 3, 3
        AB:
        DSZ PAC3.2     ;GET DATA
        LDA 0, @PAC3.2  ;GET DATA
        ISZ PAC3.2     ;GET DATA
        MOVZR 0, 0
        MOV R 1, 1     ;"TWO WORD" / 2
        ADD 3, 1
        ADD CARRY
        LDA 3, @PAC3.2  ;GET DATA
        ADD 3, 1
STA 1. @PAC3, 2
SUBCL 3.3
DSZ PAC3, 2
LDA 1. @PAC3, 2
DSZ M, 2
JMP AB
STA 3. TEMP, 2
MOVZR 1. 0
ADDZ 0. 1, S2C
ISZ TEMP, 2
STA 1. @PAC3, 2

* (1+1/2**4)

LDA 1. X4T, 2
STA 1. M, 2
LDA 0. X17T, 2
STA 0. PAC3, 2
LDA 1. @PAC3, 2
MOVZR 1. 1
MOVZR 1. 1
MOVZR 1. 1
MOVZR 1. 1
AC:
DSZ PAC3, 2
LDA 0. @PAC3, 2
ISZ PAC3, 2
SUBO 3. 3
MOV R 0. 0
MOV R 3. 3
MOV R 3. 3
MOV R 3. 3
MOV R 0. 0
MOV R 3. 3
ADDZ 3. 1, S2C
INC 0. 0
LDA 3. @PAC3, 2
ADDZ 3. 1, S2C
INC 0. 0
STA 1. @PAC3, 2
DSZ PAC3, 2
MOV 0. 1
DSZ M, 2
JMP AC
LDA 3. @PAC3, 2
ADDZ 3. 1, S2C
ISZ TEMP, 2
STA 1. @PAC3, 2

* (1+1/2**8)

LDA 0. X4T, 2
STA 0. M, 2
LDA 0. X17T, 2
STA 0. PAC3, 2
LDA 1. @PAC3, 2
MOV 1. 3
LDA 0. REMSK, 2
AND 3. 0
SUB 3. 3

118
AD:

ADD 0, A
ISZ TEMP, 2
DSZ PAC3, 2
LDA 0, APAC3, 2
ISZ PAC3, 2
ADD 3, 1, S2C
ISZ TEMP, 2
MOV 0, 0
LDA 3, MS256, 2
AND 0, 3
ADD 3, 1, S2C
ISZ TEMP, 2
STA 1, APAC3, 2
LDA 3, TEMP, 2
SUB 1, 1
STA 1, TEMP, 2
LDA 1, PBNSK, 2
AND 1, 0
SUB 1, 1
STA 1, TEMP, 2
LDA 1, APAC3, 2
DSZ PAC3, 2
LDA 1, APAC3, 2
DSZ M, 2
JMP AD
MOV 3, 3, S2R
ISZ TEMP, 2
ADD 0, 1, S2C
ISZ TEMP, 2
STA 1, APAC3, 2

AG:

LDA 1, X4T, 2
LDA 0, X17T, 2
STA 0, PAC3, 2
STA 1, M, 2
LDA 3, TEN, 2
LDA 1, M, 2
INC 1, 1
STA 1, NUM, 2
LDA 0, PAC3, 2
STA 0, X17T, 2
MOV 3, 1
SUB 3, 3
LDA 0, 0X17T, 2
ADD 0, 1, S2C
INC 3, 3
DSZ X17T, 2
DSZ NUM, 2
JMP AH
STA 1, APAC3, 2
DSZ PAC3, 2
DSZ M, 2
JMP AG
LDA 0, TEMP, 2
SUB 1, 1
ADD 0, 3, S2R
SUB 2, 1
MOV 1, 1
JSR 0RTH, 2

AH:

LDA 0, 0X17T, 2
ADD 0, 1, S2C
INC 3, 3
DSZ X17T, 2
DSZ NUM, 2
JMP AH
STA 1, APAC3, 2
DSZ PAC3, 2
DSZ M, 2
JMP AG
LDA 0, TEMP, 2
SUB 1, 1
ADD 0, 3, S2R
SUB 2, 1
MOV 1, 1
JSR 0RTH, 2

119
DUMMYENTRIES

NREL

ENT   CLG LIST

EXTN  PPREP

CLG:  JSR  @PREPP
LIST:  JSR  @PREPP
PREPP: PREPP
TITLE DSKFL
ENT DSKFL
NREL

INPUT ARGUMENTS IN IBUF (+USP)

O-255
IFCN=-1
IO=-2
STD=-3
ICT=-4
IVEC=-6
IER=-7

IO BUFFER
CHANNEL #
STARTING BYTE IN DISK FILE
NUMBER OF BYTES TO PROCESS
ADDR OF DATA VECTOR TO PROCESS
ERROR FLAG

TEMPORARY DATA STORAGE

PSECT=-10
WFLG=-11
SECTR=-12
DISP=-13
WORD=-14
LIM=-15
CPR=-20
FPTR=-21
TPTR=-22

PREVIOUS SECTOR # OF BUFFER
0-CLEAN, 1-HAS BEEN UPDATED BUT NOT WRITTEN
SECTOR # OF CURRENT DATA
DISP OF DATA WORD FROM LOOP COUNTER
WORD TO START LOOP ON
WORD TO END LOOP ON

DSKFL: ISZ RTST. 2
STA 3,RTST. 2
LDA 3,IBUF. 2
LDA 0,STWD. 3
NEG 0,0
COM 0,0
LDA 1,MZ256.2
AND 0,1
SECTR=256
INC 0,0
SUB 1,0
NEG 0,0
INC 0,0
STA 0,DISP. 3
SECTR=256-STWD+1
MOVS 1,1
STA 1,SECTR. 3
STWD-1)/256
SUBZ 1,1
STA 1,IER. 3
IER=1 IF NO ERROR
SUB 0,1
STA 1,WORD. 3
1-DISP
LDA 0,THD. 2
LDA 1,IFCN. 3
SUBZ 0,1,SNR
GO TO (100, 200, 300), IFCN
JMP D200
IFCN=2
MOV 1,1,SNR
JMP D100
IFCN=1
JMP @D300P
IFCN=3

READ

D100:
LDA 0,SECTR. 3
LDA 1,SECTR. 3
SUB 0,1,SNR
JMP D120
CURRENT SECTOR
LDA 0,WFLG. 3
MOV 0,0,SNR
JMP D110
PREV SECTOR WRITTEN
LDA 1, PSECT, 3
JSP WRITP
JSP @05AA0P
JSP @05AA0P
LDA 3, IRUF, 2
D110: 
JSP @06AA0P
JSP @06AA0P
LDA 3, IRUF, 2
D120: 
LDA 1, ICT, 3
LDA 0, WORD, 3
ADD 0, 1
ADC 0, 0
ADD 1, 0
STA 0, LIM, 3
LDA 1, TH056, 2
SUBZ 0, 1, SZC
LIM LE 256?
JMP D130
YES
SUB 1, 0
STA 0, ICT, 3
LIM-256
STA 1, LIM, 3
LIM
JMP D140
D130: 
SUB 0, 0
STA 0, ICT, 3
D140: 
LDA 0, WORD, 3
ADD 1, 1
ADD 1, 0
WORD-1
MOV 3, 1
ADD 0, 1
IBUF(1)
STA 1, FPTR, 3
LDA 1, IVEC, 3
ADD 0, 1
LDA 2, DISP, 3
LDA 1, 2
IVEC(1+DISP)
ADD 1, 2
IVEC([1+DISP]=IBUF(1))
STA 2, TPTR, 3
LDA 2, USP
LDA 1, LIM, 3
SUB 0, 1
STA 1, CTR, 3
CTR FOR # TIMES THROUGH LOOP
D150: 
LDA 0, FPTR, 3
STA 0, TPTR, 3
IVEC(1+DISP)=IBUF(1)
ISZ FPTR, 3
ISZ TPTR, 3
DSZ CTR, 3
JMP D150
LDA 0, ICT, 3
MOV 0, 0, SNR
JMP D160
LDA 0, DISP, 3
MOV 0, 0, SNR
JMP +3
MOV2L 0, 0, SNC
JMP D170
DISP GT 0
LDA 0, LIM, 3
LDA 1, LIM, 3
D160:
SUB 0
SUBZL 0
STA 0, LIM, 3
MOV 0, 0, SNR
ISZ SECTR, 3
JSP READ
JSP @05AA0P
JMP D170
MOV   3,1
ADD  0,1         ;IBUF(1)
STA   1,TPTR,3
LDA   1,IVEC,3
ADD  0,1
LDA   2,DISP,3
ADD  1,2         ;IVEC(1+DISP)
STA   2,FPTR,3
LDA   2,USP
LDA   1,LIM,3
SUB  0,1         ;CTR FOR # TIMES THROUGH LOOP
STA   1,CTR,3

D250:  LDA   0,APPTR,3
STA   0,APPTR,3
ISZ   FPTR,3
ISZ   TPTR,3
DSZ   CTR,3
JMP   D250
LDA   0,ICT,3
MOV   0,0,SNR
JMP   D280
LDA   0,DISP,3
MOV   0,0,SNR
JMP   +3
MOVZL 0,0,SNC
JMP   D270
LDA   0,LIM,3
LDA   1,WORD,3
SUB  1,0
INC  0,0
STA   0,DISP,3       ;LIM-WORD+1
SUBZL 0,0
STA   0,WORD,3       ;1
LDA   1,SECTR,3
JSR   WRITE          ;WRITE CURRENT SECTOR
JSR   @D500P
LDA   3,IBUF,2
ISZ   SECTR,3
JSR   READ          ;READ NEXT SECTOR
JSR   @D500P
LDA   3,IBUF,2
LDA   0,ICT,3
LDA   1,TH056,2
SUBZL 0,1,SZC         ;LIM LE 256?
JMP   D220          ;YES
STA   1,LIM,3
SUB  1,0
STA   0,ICT,3         ;ICT-256
JMP   D24A
D270:  LDA   0,DISP,3
LDA   1,TH056,2
ADD  1,0
JMP   D26A          ;DISP+256
D280:  LDA   0,SECTR,3
STA   0,PSECT,3
SUB   0,0
STA   0,HLFLG,3
JSR   @PTH,2
D300:  LDA   1,PSECT,3
JSR   WRITE          ;WRITE PREVIOUS SECTOR
JSR   D500
LDA   3, IBUF, 2
SUB   0, 0
STA   0, WFLG, 3
JSR   @RTN, 2
READ:
ISZ  RTST, 2
STA   3, @RTST, 2
LDA   3, IBUF, 2
MOV   3, 0
LDA   1, SECTR, 3
LDA   2, ONEB7, 2
MOV   1 BLOCK (LFT BYTE)
LDA   3, IO, 3
MOV   CHANL RT BYTE
ADD   3, 2
. SYSTH
RDB   77
JSR   @RTER, 3
LDA   2, USP
JSR   @RTN, 2
WRITE:
ISZ  RTST, 2
STA   3, @RTST, 2
LDA   3, IBUF, 2
MOV   3, 0
LDA   2, ONEB7, 2
MOV   1 BLOCK LFT BYTE
LDA   3, IO, 3
MOV   CHANL RT BYTE
ADD   3, 2
. SYSTH
RWB   77
JSR   @RTER, 3
LDA   2, USP
JSR   @RTN, 2
D500:
LDA   3, IBUF, 2
STA   0, IER, 3
SUB   0, 0
STA   0, WFLG, 3
STA   0, SECTR, 3
JSR   @RTER, 2
JSR   0,ierung, 3
JSR   0, SECTR, 3
JSR   0, IER, 3
JSR   0, USP
JSR   0, WFLG, 3
INITIALIZE SAVE FILE AND CORE

.TITLE  ENTER
.NREL
.TXTM  1
.ENT  ENTER
.EXTN  RDSVF, INSVP

ENTER:  ISZ  PTEST, 2
STA  3, @PTEST, 2
JSR  @CHAP, 2
JSR  @TER, 2

SYSTEM
.PCHAR
JSR  @TER, 3
LDA  1, C1, 2
SUB  0, 1, STPR
JMP  EN1
LDA  0, LF, 2

Systm
.PCHAR
JSR  @TER, 3

ENT1:  JSR  @RDSVP
JSR  @TER, 2
LDA  0, SVFLN, 2
JSR  @OFFVP, 2
JSR  @TER, 2
STA  1, SVFCN, 2
JSR  @ALCP, 2
JSR  @TER, 2
JSR  @INSVP
JSR  @TER, 2
JSR  @RTN, 2

RDSVP:  RDSVF
INSVP:  INSVF
EXIT1:   ISZ RTST.2 GET NEXT STORE POSITION
STA 3.0RTST.2 SAVE RETURN ADDRESS
LDA 1.0NDPT.2 LOAD AC1 WITH THE END STATEMENT
LDA 3.OC.2 LOAD AC3 WITH THE OC
SUB 3.1.S2R COMPARE
JSR @RTN.2 IF NOT EQUAL, RETURN
EXIT2:   DSZ @CNT.2 DECREMENT COUNTER
JMP EXIT3 IF NOT ZERO, JUMP TO EXIT3
SUB 3.3 CLEAR AC3 TO ZERO
STA 3.0NDPT.2 STORE 0 IN NDPT
STA 3.0SAVE.2 STORE 0 IN SAVE
STA 3.0CNT.2
LDA 1.0RTPT.2 LOAD AC1 WITH RETURN STATEMENT
STA 1.TOC.2 STORE RETURN STATEMENT IN TOC
STA 3.0RTPT.2 STORE 0 IN RTPT
DSZ CNT.2
DSZ NDPT.2
DSZ SAVE.2
DSZ RTPT.2
JSR @RTN.2
EXIT3:   LDA 3.0SAVE.2 LOAD AC3 WITH BEGINNING PT.
STA 3.TOC.2 STORE BEGINNING PT. IN TOC
JSR @RTN.2 PROCESSING COMPLETE, RETURN
EXECUTE QUINT LOCATED IN OP

.TITLE EXECUTEQUINT

.NREL
.TXTM 1
.ENT EXONT, PRONT
.EXTN BTL

EXONT: ISZ RTST, 2
       STA 3, @RTST, 2
       LDA 3, @SP, 2
       MOV 3, SJZC, 2, ADDRESS?
       JMP EX02, JYES
       JSR 0, 3, EXECUTE STATE ROUTINE
       JSR @RTER, 2
       JSR SP, 2
       JMP EX01

EX01: MOVZL 3, SZR, 2
       JMP EX03, JYES
       LDA 0, TOC, 2
       MOV 0, 0, SZR
       JMP EX04
       JMP EX04+1

EX02: MOVZR 3, 2
       STRP OFF BA
       STA 3, SP, 2
       JMP EX01

EX03: STA 0, OC, 2
       SUB 0, 0
       STA 0, TOC, 2
       LDA 3, BTLP
       LDA 0, 3
       STA 0, TEMP, 2
       INC 3, 3
       ADDR BREAK TABLE
       INC 3, 3
       LDA 0, OC, 2
       JMP EX05

EX04: LDA 1, 1, SNR
       JMP 1, 1, +3
       SUB 0, 1, SNR
       JMP EX06
       INC 3, 3
       DSZ TEMP, 2
       JMP EX05
       JSR @RTN, 2
       JSR @STPFL, 2
       JSR @PRMP, 2
       JSR @RTER, 2
       JSR @RTN, 2
       JSR @PRMP, 2
       JSR @RTER, 2
       LDA 0, 0

PRONT: ISZ RTST, 2
       STA 3, @RTST, 2
       JSR @RTER, 2
       JSR @RTN, 2
       JSR @RTER, 2
       LDA 0, AAA

.SVSTM
.PCHAR
.JSP @RTER, 2
.LDA 0, AAA

.SVSTM
.PCHAR
JSP .@RTEP.2
LDA 0, DC, 2
JSP .@PROC, 2
JSP .@RTER.2
LDA 0, FIVE, 2
STA 0, SAV3, 2
EX07:
LDA 3, OP, 2
LDA 0, A, 3
JSP .@PROC, 2
JSP .@RTER.2
ISZ OP. 2
DSZ SAV3, 2
JMP EX07
JSP .@PRCRP, 2
JSP .@RTER.2
JSP .@RTHN.2
BTLB:
BTL
Q: .TXTM 0
AAA: .TXT /#/
GETARGMENTS
OP1D, OP2D, OP3N, OP4N, OP5N, OP6N
EXTN OP1N, OP2N, OP3N, OP4N, OP5N, OP6N
EXTN RETRN, CONST, GINST, GIST, ERTRN

OP1D: ISZ RTST, 2 ; SAVE RETURN ADDRESS
STA 3 RTST, 2 ; SAVE RETURN ADDRESS
LDA 3 OP, 2 ; LOAD QUANT POINTER
LDA 0, 3 ; GET FIRST ARGUMENT
LDA 1, SPFI, 2 ; LOAD SUBSCRIPT FLAG
MOV 1, 1, SZR ; IS FLAG = 0
JMP NEW1 ; NO
MOV 0, 0, SZR ; YES, CHECK IF OPER. = 0
JMP +11 ; NO
LDA 1, 1, 3 ; YES, LOAD OPCODE
MOVZR 1, 1
MOVZR 1, 1
MOVZR 1, 1
LDA 0, C3I, 2 ; LOAD OPCODE 25
SUB 1, 0, SZR ; CHECK IF EQUAL
JSR @RTPE, 2 ; NO. ERROR RETURN
JSR @RTN, 2 ; YES. NORMAL RETURN
JSR @NSTP, 2
JSR @TER, 2
MOV 1, 1
LDA 0, 0, 3 ; LOAD AC3 TO AC8
STA 0, SAVI, 2
JSR @PSTP, 2
JSR @TER, 2
STA 0, OP1DP, 2 ; GET BYTE POINTER
MOV 1, 3
LDA 0, 1, 3
STA 0, OPIDT, 2 ; GET TYPE
LDA 0, 2, 3
STA 0, OPIDL, 2 ; GET LENGTH
LDA 0, 3, 3
LDA 1, MSK4, 2 ; LOAD MASK 377
AND 1, 0 ; SAVE RIGHT BYTE
STA 0, OPIDX, 2 ; GET NO. OF OCCUR.
LDA 0, 3, 3
MOVS 0, 0 ; SWAP BYTES
LDA 1, MSK3, 2
AND 1, 0 ; SAVE RIGHT BYTE
MOVZ 0, 0
MOVZ 0, 0
STA 0, DEC1, 2 ; GET NO. DECIMAL POSITIONS
JMP @PTN, 2 ; PROCESSING COMPLETE. RETURN

NEW1: ISZ SP, 2 ; INCREMENT STATE POINTER
LDA 1, SP, 2 ; LOAD STATE PTR.
SUBZR 0, 0 ; SET RITA
ADD 0, 1 ; SET RITA
STA 1, SP, 2 ; STORE IN OP1N
LDA 1, PSI, 2 ; LOAD DUMMY STATE TABLE
STA 1, SP, 2 ; STORE IN STATE PTR.
JSR @PTN, 2 ; RETURN

OP2D: ISZ RTST, 2 ; GET NEXT STORE POSITION
STA 2, RTST, 2 ; SAVE RETURN ADDRESS
LDA 3, OP, 2 ; LOAD QUANT POINTER
LDA 0, 3, 3 ; GET SECOND ARGUMENT
LDA 1, SPFI, 2 ; LOAD SUBSCRIPT FLAG
MOV 1, 1, SZR ; IS FLAG = 0
JMP NEW2, NO
MOV 0, 0, SZR, YES, CHECK IF OPER = 0
JMP +411, NO
LDA 1, 1, 3, LOAD OPCODE
MOVZ #1
MOVZ #1
LDA 0, C31, 2, LOAD OPCODE 25
SUB 1, 0, SZR, CHECK IF EQUAL
JSR @RTER, 2, NO. EPERROR RETURN
JSR @RTN, 2, YES, NORMAL RETURN
JSR @NSTP, 2
JSR @RTER, 2
MOV 1, 3
LDA 0, 0, 3, LOAD A3 TO AC0
STA 0, SAV2, 2
JSR @PSTP, 2
JSR @RTER, 2
STA 0, OP2DP, 2, GET BYTE POINTER
MOV 1, 3
LDA 0, 1, 3
STA 0, OP2DT, 2, GET TYPE
LDA 0, 2, 3
STA 0, OP2DL, 2, GET LENGTH
LDA 0, 0, 3
LDA 1, MSK4, 2, LOAD MASK 377
AND 1, 0, SAVE RIGHT BYTE
STA 0, OP2DX, 2, GET NO. OF OCCUR.
LDA 0, 0, 3
MOVZ 0, 0
MOVZ 1, MSK3, 2
AND 1, 0
SAVE RIGHT BYTE
MOVZR 0, 0
MOVZR 0, 0
STA 0, DEC2, 2, GET NO. DECIMAL POSITIONS
LDA 0, 4, 3
LDA 1, EDST, 2, GET BEGINNING OF EDIT TABLE
ADD 1, 0
ADD DISPLACEMENT
MOVZ 0, 0
CONVERT TO BYTE POINTER
STA 0, OP2DE, 2
GET EDIT POINTER
JMP @RTN, 2
PROCESSING COMPLETE, RETURN
NEW2:
ISZ SP, 2
INCREMENT STATE PTR.
LDA 1, SP, 2
LOAD STATE PTR
SUBZ 0, 0
SET BIT 0
ADD 0, 1
SET BIT 0
STA 1, DOP2, STORE IN OP2DP
LDA 1, D62, LOAD DUMMY STATE TABLE
STA 1, SP, 2
STORE IN SP
JSR @RTN, 2
RETURN
OP3:
ISZ RTST, 2
GET NEXT STORE POSITION
STA 3, ORST, 2
SAVE RETURN ADDRESS
LDA 3, DOP, 2
LOAD DINTO POINTER
LDA 0, 4, 3
GET THIRD ARGUMENT
LDA 1, SRFI, 2
LOAD SUBSCRIPT FLAG
MOV 1, 1, SZR
JMP NEW3
MOV 0, 0, SZR
JMP +411
LDA 1, 1, 3
MOVZR 1, 1
MOVZR 1, 1
132
MOVZR 1.1
LDA 0, C31, 2
SUB 1, 0, S7P
JSR @RTER, 2
JSR @RTH, 2
JSR @WSTP, 2
JSR @RTER, 2
MOV 1.3
LDA 0, 0, 3
STA 0, SA3, 2
JSR @RSTP, 2
JSR @RTER, 2
STA 0, OP30P, 2
GET BYTE POINTER
MOV 1.3
LDA 0, 1.3
STA 0, OP30T, 2
GET TYPE
LDA 0, 2.3
STA 0, OP30L, 2
GET LENGTH
LDA 0, 3.3
LDA 1, MSK4, 2
LOAD MASK 377
AND 1, 0
SAVE RIGHT BYTE
STA 0, OP30X, 2
GET NO. OF OCCUR.
LDA 0, 3.3
MOV 0, 0
SWAP BYTES
LDA 1, MSK3, 2
AND 1, 0
SAVE RIGHT BYTE
MOVZR 0, 0
MOVZR 0, 0
STA 0, DEC3, 2
GET NO. DECIMAL POSITIONS
JMP @RTH, 2
PROCESSING COMPLETE, RETURN

NEW3:
ISZ SP, 2
LDA 1, SP, 2
SUBZR 0, 0
ADD 0, 1
STA 1, OP3
LDA 1, DS3
STA 1, SP, 2
JSR @RTH, 2
DS1: . GADO OP1DS, -1
DS2: . GADO OP2DS, -1
DS3: . GADO OP3DS, -1
DP1: . GADO OP1DR
DP2: . OP2DR
DP3: . OP3DR
OP4D: ISZ RTST, 2
GET NEXT STORE POSITION
STA 3, RTST, 2
SAVE RETURN ADDRESS
LDA 3, OP, 2
LOAD QUINT POINTER
LDA 1, 4, 3
GET THE THIRD ARGUMENT
STA 1, TOC, 2
STORE IN TEMP QUINT CTR
JMP @RTH, 2
PROCESSING COMPLETE, RETURN

OP5D: ISZ RTST, 2
GET NEXT STORE POSITION
STA 3, RTST, 2
SAVE RETURN ADDRESS
LDA 3, OP, 2
LOAD QUINT POINTER
LDA 0, 2, 3
GET ARGUMENT 1
JSR @FQST1
JSR @RTER, 2
MOV 1.3
LDA 0, 1.3
STA 3, PNT1, 2
JSR @WSTP, 2
JSR @RTER, 2
MOV 1.3
LDA 0.0.3
JSR @RSSTP, 2
JSR @TER, 2
STA 0.OP1DP, 2 //GET BYTE POINTER
MOV 1.3
LDA 0.2.3 //GET THE LENGTH
STA 0.OP1DL, 2
LDA 3.PNT1, 2
LDA 0.2.3
JSR @IOD1
JSR @TER, 2
MOV 1.3
LDA 0.5.3 //PICK-UP CHANNEL NO.
STA 0.CHANL, 2 //STORE IN DATA-AREA
JMP @RTN, 2 //PROCESSING COMPLETE, RETURN

OPED:
ISZ RTST, 2 //GET NEXT STORE POSITION
STA 3.ORST, 2 //SAVE RETURN ADDRESS
LDA 3.OP, 2 //LOAD QUINT POINTER
LDA 0.2.3 //GET FIRST ARGUMENT
JSR @FDST1
JSR @TER, 2
MOV 1.3
LDA 0.2.3
JSR @IOD1
JSR @TER, 2
MOV 1.3
LDA 0.5.3 //PICK-UP CHANNEL NO.
STA 0.CHANL, 2 //STORE IN DATA-AREA
MOVZ 3.3
STA 3.OP1DP, 2 //GET I/O BYTE POINTER
JMP @RTN, 2 //PROCESSING COMPLETE, RETURN

FDST1: GFDEST
IOD1: GIOST
.TITL GETEDIT
.ENT EDIT1
.EXTD GET

EDIT1: ISZ RTST.2 GET NEXT STORE POSITION
STA 3.OPRT.2 SAVE RETURN ADDRESS
LDA 1.SHRT.2 CHECK TO SEE WHICH FORMAT TO GO TO
MOVX 1.1.SZC NOT GET TEST TO SEE WHICH FORMAT
JMP SHORT LAST EDIT CODE HAS IN SHORT FORMAT
LDA 3.OP2DE.2 GET EDIT FORMAT
MOVZ 3.3
STA 3.OPDE.2 STORE IN TEMP FIELD
LDA 3.OPDE.2 STORE EDIT FIELD
MOVX 3.3.SNC TEST TO SEE IF FORMAT IS LONG OR SHORT
JMP SHORT NOT ON SHORT FORMAT
LDA 3.OPDE.2 GET EDIT FORMAT
MOV 3.1 MOVE TO AC1
LDA 0.EMSK1.2 GET EDIT MASK 3000
AND 0.1 TO GET REPITION COUNT
STA 1.RCNT.2 STORE REPIATION COUNT
LDA 1.EMSK2.2 GET EDIT MASK 74000
AND 1.3 TO GET EDIT CODE
LDA 1.NINE.2 GET A ELEVEN
STA 1.X11T.2 STORE IN COUNTER
MOVX 3.3 SHIFT EDIT CODE TO RIGHT MOST POSITION
DSZ X11T.2 DECREMENT COUNTER
JMP .-2 DO TILL DONE
STA 3.ECODE.2 STORE IN EDIT CODE
ISZ 0.RTHN.2

SHORT: LDA 3.OP2DE.2 GET EDIT FORMAT
STA 3.EPVR.2 STORE IN BYTE POINTER
JSR 0.GET GET A BYTE ROUTINE
JMP 0.PTER.2
MOV 1.3 MOVE EDIT FORMAT TO AC3
LDA 0.EMSK5.2 GET EDIT MASK 20A
AND 0.1 TO CHECK WHAT FORMAT THIS IS
LDA 0.BELL.2 GET A SEVEN
STA 0.X11T.2 STORE IN A COUNTER
MOVX 1.1 SHIFT TO RIGHT MOST POSITION
DSZ X11T.2 DECREMENT COUNTER
JMP 0.
MOVX 1.1.SNC TEST TO SEE IT ON
JMP .+3 NOT ON
ISZ 0.OP2DE.2 INCREMENT EDIT FORMAT BYTE POINTER
JMP EDIT1+5 GET NEXT EDIT FORMAT
MOV 3.1 GET THE EDIT FORMAT
LDA 0.EMSK3.2 GET EDIT MASK 7
AND 0.1 TO GET REPIATION COUNT
STA 1.RCNT.2 STORE IN REPIATION COUNT
ISZ RCNT.2
LDA 1.EMSK4.2 GET EDIT MASK 170
AND 1.3 TO GET EDIT CODE
LDA 1.THEP.2 GET A THREE FOR A COUNTER
STA 1.X3T.2 STORE IN COUNTER
MOVX 3.3 SHIFT EDIT CODE TO RIGHT MOST POSITION
DSZ X3T.2 DECREMENT COUNTER
JMP 0.
STA 3.ECODE.2 STORE IN EDIT CODE
SUBZL 1.1 GENERATE A 0IF
STA 1.SHRT.2 STORE IN SHORT FORMAT CODE TO RETURN TO HERE
ISZ     OPZOE. 2
JMP    ORTN. 2    RETURN
.TITLE GETTABLES
.ENT GDNST, GFDST, GIOST, GRSST

.GDNST:
ISZ RTST, 2  ; GET NEXT STORE POSITION
STA 3, @RTST, 2  ; SAVE RETURN ADDRESS
NEG 0, 0
COM 0, 0
MOV 0, 1  ; MULTIPLY BY FIVE
ADDZL 0, 0
ADD 1, 0
LDA 1, DNST, 2  ; LOAD DATA NAME TABLE PTR.
ADD 0, 1  ; GET DATA NAME ENTRY
JSR @RTN, 2

.GFDST:
ISZ RTST, 2  ; GET NEXT STORE POSITION
STA 3, @RTST, 2  ; SAVE RETURN ADDRESS
NEG 0, 0
COM 0, 0
MOV 0, 1  ; MULTIPLY BY FIVE
ADDZL 0, 0
ADD 1, 0
LDA 1, FDST, 2  ; LOAD FD TABLE PTR.
ADD 0, 1  ; GET FD ENTRY
JSR @RTN, 2

.GIOST:
ISZ RTST, 2  ; GET NEXT STORE POSITION
STA 3, @RTST, 2  ; SAVE RETURN ADDRESS
NEG 0, 0
COM 0, 0
MOVZL 0, 1  ; MULTIPLY BY SIX
ADDZL 1, 0
LDA 1, IOST, 2  ; LOAD I/O TABLE PTR.
ADD 0, 1  ; GET I/O ENTRY
JSR @RTN, 2

.GRSST:
ISZ RTST, 2  ; GET NEXT STORE POSITION
STA 3, @RTST, 2  ; SAVE RETURN ADDRESS
LDA 3, RST, 2  ; LOAD RAM STORAGE WORD PTR.
MOVZL 3, 3  ; CONVERT TO RYTE PTR.
ADD 3, 0  ; ADD DATA-NAME DISPLACEMENT
JSR @RTN, 2
GOTO1:  ISZ    RTST.2    GET NEXT STORE POSITION
       STA    3. RTST.2    SAVE RETURN ADDRESS
       LDA    1. SFLAG.2    LOAD SIZE ERROR FLAG
       MOV.   1.1. SZE    SEE IF SET
       JMP    GOTO2    YES
       LDA    1. MPFLAG.2    NO, LOAD MATH-PACK FLAG
       MOV.   1.1. SZE    SEE IF SET
       JMP    GOTO3    YES
       JMP    @RTN.2    UNO
GOTO2:  LDA    1. MPFLAG.2    LOAD IN MATH-PACK FLAG
       MOV.   1.1. SZE    CHECK TO SEE IF SET
       JMP    @RTN.2    YES, DO THE GOTO
GOTO3:  SUB    1.1    CLEAR AC1
       STA    1. T0C.2    STORE IN TEMP, QUINT CTR
       JMP    @RTN.2    PROCESSING COMPLETE, RETURN
LESS:  ISZ  RTST.2  GET NEXT STORE POSITION
STA  3.0RTST.2  SAVE RETURN ADDRESS
LDA  1.ONE.2  LOAD A ONE
STA  1.ROCK.2  STORE IT IN ROCKK
LDA  1.OPFLG.2  LOAD OPCODE FLAG
MOVR 1.1.SZC  CHECK STATUS
JMP  FLG01  JON. NLESS
JMP  FLG00  JSET NFLAG TO 0

GREAT: ISZ  RTST.2  GET NEXT STORE POSITION
STA  3.0RTST.2  SAVE RETURN ADDRESS
LDA  1.THO.2  LOAD A TWO
STA  1.ROCK.2  STORE IT IN ROCKK
LDA  1.OPFLG.2  LOAD OPCODE FLAG
MOVR 1.1.SZC  CHECK STATUS
JMP  FLG01  JON. NGREATER
JMP  FLG00  JSET NFLAG TO 0

EQUAL: ISZ  RTST.2  GET NEXT STORE POSITION
STA  3.0RTST.2  SAVE RETURN ADDRESS
LDA  1.THRP.2  LOAD A THREE
STA  1.ROCK.2  STORE IT IN ROCKK
LDA  1.OPFLG.2  LOAD OPCODE FLAG
MOVR 1.1.SZC  CHECK STATUS
JMP  FLG01  JON. NEQUAL
JMP  FLG00  JSET NFLAG TO 0

FLG01: LDA  1.ONE.2  LOAD A ONE
STA  1.NFLAG.2  STORE IT IN NFLAG
JMP  @RELAT  JPROCESS RELATIONAL TEST

FLG00: SUB  1.1  JCLEAR AC1 TO ZEPO
STA  1.NFLAG.2  STORE IT IN NFLAG
JMP  @RELAT  JPROCESS RELATIONAL TEST

ALPHA: ISZ  RTST.2  GET NEXT STORE POSITION
STA  3.0RTST.2  SAVE RETURN ADDRESS
LDA  1.THO.2  LOAD A TWO
STA  1.ROCK.2  STORE IT IN ROCKK
LDA  1.OPFLG.2  LOAD OPCODE FLAG
MOVR 1.1.SNC  CHECK STATUS
JMP  FLG01  JON. NALPHA
JMP  FLG01  JSET NFLAG TO -1

NUMRC: ISZ  RTST.2  GET NEXT STORE POSITION
STA  3.0RTST.2  SAVE RETURN ADDRESS
LDA  1.THRP.2  LOAD A THREE
STA  1.ROCK.2  STORE IT IN ROCKK
LDA  1.OPFLG.2  LOAD OPCODE FLAG
MOVR 1.1.SNC  CHECK STATUS
JMP  FLG01  JON. NUMRIC
JMP  FLG01  JSET NFLAG TO -1

FLAG0: SUB  1.1  JCLEAR AC1 TO ZEPO
STA  1.NFLAG.2  STORE IT IN NFLAG
JMP  CLASS  JPROCESS CLASS TEST

FLAG1: SUB2L  1.1
STA  1.NFLAG.2  STORE IT IN NFLAG

CLASS: LDA  1.OPTD.2  LOAD OPERAND 1 TYPE
MOVR 1.1.SZC  JCHECK TYPE <1 2 OR GREATER
JMP  HSXPG  J2 OR GREATER (NO)
LDA  1.THRP.2  LOAD A THREE
STA  1.CKND.2  STORE IT IN CKND
LDA 1.ONE, 2   ;LOAD A ONE
STA 1.BTCNT, 2   ;STORE IT IN BTCNT
JMP A2
A2

HSKPG: LDA 1.OPDL, 2   ;LOAD OPERAND 1 LENGTH
STA 1.BTCNT, 2   ;STORE IT IN BTCNT
LDA 1.OPDP, 2   ;LOAD OPERAND 1 BYTE PTR.
STA 1.OPDP, 2   ;STORE IN OPDP
JMP A1
A1

SUB 1, 1   ;CLEAR AC1 TO ZERO
STA 1.CHKND, 2   ;STORE IT IN CHKND
LDA 3.OPDP, 2   ;LOAD OPERAL 1 BYTE PTR.
STA 3.BPTR, 2   ;STORE IN BYTE POINTER
JMP &GET   ;JUMP TO GET A CHARACTER
JMP &TER, 2
LDA 3.MINUS, 2   ;TEST FOR MINUS
SUB# 3.1.SNR
JMP AB
LDA 3.PLUS, 2   ;LOAD A PLUS SIGN
SUB# 3.1.SNR   ;CHECK IF EQUAL
JMP AB   ;EQUAL TO PLUS SIGN
LDA 0.CAO, 2   ;LOAD AN ASCII ZERO
LDA 3.CSI, 2   ;LOAD AN ASCII NINE
ADC# 3.1.SNC   ;SKIPS IF > 9
ADC# 1.0.SZC   ;SKIPS IF > OP = 9
JMP .+42   ;NOT EQUAL TO A DIGIT
JMP INC3   ;EQUAL TO A DIGIT
LDA 3.THERE, 2   ;LOAD A THREE
LDA 0.RCH, 2   ;LOAD REQUEST CHECK CODE
SUB# 3.8.SNR   ;CHECK IF EQUAL
JMP AA   ;EQUAL

A3: LDA 3.BLK, 2   ;SUB# 3.1.SNR
JMP INC2
MOV 1, 3   ;SWAP BYTES OF WORD
MOVL 3, J   ;MOVE BIT 0 INTO CARRY
MOVL 3, 3.SNC   ;MOVE NEXT BIT TO CARRY
JMP INC2   ;CARRY EQUAL TO 1
INC1: ISZ CHKND, 2   ;INCREMENT CHKND (WILL END UP A 3)
INC2: ISZ CHKND, 2   ;INCREMENT CHKND (WILL END UP A 2)
INC3: ISZ CHKND, 2   ;INCREMENT CHKND (WILL END UP A 1)
A2: LDA 3.RCH, 2   ;LOAD REQUEST CHECK
LDA 1.CHKND, 2   ;LOAD CHECK WORD
SUB 3.1.SNR   ;CHKND - RCH
JMP .+7
A4: LDA 3.NFLAG, 2   ;LOAD NFLAG
MOV 3, 3.SZR   ;RESULT OF CHKND-RCH = NFLAG
JMP &RTH, 2   ;NOT IN TEMP. QUINT CTR.
SUB 1, 1   ;CLEAR AC1 TO ZERO
STA 1.TOC, 2   ;STORE IN TEMP. QUINT PTR.
JMP &RTH, 2
ISZ OPDP, 2   ;INCREMENT BYTE POINTER
DSZ BTCNT, 2   ;DECREMENT LENGTH
JMP A1   ;NOT EQUAL TO ZERO. GET NEXT CHARACTER
LDA 1.NFLAG, 2
MOV 1.1.SNR   ;TEST TO SEE IF NFLAG IS SET
JMP &RTH, 2
SUB 1, 1   ;YES
STA 1.TOC, 2
JMP &RTH, 2
AB: LDA 1.OPDP, 2
LDA 0.FOUR, 2

140
SUB 1.0, ZR  ; TEST TO SEE IF A SIGN FIELD
JMP AA  ; NO
JMP INC3
RELAT: PLAT1
IMMEDIATE/INSERT COMMAND

.TITLE IMINS
.NREL
.TXTM 0
.ENT IMINS, VALLOG, NMCH, NMCH

IMINS:
ISZ RST.2
STA 3, ARTST.2
JSR @NMHC.2  ; READ NUMBER
JMP +3
ISZ CERFL.2
JSR @RTER.2  ; CARR RET
STA 1, TEMP.2  ; NUMBER
LDA 3, CMDCT

IMIN1:
LDA 1, 0, 3  ; COMMAND CHARACTER
MOV 1, 1, SNR  ; END OF TABLE?
JSR @RTER.2  ; YES
SUB 0, 1, SNR  ; THIS CHAR?
JMP @DISPL.3  ; NO
INC 3, 3
JMP IMIN1  ; EXECUTE

CMDCT:
+1  ; COMMAND CHARACTER & DISPATCH TABLE
.TXT *//  ; IO FILE INSERT
.TXT *//  ; FD FILE INSERT
.TXT *//  ; DATA NAME FILE INSERT
.TXT *//  ; QUINT FILE INSERT
.TXT *//  ; IMMEDIATE
.TXT *//  ; IMMEDIATE

0
.TXTM 1
.DISPL=7  ; DISPLACEMENT TO DISPATCH TABLE
JOIN
FDIN
DNIN
QNIN
IMMED
IMMED

JOIN:
JSR @PREP.2
FDIN:
JSR @PREP.2
QNIN:
JSR @PREP.2

; INSERT DATA NAME ENTRY

DNIN:
DSZ TEMP.2
LDA 0, TEMP.2  ; DATA NAME ENTRY -1
JSR @NASTP.2
JSR @RTER.2
STA 1, TEMP.2  ; STARTING ADDRESS OF ENTRY
LDA 1, FOUP.2
STA 1, TEMP.2.2

.DNZ:
JSR @NMHC.2  ; READ NUMBER
JMP +2
JMP DN2
ISZ TEMP.2
STA 1, @TEMP.2
DSZ TEMP.2.2
JMP DN1
ISZ CERFL.2
JSR @PTFP.2
DNZ:
DSZ TEMP.2.2
JMP .+2
JMP .+3
ISZ CERFL, 2
JSR @RTER, 2
ISZ TEMP, 2
STA 1, @TEMP, 2
LDA 3, TEMP, 2
LDA 0, -3, 3
LDA 1, ONEE7, 2
BRING IN A 400
MOVZL 1, 1
AND 0, 1, SNR
JMP @RTN, 2
LDA 0, RSMX, 2
LDA 1, -2, 3
MOVZR# 0, 0, S2C
INC 0, 0
STA 0, -4, 3
DRAW STORAGE BYTE PTR
ADD 1, 0
STA 0, RSMX, 2
NEW MAX
INM1: LDA 1, TEMP, 2
LDA 3, IMONT
INC 3, 3
STA 1, 0, 3
INC 3, 3
STA 3, TEMP, 2
LDA 1, THREE, 2
STA 1, TEMP2, 2
INM2: JSR @NMMCP, 2
JMP .+2
JMP IMM2
STA 1, @TEMP, 2
ISZ TEMP, 2
DSZ TEMP2, 2
JMP IMM1
JSR @RPERP, 2
DSZ TEMP2, 2
JSR @RPERP, 2
STA 1, @TEMP, 2
LDA 3, IMONT
STA 3, OP, 2
LDA 3, 1, 3
LDA 1, BELL, 2
AND 3, 1
STA 1, OPFLG, 2
MOVZR 3, 3
MOVZR 3, 3
MOVZR 3, 3
LDA 1, OPDSP, 2
ADD 1, 3
LDA 1, SP, 2
STA 1, OP1DE, 2
STORE OLD SP
LDA 3, 0, 3
LDA 1, O, 3
STA 1, SP, 2
JSR @ENKDP, 2
JSR @RTER, 2
LDA 1, OP1DE, 2
STA 1, SP, 2
DSZ QC, 2
JMP .+1

143
IMONT:  +1
BLK  5

; CHECKS FOR VALID DIGIT

VALDG:  ISZ  RTST, 2
        STA  3, ORTST, 2
        LDA  3, C71, 2
        ADCZ#  3, 0, SZC
        JSR  @RTER, 2
                AC0 > 9
        LDA  3, C60, 2
        ADCZ#  0, 3, SZC
        JSR  @RTER, 2
                AC0 < 0
        JSR  @RTN, 2
                AC0 DIGIT

; READS 5 DIGITS, ACCUMULATES IN AC1

NMNCH:  ISZ  RTST, 2
        STA  3, ORTST, 2
        JSR  @GCHAP, 2
        JSR  @RTER, 2
        .SYSTEM
        .PCHAR
        JSR  @RTER, 3
        JMP  NM1

; READS 4 DIGITS, ACCUMULATES IN AC1, EXPECTS RT ADJ ASCII CHAR IN AC0

NMUCH:  ISZ  RTST, 2
        STA  3, ORTST, 2
        NM1:  SUB  1, 1
        JSR  VALDG
                DIGIT?
        JSR  NM3
                NO
        MOY  0, 1
        LDA  0, C60, 2
        SUB  0, 1
        LDA  3, FOUR, 2
        STA  3, X17T, 2
        NM2:  JSR  @GCHAP, 2
                NEXT DIGIT
        JSR  NM3
        .SYSTEM
        .PCHAR
        JSR  @RTER, 3
        MOY  3, 2
        JSR  VALDG
                DIGIT?
        JSR  NM3
                NO
        MOVZL  1, 3
        MOVZL  3, 3
        ADDZL  3, 1
        LDA  3, C60, 2
        SUB  3, 0
        ADD  0, 1
        DSZ  X17T, 2
        JMP  NM2
        JSR  @GCHAP, 2
        JSR  NM3
        .SYSTEM
        .PCHAR
        JSR  +1

NM3:  LDA  3, C1, 2
SUBR 0.3:SZR ; CARRIER RETURN
JSR @RTF.2 ; NO
LDA 0. LF.2
; SYSTEM
; FCHAR
JSR @RTF. 3
MOV 3, 2
JSR @RTN. 2
INITIALIZE SAVE FILES

.TITLE INSVF
.NREL
.TXTM 1
.ENT INSVF
.EXTN DSKFL

INSVF:
ISZ RTST, 2
LDA 3, IBUF, 2
LDA 0, SVFCN, 2
STA 0, IO, 3
SUBZL 0, 0
STA 0, IFCN, 3
LDA 0, IOSTW, 2
STA 0, STND, 3
LDA 0, IOCNT, 2
STA 0, ICT, 3
LDA 0, IOST, 2
STA 0, IVEC, 3
JSR @DSKFP
JSR @RTEP, 2
LDA 3, IBUF, 2
LDA 0, FDSTW, 2
STA 0, STND, 3
LDA 0, FDCNT, 2
STA 0, ICT, 3
LDA 0, FDST, 2
STA 0, IVEC, 3
JSR @DSKFP
JSR @RTEP, 2
LDA 3, IBUF, 2
LDA 0, EDSTW, 2
STA 0, STND, 3
LDA 0, EDCNT, 2
STA 0, ICT, 3
LDA 0, EDST, 2
STA 0, IVEC, 3
JSR @DSKFP
JSR @RTEP, 2
LDA 3, IBUF, 2
LDA 0, DSTW, 2
STA 0, STND, 3
LDA 0, DNCNT, 2
STA 0, ICT, 3
LDA 0, DST, 2
STA 0, IVEC, 3
JSR @DSKFP
JSR @RTEP, 2
LDA 3, IBUF, 2
LDA 0, RSSTW, 2
STA 0, STND, 3
LDA 0, PSCNT, 2
STA 0, ICT, 3
LDA 0, RSST, 2
STA 0, IVEC, 3
JSR @DSKFP
JSR @RTEP, 2
LDA 3, IBUF, 2
LDA 0, ONSTW, 2
STA 0, STMD, 3
LDA 0, ONCNT, 2
STA 0, ICT, 3
LDA 0, OHST, 2
STA 0, IVEC, 3
JSP $00SKFP
JSR @RTFP, 2
JSR @PTN, 2

DSKFP: DSKFL
.TITL    LEVEL
.ENTER    AG1, AG2
. NREL

AG1:
ISZ     RTST, 2
STA    3, @RTST, 2
LDA    1, OP1DT, 2
MOV    1, 3
LDA    0, PBMSK, 2
AND    0, 1
STA    1, OP1DT, 2
MOVS   3, 3
AND    0, 3
STA    3, OP1LY, 2
JSR    @RTN, 2

AG2:
ISZ     RTST, 2
STA    3, @RTST, 2
LDA    1, OP2DT, 2
MOV    1, 3
LDA    0, PBMSK, 2
AND    0, 1
STA    1, OP2DT, 2
MOVS   3, 3
AND    0, 3
STA    3, OP2LY, 2
JSR    @RTN, 2
.LIGHT LOADPACK
.ENT LDPK1, LDPK2, LDPK3, SFLDP
.EXTD GET, PUT

SFLDP: SUBZL 1, 1
STA 1, SAV, 2

LDPK1: LDA 1, PACK1, 2 ; INITIALIZE LOADPACK ROUTINE
STA 1, PACTL, 2 ; FOR PACK1
LDA 1, OPI2T, 2
STA 1, OPI2T, 2
LDA 1, OPI2DL, 2
STA 1, OPI2DL, 2
LDA 1, OPDP, 2
STA 1, OPDP, 2
LDA 1, DEC1, 2
STA 1, OPDE, 2
JMP PAKIT

LDPK2: LDA 1, PACK2, 2 ; INITIALIZE LOADPACK ROUTINE
STA 1, PACTL, 2 ; FOR PACK2
LDA 1, OPI2T, 2
STA 1, OPI2T, 2
LDA 1, OPI2DL, 2
STA 1, OPI2DL, 2
LDA 1, OPDP, 2
STA 1, OPDP, 2
LDA 1, DEC2, 2
STA 1, OPDE, 2
JMP PAKIT

LDPK3: LDA 1, PACK2, 2 ; INITIALIZE LOADPACK ROUTINE
STA 1, PACTL, 2 ; FOR OPERAND THREE
LDA 1, OPI2T, 2
STA 1, OPI2T, 2
LDA 1, OPI2DL, 2
STA 1, OPI2DL, 2
LDA 1, OPDP, 2
STA 1, OPDP, 2
LDA 1, DEC3, 2
STA 1, OPDE, 2

PAKIT: ISZ RTST, 2 ; GET NEXT STORE POSITION
STA 3, RTST, 2 ; SAVE RETURN ADDRESS
LDA 3, OPI2T, 2 ; GET ARGUMENT 1 TYPE
SUB 0, 0 ; GENERATE A ZERO
SUBB 3, O, 2 ; CHECK TO SEE IF EQUAL TO ZERO
JMP COMP ; YES A COMP TO COMP
LDA 0, TEN, 2 ; GET A TEN
STA 0, N, 2 ; STORE IN N
LDA 1, OACLT, 2 ; ZERO-OUT PACK1
STA 1, PAC2, 2
JSR @ZER, 2 ; NO, ZERO OUT PAC1
JSR @RTER, 2
LDA 0, OPDP, 2 ; GET LENGTH OF FIRST ARGUMENT
STA 0, MM, 2 ; STORE IN M
LDA 1, FOUR, 2
STA 1, N, 2
LDA 1, PACT, 2
STA 1, PAC2, 2
JSR @ZER, 2
JSR @RTER, 2

AA: LDA 3, ODP, 2 ; GET BYTE POINTER
STA 3, BVPTP, 2 ; STORE IN BYTE POINTER
JSR @GET ; GET 2 BYTE

149
JSR BRTER.2
LDA 0, MSK2.2 ; GET MASK 17
AND 0, 1 ; SAVE CHAR DESTROY AC1
ISZ PACT.2 ; TO GET PROPER POSITION
ISZ PACT.2 ; FOR STORING OF CHARACTER
ISZ PACT.2
STA 1, PACT.2 ; STORE THE CHARACTER
DSZ PACT.2 ; DECREMENT RECEIVING
DSZ PACT.2 ; FIELD TO GET THE
DSZ PACT.2 ; BEGINNING OF IT
LDA 0, PACT.2 ; GET BEGINNING OF TEMP
STA 0, PAC1.2 ; INITIALIZE
LDA 0, PAC2.2 ; THESE FOR THE ADD ROUTINE
STA 0, PAC2.7
LDA 0, PAC3.2
STA 0, PAC3.2
JSR @ADD.2
JSR @STEP.2
LDA 1, PACK3.2
STA 1, PAC1.2
LDA 1, PAC1.2
STA 1, PACK2.2
JSR @MVKOV.2
JSR @BRTER.2
DSZ MM.2
JMP +2 ; NOT EQUAL TO ZERO
JMP EXIT1 ; EQUAL TO ZERO
JSR @MVEN.2
JSR @BRTER.2
JSZ @DPDP.2
JMP AA
EXIT1: LDA 1, SAV.2
MOLP 1, 1,52C ; TEST TO SEE IF SET
JMP MNAV ; SET
LDA 0, DPPE.2 ; GET DECIMAL PTR
SUBZ 1, 1 ; CLEAR AC1
SUB# 1, 0, SNR ; TEST TO SEE IF ZER0
JMP @RTN.2 ; GIVES RETURN
LDA 1, TEN.2 ; NO SET N TO EIGHT
STA 1, N.2
STA 0, MM.2 ; SET M TO NUMBER OF DEC POS.
JSR @DVI0.2
JSR @BRFR.2
DSZ MM.2 ; DECREMENT DECIMAL POINT
JMP EXIT1+7 ; DO AGAIN TILL ZERO
JMP @RTN.2 ; RETURN
COMP: LDA 1, FIVE.2 ; GET A FIVE
STA 1, N.2 ; STORE IN N
STA 1, MM.2 ; INITIALIZE A COUNTER
LDA 1, PAC2.2 ; ZERO OUT RECEIVING PACK AREA
STA 1, PAC2.2
JSR @ZEP.2
JSR @BRFR.2
LDA 1, PAC2.2 ; GET RECEIVING PACK FIELD
LDA 0, DPPE.2 ; GET DECIMAL POSITION
MOV# 0, 0, SNR ; CHECK TO SEE IT EQUAL TO ZERO
JMP +5 ; YES
ADD 0, 1
STA 1, PAC1.2 ; STORE THE ADDRESS IN PAC1
DSZ PAC1.2
JMP +2
STA 1, PAC1, 2
LDA 0, ODOP, 2  ; GET BYTE POINTER
MOVZ 0, 0  ; CHANGE TO WORD POINTER
STA 0, ODOP, 2  ; STORE BACK OUT
LDA 0, ODOP, 2  ; GET FIRST BINARY WORD
STA 0, APAC1, 2  ; STORE IT
DSZ OPDL, 2  ; DECREMENT LENGTH
JMP .+2  ; NOT ZERO
JMP @RTN, 2  ; ZERO
ISZ ODOP, 2  ; GET NEXT WORD
ISZ PAC1, 2  ; GET NEXT STORE POSITION
DSZ MN, 2  ; DECREMENT COUNTER
JMP .-10  ; DO TILL ZERO
JMP @RTN, 2  ; RETURN TO INTERRUPTER

MMAY:

SUB 1, 1
STA 1, SAV, 2
LDA 1, PACTL, 2
STA 1, PAC1, 2
LDA 0, FOUR, 2
STA 0, N, 2
ADD 1, 0
STA 0, PAC2, 2
JSR @MMAY, 2
JSR @TER, 2
LDA 1, PACTL, 2
STA 1, PAC2, 2
JSR @ZEP2, 2
JSR @TER, 2
JSR @RTN, 2
.TITLE  MATHMOVE
.END

; CALL TO THIS ROUTINE IS (N, PAC1, PAC2)

M3T01:  LDA  1, TEN, 2 ; GET A TEN FOR THE NUMBER OF WORDS TO BE MOVED
STA  1, NT, 2  ; INITIALLY A COUNTER
LDA  1, PAC3, 2 ; GET RECEIVING PACK AREA
STA  1, PAC1, 2 ; STORE IN PAC1, 2
LDA  1, PAC1, 2
STA  1, PAC2, 2
JMP  HERE

M2T03:  LDA  1, TEN, 2 ; TO INITIALLY A COUNTER WITH A TEN
STA  1, NT, 2
LDA  1, PAC2, 2  ; MOVE PACK2 TO PACK3
STA  1, PAC1, 2
LDA  1, PAC3, 2
STA  1, PAC2, 2
JMP  HERE

MOVA:  ISZ  RTST, 2  ; GET NEXT STORE POSITION
STA  3, @RTST, 2  ; SAVE RETURN ADDRESS
LDA  3, N, 2  ; GET NUMBER OF WORDS LONG
STA  3, NT, 2  ; STORE IN A COUNTER
JMP +43

HERE:  ISZ  RTST, 2  ; GET NEXT STORE POSITION
STA  3, @RTST, 2  ; SAVE RETURN ADDRESS
LDA  1, @PAC1, 2  ; GET FIRST ARGUMENT TO BE MOVED
STA  1, @PAC2, 2  ; AND STORE IT IN RECEIVING FIELD
ISZ  PAC1, 2  ; GET NEXT WORD
ISZ  PAC2, 2  ; GET NEXT STORE POSITION
DSZ  NT, 2
JMP  HERE+2  ; NOT ZERO GO TO HERE
JMP  @RTN, 2  ; RETURN
MATHPACKIF

SUB 1, 1
STA 1, CHKWD, 2
JSR @ARG1
JSR @RTEP, 2
JSR @ARG2
JSR @RTEP, 2
JSR @CPARE
JSR @RTEP, 2
LOAD AN EIGHT
STA 1, X4T, 2
STORE IN TEMP. CNTR.
LDA 1, @PAC3, 2
LOAD WORD FROM PAC3
MOV 1, 1, SZR
CHECK IF ZERO
JMP +6
NOT ZERO
DSZ X4T, 2
ZERO, DECREMENT COUNTER
JMP +2
NOT ZER0
JMP INC3
ZERO (@ARG1 = @ARG2)
ISZ PAC3, 2
GET NEXT WORD
JMP -7
DO NEXT WORD
LDA 1, @PAC3, 2
LOAD FIRST WORD OF ANSWER
MOVL 1, 1, SIC
SHIFT BY2 INTO CARRY
JMP INC3
YES (@ARG1 > @ARG2)
JMP INC1
NO (@ARG1 < @ARG2)

INC3: ISZ CHKWD, 2
INCREMENT CHKWD (HILL END UP 3)
INC2: ISZ CHKWD, 2
INCREMENT CHKWD (HILL END UP 2)
INC1: ISZ CHKWD, 2
INCREMENT CHKWD (HILL END UP 1)
LDA 3, RCKH, 2
LOAD REQUEST CHECK
LDA 1, CHKWD, 2
LOAD CHECK WORD
SUB 3, 1, SIC
CHKWD = RCKH
JMP C11+3
RESULT = ZERO
LDA 1, NFLAG, 2
LOAD NOT FLAG
MOV 1, 1, SZR
SEE IF FLAG IS SET
JMP B1
FLAG IS SET
C11: SUB 1, 1
CLEAR AC1
STA 1, T0C, 2
CLEAR T0C
JMP @RTN, 2
RETURN, BAD COMPARE
LDA 1, NFLAG, 2
LOAD NOT FLAG
MOV 1, 1, SZR
SEE IF FLAG IS SET
JMP C11
FLAG IS SET
B1: JMP @RTN, 2
RETURN, GOOD COMPARE
ARG1: LDPK1
ARG2: LDPK2
CPARE: SUBPK
MOVE
EXTN VALID, TYPE1, SETED, EDIT
EXTD GET, PUT
ENT MOVE, B1, AAB
NREL

MOVE: ISZ RTST, 2 ;GET NEXT STORE POSITION
STA 3, RTST, 2 ;SAVE RETURN ADDRESS
LDA 3, OP1DP, 2 ;GET OP1 BYTE PTR
STA 3, PNT1, 2 ;STORE IN PNT1
LDA 3, OP1DL, 2 ;GET LENGTH FOR OP1
STA 3, CNT1, 2 ;STORE CNT1
LDA 3, OP2DP, 2 ;GET RAW DATA BYTE PTR FOR OP2
STA 3, PNT2, 2 ;STORE IN PNT2
LDA 3, OP2DL, 2 ;GET LENGTH OF RECEIVING FIELD
STA 3, CNT2, 2 ;STORE IN CNT2
JSR @, VAL, 2 ;GO TO VALIDITY ROUTINE
JSR @RTER, 2
SUB 0, 0 ;CLEAR ACO
STA 0, EFLAG, 0
JSR @TPVE ;GO TO VALID TYPE ROUTINE
JSR @RTER, 2
LDA 1, EFLAG, 2 ;GET EDIT FLAG
MOV 1, GTC, 1 ;CHECK TO SEE IF SET
B1: JMP @, SETD, 2 ;JMP T SET EDIT ROUTINE
LDA 3, PNT0, 2 ;GET BYTE POINTER
MOV 3, SNR, 1 ;TEST TO SEE IF EQUAL TO ZERO
JMP PT1 ;NOJMP TO PT1
LDA 1, C60, 2 ;YES LOAD IN ASCI1 ZERO
LDA 3, PNT2, 2 ;LOAD IN BYTE PTR
STA 3, BVPTR, 2 ;STORE IN BYTE POINTER
JSR @PUT ;GO TO STORE ROUTINE
JMP @RTER, 2
ISZ PNT2, 2 ;ISZ STORE BYTE PTR
DSZ CNT0, 2 ;DECREMENT CNT1
JMP +7 ;NOT ZERO
DSZ CNT2, 2
JMP +2
JSR @RTN, 2
SUB 1, 1 ;CLEAR AC1
STA 1, PNT0, 2 ;CLEAR ZERO FLAG BYTE POINTER
JMP PT1 ;CNT1 IS ZERO
DSZ CNT2, 2 ;DECREMENT CNT2
JMP PT ;NOT ZERO
JMP @RTN, 2 ;ZERO RUN TIME ERROR
PT1: LDA 3, PNT1, 2 ;GET BYTE PTR SENDING FIELD
STA 3, BVPTR, 2 ;STORE IN BYTE POINTER
JSR @GET ;GO TO GET A BYTE ROUTINE
JMP @RTER, 2
LDA 3, PNT2, 2 ;LOAD IN RECEIVING BYTE PTR
STA 3, BVPTR, 2 ;STORE IN BYTE POINTER
JSR @PUT
JMP @RTER, 2
ISZ PNT1, 2
ISZ PNT2, 2
DSZ CNT1, 2 ;DSZ LENGTH OP1
JMP +2
JMP PT2 ;CHECK FOR PNT1 & 0
DSZ CNT2, 2 ;DSZ LENGTH OP2
JMP PT1
JMP @RTN, 2 ;JMP RUN TIME ERROR
PT2: DSZ CNT2, 2
JMP .+2
JMP @RTN. 2 ; RETURN TO INTERPRETER
LDA 1, OP1DT. 2 ; GET TYPE
LDA 0, FILL. 2 ; GET A SEVEN
SUB# 1, 0, SNR
JMP A0
LDA 0, SIX. 2
SUB# 1, 0, SNR
JMP AA ; ZERO FILL
LDA 1, INDIC. 2 ; GET INDICATOR
LDA 0, FOUR. 2 ; GET A FOUR FOR A TEST
SUB# 1, 0, SNC ; TEST TO SEE IF INDICATOR IS # = 4
JMP AA ; NUMERIC FILL WITH ZEROS
A0:
LDA 1, BLK. 2 ; BLANK FILL
LDA 3, PNT2. 2 ; GET BYTE PTR FOR SECOND OPERAND
STA 3, BVPTR. 2 ; STORE IN BYTE POINTER
JSR @PUT ; STORE BLANK
JMP @RTER. 2
ISZ PNT2. 2
DSZ CNT2. 2 ; DSZ COUNT TWO
JMP -7
JMP @RTN. 2 ; EXIT ROUTINE
AAB:
DSZ CNT2. 2
JMP .+2
JMP B1 ; TOTAL LENGTH HUNT TO ZERO GET END OF EDIT CODE
AA:
LDA 1, C60. 2
LDA 3, PNT2. 2 ; ZERO FILL BYTE PTR FOR 2ND OP.
STA 3, BVPTR. 2 ; STORE BYTE POINTER
JSR @PUT; STORE ZERO FILL
JMP @RTER. 2
ISZ PNT2. 2
DSZ CNT2. 2 ; DSZ COUNT TWO
JMP AA
JMP @RTN. 2
.VALID: VALID
.TPVE: TYPE1
.SETD: SETED
.EDIT: EDITT
IDENTIFICATION DIVISION
PROGRAM-ID. ABC1P
ENVIRONMENT DIVISION
CONFIGURATION SECTION
SOURCE-COMPUTER UNIVAC-1100
OBJECT-COMPUTER UNIVAC-1100
INPUT-OUTPUT SECTION
FILE-CONTROL

DATA DIVISION
FILE SECTION
FD COBIN
LABEL RECORDS OMITTED
001000 SIZE IS 80
001000 01 SOR
001020 02 FILLER PIC 9(6)
001020 02 CONT PIC X
001040 02 PARA PIC X
001050 03 PARA PIC X(4)
001060 03 INSTR
001070 04 INSA PIC X(2)
001080 04 FILLER PIC X(2)
001090 04 INSR PIC X(2)
001100 02 DATAR PIC X(6)
001110 02 IGN-1 PIC X(6)
001120 02 COND
001130 03 DATA2 PIC X(6)
001140 03 DATA3 PIC X(6)
001150 03 DATA4 PIC X(6)
001160 03 DATA5 PIC X(6)
001170 03 DATA6 PIC X(6)
001180 03 DATA7 PIC X(6)
001190 02 STEND PIC X
001200 02 PFFILL PIC X(6)
001210 FD MINI?
RECORD 30
LABEL RECORD OMITTED
001230 SIZE IS 80
001240 01 MINI
001250 02 MASM PIC X(30)
FD NIV
RECORD 23
LABEL RECORD OMITTED
01 POUTS PIC X(23)
FD LITF
RECORD 68
LABEL RECORD OMITTED
01 LITR PIC X(60)
FD PRNF
RECORD 86
LABEL RECORD OMITTED
01 PRNF
02 PRN PIC X(4)
02 FILLER PIC X(2)
02 PRF PIC X(80)

156
001270 WORKING-STORAGE SECTION
  77 DEFG  PIC 9 VALUE ZERO
  77 FLOG  PIC X
  77 BKLOC  PIC 9(5) VALUE ZERO
  77 COLRC  PIC 9(5) VALUE 0
  001300 SIZE IS  26
  001310 01 DUORD
  001320 02 IDLBL  PIC XX(5)
  001330 02 ADR  PIC XX(5)
  001340 02 IDPIC  PIC X
  001360 02 IDLNQ  PIC XX(4)
  001370 02 IDDEC  PIC XX(2)
  001380 02 PAK  PIC 9
  001390 02 SVNQ  PIC 9
  001400 02 USGE  PIC 9
  001410 02 SINC  PIC 9
  001420 02 OCP  PIC 9
  001421 02 EDR  PIC XX(2)
  001430 02 VLR  PIC XX(3)
  001440 SIZE IS  23
  001450 01 PHORD
  02 CSEP  PIC XX(4)
  02 FLOP  PIC 9
  02 RTNQ1  PIC XX(3)
  02 PBDV
  03 FRST1  PIC XX(5)
  03 SECO1  PIC XX(5)
  03 LAST1  PIC XX(5)
  001500 SIZE IS  3000
  001570 01 EDITT
  02 EDT OCCURS 20 TIMES
  03 FILLER PIC XX(18)
  03 BB  PIC XX(6)
  03 AA  PIC XX(6)
  001620 SIZE IS  2000
  001630 01 JNPTA
  02 JNPTA OCCURS 258
  03 JNPSQ  PIC XX(5)
  03 JNPHA  PIC XX(5)
  001650 SIZE IS  60
  001670 01 TAB1
  02 T1  PIC X OCCURS 62 TIMES
  *SIZE IS  119
  001710 01 TAB2
  02 FILLER PIC XX(11)
  001730 02 TA
  03 T2  PIC XX(6) OCCURS 18
  001750 SIZE IS  6
  001760 01 TAB3
  001770 02 T3  PIC X OCCURS 6 TIMES
  001780 SIZE IS  676
  001790 01 ONET
  02 ONEDE OCCURS 50 TIMES
  001810 03 NA  PIC XX(5)
  03 FILLER PIC XX(5)
  03 IPICT  PIC X
  03 IDLN  PIC XX(4)
  001840 03 FILLER PIC XX(12)
  001850 SIZE IS  6
  001860 01 IGNC-2
  001870 02 COL-1  PIC X
001690  02 COL-2  PIC  X               .00012400
001690  02 COL-3  PIC  X               .00012400
001900  02 FILLER  PIC  X(23)            .00012400
001910+SIZE IS  4                      .00012400
001920  01 PPNAH                        .00012400
001930  02 PPCODE  PIC  X               .00012400
001940  02 PPNO  PIC  X(23)             .00012400
001950+SIZE IS  4                      .00012400
001960  01 VNM                          .00012400
001970  02 TST  PIC  X                  .00012400
001980+SIZE IS  4                      .00012400
001990  01 LITCN  PIC  X(24) VALUE ZERO .00012400
002000  01 SUBS1                        .00012400
002010  02 SUC                          .00012400
002020  03 S1  PIC  9                   .00012400
002030  03 S2  PIC  9                   .00012400
002040  03 S3  PIC  9                   .00012400
002050  02 RDNLG  PIC  9                .00012400
002060  02 SUBP  PIC  X(24)             .00012400
002070  02 SBS  PIC  X(24)              .00012400
002080  02 SUBZ  PIC  X(24)             .00012400
002090  02 OCLN  PIC  X(24)             .00012400
002100  02 SUB  PIC  X(24)              .00012400
002110  02 SURB  PIC  X                 .00012400
002120  02 FLGNY  PIC  9                .00012400
002130  02 LEVEL  PIC  X(22)            .00012400
002140  02 FDCC  PIC  X(22)             .00012400
002150  02 GF  PIC  X(22)               .00012400
002160  02 UF  PIC  9                   .00012400
002170  02 SF  PIC  9                   .00012400
002180  02 G2  PIC  X(22)               .00012400
002190  02 G3  PIC  X(22)               .00012400
002190  02 G4  PIC  X(22)               .00012400
002190  02 G5  PIC  X(22)               .00012400
002190  02 G6  PIC  X(22)               .00012400
002190  02 G7  PIC  X(22)               .00012400
002190  02 G8  PIC  X(22)               .00012400
002190  02 G9  PIC  X(22)               .00012400
002190  02 CG5  PIC  X(24)              .00012400
002190  02 CG6  PIC  X(24)              .00012400
002190  02 CG7  PIC  X(24)              .00012400
002190  02 CG8  PIC  X(24)              .00012400
002190  02 CG9  PIC  X(24)              .00012400
002190  02 ODFLG  PIC  9                .00012400
002190  02 SUFLG  PIC  9                 .00012400
01 IDTB1
  02 IOTB2  OCCURS  15 TIMES            .00012400
  03 RELF  PIC  X(23)                   .00012400
  03 RELF  PIC  X(23)                   .00012400
  03 FNM  PIC  X(25)                    .00012400
  03 DTYPE  PIC  X(25)                  .00012400
  03 DFNAM  PIC  X(25)                  .00012400
01 JMP2T
  02 J502  PIC  X(25) OCCURS  400       .00012400
  01 DSRUA
    02 XYZS  PIC  X(25) OCCURS  3        .00012400
002200 PROCEDURE DIVISION              .00012400
002210+-----------------------------------------------.00012400
002220+  OPENES THE INPUT AND OUTPUT FILES          .00012400
          .00012400
158
ADD 1 TO SUB
GO TO SELG
IF INSTR EQUAL @ORGANIZATION@
GO TO ORASG
IF INSTR EQUAL @ACCESS@
GO TO ACASG
IF PARA EQUAL @DATA@
MOVE ZERO TO SUB
GO TO ASPRT

SELG
MOVE ZERO TO RELF XSUBC
MOVE DATA1 TO FNAM XSUBC
MOVE DATA3 TO DTYP XSUBC
MOVE SPACE TO DFNAM XSUBC
IF DATA4 NOT EQUAL SPACE
MOVE DATA4 TO DFNAM XSUBC
GO TO ASSGL

ORASG
IF DATA1 EQUAL @RELATION@
MOVE 100 TO RELF XSUBC
GO TO ASSGL

ACASG
IF DATA2 EQUAL @RANDOM@
ADD 10 TO RELF XSUBC
IF DATA2 EQUAL SPACE
GO TO ASSGL
IF DATA4 EQUAL SPACE
GO TO ASSGL
MOVE DATA4 TO KEYN XSUBC
ADD 1 TO RELF XSUBC
GO TO ASSGL

ASPRT
ADD 1 TO SUB
IF IOTR2 XSUBC EQUAL SPACE
GO TO ASXIT
MOVE IOTR2 XSUBC TO MIASM
PERFORM WRIT THRU WRXIT
GO TO ASPRT

ASXIT
EXIT

002000 ASXIT
002810 EXIT
002820 DATA DIVISION CONTROL
002840 DDNOW

002870 MOVE @ 30 TO MIASM
002260 MOVE SPACE TO EDITT
002960 PERFORM RDCOB THRU CORDX
002980 D77
002910 IF PARA EQUAL @77 @
002920 MOVE @01 @ TO PARA
002930 IF PARA EQUAL AFD @
002940 PERFORM FDD THRU FD0X
002950 PERFORM WRIT THRU WRXIT
002960 IF PARA EQUAL @91 @
002970 PERFORM ONE THRU ONEX
002980 GO TO D77
002990 IF PAPAG EQUAL @WORKING-STOPAGE @
MOVE @ 40 TO ILLBL
MOVE BKLOC TO ADR
003820  MOVE  DWORD TO  MIASM
003830  PERFM  WRIT THRU  WRXIT
003840  MOVE  @A1 TO  LAST1
003850  GO TO  DDRD
003860  IF  PARAG NOT  EQUAL @PROCEDURE DIVISION
003868  GO TO  DDRD
003870  MOVE  0  TO  IDLAL
003880  MOVE  COPLC TO  ADD
003890  MOVE  DWORD TO  MIASM
0038A0  PERFM  WRIT THRU  WRXIT
0038B0  MOVE  1  TO  SUB
0038C0  EVDDPA
0038D0  IF  EDJ  XSUBC NOT  EQUAL SPACE
0038E0  MOVE  EDJ  XSUBC TO  MIASM
0038F0  PERFM  WRIT THRU  WRXIT
003900  ADD  1  TO  SUB
003910  GO TO  EVDDPA
003920  DOXIT
003930  EXIT

For PROCEDURE DIVISION CONTROL

003940  PROC
003950  MOVE  SPACE TO  JMP2T
003960  MOVE  SPACE TO  EDJTT
003970  MOVE  1  TO  PPDN
003980  MOVE  ZERO TO  FLOP
003990  MOVE  SPACE TO  TABS2
0039A0  MOVE  #0 TO  TST
0039B0  MOVE  #0 TO  PPCOD
0039C0  MOVE  ZERO TO  SUBS1
0039D0  MOVE  ZERO TO  PTH01
0039E0  MOVE  ZERO TO  PROV
0039F0  OPEN  OUTPUT  PDI

0039F8  PROC1
004000  IF  RDFLG NOT  EQUAL 1
004010  PERFM  RDODB THRU  CORODX
004020  MOVE  ZERO TO  RDFLG
004030  ADD  1  TO  SEO
004040  IF  PARAG NOT  EQUAL SPACE
004050  MOVE  PARAG TO  JMPFA XSUBRC
004060  MOVE  SEO TO  JNPSO XSUBRC
004070  SUBTRACT  1  FROM  SEO
004080  ADD  1  TO  SUBER
004090  GO TO  PROC1
0040A0  IF  BAK  EQUAL ZERO
0040B0  GO TO  VRPRP
0040C0  IF  STEND  EQUAL SPACE
0040D0  GO TO  VRPRP
0040E0  MOVE  ZERO TO  S2
0040F0  MOVE  IGN-1 TO  IGN-2
004100  IF  COL-1  EQUAL  #0
004110  ADD  1  TO  SEO
004120  MOVE  1  TO  S2
004130  GO TO  FLOD
004140  IF  CDFP  EQUAL SPACE
004150  GO TO  FLOPD
004160  GO TO  FLOP
004170  MOVE  CDFP TO  TAP1
004180  MOVE  1  TO  SUB

PLOOP
IF T1 *SUBC EQUAL 0 0
MOVE 1 TO S2
ADD 1 TO SEQ
GO TO PLOPD
IF SUB NOT GREATER THAN 30
ADD 6 TO SUB
GO TO PLOPD
ADD 1 TO SEQ
MOVE SEQ TO JS02 XPPNC
003560 SUBTRACT 1 FROM SEQ
IF S2 EQUAL 1
SUBTRACT 1 FROM SEQ
MOVE ZERO TO BAK
ADD 1 TO PPNO
003590 VRPPR
003600 PERFORM FCVPR THRU VRAXT
MOVE PWORD TO POUTS
PERFORM WMS THRU WRXT
MOVE SPACE TO TAB2
MOVE SPACE TO EDITT
003630 MOVE ZFPR TO PBDY
003640 GO TO PROC1
003650 PRXIT
003660 EXIT
003670***************************************************************************
003680 PERFORM FDCOB THRU CORDX READS ONE 11K CARD IMAGE
003690 AND
003700***************************************************************************
003710 FINIS PRINTS OUT THE VALUE TABLE AND IS THE
003720***************************************************************************
003730***************************************************************************
003740***************************************************************************
003750***************************************************************************
003760***************************************************************************
003770***************************************************************************
003780***************************************************************************
003790***************************************************************************
003800***************************************************************************
003810***************************************************************************
003820***************************************************************************
003830***************************************************************************
003840***************************************************************************
003850***************************************************************************
003860***************************************************************************
003870***************************************************************************
003880***************************************************************************
003890***************************************************************************
003900***************************************************************************
003910***************************************************************************
003920***************************************************************************
003930***************************************************************************
003940***************************************************************************
003950***************************************************************************
003960***************************************************************************
003970***************************************************************************
003980***************************************************************************
003990***************************************************************************
004000***************************************************************************
004010***************************************************************************
004020***************************************************************************
004030***************************************************************************
004040***************************************************************************
004050***************************************************************************
004060***************************************************************************
004070***************************************************************************
004080***************************************************************************
004090***************************************************************************
004100***************************************************************************
004110***************************************************************************
004120***************************************************************************
004130***************************************************************************
004140***************************************************************************
004150***************************************************************************
004160***************************************************************************
004170***************************************************************************
004180***************************************************************************
004190***************************************************************************
004200***************************************************************************
004210***************************************************************************
004220***************************************************************************
004230***************************************************************************
004240***************************************************************************
004250***************************************************************************
004260***************************************************************************
004270***************************************************************************
003920  STOP       RUN
003930  CORDEX
003940  EXIT
003950  WRITE THRU XRXT  WRITES ONE LINE
003960  WRITE THRU XRXT  WRITES ONE LINE
003970  WRITE
003980  WRIT
004000  MOVE SPACE TO MINI
004010  XRXT
004020  EXIT
004300  WRIT
004310  WRITE POUTS
004320  MOVE SPACE TO POUTS
004330  WRIT
004340  EXIT
004400  P
004410  READ PDIV END GO TO PEND
004420  MOVE POUTS TO PEND
004430  IF RTNO1 EQUAL 070
004440  GO TO PROCE
004450  IF RTNO1 EQUAL 130
004460  GO TO PROCE
004470  IF RTNO1 EQUAL 120
004480  GO TO PROCB
004490  IF RTNO1 EQUAL 190
004500  GO TO PPCE
004510  IF RTNO1 EQUAL 281
004520  GO TO PROCE
004530  IF RTNO1 GREATER THAN 241
004540  GO TO P1
004550  IF RTNO1 LESS THAN 200
004560  GO TO PROCE
004600  P1
004610  MOVE PEND TO MIASM
004620  PERFORM WRIT THRU XRXT
004630  GO TO P
004650  PROCE
004660  MOVE 1 TO SUB
004670  MOVE LAST1 TO IGN-2
004680  IF COL-1 EQUAL A+0
004690  MOVE LAST1 TO PPNAM
004700  MOVE J502 XPNNOC TO LAST1
004710  MOVE PEND TO MIASM
004720  PERFORM WRIT THRU XRXT
004730  GO TO P
004740  PERFORM SEDFD THRU SFOD
004750  MOVE JMP5A XSUBC TO LAST1
004760  MOVE PEND TO MIASM
004770  PERFORM WRIT THRU XRXT
004780  GO TO P
004790  PROCB
004800  MOVE 1 TO SUB
004810  MOVE FIRST1 TO IGN-2
004820  PERFORM SEDFD THRU SFOD
004830  MOVE JMP5A XSUBC TO FIRST1
004840  MOVE SECD1 TO IGN-2
004850  PERFORM SEDFD THRU SFOD
004860  MOVE JMP5A XSUBC TO SECD1
004870  MOVE PEND TO MIASM
004880  PERFORM WRIT THRU XRXT
PERFORM WRIT THRU WRXIT
GO TO P
PEND
EXIT
SE0FD
IF JMPNA XSUBC EQUAL SPACE
MOVE @PPNA REFERENCED NOT FOUND TO MIASM
PERFORM WRIT THRU WRXIT
GO TO SE0D
IF JMPNA XSUBC EQUAL IGN-2
GO TO SE0D
ADD 1 TO SUB
GO TO SE0FD
SE0D
EXIT

004030 FEED PROCESSING

004050 FDD

004060 M0VE 1 TO PPNO

004080 M0VE 00 TO RTNO1

004100 M0VE ZERO TO DWORD

004120 M0VE PARA EQUAL @FD @

004140 M0VE INSTR TO IDLBL

004160 M0VE FDRO

004180 M0VE DATA1 TO IGN-2

004200 M0VE OOCR THRU OCPXT

004220 M0VE SUC TO PPNO

004240 M0VE FDRO

004260 M0VE DATA2 EQUAL @COMITTE@

004280 M0VE DATA2 EQUAL @STANDAD@

004300 M0VE @1 TO IDDEC

004320 M0VE DATA2 TO FDRO

004340 M0VE PARA NOT EQUAL @1 @

004360 M0VE PPNO GREATER THAN 1

004380 MULTIPY RTNO1 BY PPNO GIVING IDLNG

004400 MOVE BKLOC TO ADR

004420 ADD IDLNG TO BKLOC

004440 MOVE 8 TO IDPIC

004460 MOVE DWORD TO MINI

004480 GO TO FD0X

004500 PERFORM ROCOB THRU CORDX

004520 GO TO FAD

004540 EXIT

105920 HANDLES 01S, 02S, 03S, 04S, 05S AND 77S

105940 ONE

105960 MOVE ZERO TO DWORD

105970 IF DATA1 NOT EQUAL @PIC @
010880 GO TO ONEAA
010890 PERFORM PI THRU PIX
011440 MOVE INSTR TO IDLBL
011450 IF CODF NOT EQUAL SPACE
011460 PERFORM CODE36 THRU CODEX
011480 MOVE CORLC TO ADR
011490 ADD ILDNG TO COPLC
011500 MOVE 9 TO OCR
011510 MOVE DWORD TO MINSI
011520 PERFORM WRIT THRU WRXIT
011530 IF DATA1 EQUAL @REDEFI@
011540 SUBTRACT ILDNG FROM CORLC
011550 IF LAST1 EQUAL $0001
011560 GO TO ONE
011570 IF PARA EQUAL @G1 @
011580 SUBTRACT ILDNG FROM CORLC
011590 GO TO ONE
011600 GO TO ONE
011610 ONEAA
011620 MOVE ZERO TO ONET
011630 MOVE ZERO TO SUBSI
011640 MOVE 1 TO SUB
011650 MOVE INSTR TO IDLBL
011660 MOVE CORLC TO ADR
011670 IF CODF NOT EQUAL SPACE
011680 PERFORM CODEG THRU CODEGX
011690 MOVE 9 TO OCR
011700 ONEA
011710 MOVE DWORD TO ONEDE XSUBC
011720 ADO 1 TO SUB
011730 IF SUB EQUAL $1
011740 MOVE @810BIG@ TO MIASTM
011750 PERFORM WRIT THRU WRXIT
011760 GO TO ONE
011770 PERFORM RDCOB THRU CORDX
011780 MOVE ZERO TO DWORD
011790 IF PARA EQUAL @WORK@@
011800 GO TO ONEJ
011810 IF PARA EQUAL @PROC@@
011820 GO TO ONEJ
011830 IF PARA EQUAL @F@D @
011840 GO TO ONEJ
011850 IF PARA EQUAL @G1 @
011860 GO TO ONEJ
011870 MOVE INSB TO IDLBL
011880 IF INSB EQUAL @FILLER @
011890 MOVE SPACE TO IDLBL
011900 IF DATA1 NOT EQUAL @PIC @
011910 PERFORM PIS NOT EQUAL PIEXT
011920 GO TO ONEA
011930 PERFORM PI THRU PIX
011940 MOVE INSA TO LEVL
011950 IF GF NOT EQUAL ZERO
011960 PERFORM ONEGF THRU ONFX
011970 IF CODF NOT EQUAL SPACE
011980 PERFORM CODE36 THRU CODEX
011990 IF FOCF NOT EQUAL ZERO
012000 IF FOCF LESS THAN LEVL
012010 MOVE OCLN TO VLR
ADD IDNLG TO OCLN GIVING IDNLG
011540
MULTIPLY IDNLG BY PPNO GIVING IDNLG
011550
MOVE 1 TO OCR
011560
GO TO ONEOR
MOVE 70 TO FOCC
MOVE ZERO TO OCLN
011580
IF QR EQUAL ZERO
011590
GO TO ONE3
IF QR LESS THAN LEVL
011610
ADD IDNLG TO SUBR
GO TO ONE3
MOVE SUBR TO INLN XORC
011640
MOVE ZERO TO SUBR
011660
MOVE ZERO TO QR
ONE3
IF QR EQUAL ZERO
011690
GO TO ONE4
IF QR LESS THAN LEVL
011700
ADD IDNLG TO SUA
GO TO ONE4
MOVE SUA TO INLN XORC
011730
MOVE ZERO TO SUA
011750
MOVE ZERO TO QR
ONE4
IF QR EQUAL ZERO
011770
GO TO ONE5
IF QR LESS THAN LEVL
011790
ADD IDNLG TO TNU
GO TO ONE5
MOVE TNU TO INLN XORC
011820
MOVE ZERO TO TNU
011840
MOVE ZERO TO QR
ONE5
IF QR EQUAL ZERO
011860
GO TO ONE6
IF QR LESS THAN LEVL
ADD IDNLG TO CGS
GO TO ONE6
MOVE CGS TO INLN XORC
MOVE ZERO TO CGS
MOVE ZERO TO QR
ONE6
IF QR EQUAL ZERO
011890
GO TO ONE7
IF QR LESS THAN LEVL
ADD IDNLG TO CG7
GO TO ONE7
MOVE CG7 TO INLN XORC
MOVE ZERO TO CG7
MOVE ZERO TO QR
ONE7
IF QR EQUAL ZERO
011920
GO TO ONE8
IF QR LESS THAN LEVL
ADD IDNLG TO CG7
GO TO ONE8
MOVE CG7 TO INLN XORC
MOVE ZERO TO CG7
MOVE ZERO TO QR
ONE8
IF  G8  EQUAL  ZERO
  GO TO  ONE9
IF  G8  LESS  THAN  LEVL
  ADD  ID LNG TO  CG8
  GO TO  ONE9
  MOVE  CG8  TO  IDLN  XG8C
  MOVE  ZERO  TO  CG8
  MOVE  ZEPO  TO  G8
ONE9
IF  G9  EQUAL  ZERO
  GO TO  ONEE
IF  G9  LESS  THAN  LEVL
  ADD  ID LNG TO  CG9
  GO TO  ONEE
  MOVE  CG9  TO  IDLN  XG9C
  MOVE  ZERO  TO  CG9
  MOVE  ZERO  TO  G9
011859  ONEE
  ADD  ID LNG TO  SUBZ
011870  MOVE  COPLC  TO  ADR
011880  ADD  ID LNG TO  COPLC
  IF  FOCC  EQUAL  70
  MOVE  ZERO  TO  FOCC
  PERFORM  OCCR  THRU  OCGPX
011890  GO TO  ONEA
011900  ONEA
011910  MOVE  SUBZ  TO  IDLN  X1C
011920  IF  G2  NOT  EQUAL  ZERO
  ADD  SUBR  TO  IDLN  X02C
011930  IF  G3  NOT  EQUAL  ZERO
  MOVE  SUA  TO  IDLN  XG3C
011940  IF  G4  NOT  EQUAL  ZERO
  MOVE  TNU  TO  IDLN  XG4C
011950  IF  G5  NOT  EQUAL  ZERO
  MOVE  CG5  TO  IDLN  XG5C
  IF  G6  NOT  EQUAL  ZERO
  MOVE  CG6  TO  IDLN  XG6C
  IF  G7  NOT  EQUAL  ZERO
  MOVE  CG7  TO  IDLN  XG7C
  IF  G8  NOT  EQUAL  ZERO
  MOVE  CG8  TO  IDLN  XG8C
  IF  G9  NOT  EQUAL  ZERO
  MOVE  CG9  TO  IDLN  XG9C
  IF  FOCC  NOT  EQUAL  ZERO
  PERFORM  OCCR  THRU  OCGPX
011990  MOVE  1  TO  SUB
011990  ONEK
012000  MOVE  ONEDE  XSURC  TO  MIASM
012010  PERFORM  WRIT  THRU  WRXIT
012020  ADD  1  TO  SUB
012030  IF  ONEDE  XSUBC  NOT  EQUAL  ZERO
  GO TO  ONEK
  IF  DATA1  EQUAL  #PDEFI|
  SUBTRACT  IDLN  X1C  FROM  CORLC
012050  IF  LAST2  EQUAL  #0001
  GO TO  ONEK
  IF  PARA  EQUAL  #01  @
  SUBTRACT  IDLN  X1C  FROM  CORLC
012090  GO TO  ONEK
012100  ONEK
012110  EXIT
VERB PROCESSING

004480 FCVPB
004490 IF INSTP EQUAL TO @MOVE B
004500 PERFORM MOVE1 THRU MVXIT
004510 GO TO VBPX B
004520 IF INSTP EQUAL TO @IF B
004530 PERFORM IF-1 THRU IFXIT B
004540 GO TO VBPX B
004550 IF INSTP EQUAL TO @GO TO B
004560 PERFORM JMP-1 THRU JPXIT B
004570 GO TO VBPX B
004580 IF INSTP EQUAL @GO B
004590 PERFORM JMP-1 THRU JPXIT B
004600 GO TO VBPX B
004610 IF INSTP EQUAL TO @PERFORM B
004620 PERFORM PERF1 THRU PFXIT B
004630 GO TO VBPX B
004640 IF INSTP EQUAL TO @ADD B
004650 PERFORM RT-0 THRU OPTXT B
004660 GO TO VBPX B
004670 IF INSTP EQUAL TO @SUBTRACT B
004680 PERFORM RT-0 THRU OPTXT B
004690 GO TO VBPX B
004700 IF INSTP EQUAL @EXIT B
004710 PERFORM EXE1 THRU EXIT B
004720 GO TO VBPX B
004730 IF INSTP EQUAL TO @MULTIPLY B
004740 PERFORM RT-0 THRU OPTXT B
004750 GO TO VBPX B
0048 60 IF INSTP EQUAL TO @DIVIDE B
004770 PERFORM RT-0 THRU OPTXT B
004780 GO TO VBPX B
004790 IF INSTP EQUAL TO @READ B
004800 PERFORM READ1 THRU RDXIT B
004810 GO TO VBPX B
004820 IF INSTP EQUAL TO @WRITE B
004830 PERFORM RT1 THRU RIXIT B
004840 GO TO VBPX B
004850 IF INSTP EQUAL TO @OPEN B
004860 PERFORM OPEN1 THRU OPXIT B
004870 GO TO VBPX B
004880 IF INSTP EQUAL TO @CLOSE B
004890 PERFORM CLOSI THRU CLXIT B
004900 GO TO VBPX B
004910 IF INSTP EQUAL TO @STOP B
004920 PERFORM STOP1 THRU STXIT B
004930 GO TO VBPX B
004940 IF INSTP EQUAL TO @DISPLAY B
004950 PERFORM DPLAY1 THRU DXIT B
004960 GO TO VBPX B
004970 IF INSTP EQUAL TO @ACCEPT B
004980 PERFORM ACFP1 THRU ACXIT B
004990 GO TO VBPX B
005000 IF INSTP EQUAL TO @ENTER B
005010 PERFORM ENTR1 THRU ENXIT B
005020 GO TO VBPX B
005030 IF INSTP EQUAL @DELETE B
005040 PERFORM DELT1 THRU DELTX B
005050 GO TO VBPX B
IF INSTR EQUAL @CALL @ CALLI THRU CALLX GO TO VRXT
005030 IF INSTR EQUAL TO @REWRITE @
005040 PERFORM RTPAR THRU RTXIT
005060 EXIT

ALL MODULES BELOW HERE ARE @SUB-Routines@

005070

IF IGN-1 EQUAL SPACE
005100 MOVE DATA1 TO FRST1
005110 MOVE 016 TO RTN01
005120 GO TO ACXIT
005230 MOVE 4 TO SB5
005240 MOVE SOR TO TAB2
005250 MOVE 280 TO RTN01
005260 MOVE TAB1 TO FRST1
005270 MOVE PHORD TO POUTS
005280 PERFORM WMS THRU WMXT
005290 ADD 1 TO SEX
005360 MOVE 016 TO RTN01
005370 MOVE DATA1 TO FRST1
005380 EXIT

CLOSE VEPB

005410

005420

005430 CLOS1
005440 MOVE 070 TO RTN01
005450 MOVE DATA1 TO FRST1

005550 EXIT

DISPLAY VEPB

005620

005630

005640 DPLAY
005660 PERFORM BKDW THRU BPKXT
005670 IF BB X1C NOT EQUAL SPACE
005680 MOVE 250 TO RTN01
005690 MOVE BR X1C TO FRST1
005700 MOVE PHORD TO POUTS
005710 PERFORM WMS THRU WMXT
005720 ADD 1 TO SEX
005730 MOVE AA X1C TO FRST1
005740 MOVE 048 TO RTN01
005750 IF AA X2C EQUAL SPACE
005760 GO TO DPXIT
005770 MOVE PHORD TO POUTS
005780 PERFORM WMS THRU WMXT
005790 ADD 1 TO SEX
005800 IF BB X2C NOT EQUAL SPACE
005810 MOVE 250 TO RTN01
005820 MOVE BR X2C TO FRST1
005830 MOVE PHORD TO POUTS
005840 PERFORM WMS THRU WMXT
005850

169
PERFORM WMS THRU WMXT
ADD 1 TO SEQ
GO TO JWR
006950 JPXIT
006950 EXIT
007210 OPEN1
007230 OPEN VERA
007240 OPEN1
007250 IF DATA1 NOT EQUAL @OUTPUT@
007260 GO TO DI-10
007270 MOVE 112 TO RTNO1
007280 MOVE DATA2 TO FRST1
007290 GO TO OPXIT
007300 DI-10
007310 IF DATA1 EQUAL @INPUT @
007320 MOVE 110 TO RTNO1
007330 IF DATA1 EQUAL @I-O @
007340 MOVE 111 TO RTNO1
007350 MOVE IGN-1 TO FRST1
007360 OPXIT
007370 EXIT
007380 STOP VERA
007390 STOP1
007400 MOVE 150 TO RTNO1
007410 IF DATA1 EQUAL @RUN @
007420 GO TO STXIT
007430 MOVE 3 TO SBS
007440 MOVE SOR TO TAB2
007450 PERFORM D21 THRU D6X
007460 MOVE LITCN TO TNU
007470 MOVE VNAM TO FRST1
007480 MOVE 280 TO RTNO1
007490 STXIT
007500 EXIT
007510 STOP1
007520 MOVE VERA
007530 MOVE VERB
007540 MOVE1
007550 MOVE BRKW THRU BRKXT
007560 MOVE BB X1C TO FPST1
007570 MOVE BB X3C TO SECD1
007580 MOVE SPACE TO LAST1
007590 IF PRDY EQUAL SPACE
007600 GO TO MV1
007610 MOVE 250 TO RTNO1
007620 MOVE PHORD TO POUTS
007630 PERFORM WMS THRU WMXT
007640 ADD 1 TO SEQ
007650 MV1
007660 MOVE AA X1C TO FRST1
007670 MOVE AA X3C TO SECD1
007680 MOVE ZERO TO LAST1
007690 MOVE 090 TO RTNO1
007700 MVXIT
007710 EXIT
007720 STOP1
007730 IF VERB
007740 STOP1
000320 IF-1
000360 PERFORM BRKDN THRU BRKXT
000370 MOVE BB X1C TO FRST1
000390 MOVE SPACE TO LAST1
000390 MOVE BB X1C TO SECD1
000410 IF BB X6C NOT EQUAL SPACE
000430 MOVE BB X6C TO SECD1
000450 IF BB X5C NOT EQUAL SPACE
000470 MOVE BB X4C NOT EQUAL SPACE
000490 MOVE BB X4C TO SECD1
000490 IF PBDV EQUAL SPACE
000510 GO TO IF-1A
000530 MOVE 250 TO RTN01
000550 MOVE PWORD TO POUTS
000570 PERFORM WMS THRU WMKT
000590 MOVE ZERO TO PBDV
000610 ADD 1 TO SE0
000630 MOVE ZERO TO PBDV
000640 IF-1A
000660 MOVE 200 TO RTN01
000680 MOVE AA X1C TO FRST1
000700 MOVE 2 TO SUB
000720 IF AA XSUBC EQUAL @NOT @
000740 ADD 1 TO RTN01
000760 ADD 1 TO RTN01
000780 IF AA XSUBC EQUAL @EQUAL @
000800 GO TO IF-2
000820 IF AA XSUBC EQUAL @LESS @
000840 ADD 10 TO RTN01
000860 GO TO IF-2
000880 IF AA XSUBC EQUAL @GREATER @
000900 ADD 20 TO RTN01
000920 ADD 1 TO SUB
000940 GO TO IF-2
000960 IF AA XSUBC EQUAL @NUMERIC @
000980 ADD 30 TO RTN01
000990 GO TO IF-4
000990 IF AA XSUBC EQUAL @ALPHAB @
001000 ADD 40 TO RTN01
001020 ADD 50 TO RTN01
001040 IF-2
001060 ADD 1 TO SUB
001080 IF AA XSUBC EQUAL @TO @
001100 ADD 1 TO SUB
001120 IF AA XSUBC EQUAL @THAN @
001140 ADD 1 TO SUB
001160 IF-3
001180 MOVE AA XSUBC TO SECD1
001200 IF-4
001220 IF RELF X1C EQUAL 1
001240 GO TO IFXIT
001260 PERFORM RCDOC THRU RCODX
001280 IF INSTR EQUAL @GO @
001300 MOVE DATA1 TO LAST1
001320 GO TO IFXIT
001340 IF INSTR EQUAL @GO TO @
001360 MOVE DATA1 TO LAST1
001380 GO TO IFXIT
001400 MOVE PNAME TO LAST1
000880  MOVE    1     TO    RDELG
000890  MOVE    RTNO1    TO    FLNX
      MOVE    1     TO    BAK
0008A0  IF    FLNX    EQUAL    1
0008C0  SUBTRACT    1    FROM    RTNO1
0008C0  GO TO    IFXIT
0008C0  ADD    1    TO    RTNO1
0008D0  IFXIT
0008E0  EXIT
000900  ********************************
000900  ADD, SUBTRACT, MULTIPLY, AND DIVIDE VERBS
000900  000940  RT-0
000940  000940  PERFORM    BRKDW    THRU    BRKXT
000950  000950  MOVE    4    TO    SUB
000960  000960  RT-1
000960  000960  IF    AA    XSUBC    EQUAL    SPACE
000970  000970  GO TO    RT-2
000970  000970  IF    AA    XSUBC    NOT    EQUAL    @ROUNDED
000980  000980  ADD    1    TO    SUB
000990  000990  GO TO    RT-1
0009A0  0009A0  MOVE    180    TO    RTNO1
0009A0  0009A0  MOVE    PWORD    TO    POUTS
0009B0  0009B0  PERFORM    WMS    THRU    WMXT
0009C0  0009C0  ADD    1    TO    SEQ
0009D0  0009D0  RT-2
0009D0  0009D0  MOVE    BB    X1C    TO    FRST1
0009E0  0009E0  MOVE    BB    X2C    TO    SEC1D
0009E0  0009E0  MOVE    BB    X4C    TO    LAST1
0009F0  0009F0  IF    BB    X5C    NOT    EQUAL    SPACE
000A00  000A00  MOVE    BB    X5C    TO    LAST1
000A10  000A10  IF    BB    X3C    NOT    EQUAL    SPACE
000A20  000A20  MOVE    BB    X3C    TO    SEC1D
000A30  000A30  IF    PROV    EQUAL    SPACE
000A40  000A40  GO TO    RT-3
000A50  000A50  IF    SECD1    EQUAL    SPACE
000A60  000A60  GO TO    RT-2A
000A70  000A70  IF    LAST1    NOT    EQUAL    SPACE
000A80  000A80  GO TO    RT-2A
000A90  000A90  IF    AA    X4C    NOT    EQUAL    SPACE
000AA0  000AA0  GO TO    RT-2A
000AB0  000AB0  MOVE    SECD1    TO    LAST1
000AC0  000AC0  RT-2A
000AC0  000AC0  MOVE    250    TO    RTNO1
000AD0  000AD0  MOVE    PWORD    TO    POUTS
000AE0  000AE0  PERFORM    WMS    THRU    WMXT
000AF0  000AF0  ADD    1    TO    SEQ
000B00  000B00  RT-3
000B00  000B00  MOVE    ZERO    TO    PROV
000B10  000B10  MOVE    AA    X1C    TO    FRST1
000B20  000B20  IF    T2    X1C    EQUAL    @ADD
000B30  000B30  MOVE    020    TO    RTNO1
000B40  000B40  RT-2A
000B40  000B40  GO TO    ADDR-1
000B50  000B50  IF    T2    X1C    EQUAL    @SUBTRACT
000B60  000B60  MOVE    160    TO    RTNO1
000B70  000B70  RT-3
000B70  000B70  GO TO    SUB-1
000B80  000B80  IF    T2    X1C    EQUAL    @MULTIPLY
000B90  000B90  MOVE    160    TO    RTNO1
000BA0  000BA0  IF    T2    X1C    EQUAL    @DIVIDE
000BB0  000BB0  MOVE    050    TO    RTNO1
000BC0  000BC0  MOVE    AA    X3C    TO    SEC1D
000BD0  000BD0  ********************************
PERFORM WMS THRU WMXT
ADD 1 TO SEQ
ADD 1 TO RTN01
MOVE AA %5C TO FRST1
MOVE AA %5C TO SECD1
MOVE AA %57C TO LAST1
CALLX
EXIT
***************************************************************************
DELETE VERB
***************************************************************************

DELT1
MOVE 30 TO RTN01
MOVE DATA1 TO FRST1
IF IGN-1 EQUAL SPACE
GO TO DELTX
ADD 1 TO RTN01
PERFORM RDCRD THRU CORDX
IF INSTR EQUAL %20 TO 
GO TO DELTX
MOVE DATA1 TO LAST1
GO TO DELTX
IF INSTR EQUAL %20 TO 
GO TO DELTX
MOVE DATA1 TO LAST1
GO TO DELTX
MOVE PPNUM TO LAST1
MOVE 1 TO RDFLG
MOVE 1 TO BAK
DELTX
EXIT
***************************************************************************
EXIT VERB
***************************************************************************
007830************ EXIT VERB
007850************ EXIT VERB
007960************ EXIT VERB
007970 EXEX
007980 MOVE 170 TO RTN01
IF DATA1 NOT EQUAL SPACE
ADD 1 TO RTN01
007990 EXIT
008000 EXIT
007380************ PERFORM VERB
007400************ PERFORM VERB
007410************ PERFORM VERB
007420 PERFI
MOVE 120 TO RTN01
MOVE DATA1 TO FRST1
MOVE DATA2 TO SECD1
MOVE %80010 TO LAST1
IF DATA3 NOT EQUAL SPACE
MOVE DATA3 TO IGN-2
PERFORM OCR TTHRU OCRXT
MOVE SUJ TO VLR
MOVE VLR TO ADR
MOVE ADR TO LAST1
007490 PFXIT
007500 EXIT
007510************ READ VERB
007520************ READ VERB
007530************ READ VERB
007540************ READ VERB
007550 READ1
007560 MOVE 130 TO RTN01
007570 MOVE DATA1 TO FRST1
MOVE DATA3 TO LAST1
005860 BRKD1
  MOVE   SPACE TO  0SUB
  MOVE   1 TO  SUBLG
  MOVE   1 TO  ODFLG
  MOVE   SOR TO  TAR2
  MOVE   SPACE TO  SOR
  MOVE   0 TO  SUBZ
  MOVE   2 TO  SBS

BRKD1
005870 ADD   1 TO  SBS
  IF      SBS EQUAL 11
  PERFORM  RDCOB THRU  CORDX
  MOVE   DAT1 TO  T2  X11C
  MOVE   IGN-1 TO  T2  X12C
  MOVE   DAT2 TO  T2  X13C
  MOVE   DAT3 TO  T2  X14C
  MOVE   DAT4 TO  T2  X15C
  MOVE   DAT5 TO  T2  X16C
  MOVE   DAT6 TO  T2  X17C
  MOVE   DAT7 TO  T2  X18C
  IF      PRRAG NOT  EQUAL  SPACE
  MOVE   1 TO  RDFLG
  MOVE   ZERO TO  SUBLG
  MOVE   ZERO TO  ODFLG
  PERFORM  GDUMP THRU  ODMPX
  GO TO   BRKXT

005880 IF      T2  X5BSCEQUAL SPACE
005890 ADD   1 TO  SBS
005900 IF      T2  X5BSCEQUAL SPACE
005910 PERFORM  GDUMP THRU  ODMPX
  GO TO   BRKXT

005920 MOVE   T2  X5BSCTO  IGN-2
005930 IF      COL-1  EQUAL  X008
005940 PERFORM  SURED THRU  SUREX
005950 MOVE   TAB1 TO  BB  XSUBZC
  GO TO   BRKD1
005960 IF      COL-1  EQUAL  QUOTE
010000 PERFORM  CO1 THRU  O6X
010030 MOVE   LITCN TO  TNU
  AND     1 TO  SUEZ
010020 MOVE   VNAME TO  AA  XSUBZC
  MOVE   TAB1 TO  XYSZ  X0DFLG
  ADD     1 TO  ODFLG
  GO TO   BRKD1
010050 IF      COL-1  NUMERIC
010060 PERFORM  CO1 THRU  O6X
  MOVE   LITCN TO  TNU
  AND     1 TO  SUEZ
010090 MOVE   VNAME TO  AA  XSUBZC
  MOVE   TAB1 TO  XYSZ  X0DFLG
  ADD     1 TO  ODFLG
  GO TO   BRKD1
010120 ADD     1 TO  SUEZ
010110 MOVE   T2  X5BSCTO  AA  XSUBZC
  GO TO   BRKD1
010140 BRKXT
010150 EXIT
  GDUMP
ADD 1 TO QDFLAG
IF XVSZ XORDFLG EQUAL LITR
GO TO QDMPX
MOVE XVSZ XORDFLG TO LITR
WRITE LITR
GO TO QDUMP
QDMPX

EXIT

* EXTRACTS SUBSCRIPTS USED IN SOURCE CODE

010170 SUBED
010180 MOVE SPACE TO TAB1
010190 MOVE 1 TO SUB
010200 MOVE 2 TO SUA
010210 MOVE T2 XSUBC TO TAB3
010220 SUBXT
010230 MOVE T3 XSUBC TO T1 XSUBC
010240 ADD 1 TO SUA
010250 ADD 1 TO SUB
010260 IF SUA EQUAL 7
010270 ADD 1 TO SBS
010280 MOVE 1 TO SUA
010290 MOVE T2 XSUBC TO TAB3
010300 IF T3 XSUBC NOT EQUAL 0
010310 GO TO SUBXT
MOVE TAB1 TO IGN-2
IF COL-1 NOT NUMERIC
GO TO SUBEX
MOVE ZERO TO SUFLG
PERFORM VALK THRU 06X
MOVE 1 TO SUFLG
MOVE LITCH TO THU
MOVE VNAM TO TAB1
010380 SUBEX
010390 EXIT

010400*
010410* EXTRACTS THE VALUES FOR BOTH NUMERIC AN ALPHANUMERIC AND
010420* BUILDS THE VALUE TABLE WITH THE ACTUAL VALUE

010440 021
010450 MOVE ZERO TO FLONG
MOVE SPACE TO TAB1
010460 MOVE 1 TO OCR
010470 MOVE T2 XSUBC TO TAB3
010480 IF T3 XOCRC EQUAL QUOTE
MOVE 3 TO FLONG
010510 MOVE 1 TO VR
010520 VALB
010530 MOVE T3 XOCRC TO T1 XVLRC
010540 ADD 1 TO VR
010550 ADD 1 TO OCR
IF OCR EQUAL 7
GO TO VALA
IF T3 XOCRC EQUAL QUOTE
GO TO VALDD
010580 VALD
010590 IF FLONG EQUAL 3
010600 GO TO VALDD
010610 IF T3 XOCRC EQUAL SPACE
010620 GO TO VALK
IF INSA EQUAL 05
PERFORM P155 THRU P159X
MOVE SUB TO 05
GO TO P15RO
IF INSA EQUAL 06
PERFORM P156 THRU P159X
MOVE SUB TO 06
GO TO P15RO
IF INSA EQUAL 07
PERFORM P157 THRU P159X
MOVE SUB TO 07
GO TO P15RO
IF INSA EQUAL 08
PERFORM P158 THRU P159X
MOVE SUB TO 08
GO TO P15RO
IF INSA EQUAL 09
PERFORM P159 THRU P159X
MOVE SUB TO 09

012490 P15RO
012500 MOVE INSA TO LEVL
012510 IF GF NOT EQUAL ZERO
012520 PERFORM ONGF THRU ONGFX
012530 IF F0CC EQUAL ZERO
012540 GO TO P152P
012550 IF F0CC LESS THAN LEVL
012560 MOVE 1 TO OCR
012570 MOVE OCLN TO VLR
012570 GO TO P152P
PERFORM OCRX THRU OCRFX
MOVE ZERO TO OCLN
012580 MOVE ZERO TO F0CC
012590 P152P
012600 IF DATR1 EQUAL 00OCCURS0
012610 MOVE DATR2 TO IGN-2
012620 PERFORM OCRX THRU OCRX
012630 MOVE SUC TO VLR
012640 MOVE SUC TO PPN0
012650 MOVE INSA TO F0CC
012660 MOVE 2 TO IPR0C
012670 MOVE 2 TO OCR
012670 GO TO P15R1
012680 IF DATR1 EQUAL 00RED0F
012690 PERFORM REDX THRU REDX
012700 GO TO P15R1
012710 IF COGF NOT EQUAL SPACE
012720 PERFORM COGM THRU COGMX
012730 P15R1
012740 MOVE CORLC TO ADR
012750 PIEXT
012760 EXIT

OCGR
MOVE SUB TO OCLN
OCGRA
SUBTRACT 1 FROM SUB
IF IPICX SXSUBC EQUAL 2
MOVE ZERO TO IPICX SXSUBC
MOVE OCLN TO SUB
GO TO OCRX
IF SUB EQUAL 1
GO TO OCRX
<table>
<thead>
<tr>
<th>DIVIDE</th>
<th>PPNQ</th>
<th>INTO</th>
<th>IDLN</th>
<th>XSUBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO TO</td>
<td>OCRX</td>
<td>EXIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCGRX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>012770</td>
<td>TERMINATES, PROCESSING FOR A GROUP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>012790</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**012600 PIS2**

| 012610 | IF   | G2   | NOT EQUAL ZERO |
| 012620 | MOVE | SUBR | TO     | IDLN | XG2C |
| 012630 | MOVE | ZERO | TO     | SUBR |

**012640 PIS3**

| 012650 | IF   | G3   | NOT EQUAL ZERO |
| 012660 | MOVE | SUBR | TO     | IDLN | XG3C |
| 012670 | MOVE | ZERO | TO     | G3   |
| 012680 | MOVE | ZERO | TO     | SUBR |

**012690 PIS4**

| 012700 | IF   | G4   | NOT EQUAL ZERO |
| 012710 | MOVE | TNV  | TO     | IDLN | XG4C |
| 012720 | MOVE | ZERO | TO     | G4   |
| 012730 | MOVE | ZERO | TO     | TNV  |

**PIS5**

| 012740 | IF   | G5   | NOT EQUAL ZERO |
| 012750 | MOVE | CG5  | TO     | IDLN | XG5C |
| 012760 | MOVE | ZERO | TO     | G5   |
| 012770 | MOVE | ZERO | TO     | CG5  |

**PIS6**

| 012780 | IF   | G6   | NOT EQUAL ZERO |
| 012790 | MOVE | CG6  | TO     | IDLN | XG6C |
| 012800 | MOVE | ZERO | TO     | G6   |
| 012810 | MOVE | ZERO | TO     | CG6  |

**PIS7**

| 012820 | IF   | G7   | NOT EQUAL ZERO |
| 012830 | MOVE | CG7  | TO     | IDLN | XG7C |
| 012840 | MOVE | ZERO | TO     | G7   |
| 012850 | MOVE | ZERO | TO     | CG7  |

**PIS8**

| 012860 | IF   | G8   | NOT EQUAL ZERO |
| 012870 | MOVE | CG8  | TO     | IDLN | XG8C |
| 012880 | MOVE | ZERO | TO     | G8   |
| 012890 | MOVE | ZERO | TO     | CG8  |

**PIS9**

| 012900 | IF   | G9   | NOT EQUAL ZERO |
| 012910 | MOVE | CG9  | TO     | IDLN | XG9C |
| 012920 | MOVE | ZERO | TO     | G9   |
| 012930 | MOVE | ZERO | TO     | CG9  |

**012940 PIS9X**

| 012950 | EXIT |

**012960**

| 012970 | PROCESSES THE PICTURE CLAUSE |

**012980**

| 012990 |      |

**013000**

| 013010 | MOVE | SDR  | TO     | TARI |
| 013020 | MOVE | 30   | TO     | ADR  |
| 013030 | IF   | T1   | XADRC | EQUAL | 8X0 |
| 013040 | MOVE | 8X0  | TO     | IDPIC|
| 013050 | GO TO | PIA  |
| 013060 | IF   | T1   | XADRC | EQUAL | 9X0 |
| 013070 | GO TO | PIA  |
| 013080 | IF   | T1   | XADRC | EQUAL | 059 |

181
013900  ADD  1  TO  ADR
013900  MOVE  3  TO  IDPIC
013910  MOVE  1  TO  SYGN
013920  GO TO  PIA
013930  IF  T1  XADR<\text{C} EQUAL @ @
013940  MOVE  1  TO  IDDEC
013950  MOVE  2  TO  FLONG
013960  GO TO  PID
013970  IF  T1  XADR<\text{C} EQUAL @ @
013980  MOVE  9  TO  IDPIC
013990  GO TO  PIV
014000  GO TO  PID
014000  PIA
014010  ADD  1  TO  ADR
014020  IF  T1  XADR<\text{C} EQUAL SPACE
014030  MOVE  1  TO  IDLNG
014040  MOVE  PIK
014050  IF  T1  XADR<\text{C} NOT EQUAL @ @
014060  ADD  1  TO  IDLNG
014070  GO TO  PIV
014080  MOVE  ZERO TO SUC
014090  PERFORM  PIBBB THRU PIKX
014100  MOVE  SUC TO RTNO1
014110  MOVE  RTNO1 TO IDLNG
014120  IF  T1  XADR<\text{C} EQUAL SPACE
014130  GO TO  PIX
014140  IF  IDPIC EQUAL 9
014150  GO TO  PIV
014160  PID
014170  ADD  1  TO  ADR
014180  IF  T1  XADR<\text{C} EQUAL @ @
014190  MOVE  ZERO TO SUC
014200  PERFORM  PIBBB THRU PIKX
014210  MOVE  SUC TO RTNO1
014220  ADD  RTNO1 TO IDLNG
014230  GO TO  PIDD
014240  ADD  1  TO  IDLNG
014250  PIDD
014260  IF  T1  XADR<\text{C} EQUAL @ @
014270  MOVE  IDLNG TO IDDEC
014280  MOVE  2  TO FLONG
014290  ADD  1  TO ADR
014300  GO TO  PID
014310  PIDDD
014320  IF  T1  XADR<\text{C} EQUAL @ @
014330  ADD  1  TO IDLNG
014340  MOVE  IDLNG TO IDDEC
014350  MOVE  2  TO FLONG
014360  ADD  1  TO ADR
014370  GO TO  PID
014380  IF  T1  XADR<\text{C} NOT EQUAL @ @
014390  MOVE  @TOBIEEDITPIC<\text{C} TO MIASM
014400  PERFORM  WRIT THRU WRXIT
014410  GO TO  PIX
014420  PIV
014430  IF  T1  XADR<\text{C} NOT EQUAL @ @
014440  GO TO  PIDDD
014450  MOVE  2  TO FLONG

013660 MOVE IDNGL TO IDDEC
013670 ADD I TO ADR
013680 IF T1 XADRC NOT EQUAL 9
013690 GO TO PID
013700 ADD I TO ADR
013710 IF T1 XADRC EQUAL 0%9
013720 MOVE ZERO TO SUC
013730 PERFORM PIBBA THRU PIRX
013740 MOVE SUC TO RTNO1
013750 ADD RTNO1 TO IDNLG
013760 GO TO PIVA
013770 IF T1 XADRC EQUAL SPACE
013780 ADD I TO IDNLG
013790 MOVE I TO IDDEC
013800 MOVE ZERO TO FLONG
013810 GO TO PIX
013820 ADD I TO IDNLG
013830 GO TO PID
013840 PIVA
013850 IF T1 XADRC EQUAL SPACE
013860 MOVE ZERO TO FLONG
013870 SUBTRACT IDDEC FROM IDNLG GIVING IDDEC
013880 GO TO PIX
013890 GO TO PID
013900 PIP
013910 IF FLONG EQUAL 2
013920 SUBTRACT IDDEC FROM IDNLG GIVING IDDEC
013930 MOVE ZERO TO FLONG
013940 MOVE #9 TO IDPIC
013950 GO TO PIPA
013960 MOVE 38 TO EDR
013970 PIPB
013980 IF T1 XEDRC EQUAL 0%9
013990 MOVE #9 TO IDPIC
014000 GO TO PIPA
014010 IF EDR EQUAL 59
014020 MOVE #9 TO IDPIC
014030 GO TO PIPA
014040 ADD I TO EDR
014050 GO TO PIPB
014060 PIPA
014070 MOVE I TO EDR
014080 MOVE SPACE TO TA
014090 MOVE IGN-1 TO T2 XEDRC
014100 IF ADR LESS THAN 36
014110 GO TO PIK
014120 ADD I TO EDR
014130 MOVE DATA2 TO T2 XEDRC
014140 IF ADR LESS THAN 42
014150 GO TO PIK
014160 ADD I TO EDR
014170 MOVE DATA3 TO T2 XEDRC
014180 IF ADR LESS THAN 48
014190 GO TO PIK
014200 ADD I TO EDR
014210 MOVE DATA4 TO T2 XEDRC
014220 IF ADR LESS THAN 54
014230 GO TO PIK
014240 ADD I TO EDR
014250 MOVE DATA5 TO T2 XEDRC

PIK
014270  MOVE   1 TO EDR
014280  PIL
014290  IF    TA EQUAL EDR XEDRC
014300  GO TO PIX
014310  IF    EDR XEDRC EQUAL SPACE
014320  MOVE   TA TO EDR XEDRC
014330  GO TO PIX
014340  ADD   1 TO EDR
014350  IF    EDR NOT EQUAL 100
014360  GO TO PIL
014370  MOVE   @EDITAPEPROPO TO MIASM
014380  PERFORM WRIT THRU WRXIT
014390  PIX
014400  EXIT
014410  EXTRACTS THE PICTURE SIZE
014420  PIBBB
014430  ADD   2 TO ADR
014440  IF    T1 XEDRC EQUAL 008
014450  SUBTRACT 1 FROM ADR
014460  MOVE   T1 XEDRC TO S3
014470  ADD   2 TO ADR
014480  GO TO PIBX
014490  ADD   1 TO ADR
014500  IF    T1 XEDRC EQUAL 008
014510  SUBTRACT 2 FROM ADR
014520  MOVE   T1 XEDRC TO S2
014530  ADD   1 TO ADR
014540  ADD   1 TO ADR
014550  MOVE   T1 XEDRC TO S3
014560  ADD   2 TO ADR
014570  GO TO PIBX
014580  SUBTRACT 2 FROM ADR
014590  MOVE   T1 XEDRC TO S1
014600  ADD   1 TO ADR
014610  ADD   1 TO ADR
014620  MOVE   T1 XEDRC TO S2
014630  ADD   1 TO ADR
014640  MOVE   T1 XEDRC TO S3
014650  ADD   2 TO ADR
014660  PIBX
014670  EXIT
014680  GROUP OPTIONS CONTROL.
014690  CODG
014700  CDDG
014710  CODG
014720  MOVE   SOR TO TAB2
014730  MOVE   5 TO S0
014740  CODG
014750  IF    T2 XSEC EQUAL @VALUE @
014760  PERFORM VAL THRU VALX
014770  GO TO CODG
014780  CODGK
014790  IF    T2 XSEC EQUAL @USAGE @
014800  MOVE   INSA TO GF
014810  PERFORM USA THRU USAX
014820  MOVE   USAX TO UF
014830  GO TO CODG
014840  IF    T2 XSEC EQUAL @SIGN @
014850  MOVE   INSA TO GF
014860  PERFORM SUG THRU SIGN
014870  MOVE   SVGN TO SF
014880  MOVE  ZERO TO  SYGN
014890  C0GA
014900  IF  SEQ  NOT GREATER 9
014910  ADD 1 TO SEQ
014920  GO TO C0D0B
014930  IF  STEND  NOT EQUAL 00
014940  PERFORM RDC0B THRU C0D0X
014950  MOVE SOR TO TAB2
014960  MOVE 3 TO SEQ
014970  GO TO C0DGK
014980  IF  GF  EQUAL ZERO
014990  GO TO C0DGX
015000  IF  PARA  EQUAL 001
015010  MOVE 01 TO GF
015020  C0DGX
015030  EXIT
015040  PROCESSES THE VALUE CLAUSE
015050  PROCESSES THE VALUE CLAUSE
015060  PROCESSES THE VALUE CLAUSE
015070  VAL
015080  ADD 1 TO SEQ
015090  IF  T2 XSE0C EQUAL @SPACE @
015100  MOVE 999 TO VLR
015110  GO TO VALX
015120  IF  T2 XSE0C EQUAL @ZERO @
015130  MOVE 998 TO VLR
015140  GO TO VALX
015150  IF  T2 XSE0C EQUAL @QUOTE @
015160  MOVE 997 TO VLR
015170  GO TO VALX
015180  MOVE SPACE TO TAB1
015190  IF  T2 XSE0C EQUAL SPACE
015200  GO TO VALN
015210  IF  SEQ  EQUAL 11
015220  GO TO VALN
015230  MOVE  SEQ  TO SBS
015240  VALH
015250  PERFORM 021 THRU 05X
015260  MOVE L1TCN TO VLR
015270  MOVE SBS TO SEQ
015280  GO TO VALX
015290  VALN
015300  PERFORM RDC0B THRU C0D0X
015310  MOVE SOR TO TAB2
015320  MOVE 1 TO SBS
015330  VALH
015340  EXIT
015350  ELEMENTARY ITEM OPTIONS CONTROL
015360  ELEMENTARY ITEM OPTIONS CONTROL
015370  ELEMENTARY ITEM OPTIONS CONTROL
015380  COD36
015390  MOVE SOR TO TAB2
015400  MOVE 5 TO SEQ
015410  CODA
015420  IF  T2 XSE0C EQUAL @VALUE @
015430  PERFORM VAL THRU VAX
015440  GO TO CONC
015450  IF  T2 XSE0C EQUAL 00CURR0
015460  MOVE 2 TO OCR
015470  ADD 2 TO SEQ
015480 MOVE T2 XE2OC TO IGN-2
015500 PERFORM OCR THRU OCRXT
015500 MOVE SUC TO VLR
015500 MULTIPY IDLN G BY VLR GIVING IDLN
015570 ADD 1 TO SEQ
015570 GO TO COCC
015570 IF T2 XE2OC EQUAL BLANK @ 0
015570 MOVE 1 TO BK
015570 ADD 1 TO SEQ
015570 GO TO COCC
015570 IF T2 XE2OC EQUAL SSYNC @ 0
015570 PERFORM SVN THRU SVNX
015570 GO TO COCC
015570 IF T2 XE2OC EQUAL IDLN @ 0
015570 PERFORM USA THRU USAX
015570 GO TO COCC
015570 IF T2 XE2OC EQUAL SIG0 @ 0
015570 PERFORM SIG0 THRU SIGX
015670 COCC
015670 IF SEQ NOT GREATER 9
015670 ADD 1 TO SEQ
015670 GO TO CODA
015670 IF STEND EQUAL 9 @ 0
015670 GO TO CODX
015670 PERFORM RDCOB THRU RDCOX
015740 MOVE SOR TO TAB2
015750 MOVE 3 TO SEQ
015760 GO TO CODB
015770 CODX
015770 EXIT
015790
015800** RIGHT JUSTIFIES NUMERIC FIELDS SUCH AS BLOCK CONTAINS
015810
015830 MOVE ZERO TO SUC
015840 IF COL-2 EQUAL TO SPACE
015850 MOVE COL-1 TO S3
015860 GO TO OCRXT
015870 IF COL-3 EQUAL TO SPACE
015880 MOVE COL-1 TO S2
015890 MOVE COL-2 TO S3
015900 GO TO OCRXT
015910 MOVE COL-1 TO S1
015920 MOVE COL-2 TO S2
015930 MOVE COL-3 TO S3
015940 OCRXT
015950 EXIT
015970** PROCESSES THE REDEFINES CLAUSE
015980
015990 REDE
016000 MOVE SUB TO SEQ
016010 REDEA
016020 SUBTRACT 1 FROM SEQ
016030 IF NON XE2OC EQUAL DATA2
016040 SUBTRACT IDLN XE2OC FROM CODLC
016050 GO TO REDER
016060 IF SEQ NOT EQUAL 1
016070 GO TO REDFA
016080 MOVE @NONANFREDEFINES TO MASM

186
01600 PERFORM WRIT THRU WPXIT
016100 GO TO REDEX
016110 REDEX
  IF FOCC NOT EQUAL ZFPO
  DIVIDE PPO INTO INN XSEOC GIVING IDLN
  SUBTRACT IDLN FROM OCLN
  MOVE OCLN TO VLR
016120 SUBTRACT IDLN XSEOC FROM SUBZ
016130 IF INSA EQUAL @A20
016140 GO TO REDEX
016150 IF INSA EQUAL @A30
016160 SUBTRACT IDLN XSEOC FROM SUBR
016170 GO TO REDEX
016180 IF INSA EQUAL @A40
016190 SUBTRACT IDLN XSEOC FROM SUBR
016200 SUBTRACT IDLN XSEOC FROM SUBR
016210 REDEX
016220 EXIT
016230 **************************************** PROCESSES SIGN CLAUSE
016240 ****************************************
016250 Sig
016260 ADD 1 TO SEQ
016270 IF SEQ GREATER THAN 9
016280 PERFORM RDCCO THRU CORDX
016290 MOVE SOR TO TAB2
016300 MOVE 3 TO SEQ
016310 IF T2 XSEOC EQUAL @LEADING
016320 MOVE 1 TO SYGN
016330 GO TO SIGA
016340 MOVE 3 TO SYGN
016350 SIGA
016360 ADD 2 TO SEQ
016370 IF SEQ LESS THAN 10
016380 GO TO SIGB
016390 IF STEND EQUAL @0
016400 GO TO SIGX
016410 PERFORM RDCCO THRU CORDX
016420 MOVE SOR TO TAB2
016430 MOVE 3 TO SEQ
016440 SIGB
016450 IF T2 XSEOC NOT EQUAL @SEPARAB
016460 SUBTRACT 1 FROM SEQ
016470 GO TO SIGX
  IF OCR EQUAL 2
016480 ADD VLR TO IDLN
016490 GO TO SIGC
016500 ADD 1 TO IDLN
016510 SIGC
016520 ADD 1 TO SYGN
016530 SIGX
016540 EXIT
016550 **************************************** PROCESSES THE USAGE CLAUSE
016560 ****************************************
016570 USA
016580 ADD 1 TO SEQ
016590 IF SEQ EQUAL 11
016600 PERFORM RDCCO THRU CORDX
016610 MOVE SOR TO TAB2
016620 MOVE 3 TO SEQ
016660 IF T2 XSEQC EQUAL @COMP @
016670 MOVE 1 TO USAGE
016680 USAX
016690 EXIT
016700 PROCESSES THE SYNC CLAUSE
016710
016720 SYN
016730 SYN
016740 ADD 1 TO SEX
016750 IF SEX EQUAL 11
016760 PERFORM RDCOB THRU CORDX
016770 MOVE SOR TO TAR2
016780 MOVE 3 TO SEX
016790 IF T2 XSEQC EQUAL @LEFT @
016800 MOVE 1 TO SINC
016810 GO TO SYN
016820 MOVE 2 TO SINC
016830 SYN
016840 EXIT
016850
016860 PHYSICAL END OF PROGRAM
016870
```
000000 IDENTIFICATION DIVISION.
000001 PROGRAM-ID    NPP.
000002 ENVIRONMENT DIVISION.
000003 CONFIGURATION SECTION.
000004 SOURCE- COMPUTER.
000005 UNIVAC-1108.
000006 OBJECT-COMPUTER.
000007 UNIVAC-1108.
000008 INPUT-OUTPUT SECTION.
000009 FILE-CONTROL.

<table>
<thead>
<tr>
<th>SELECT</th>
<th>CORIN</th>
<th>ASSIGN</th>
<th>CORIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>LITF</td>
<td>ASSIGN</td>
<td>X2</td>
</tr>
<tr>
<td>SELECT</td>
<td>COBF</td>
<td>ASSIGN</td>
<td>X3</td>
</tr>
<tr>
<td>SELECT</td>
<td>VAOE</td>
<td>ASSIGN</td>
<td>X5</td>
</tr>
<tr>
<td>SELECT</td>
<td>DNOE</td>
<td>ASSIGN</td>
<td>X6</td>
</tr>
<tr>
<td>SELECT</td>
<td>PRF</td>
<td>ASSIGN</td>
<td>X4</td>
</tr>
</tbody>
</table>

000016 DATA DIVISION.
000017 FILE SECTION.
000018 FD    LITF.
000019 RECORD 6A.
000020 LABEL RECORD OMITTED.
000021 01 LITP PIC X(60).
000022 FD    CORIN.
000023 RECORD 38.
000024 LABEL RECORDS OMITTED.
000025 01 COBR.
000026 02 IDL PIC X(5).
000027 02 FILLER PIC X(25).
000028 01 COBPH.
000029 02 SE0 PIC X(5).
000030 02 RTH01 PIC X(3).
000031 02 FRST1 PIC X(5).
000032 02 SECDPIC X(5).
000033 02 LAST1 PIC X(5).
000034 02 FILLER PIC X(7).
000035 01 IOTBL.
000036 02 PELF PIC X(3).
000037 02 KEVN PIC X(5).
000038 02 FNAM PIC X(5).
000039 02 DTYPE PIC X(5).
000040 02 DFNAM PIC X(5).
000041 02 FILLER PIC X(7).
000042 FD    CORF.
000043 RECORD 27.
000044 LABEL RECORDS OMITTED.
000045 01 COEDM.
000046 02 INBL PIC X(5).
000047 02 ADR PIC X(5).
000048 02 IDPIC PIC X.
000049 02 IDLNG PIC X(4).
000050 02 IDDEC PIC X(2).
000051 02 BAK PIC 9.
000052 02 SVN PIC 9.
000053 02 USEPIC 9.
000054 02 SINC PIC 9.
000055 02 OCR PIC 9.
000056 02 EDR PIC X(2).
000057 02 VLR PIC X(3).
000058 FD    PRF.
000059 LABEL RECORD OMITTED.
000060 01 PR.
```
000061  02  PPA  PIC  XX10C  OCCURS  8  TIMES
000062  FD  VARF
000063  LABEL  RECORD  OMITTED
000064  01  VAL
000065  02  VU  PIC  X  OCCURS  80  TIMES
000066  FD  DNOF
000067  RECORD  22
000068  LABEL  RECORD  OMITTED
000069  01  DNP
000070  02  DNLCC  PIC  9X5C
000071  02  DNTYR  PIC  9
000072  02  DNTYP  PIC  9
000073  02  DLLEN  PIC  9X4C
000074  02  OCLV  PIC  9
000075  02  SVN  PIC  9
000076  02  BKN  PIC  9
000077  02  OCC  PIC  9X3C
000078  02  DNEPT  PIC  9X3C
000079  02  DECP  PIC  9X2C
000080  WORKING-STORAGE
000081  77  SAVLG  PIC  9X3C
000082  77  SUPR  PIC  9 VALUE 1
000083  01  WKTAB
000084  02  WAPRA  PIC  X25C
000085  02  FILLER  PIC  8X4C
000086  01  QUINT
000087  02  SAN  PIC  9X5C
000088  02  COED  PIC  9X3C
000089  02  OP-1  PIC  9X4C
000090  02  OP-2  PIC  9X4C
000091  02  OP-3  PIC  9X4C
000092  01  VALP
000093  02  HYP  PIC  X
000094  02  NOV  PIC  8X4C
000095  01  WN
000096  02  VCN  PIC  9X5C VALUE ZERO
000097  02  VRDF  PIC  9 VALUE ZERO
000098  02  SUP  PIC  9X5C
000099  02  SUM  PIC  9X5C
000100  02  DNP  PIC  8X4C VALUE 1
000101  02  ECMNT  PIC  9X5C VALUE ZERO
000102  02  SUC  PIC  9X3C
000103  02  VLP  PIC  9X3C VALUE 1
000104  02  HDNP  PIC  9X4C
000105  02  H  PIC  X
000106  02  EDP  PIC  9X3C VALUE 1
000107  02  SVLOC  PIC  9X5C VALUE ZERO
000108  02  FNP  PIC  9X2C VALUE 1
000109  01  EDTA
000110  02  EDI  PIC  XX30C OCCURS  30  TIMES
000111  01  DNTAB
000112  02  DNNS  OCCURS  400
000113  03  DNN  PIC  9X5C
000114  03  DNPTZ  PIC  9X4C
000115  01  IOTAB
000116  02  IOTAB  OCCURS  10  TIMES
000117  03  ION  PIC  9X5C
000118  03  RELFA  PIC  9X3C
000119  03  KEVNA  PIC  9X5C
000120  01  FDTRB
000121  02  FDTRB  OCCURS  10  TIMES
000122  03 FDN  PIC  X(5)C  
000123  01 FD01T  
000124  02 FDN1A  OCCURS  30 TIMES  
000125  03 FD0N  PIC  X(5)C  
000126  03 FDPT  PIC  S(2)C  
000127  01 TARE  
000128  02 TE  PIC  X  OCCURS  61 TIMES  
000129  01 WKP  
000130  02 WKR  PIC  9  
000131  02 WKB  PIC  9  
000132  02 HKC  PIC  9  
000133  01 PR2  
000134  02 PRB  PIC  X(23)C  OCCURS  3 TIMES  
000135  02 FILLER  PIC  X(2)C  
000136  01 PR3  REDEFINES  PR2  
000137  02 PRD  PIC  X  OCCURS  80 TIMES  
000138  01 PR4  REDEFINES  PR2  
000139  02 PRD  PIC  X(22)C  OCCURS  3 TIMES  
000140  02 FILLER  PIC  X(4)C  
000141  01 POUT  REDEFINES  PR2  
000142  02 POUT  PIC  X(20)C  OCCURS  4 TIMES  
000143  01 EDT  
000144  02 ED  PIC  X  OCCURS  999 TIMES  
000145  01 FDT  
000146  02 FDTT  OCCURS  10 TIMES  
000147  03 FDSTL  PIC  S(5)C  
000148  03 DNPTR  PIC  S(4)C  
000149  03 IOPTR  PIC  S(2)C  
000150  03 BKLEN  PIC  S(4)C  
000151  03 LBL  PIC  9  
000152  03 RCOD  PIC  S(3)C  
000153  03 KKEYN  PIC  S(4)C  
000154  01 IOT  
000155  02 IOD  OCCURS  10 TIMES  
000156  03 IODD  PIC  X(5)C  
000157  03 IOF  PIC  X(5)C  
000158  
000159  PROCEDURE DIVISION  
000160  
000161  ST  
000162  OPEN  OUTPUT  PPF  
000163  OPEN  INPUT  CORIN  
000164  OPEN  INPUT  LITF  
000165  MOVE  SPACE TO  IOTAB  
000166  MOVE  SPACE TO  EDTA  
000167  MOVE  SPACE TO  IOT  
000168  MOVE  ZERO TO  QUINT  
000169  PERFORM  PRED  THRU  PREDX  2 TIMES  
000170  
000171  
000172  
000173  ST1  
000174  PERFORM  PRED  THRU  PREDX  
000175  IF  IDL  EQUAL  @ 2A  
000176  MOVE  1  TO  SUB  
000177  PERFORM  IOTAB  THRU  IOTX  
000178  IF  IDL  EQUAL  @ 3A  
000179  PERFORM  DABL  THRU  DABLX  
000180  IF  IDL  EQUAL  @ 5A  
000181  PERFORM  EDABL  THRU  EDABRX  
000182  IF  IDL  EQUAL  @ 8A  

191
000183 PERFORM ST6 THRU FDXT
000184 PERFORM VB THRU VRX
000185 MOVE 1 TO OP-1
000186 PERFORM SPT-1 THRU SPT-8
000187 MOVE EDP TO SAVLG
000188 MOVE ZFRD TO OP-1
000189 PERFORM PDIV THRU PDIVT
000190 GO TO ST1

000191 PERFORM CLOSE CBRF
000192 OPEN INPUT CBRF
000193 OPEN OUTPUT VARF
000194 OPEN OUTPUT DNOF
000195 MOVE SPACE TO EDT
000196 MOVE ZERO TO FDT
000197 MOVE SPACE TO DNTAB
000198 MOVE SPACE TO DINTAB
000199 PERFORM RDD THRU RDDX
000200 MOVE 1 TO SUB
000201 IF IDLB1 EQUAL 4
000202 GO TO CNTSW
000203 MOVE ZERO TO SUA
000204 PERFORM RDD THRU RDDX
000205 PERFORM FDPAR THRU FDPAK
000206 PERFORM RDD THRU RDDX
000207 CNTSF
000208 CNTSD
000209 IF OCR EQUAL 9
000210 MOVE IDLB1 TO FDNN $FNPC
000211 MOVE SUA TO FDPT $FNPC
000212 ADD 1 TO FNP
000213 PERFORM DNPAS THRU DNPAX
000214 IF IDLB1 EQUAL 4
000215 GO TO CNTSW
000216 IF IDLB1 EQUAL 5
000217 GO TO FDXT
000218 IF IDPIC EQUAL 8
000219 GO TO CNTSF
000220 GO TO CNTSD
000221 CNTSW
000222 PERFORM RDD THRU RDDX
000223 CNTSH
000224 PERFORM DNPAS THRU DNPAX
000225 IF IDLB1 NOT EQUAL 5
000226 GO TO CNTSHS
000227 CNTSHS
000228 FDXT
000229 EXIT

000230 FDPAR
000231 ADD 1 TO SUA
000232 MOVE IDLB1 TO FDNN $SUAC
000233 MOVE ADDR TO FDSTL $SUAC
000234 MOVE DNP TO DNPR $SUAC
000235 MOVE 1 TO SUB
000236 FDPAT
000237 IF IDLB1 EQUAL 4
000238 MOVE SUA TO IOPTP $SUAC
000239 MOVE RELFA $SUBC TO RCO $SUAC

192
GO TO FDPAS
IF SUB EQUAL 19
MOVE \&RRPPINF\TENTYP\A TO PR
PERFORM WPR 47 THRU WPRX.
GO TO FDPAX
ADD 1 TO SUB
GO TO FDPAT
FDPAS
MOVE IDL\#G TO AKLEN X\#SUAC
MOVE IDDE\#C TO LBL X\#SUAC
FDPAX
EXIT

OPEN OUTPUT COBF
MOVE 3 TO IDLBL
WRITE COBDW
DBL
OPEN OUTPUT COBF
MOVE 3 TO IDLBL
WRITE COBDW
DBL
OPEN OUTPUT COBF
MOVE 4 TO IDLBL
WRITE COBDW
DBL
OPEN OUTPUT COBF
MOVE 5 TO IDLBL
WRITE COBDW
EXIT

GO TO ED\#B
MOVE 1 TO SUB
EB1
PERFORM RED THRU REDX
IF IDL EQUAL @ 89
GO TO ENDBOX
MOVE COBR TO EDI XSUBC
ADD 1 TO SUB
IF SUB LESS THAN 31
GO TO EDI
MOVE \#3@EDIT\# TO PR
PERFORM WPR THRU WPRX.
EDB
EXIT

GO TO JO\#B
PERFORM RED THRU REDX
IF IDL EQUAL @ 39
GO TO IOTX
MOVE DFNPN TO IOF X\#SUAC
MOVE DTVPE TO ION X\#SUAC
MOVE F\#H TO ION X\#SUAC
00305    MOVE    RELF TO RELFA XSUBC
00306    MOVE    KEVNR TO KEVNA XSUBC
00307    ADD    1 TO SUB
00308    GO TO   IOTBD
00309    IOTX
00310    EXIT
00311
00312    DNPAR

00313    MOVE    ZERO TO DNP
00314    MOVE    INRL TO DNP XNPIC
00315    MOVE    DNP TO DNPTZ XDGPC
00316    MOVE    ADR TO DNLOC
00317    MOVE    IDNG TO DNLE
00318    MOVE    BAK TO BAK
00319    MOVE    SNC TO SVN
00320    IF    IDPIC EQUAL 8
00321    MOVE    2 TO DNTYP
00322    GO TO   DNPBR
00323    IF    IDPIC EQUAL 8
00324    MOVE    1 TO DNTYP
00325    MOVE    EGN TO DNTV
00326    MOVE    IDDEC TO DECP
00327    GO TO   DNPBR
00328    IF    IDPIC EQUAL 0
00329    MOVE    3 TO DNTYP
00330    GO TO   DNPBR
00331    IF    IDPIC EQUAL 8
00332    MOVE    IDDEC TO DECP
00333    MOVE    4 TO DNTYP
00334    PERFORM  DNPED THRU DNPBEX
00335    GO TO   DNPBR
00336    IF    IDPIC EQUAL 8
00337    MOVE    5 TO DNTYP
00338    PERFORM  DNPED THRU DNPBEX
00339    DNPBR

00340    IF    OCR EQUAL 8
00341    MOVE    2 TO OCLV
00342    GO TO   DNPV
00343    IF    OCR EQUAL ZER
00344    MOVE    DNPP TO DNLOC
00345    IF    OCR EQUAL 1
00346    MOVE    VLR TO OCCR
00347    GO TO   DNPP
00348    DIVIDE  VLR INTO IDNLG GIVING DNLEN
00349    MOVE    VLR TO OCCR
00350    MOVE    DNPP TO DNPP
00351    DNPV

00352    IF    VLR EQUAL ZER
00353    GO TO   DNPP
00354    PERFORM  DNVPZ THRU DNPVX
00355    MOVE    IDNLG TO SUC
00356    MOVE    SUC TO WCF
00357    PERFORM  DNPPZ THRU DNPVX
00358    IF    VLP EQUAL SSS
00359    MOVE    SPACE TO TABE
00360    GO TO   DNPPVG
00361    IF    VLR EQUAL SSS
00362    MOVE    ZERO TO TABE
<table>
<thead>
<tr>
<th>Offset</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000366</td>
<td>GO TO</td>
<td>DNPVG</td>
</tr>
<tr>
<td>000367</td>
<td>IF</td>
<td>VLR EQUAL S97</td>
</tr>
<tr>
<td>000368</td>
<td>MOVE</td>
<td>QUOTE TO TARE</td>
</tr>
<tr>
<td>000369</td>
<td>GO TO</td>
<td>DNPVG</td>
</tr>
<tr>
<td>000370</td>
<td>READ</td>
<td>LITF END GO TO EOF</td>
</tr>
<tr>
<td>000371</td>
<td>MOVE</td>
<td>LITR TO TARE</td>
</tr>
<tr>
<td>000372</td>
<td>IF</td>
<td>TE XSLC EQUAL QUOTE</td>
</tr>
<tr>
<td>000373</td>
<td>GO TO</td>
<td>DNPVG</td>
</tr>
<tr>
<td>000374</td>
<td>MOVE</td>
<td>ZERO TO SUC</td>
</tr>
<tr>
<td>000375</td>
<td>MOVE</td>
<td>1 TO SUB</td>
</tr>
<tr>
<td>000376</td>
<td>DNPVG</td>
<td></td>
</tr>
<tr>
<td>000377</td>
<td>IF</td>
<td>TE XSUBC EQUAL @ @</td>
</tr>
<tr>
<td>000378</td>
<td>ADD</td>
<td>1 TO SUB</td>
</tr>
<tr>
<td>000379</td>
<td>IF</td>
<td>TE XSUBC EQUAL SPACE</td>
</tr>
<tr>
<td>000380</td>
<td>SUBTRACT</td>
<td>SUC FROM IDLNG GIVING SUB</td>
</tr>
<tr>
<td>000381</td>
<td>GO TO</td>
<td>DNPVP</td>
</tr>
<tr>
<td>000382</td>
<td>ADD</td>
<td>1 TO SUB</td>
</tr>
<tr>
<td>000383</td>
<td>ADD</td>
<td>1 TO SUC</td>
</tr>
<tr>
<td>000384</td>
<td>GO TO</td>
<td>DNPV9</td>
</tr>
<tr>
<td>000385</td>
<td>DNPVP</td>
<td></td>
</tr>
<tr>
<td>000386</td>
<td>IF</td>
<td>SUB EQUAL ZERO</td>
</tr>
<tr>
<td>000387</td>
<td>GO TO</td>
<td>DNPV7</td>
</tr>
<tr>
<td>000388</td>
<td>DNPV6</td>
<td></td>
</tr>
<tr>
<td>000389</td>
<td>MOVE</td>
<td>0 TO VU XVLPC</td>
</tr>
<tr>
<td>000390</td>
<td>IF</td>
<td>VLP EQUAL 80</td>
</tr>
<tr>
<td>000391</td>
<td>PERFORM</td>
<td>HVAL THRU HVALX</td>
</tr>
<tr>
<td>000392</td>
<td>ADD</td>
<td>1 TO VLP</td>
</tr>
<tr>
<td>000393</td>
<td>SUBTRACT</td>
<td>1 FROM SUB</td>
</tr>
<tr>
<td>000394</td>
<td>IF</td>
<td>SUB NOT EQUAL 0</td>
</tr>
<tr>
<td>000395</td>
<td>GO TO</td>
<td>DNPV8</td>
</tr>
<tr>
<td>000396</td>
<td>DNPV7</td>
<td></td>
</tr>
<tr>
<td>000397</td>
<td>MOVE</td>
<td>1 TO SUB</td>
</tr>
<tr>
<td>000398</td>
<td>DNPV6</td>
<td></td>
</tr>
<tr>
<td>000399</td>
<td>IF</td>
<td>TE XSUBC EQUAL @ @</td>
</tr>
<tr>
<td>000400</td>
<td>ADD</td>
<td>1 TO SUB</td>
</tr>
<tr>
<td>000401</td>
<td>MOVE</td>
<td>TE XSUBC TO VU XVLPC</td>
</tr>
<tr>
<td>000402</td>
<td>IF</td>
<td>VLP EQUAL 80</td>
</tr>
<tr>
<td>000403</td>
<td>PERFORM</td>
<td>HVAL THRU HVALX</td>
</tr>
<tr>
<td>000404</td>
<td>ADD</td>
<td>1 TO VLP</td>
</tr>
<tr>
<td>000405</td>
<td>ADD</td>
<td>1 TO SUB</td>
</tr>
<tr>
<td>000406</td>
<td>IF</td>
<td>TE XSUBC NOT EQUAL SPACE</td>
</tr>
<tr>
<td>000407</td>
<td>GO TO</td>
<td>DNPV6</td>
</tr>
<tr>
<td>000408</td>
<td>GO TO</td>
<td>DNPV8</td>
</tr>
<tr>
<td>000409</td>
<td>DNPVG</td>
<td></td>
</tr>
<tr>
<td>000410</td>
<td>MOVE</td>
<td>2 TO SUB</td>
</tr>
<tr>
<td>000411</td>
<td>DNPV9</td>
<td></td>
</tr>
<tr>
<td>000412</td>
<td>MOVE</td>
<td>TE XSUBC TO VU XVLPC</td>
</tr>
<tr>
<td>000413</td>
<td>IF</td>
<td>VLP EQUAL 80</td>
</tr>
<tr>
<td>000414</td>
<td>PERFORM</td>
<td>HVAL THRU HVALX</td>
</tr>
<tr>
<td>000415</td>
<td>ADD</td>
<td>1 TO VLP</td>
</tr>
<tr>
<td>000416</td>
<td>ADD</td>
<td>1 TO SUB</td>
</tr>
<tr>
<td>000417</td>
<td>IF</td>
<td>TE XSUBC EQUAL QUOTE</td>
</tr>
<tr>
<td>000418</td>
<td>MOVE</td>
<td>SPACE TO TE XSUBC</td>
</tr>
<tr>
<td>000419</td>
<td>SUBTRACT</td>
<td>1 FROM IDLNG</td>
</tr>
<tr>
<td>000420</td>
<td>IF</td>
<td>IDLNG NOT EQUAL 0</td>
</tr>
<tr>
<td>000421</td>
<td>GO TO</td>
<td>DNPV9</td>
</tr>
<tr>
<td>000422</td>
<td>DNPV8</td>
<td></td>
</tr>
<tr>
<td>000423</td>
<td>ADD</td>
<td>1 TO DNP</td>
</tr>
<tr>
<td>000424</td>
<td>WRITE</td>
<td>DNR</td>
</tr>
<tr>
<td>000425</td>
<td>PERFORM</td>
<td>RDD THRU RDX</td>
</tr>
<tr>
<td>000426</td>
<td>DNPVX</td>
<td></td>
</tr>
</tbody>
</table>

195
000427    EXIT       PREPRO
000428    DNPVZ      PREPRO
000429    ADD       1  TO  VCNJ      PREPRO
000430    MOVE       1  TO  SUB      PREPRO
000431    MOVE       DNP  TO  TABC      PREPRO
000432    MOV       4      PREPRO
000433    DNPVR      PREPRO
000434    MOVE       TE  XSUBC  TO  VU  XVLPX      PREPRO
000435    IF       VLP  EQUAL  80      PREPRO
000436    PER       WWE  THRU  XVALX      PREPRO
000437    ADD       1  TO  VLP      PREPRO
000438    IF       SUB  NOT  EQUAL  4      PREPRO
000439    ADD       1  TO  SUB      PREPRO
000440    GO       TO  DNPVB      PREPRO
000441    DNPVX      PREPRO
000442    EXIT       PREPRO
000443    DNPZZ      PREPRO
000444    MOVE       WKA  TO  VU  XVLPX      PREPRO
000445    IF       VLP  EQUAL  80      PREPRO
000446    PER       WWE  THRU  XVALX      PREPRO
000447    ADD       1  TO  VLP      PREPRO
000448    MOVE       WKB  TO  VU  XVLPX      PREPRO
000449    IF       VLP  EQUAL  80      PREPRO
000450    PER       WWE  THRU  XVALX      PREPRO
000451    ADD       1  TO  VLP      PREPRO
000452    MOVE       WKC  TO  VU  XVLPX      PREPRO
000453    IF       VLP  EQUAL  80      PREPRO
000454    PER       WWE  THRU  XVALX      PREPRO
000455    ADD       1  TO  VLP      PREPRO
000456    DNPZK      PREPRO
000457    EXIT       PREPRO
000458    DNPVX      PREPRO
000459    EXIT       PREPRO
000460    VB         PREPRO
000461    VB         PREPRO
000462    IF       EDP  GREATER  THAN  1      PREPRO
000463    MOVE       E @  TO  ED  XEDPC      PREPRO
000464    MOVE       ZRO  TO  DNP      PREPRO
000465    MOV       8      PREPRO
000466    MOVE       1  TO  DNP      PREPRO
000467    MOVE       DNP  TO  DNPZ  XDNPC      PREPRO
000468    PER       DNPVZ  THRU  DNPVX      PREPRO
000469    MOVE       @       PREPRO
000470    MOVE       WKF      PREPRO
000471    PER       DNPZZ  THRU  DNPZX      PREPRO
000472    MOVE       8  TO  VU  XVLPX      PREPRO
000473    IF       VLP  EQUAL  80      PREPRO
000474    PER       WWE  THRU  XVALX      PREPRO
000475    ADD       1  TO  VLP      PREPRO
000476    MOVE       1  TO  DNLN      PREPRO
000477    MOVE       7  TO  DNTVP      PREPRO
000478    MOVE       SLOC  TO  DNLOC      PREPRO
000479    ADD       1  TO  SLOC      PREPRO
000480    MOVE       2  TO  OCLV      PREPRO
000481    WRITE      DNP      PREPRO
000482    MOVE       ZRO  TO  DNP      PREPRO
000483    ADD       1  TO  DNP      PREPRO
000484    MOVE       @QUOTE  TO  DNN  XDNPC      PREPRO
000485    MOVE       DNP  TO  DNPZ  XDNPC      PREPRO
000486    PER       DNPVZ  THRU  DNPVX      PREPRO
000487    PER       DNPZZ  THRU  DNPZX      PREPRO
000549  IF  SUXA NOT EQUAL SUC  PREPPO
000550  ADD  1 TO SUXA  PREPPO
000551  GO TO VBC  PREPPO
000552  MOVE SUXA TO DLNEN  PREPPO
000553  ADD SUXA TO SVLOC  PREPPO
000554  ADD 1 TO DNP  PREPPO
000555  WRITE DNR  PREPPO
000556  GO TO VBR  PREPPO
000557  VBS  PREPPO
000558  IF  TE XSUBC EQUAL @ @  PREPPO
000559  MOVE SUXA TO SUB  PREPPO
000560  ADD 1 TO SUXA  PREPPO
000561  MOVE TE XSUBC TO PRCH XSUBC  PREPPO
000562  ADD 1 TO SUB  PREPPO
000563  IF  TE XSUBC NOT EQUAL SPACE  PREPPO
000564  ADD 1 TO SUC  PREPPO
000565  GO TO VBS  PREPPO
000566  IF  SUB EQUAL @  PREPPO
000567  GO TO VBD  PREPPO
000568  SUBTRACT 1 FROM SUXA  PREPPO
000569  SUBTRACT SUB FROM SUB GIVING DECP  PREPPO
000570  GO TO VBD  PREPPO
000571  VBX  PREPPO
000572  EXIT  PREPPO
000573  PREPPO
000574  PREPPO
000575  DNPE  PREPPO
000576  MOVE EDR TO DNPE  PREPPO
000577  IF EDR NOT GREATER THAN ECNT  PREPPO
000578  GO TO DNPEX  PREPPO
000579  MOVE EDR TO ECNT  PREPPO
000580  MOVE EDI XEDPC TO TABE  PREPPO
000581  MOVE 1 TO SUB  PREPPO
000582  DNPEA  PREPPO
000583  MOVE TE XSUBC TO ED XEDPC  PREPPO
000584  IF  TE XSUBC EQUAL @ @  PREPPO
000585  ADD 1 TO SUB  PREPPO
000586  ADD 1 TO SUB  PREPPO
000587  IF  TE XSUBC EQUAL @ @  PREPPO
000588  ADD 1 TO SUB  PREPPO
000589  IF  TE XSUBC EQUAL @ @  PREPPO
000590  ADD 2 TO SUB  PREPPO
000591  MOVE ZERO TO WKF  PREPPO
000592  GO TO DNPE  PREPPO
000593  DNPE  PREPPO
000594  ADD 1 TO EDP  PREPPO
000595  IF  TE XSUBC EQUAL SPACE  PREPPO
000596  MOVE @ @ TO ED XEDPC  PREPPO
000597  ADD 1 TO EDP  PREPPO
000598  GO TO DNPEX  PREPPO
000599  GO TO DNPEA  PREPPO
000600  DNPE  PREPPO
000601  MOVE ED XEDPC TO H  PREPPO
000602  IF  TE XSUBC EQUAL @ @  PREPPO
000603  SUBTRACT 1 FROM SUB  PREPPO
000604  MOVE TE XSUBC TO WKC  PREPPO
000605  ADD 2 TO SUB  PREPPO
000606  GO TO DNPEA  PREPPO
000607  ADD 1 TO SUB  PREPPO
000608  IF  TE XSUBC EQUAL @ @  PREPPO
000609  SUBTRACT 1 FROM SUB  PREPPO
000670  IF     SUB  EQUAL  10
000671  MOVE   1  TO   SUB
000672  MOVE   1  TO   SUB
000673  GO TO  EOFA
000674  ADD   1  TO   SUB
000675  GO TO  EFA
000676  EFB
000677  MOVE   1  TO   HDNP
000678  IF     KEVNA  XSUBC  EQUAL  ZERO
000679  GO TO  EFC
000680  EFD
000681  IF     KEVNA  XSUBC  EQUAL  DNN  XHDNPC
000682  MOVE   HDNP  TO   FKENV  XSUBC
000683  GO TO  EFC
000684  IF     HDNP  EQUAL  508
000685  GO TO  EFC
000686  ADD   1  TO   HDNP
000687  GO TO  EFD
000688  EOFA
000689  MOVE   IOD  XSUBC  TO   PRA  XSUAC
000690  ADD  1  TO   SUB
000691  IF     IOD  XSUBC  EQUAL  SPACE
000692  SUBTRACT  1  FROM  SUB
000693  PERFORM  WPR  THRU  WPRXT
000694  MOVE   SUB  TO   PRA  X2C
000695  WRITE  PR
000696  MOVE   SPACE  TO   PRA  X2C
000697  GO TO  EOFB
000698  IF     SUA  EQUAL  8
000699  PERFORM  WPR  THRU  WPRXT
000700  MOVE   1  TO   SUA
000701  GO TO  EOFA
000702  ADD   1  TO   SUA
000703  GO TO  EOFA
000704  EOFB
000705  IF     FDTT  X1C  EQUAL  ZERO
000706  MOVE   ZERO  TO   PRB  X2C
000707  MOVE   PR2  TO   PR
000708  WRITE  PR
000709  MOVE   SPACE  TO   PRB  X2C
000710  GO TO  EOFD
000711  MOVE   1  TO   SUB
000712  MOVE   1  TO   SUA
000713  EOFC
000714  MOVE   FDTT  XSUBC  TO   PRB  XSUAC
000715  ADD  1  TO   SUB
000716  IF     FDTT  XSUBC  EQUAL  ZERO
000717  GO TO  EOFC
000718  SUB   EQUAL  10
000719  GO TO  EOFC
000720  IF     SUA  EQUAL  3
000721  MOVE   PR2  TO   PR
000722  MOVE   SPACE  TO   PR2
000723  PERFORM  WPR  THRU  WPRXT
000724  MOVE   1  TO   SUA
000725  GO TO  EOFC
000726  ADD   1  TO   SUA
000727  GO TO  EOFC
000728  EOFC
000729  SUBTRACT  1  FROM  SUB
008730  MOVE  PR2  TO  PR
008731  MOVE  SPACE  TO  PR2
008732  PERFORM  WPR  THRU  WPRXT
008733  MOVE  SUB  TO  PR  X3C
008734  IF  SON  NOT  EQUAL  ZERO
008735  SUBTRACT  1  FROM  SON
008736  MOVE  SON  TO  PRB  X3C
008737  MOVE  PR2  TO  PR
008738  PERFORM  WPR  THRU  WPRXT
008739  MOVE  SPACE  TO  PRB  X3C
008740  MOVE  SPACE  TO  PRB  X3C
008741  E0FD
008742  IF  ED  X1C  EQUAL  SPACE
008743  MOVE  ZERO  TO  PR  X3C
008744  MOVE  ZERO  TO  PR  X5C
008745  PERFORM  WPR  THRU  WPRXT
008746  GO TO  E0FG
008747  MOVE  1  TO  SUB
008748  MOVE  1  TO  SRA
008749  E0FE
008750  MOVE  ED  XSUBC  TO  PRC  XSUAC
008751  1  TO  SUB
008752  IF  ED  XSUBC  EQUAL  SPACE
008753  SUBTRACT  1  FROM  SUB
008754  MOVE  PR3  TO  PR
008755  MOVE  SPACE  TO  PR3
008756  PERFORM  WPR  THRU  WPRXT
008757  MOVE  ECNT  TO  PR  X3C
008758  MOVE  SUB  TO  PR  X5C
008759  PERFORM  WPR  THRU  WPRXT
008760  GO TO  E0FG
008761  IF  SRA  EQUAL  0A
008762  MOVE  PR3  TO  PR
008763  MOVE  SPACE  TO  PR3
008764  PERFORM  WPR  THRU  WPRXT
008765  MOVE  ZERO  TO  SRA
008766  ADD  1  TO  SRA
008767  GO TO  E0FE
008768  E0FG
008769  MOVE  0  TO  SUB
008770  MOVE  1  TO  SRA
008771  CLOSE  DNOF
008772  OPEN  INPUT  DNOF
008773  E0FH
008774  READ  DNOF  END  GO  TO  EOFI
008775  MOVE  DNR  TO  PRD  XSUAC
008776  ADD  1  TO  SUB
008777  IF  SRA  EQUAL  1
008778  MOVE  PR4  TO  PR
008779  MOVE  SPACE  TO  PR4
008780  PERFORM  WPR  THRU  WPRXT
008781  MOVE  1  TO  SRA
008782  GO  TO  E0FH
008783  ADD  1  TO  SRA
008784  GO  TO  E0FH
008785  EOFI
008786  IF  PRD  X1C  EQUAL  SPACE
008787  GO  TO  EOFII
008788  MOVE  09999  TO  PRD  XSUAC
008789  MOVE  PR4  TO  PR
008790  MOVE  SPACE  TO  PR4
000751    PERFORM   WPR   THRU   WPRXT
000752   EOFII
000753    SUBTRACT   1   FROM   SVLOC
000754    MOVE   AR8889   TO   PPD   X3C
000755    MOVE   SVLOC   TO   PPD   X3C
000756    MOVE   SUB   TO   PPD   X2C
000757    MOVE   PR4   TO   PR
000758    MOVE   SPACE   TO   PR4
000759    PERFORM   WPR   THRU   WPRXT
000760    MOVE   0   TO   SUB
000761    IF   VLP   EQUAL   1
000762    MOVE   SUB   TO   PRA   X3C
000763    MOVE   SUB   TO   PRA   X3C
000764    PERFORM   WPR   THRU   WPRXT
000765    GO TO   EOFX
000766    MOVE   VLP   TO   SUB
000767    PERFORM   NVAL   THRU   NVALX
000768    CLOSE   VAOF
000769    OPEN   INPUT   VAOF
000770    MOVE   VCNT   TO   PRA   X2C
000771    PERFORM   WPR   THRU   WPRXT
000772    MOVE   SPACE   TO   PRA   X2C
000773   EOFJ
000774    READ   VAOF   END   GO TO   EOF0
000775    ADD   1   TO   SUB
000776    MOVE   VAO   TO   PR
000777    PERFORM   WPR   THRU   WPRXT
000778   EOF0
000779    MOVE   VCNT   TO   PRA   X3C
000780    SUBTRACT   1   FROM   SUB
000781    MULTIPLY   80   BY   SUB
000782    ADD   SUB   TO   SUB
000783    MOVE   SUB   TO   PRA   X3C
000784    CLOSE   VAOF
000785    PERFORM   WPR   THRU   WPRXT
000786   EOFX
000787    EXIT
000788
000789
000790    PDIV
000791    PERFORM   RED   THRU   REDX
000792    IF   SED   EQUAL   99999
000793    PERFORM   BCKC   THRU   BCKCD
000794    MOVE   LAST1   TO   SON
000795    MOVE   QUINT   TO   POTS   X2C
000796    MOVE   PGOUT   TO   PR
000797    PERFORM   WPR   THRU   WPRXT
000798    GO TO   PDIV
000799    MOVE   SED   TO   SON
000800    MOVE   RTN01   TO   COED
000801    IF   FRTST   NUMERIC
000802    MOVE   FRTST   TO   OP-1
000803    IF   SEDC1   NUMERIC
000804    MOVE   SEDC1   TO   OP-2
000805    IF   LAST1   NUMERIC
000806    MOVE   LAST1   TO   OP-3
000807    IF   RTN01   EQUAL   000
000808    PERFORM   WPR   THRU   WPRXT
000809    GO TO   POUT
000810    IF   RTN01   EQUAL   001
000812  MOVE    LAST1 TO IDLBAL
000813  PERFORM BS-1 THRU BS-EX
000814  MOVE    DNPTZ XSVLOCK TO OP-3
      IF    SECD1 EQUAL LAST1
      MOVE    OP-3 TO OP-2
      GO TO    DO1
000815  DO12
000816  IF    SECD1 NUMERIC
000817  GO TO    DO1
000818  IF    SECD1 EQUAL SPACE
000819  MOVE    ZERO TO OP-2
000820  GO TO    DO1
000821  MOVE    SECD1 TO VALP
000822  IF    HYP EQUAL 0-0
000823  MOVE    EDP TO OP-2
000824  ADD    1 TO EDP
000825  GO TO    DO1
000826  MOVE    SECD1 TO IDNLB
000827  PERFORM BS-1 THRU BS-EX
000828  MOVE    DNPTZ XSVLOCK TO OP-2
      IF    FRST1 EQUAL SECD1
      MOVE    OP-2 TO OP-1
      GO TO    DO1XT
000829  DO1
000830  IF    FRST1 NUMERIC
000831  GO TO    DO1XT
000832  IF    FRST1 EQUAL SPACE
000833  MOVE    ZERO TO OP-1
000834  GO TO    DO1XT
000835  MOVE    FRST1 TO VALP
000836  IF    HYP EQUAL 0-0
000837  MOVE    EDP TO OP-1
000838  ADD    1 TO EDP
000839  GO TO    DO1XT
000840  MOVE    FRST1 TO IDNLB
000841  PERFORM BS-1 THRU BS-EX
000842  MOVE    DNPTZ XSVLOCK TO OP-1
000843  DO1XT
000844  EXIT
000845  WPTR
000846  WPTR1
000847  MOVE    0 TO SUB
000848  WPTR
000849  ADD    1 TO SUB
000850  IF    FDNY XSUBC NOT EQUAL FRST1
000851  GO TO    WPTR1
000852  MOVE    FDPT XSUBC TO OP-1
000853  WPTR
000854  EXIT
000855  FDPT
000856  FDPT1
000857  MOVE    0 TO SUB
000858  FDPT1
000859  ADD    1 TO SUB
000860  IF    FDNY XSUBC NOT EQUAL FRST1
000861  GO TO    FDPT1
000862  MOVE    SUB TO OP-1
000863  FDPT1
000864  EXIT
000865  WPTR
000866  WPTR
000867  WPTR
000868  WPTR
000967    WRITE    PR
000968    MOVE    SPACE TO    PR
000969    WPRXT
000970    EXIT
000971
000972    WVAL
000973    WRITE    VAL
000974    MOVE    SPACE TO    VAL
000975    MOVE    ZERO TO    VLP
000976    WVALX
000977    EXIT
000978
000979    RCKC
000980    IF    POUT NOT    EQUAL SPACE
000981    MOVE    POUT TO    PR
000982    PERFORM    WPR THRU    WPRXT
000983    MOVE    SPACE TO    POUT
000984    BCKCD
000985    EXIT
000986
000987    SRT-1
000988    PERFORM    SRT-2 THRU    SRTX
000989    ADD    1 TO    OP-1
000990    IF    OP-1 NOT    EQUAL EDP
000991    GO TO    SRT-1
000992    SRT-X
000993    EXIT
000994    SRT-2
000995    MOVE    DNN  XOP-1C TO    WKTAB
000996    ADD    1 TO    OP-1 GIVING    OP-2
000997    MOVE    OP-1 TO    OP-3
000998    SRT21
000999    PERFORM    SRT-3 THRU    SRT3X
001000    ADD    1 TO    OP-2
001001    IF    OP-2 NOT    GREATER EDP
001002    GO TO    SRT21
001003    MOVE    DNN  XOP-1C TO    DNN  XOP-3C
001004    MOVE    WKTAB TO    DNN  XOP-1C
001005    SRT3X
001006    EXIT
001007    SRT-3
001008    IF    DNN  XOP-2C LESS THAN    WAREA
001009    MOVE    DNN  XOP-2C TO    WKTAB
001010    MOVE    OP-2 TO    OP-3
001011    SRT3X
001012    EXIT
001013
001014    BS-1
001015    ADD    SAVLG 1 GIVING    HDNP
001016    MOVE    ZERO TO    VLP
001017    BS-2
001018    ADD    VLP  HDNP GIVING    SVLOC
001019    DIVIDE    2 INTO    SVLOC GIVING    SVLOC
001020    IF    IDLBL EQUAL    DNN  XSVLOC
001021    GO TO    BS-EX
001022    IF    IDLBL LESS THAN    DNN  XSVLOC
001023    MOVE    SVLOC TO    HDNP
001024    IF    IDLBL GREATER THAN    DNN  XSVLOC
001025    MOVE    SVLOC TO    VLP
001026    ADD    1 TO    VLP GIVING    VCNT
001027    IF    VCNT NOT    EQUAL    HDNP
205
001020  GO TO  BS-2
001029  DISPLAY  IDLRL
001030  BS-EX
001031  EXIT
001032
001033
MULT:  ISZ PTST, 2  :GET NEXT STORE POSITION
       STA 3, PTST, 2  :SAVE RETURN ADDRESS
       LDA 1, PACK1, 2  :GET BEGINNING ADDRESS OF
       STA 1, PACK1, 2  :PACK1, PACK2, PACK3, AND
       LDA 1, PACK2, 2  :INITIALIZE TEMPORARY ADDRESS
       STA 1, PACK2, 2  :AREAS
       LDA 1, PACK3, 2
       STA 1, PACK3, 2
       LDA 1, TEN, 2  :GET AN EIGHT FOR EIGHT WORD MULTIPLY
       STA 1, N, 2  :STORE IN N
       LDA 1, FOUR, 2  :GET A FOUR
       STA 1, X4T, 2  :INITIALIZE A COUNTER
       LDA 1, N, 2  :GET NUMBER OF WORDS LONG
       MOV2L 1, 1  :MULTIPLY THE WORD
       DSZ X4T, 2  :BY 16 TO GET PROPER
       JMP -2  :VALUE FOR MULTIPLICATION FACTOR
       STA 1, M, 2  :STORE FINAL VALUE IN M
       LDA 1, PACK3, 2  :GET RECEIVING FIELD FOR
       STA 1, PAC2, 2  :ZEROING OUT ROUTINE
       JSR @ZERA, 2  :GO TO ZERO ROUTINE
       JSR @TER, 2
START:  LDA 1, PACK2, 2  :GET SECOND ARGUMENT
       STA 1, PAC1, 2  :INITIALIZE ADDRESS FOR SHIFTRIGHT
       STA 1, PAC2, 2  :ROUTINE
       JSR @SHFR  :GO TO SHIFTRIGHT ROUTINE
       JSR @TER, 2
       MOV 1, 1, SNC  :CHECK TO SEE IF CARRY BIT IS SET
       JMP +2  :NO
       JMP AA  :YES
AD:  LDA 1, PACK3, 2  :INITIALIZE ADDRESS FOR
       STA 1, PAC1, 2  :SHIFTRIGHT ROUTINE
       STA 1, PAC2, 2
       JSR @SHFR  :GO TO SHIFTRIGHT ROUTINE
       JSR @TER, 2
       DSZ M, 2  :DECREMENT M
       JMP START  :NO ZERO DO AGAIN
       JMP AB  :ZERO
AA:  LDA 1, PACK1, 2  :INITIALIZE ADDRESS FOR
       STA 1, PAC1, 2  :ADDPACK ROUTINE
       LDA 1, PACK3, 2
       STA 1, PAC2, 2
       LDA 1, PAC2, 2
       STA 1, PAC3, 2
       JSR @ADDT, 2  :GO TO ADDPACK ROUTINE
       JSR @TER, 2
       JSR XX  :GO TO INITIALIZE PACK3
       JSR @TER, 2
       JMP AD
AB:  LDA 1, PACK2, 2  :GET SECOND ARGUMENT
       STA 1, PAC1, 2  :INITIALIZE ADDRESS FOR SHIFTRIGHT
       STA 1, PAC2, 2  :ROUTINE
       JSR @SHFR  :GO TO SHIFTRIGHT ROUTINE
       JSR @TER, 2
       LDA 1, FOUR, 2  :GET A FOUR
       STA 1, X4T, 2  :STORE IT
       LDA 0, PACK3, 2  :GET RECEIVING FIELD
       ADD 1, 0  :MODIFY ADDRESS
STA 0, PACT3.2 ; STORE NEW ADDRESS
DSZ PACT3.2
Ag:
LDA 1, 0PACT3.2 ; GET CONTENTS OF RECEIVING FIELD
MOV 1, 1, SNR ; CHECK TO SEE IF EQUAL TO ZERO
JMP +2 ; EQUAL TO ZERO
JMP AX ; NOT EQUAL
DSZ WAT.2 ; DECREMENT COUNTER
JMP +2 ; NOT ZERO
JMP AX+2 ; ZERO
DSZ PACT3.2 ; DECREMENT ADDRESS
JMP Ag ; DO AGAIN
Ax:
LDA 1, ONE.2 ; GENERATE A ONE
STA 1, SFLOG.2 ; SET SIZE ERROR FLAG
LDA 1, N.2 ; GET NUMBER OF WORDS TO BE MOVED
MOVZR 1, 1, SZR ; DIVIDE BY TWO
JMP +2
INC 1, 1 ; IF AC1 IS ZERO INITILIZE IT TO ONE
STA 1, NT.2 ; STORE IN NT
LDA 0, PACK3.2 ; GET RECEIVING FIELD
STA 0, PAC2.2 ; STORE IN SECOND ARGUMENT
ADD 1, 0 ; FOR MATHMOVE ROUTINE
STA 0, PAC1.2 ; STORE MODIFIED ADDRESS FOR MATHMOVE
JSR @HERE1 ; GO TO MOVE ROUTINE
JSR @RTER.2
LDA 1, N.2 ; GET NUMBER OF WORDS TO BE MOVED
MOVZR 1, 1, SZR ; DIVIDE BY TWO
JMP +2
INC 1, 1 ; IF AC1 IS ZERO INITILIZE IF ONE
STA 1, NT.2 ; STORE IN NT
LDA 0, PAC2.2 ; GET SECOND ARGUMENT
LDA 0, PAC3.2 ; SET ADDRESS FOR MATHMOVE ROUTINE
ADD 1, 0 ; FOR PROPER ALIGNMENT
STA 0, PAC2.2 ; SECOND ARGUMENT FOR MATHMOVE ROUTINE
JSR @HERE1 ; GO TO MATHMOVE ROUTINE
JSR @RTER.2
LDA 1, N.2 ; GET NUMBER OF WORDS
MOV 1, 1 ; DIVIDE BY TWO
INC 1, 1 ; INCREMENT BY ONE
LDA 0, PACT2.2 ; GET SECOND ARGUMENT
ADD 1, 0 ; GET NEW ADDRESS
STA 0, PAC2.2 ; STORE IT
LDA 1, OPAC2.2 ; GET CONTENTS OF THAT POSITION
MOVFL 1, 1, SNR ; CHECK TO SEE IF BIT ZERO IS ON
JMP @RTN.2 ; NO, RETURN
LDA 1, PACK2.2 ; GET BEGINNING ADDRESS
STA 1, PAC2.2 ; OF PACK2 FOR ZERO OUT ROUTINE
JSR @ZER.2 ; GO TO ZERO-OUT ROUTINE
JSR @RTER.2
LDA 1, ONE.2 ; GENERATE A ONE
STA 1, 0PAC2.2 ; STORE IT
LDA 1, PACK2.2 ; GET ADDRESSES FOR
STA 1, PAC1.2 ; ADD PACK ROUTINE
LDA 1, PACK3.2
STA 1, PAC2.2
LDA 1, PAC3.2
STA 1, PAC3.2
JSR @ADD.2 ; GO TO ADDPACK ROUTINE
JMP +1
JSR AXX
JSR @RTER.2 ; INITILIZE PACK3
JMP @RTN. 2  RETURN
XX:                  RTST. 2
STA 3, @RTST. 2  SAVE RETURN ADDRESS
LDA 1, PACT. 2  INITIALIZE ADDRESSES
STA 1, PAC1. 2  FOR MATHMOVE
LDA 1, PACK3. 2  FOR ROUTINE
STA 1, PAC2. 2
JSR @MMOV. 2  GO TO MATHMOVE ROUTINE
JSR @RTER. 2
JSR @RTN. 2  RETURN
HERE1:  HERE
SHIFT:  SHIFT
.TITLE MULTIPLY10
.ENT MUL10

; THIS ROUTINE MULTIPLIES ONE PACK BY 10 AND STORES BACK IN SAME PACK
; THE PACK FIELD TO BE USED IS PACTL
MUL10:
    ISZ RTST, 2
    STA 3, @RTST, 2
    LDA 1, PACTL, 2
    LDA 0, N, 2
    NEG 0, 3
    COM 3, 3
    ADD 3, 1
    STA 1, X4T, 2
    ; MIDDLE WORD OF PAC
    STA 0, M, 2
    ; M
    SUB 0, 0
    STA 0, X17T, 2
    ; SET PREV HIGH ORDER VALUE TO ZERO

AA:
    SUB 0, 0
    LDA 3, RX4T, 2
    ; CURRENT PACK WORD CONTENTS
    MOVZL 3, 1
    MOVL 0, 0
    MOVL 1, 1
    ; LOW ORDER * 4
    MOVL 0, 0
    ; HIGH ORDER * 4
    ADDZ 3, 1, SZC
    ; LOW & HIGH ORDER * 5
    INC 0, 0
    ; ADD 1 TO HIGH ORDER IF OVERFLOW
    MOVZL 1, 1
    ; LOW ORDER * 10
    MOVL 0, 0
    ; HIGH ORDER * 10
    LDA 3, X17T, 2
    ; PREV HIGH ORDER * 10
    ADDZ 3, 1, SZC
    ; COMBINE WITH LOW ORDER
    INC 0, 0
    ; INC HI ORDER IF OVERFLOW
    STA 0, X17T, 2
    ; NEXT WORD
    STA 1, RX4T, 2
    ; * 10J
    DSZ X4T, 2
    DSZ M, 2
    JMP AA
    MOVZ 0, 0, SZR
    ; ZERO CARRY, CHECK IF OVERFLOW
    SUB 1, 1
    ; YES, SET CARRY
    JSR @RTN, 2
NEGATIVENUMBER

; GET NEXT STORE POSITION
ISZ RTST, 2
; SAVE RETURN ADDRESS
STA 3, RTST, 2
; CLEAR AC0
SUB 0, 0
; CLEAR NEGATIVE NUMBER FLAG
STA 0, NGNO, 2
; GET BYTE POINTER TO DATA
LDA 3, PVTR, 2
; STORE IN BYTE POINTER
JSR GET
; GET A BYTE
JMP GTER, 2
; SAVE AC1
MOV 1, 3
; TEST BYTES TO SEE IF ASCII NUMBER
AND 0, C60, 2
; NO NOT A NUMBER
SUB# 0, 3, SZR
; TEST
JMP .+11
; GET MASK OF 200
LDA 0, ESMKS, 2
; TEST TO SEE IF BIT EIGHT IS SET
SUB 0, 3, SNR
; NOT SET NUMBER IS POSITIVE
JMP ORTN, 2
; NUMBER IS NEGATIVE GENERATE A ONE
SUBZL 0, 0
; STORE IN NEGATIVE NUMBER FLAG
JMP ORTN, 2
; RETURN
MOV 1, 3
; SAVE AC1
LDA 0, C100, 2
; TEST TO SEE IF BIT SEVEN IS SET
SUB# 0, 3, SNR
; YES IT IS A LETTER
JMP LETT
; GENERATE A ONE
LDA 1, C60, 2
; STORE IN NEGATIVE NUMBER FLAG
JMP ORTH, 2
; RETURN
LETTP: LDA 0, C147, 2
; CHANGE LETTER TO NUMBER
ADD 0, 1
; DO THE CHANGE
JMP -5

211
NEW COMMAND

.TITLE NEW
.NREL
.TXTH 1
.ENT NEW

ISZ RTST, 2
STA J. ORTST, 2
JSR @GCHAP, 2
JSR @ORTE, 2

.SYSTEM
.PCHAR
JSR @ORTE, 2
LDA 0, LF, 2

.SYSTEM
.PCHAR
JSR @ORTE, 2
ISZ NEWF, 2
JSR @ALCRP, 2
JMP +4
DSZ NEWF, 2
JSR @ORTE, 2
JSR @ORTN, 2
DSZ NEWF, 2
JSR @ORTE, 2
JSR @ORTE, 2
PERFORM

PFRM1:

ISZ RTST.2 ; GET NEXT STORE POSITION
STA 3.0RTST.2 ; SAVE RETURN ADDRESS
LDA 3.0P.2 ; GET QUINT POINTER
LDA 0.2.3 ; STORE OPERAND 1 IN SAVE
ISZ SAVE.2
STA 0.0SAVE.2 ; THIS IS THE FIRST STATEMENT
TO EXECUTE FROM THE PERFORM
STA 0.0DTC.2 ; STORE IN TEMPORARY QUINT COUNTER
LDA 0.3.3 ; STORE OPERAND 2 IN NDPT.
ISZ NDPT.2
STA 0.0NDPT.2 ; THIS IS THE LAST STATEMENT
TO EXECUTE FROM THE PERFORM
LDA 0.4.3 ; STORE OPERAND 3 IN CNT.
ISZ CNT.2
STA 0.0CNT.2 ; THIS IS THE NUMBER OF TIMES
TO PERFORM THIS SERIES OF QUNINTS
ISZ RTPT.2
LDA 3.0C.2
INC 3.3
STA 3.0RTPT.2
JMP @RTIN.2 ; PROCESSING COMPLETE. RETURN
.TITL OPEN
.ENT OPEN

OPEN:
ISZ RTST, 2, GET NEXT STORE POSITION
STA 3, @RTST, 2, SAVE RETURN ADDRESS
LDA 1, OFFLG, 2, LOAD_OPCODE_FLAG
MOVZ# 1, 1, .SEZ, FIND OUT CONTENTS (0, 1, 2, OR 3)
JMP RTEP, 2, 3
MOV 1, 1, .SNR
JMP RPEN
MOVZ# 1, 1, .SZP
JMP EPEN
LDA 0, ONP
JMP STAR
DO THE OPEN
RPEN:
LDA 0, ROPEN
JMP STAR
DO THE OPEN
EPEN:
LDA 0, EOPEN
STAR:
STA 0, BLNK
LDA 0, OPIND, 2
SYSTEM
GCHN
JMP -.2
NO FREE CHANNEL AVAILABLE
BLNK:
0
JMP ERR
BAD OPEN
STA 2, CHNL, 3
STORE AC1 IN DATA BASE
MOV 3, 2, RESTORE AC2
JSR @RTN, 2, PROCESSING COMPLETE, RETURN
ERR:
STA 2, ERCODE, 3
STORE ERROR CODE
JSR @RTER, 2, ERRORMT, RETURN
ROPEN:
. ROPEN 77
EOPEN:
. EOPEN 77
OPEN:
. OPEN 77
SUBROUTINE OPENS AN AVAILABLE CHANNEL

INPUT: AC0 BYTE PTR TO NAME OF FILE
       AC2 DATA PTR

OUTPUT: AC1 CHNUM. 2 CHANNEL #

.TITLE OPAVC
.NREL
.TXTM 1
.ENT OPAVC

OPAVC:
  ISZ RTST.2
  STA 3, @RTST.2

OP1:
  SYSTM
  .GCNH
  JSR @RTER.3
  STA 2, CHNUM. 3
  SUB 1, 1
  SYSTM
  .OPEN 77

IÓN: OPEN IT
  JMP .+2
  JMP OP2
  LDA 1, TUNV1
  SUB 2, 1, SNR
  JMP OP1
  JSR @RTER.3

OP2:
  MOV 2, 1
  MOV 3, 2
  JNR @RTN.2

TUNV1: 21
THIS ROUTINE PRINTS SIX OCTAL DIGITS FOR THE CONTENTS OF ACO ON THE MASTER CONSOLE, PRECEDED BY ONE BLANK

.TITLE PROCT
.NREL
.TXTH 1
.ENT PROCT, PRCRL, RTAECl

PROCT:
ISZ RTST, 2
STA 3, ORTST, 2
STA 0, TEMP, 2
LDA 0, FIVE, 2
STA 0, TEMP2, 2
LDA 0, BLK, 2
.SYSTM
.PCHAR
JSR ORTER, 3
SUB 0, 0
LDA 1, TEMP, 2
MOVZL 1, 1
MOVVL 0, 0
MOVVL 1, 1
MOVVL 0, 0
STA 1, TEMP, 2
LDA 1, C60, 2
ADD 1, 0
.SYSTM
.PCHAR
JSR ORTER, 2
LDA 1, TEMP, 2
SUB 0, 0
MOVZL 1, 1
MOVVL 0, 0
MOVVL 1, 1
MOVVL 0, 0
STA 1, TEMP, 2
LDA 1, C60, 2
ADD 1, 0
.SYSTM
.PCHAR
JSR ORTST, 2
DSZ TEMP2, 2
JMP PROT
JSR ORTN, 2

PRINT CARRIAGE RETURN LINE FEED

PRCRL:
ISZ RTST, 2
STA 3, ORTST, 2
LDA 0, CI, 2
.SYSTM
.PCHAR
JSR ORTER, 3
LDA 0, LF, 2
.SYSTM
.PCHAR
JSR ORTER, 3
JSR ORTN, 2

READ CHAR AND ECHO, IF CARRIAGE RETURN PRINT LF AND RETURN +3
RDAEC:  ISZ RTST, 2
STA 3, @RTST, 2  ; READ
JSR @GCHAP, 2  ; ECHO
JSR @RTER, 2
  . SYSTM
  . PCHAR
JSR @RTER, 2
LDA 1, CI, 2  ; CARRYAGE RETURN?
SBA 0, 1, S3R
JSR @RTN, 2  ; NO
ISZ @RTST, 2  ; RETURN +3
LDA 0, LF, 2
  . SYSTM
  . PCHAR
JSR @RTER, 2
JSR @RTN, 2
EDITT: LDA 3.ECHAR.2 GET EDIT TABLE DISPLACEMENT
LDA 0.TABLE GET BEGINNING OF TABLE
ADD 0.3 GET EDIT TABLE DISPLACEMENT
LDA 3.0.3 GET THE ROUTINE TO GO TO
JMP 0.3 GO TO ROUTINE

TABLE: TABL:
NULL : NULL CHARACTER FILL ROUTINE
BLANK : BLANK-FILL CHARACTER ROUTINE
SLASH : SLASH FILL ROUTINE
ZEROT : ZERO-FILL CHARACTER ROUTINE
COMMA : COMMA-FILL CHARACTER ROUTINE
DEBIT : DEBIT SIGN FILL ROUTINE
CREDIT : CREDIT SIGN ROUTINE
PERIOD : PERIOD FILL CHARACTER ROUTINE
DOLAR : DOLLAR SIGN FILL CHARACTER ROUTINE
MINUS : MINUS SIGN FILL CHARACTER ROUTINE
PLUSS : PLUS SIGN FILL CHARACTER ROUTINE
ASTK : ASTERISK FILL ROUTINE
ZEE : ZERO FILL ZERO SUPPRESS ROUTINE
TEST1 : CHARACTER FROM DATA STRING FILL ROUTINE
PUTCH : "V" FILL CHARACTER ROUTINE
RTN : RETURN SHOULD NOT HAVE COME INTO THIS ROUTINE

NULL: LDA 1.NUL.C2 GET A NULL CHARACTER
STA 1.ECHAR.C2 STORE EDIT CHARACTER
JMP PUTIT GO TO STORE ROUTINE
SLASH: LDA 1.SLASH.C2 GET A SLASH
STA 1.ECHAR.C2 STORE EDIT CHARACTER
JMP PUTIT GO TO STORE ROUTINE
BLANK: LDA 1.BLK.C2 GET A BLANK
STA 1.ECHAR.C2 STORE EDIT CHARACTER
JMP PUTIT GO TO STORE ROUTINE
ZEROT: LDA 1.ZER.C2 GET AN ASCII ZERO
STA 1.ECHAR.C2 STORE EDIT CHARACTER
PUTIT: DSZ CNT.C2 DECREMENT TOTAL LENGTH
JMP +2
JSR @RTN.C2 WENT TO ZERO SOMETHING WRONG
LDA 1.ECHAR.C2 LOAD EDIT CHARACTER
LDA 3.PNT.C2 GET BYTE POINTER TO RECEIVING FIELD
STA 3.BYTPTR.C2 STORE IN BYTE POINTER
JSR @PUT GO TO PUT BYTE ROUTINE
JSR @RTER.C2
ISZ PNT.C2 GET NEXT STORE POSITION
DSZ RCNT.C2 DECREMENT REPITION COUNT
JMP +2 NOT ZERO
JMP @B1 ZERO GET ANOTHER EDIT CHARACTER
DSZ CNT.C2 DECREMENT TOTAL LENGTH
JMP PUTIT+1 DO AGAIN
JMP @B1 BETTER BE END OF EDIT FORMAT
PERIOD: DSZ CNT.C2
JMP +2
JSR @RTN.C2 ERROR RETURN
LDA 1.PERD.C2 GET A PERIOD
LDA 3.PNT.C2 GET BYTE POINTER TO RECEIVING FIELD
STA 3.BYTPTR.C2 STORE IN BYTE POINTER
JSR @PUT GO TO PUT BYTE ROUTINE
JSR @RTER.C2
ISZ PNT2.2 ; GET NEXT STORE POSITION
LDA 1,CNT1.2 ; GET COUNT ONE
MOV# 1.1,SZP ; TEST TO SEE IF ZERO
JMP @ B1 ; GET ANOTHER EDIT STRING
LDA 1,CNT2.2
STA 1,RCNT.2
JMP ZEPOT

DEBIT:
LDA 0,NEGNO.2 ; GET NEGATIVE NUMBER FLAG
MOVR 0,0,SNC ; CHECK TO SEE IF SET
JMP BLANK ; FIELD IS POSITIVE
LDA 1,DR.2 ; GET DEBIT SIGN
STA 1,ECHAR.2 ; STORE IN EDIT CHARACTER
LDA 3,PNT2.2 ; GET BYTE POINTER TO STORE POSITION
STA 3,BYPTR.2 ; STORE IN BYTE POINTER
JSR @PUT ; STORE IT
JSR @TERP.2
DSZ RCNT.2
JMP .+2 ; NOT ZERO
JMP @ B1 ; GET ANOTHER EDIT CHARACTER
ISZ ECHAR.2 ; GET LAST HALF OF WORK
ISZ PNT2.2 ; GET NEXT STORE POSITION
JMP .-10 ; DO AGAIN

CREDIT:
LDA 0,NEGNO.2 ; GET NEGATIVE NUMBER FLAG
MOVR 0,0,SNC ; CHECK TO SEE IF SET
JMP BLANK ; FIELD IS POSITIVE
LDA 1,CR.2 ; GET CREDIT SIGN
JMP DEBIT+4

TEST1:
LDA 0,OP2DT.2 ; GET TYPE OF MOVE
LDA 1,FOUR.2 ; NUMERIC TYPE MOVE
SUB 1,0,SZP ; TEST TO SEE IF EQUAL
JMP .+11 ; NO ALPHANUMERIC EDIT MOVE
LDA 1,PNT0.2 ; YES, SEE IF BYTE POINTER IS SET
MOV 1,1,SNR
JMP @PUCH ; YES CHARACTER FILL
LDA 1,CO60.2 ; NO GET A ZERO
STA 1,ECHAR.2 ; STORE IN ECHAR
LDA 1,CL.2
STA 1,ECHT.2
JMP @PUTT1
LDA 1,PNT0.2 ; TEST TO SEE IF SET
MOV 1,1,SNR
JMP @PUCH ; YES PUTCHARACTER OUT
LDA 1,BLK.2 ; GET A BLANK
STA 1,ECHAR.2 ; STORE IN EDIT CHARACTER
STA 1,ECHT.2 ; STORE IN TEMPORARY EDIT CHARACTER
JMP @PUTT1

.B1: B1
PUTT1: PUTTT
PUCH: PTT
DOLR: DOLAR
ASKT: ASKTK
ZER: ZEE
COM1: COMMA
POST: PLUSS
NEGI: MINUS
; READ SAVE FILE TO BE EXECUTED

.TITLE RDSVF
.NREL
.TKTN 1
.ENT RDSVF

RDSVF:  ISJ PTST, 2
        STA 3, OPTST, 2
        LDA 0, TTOP, 2
        JSR @OPAVP, 2
        JSR @ORTEP, 2
        MOV 1, 2
        LDA 0, MSG
        .SVSTM
        .MLL 77
        JSR @ORTEP, 2 ;ERROR: DATA PTR IS IN ACP AFTER .SVSTM
        LDA 2, CHNUM, 3
        .SVSTM
        .CLOSE 77
        JSR @ORTEP, 2
        MOV 3, 2
        LDA 0, TTIN
        JSR @OPAVP, 2 ;OPEN TTI
        JSR @ORTEP, 2
        LDA 0, SVPLN, 2
        MOV 1, 2
        .SVSTM
        .RDL 77 ;READ SAVE FILE NAME
        JSR @ORTEP, 2
        LDA 2, CHNUM, 3
        .SVSTM
        .CLOSE 77
        JSR @ORTEP, 2 ;RESET DATAP IN ACP
        JSR @ORTEP, 2
        .MSG: .XT 768*2
        .TXT /ENTER SAVE FILE NAME(15) /
        .TTIN: .XT 768*2
        .TXT /#TTI/
READ:
ISZ RTST, 2 \GET NEXT STORE POSITION
STA 3 ORTST, 2 \SAVE RETURN ADDRESS
LDA 0 OP1DP, 2 \GET BYTE POINTER
LDA 1 OP1NL, 2 \GET LENGTH OF INPUT
LDA 2 CHANL, 2 \GET THE CHANNEL NUMBER

SYSTEM
PDL 77
JMP EROUT \BAD READ?
JMP FNISH

EROUT:
LDA 0 EOFN, 3 \LOAD END OF FILE CODE
SUB 2, 0, SNR \COMPARE
JMP +3 \END OF FILE
STA 2 ERCD, 3
JSR @RTER, 3
MOV 3, 2
JSR @RTN, 2

FNISH:
MOV 3, 2
LDA 0 OP1NL, 2
STA 0 CNTB, 2
LDA 1 OP1DP, 2 \GET BYTE POINTER
STA 1 OPDP, 2 \STORE IN TMP
AA:
LDA 1 OPDP, 2 \GET BYTE POINTER
STA 1 BVPTR, 2 \STORE IN BYTE POINTER
JSR @GET \GET A BYTE
JSR @RTER, 2 \ERROR
LDA 0 CL, 2 \GET A CARRAGE RETURN
SUB 0, 1, SNR \TEST TO SEE IF EQUAL
JMP +42 \NO NOT A CARRAGE RETURN
JMP +45 \CARRAGE RETURN
ISZ OPDP, 2 \GET NEXT POSITION
DSZ CNTB, 2 \DECREMENT LENGTH
JMP AA
JMP +411
LDA 1 BLK, 2 \STORE OUT A BLANK
LDA 3 OPDP, 2 \GET BYTE POINTER
STA 3 BVPTR, 2 \STORE IN BYTE POINTER
JSR @PUT \STORE OUT BLANK
JSR @RTER, 2
JSZ OPDP, 2 \DECREMENT COUNTER
JMP -7
SUB 1, 1 \CLEAR AC1
STA 1 TOC, 2 \STORE IN TEMP QUINT CTR
JSR @RTN, 2 \PROCESSING COMPLETE, RETURN
.TITL RELATIONAL
.ENT RLAT1
.EXTN MATHP
.EXTD GET
.NREL

MPACK: MATHP

RLAT1: LDA 1,0.P1DT,2, LOAD OPER1 TYPE
        LDA 0,0.P2DT,2, LOAD OPER2 TYPE
        ADD 0,1, ADD THE TWO TOGETHER
        MOVZP 1,1.SNR, CHECK RESULT FOR 2 OR GREATER
        JMP @MPACK, +< 2, GO TO THE MATH-PACKAGE
        LDA 0,THO,2, LOAD A 2
        JMP 1,0.SNR, CHECK IF 2 OR > 2
        JMP @MPACK, EQUAL TO 2, GO TO THE MATH-PACKAGE
        LDA 1,0.P1DT,2, LOAD ARG1 TYPE
        LDA 0,ONE,2, LOAD A ONE
        SUB 1,0.SZR, CHECK IF TYPY = 1
        JMP CARG2, NOT 1, CHECK ARG2
        LDA 1,DECI,2, LOAD AC1 WITH ARG DECIMAL PTR.
        JMP CNTR, JUMP TO CHECK IF INTEGER
        CARG2: LDA 1,0.P2DT,2, LOAD ARG2 TYPE
        LDA 0,ONE,2, LOAD A 0
        SUB 1,0.SZR, CHECK IF TYPE EQUAL 1
        JMP $SKPG, JUMP TO RELATIONAL TEST
        LDA 1,DEC2,2, LOAD AC1 WITH DECIMAL PTR.
        JMP $SKPG, JUMP TO RELATIONAL TEST
        CNTR: MOV 1,1.SZR, CHECK TO SEE IF = 0
        JMP @RTER,2, NOT ZERO, RUNTIME ERROR
        $SKPG: LDA 1,OP1DL,2, LOAD OPER1 LENGTH
        STA 1,CNT1,2, STORE IN CNT1
        LDA 1,OP2DL,2, LOAD OPER2 LENGTH
        STA 1,CNT2,2, STORE IN CNT2
        LDA 1,OP1P,2, LOAD OPER1 BYTE PTR
        STA 1,PNT1,2, STORE IN PNT1
        LDA 1,OP2P,2, LOAD OPER2 BYTE PTR
        STA 1,PNT2,2, STORE IN PNT2
        A: SUB 0,0, CLEAR AC0
        STA 0,CHKND,2, STORE IN CHKND
        LDA 1,CNT2,2, LOAD CNT2
        MOV 1,1.SNR, CHECK TO SEE IF = 0
        JMP +3, NOT ZERO
        MOV# 1,1.SNC, JNEG?
        JMP AL, EQUAL TO ZERO
        LDA 1,CNT1,2
        MOV 1,1.SNR, EQUAL TO ZERO
        JMP NTEST, GO TO NEGATIVE TEST
        MOV# 1,1.SZC, JNEGATIVE?
        JMP NTEST
        LDA 1,OP2DT,2
        LDA 0,SIX,2
        SUB 1,0.SNR, CHECK TO SEE IF ZERO FILL
        JMP +4
        LDA 1,PLK,2
        STA 1,CHAR2,2
        JMP +3
        LDA 1,C60,2
        STA 1,CHAR2,2
        JSR ARG1
        JSR @RTER,2
        JMP CHECK
        NTEST: LDA 1,MFLAG,2
        MOV 1,1.SNR, TEST T SEE IF SET

222
RETURNPOINT

LDA 2, USP ; RESTORE AC2
STA 0, X17T, 2
ISZ RTST, 2 ; GET NEXT STORE POSITION
LDA 0, ORTST, 2 ; BRING INTO AC0
MOVL 0, 0, SZC ; CHECK TO SEE IF BIT ZERO IS SET
JMP ERTRI
MOVL 3, 3 ; SHIFTL R EMTE RETURN POINT
MOVOR 3, 3 ; SET CARRY PUT INTO BIT ZERO
STA 3, ORTST, 2

ERTRI:
DSZ RTST, 2
LDA 3, ORTST, 2
MOVL 3, 0 ; DECREMENT RETURN POINTER, MOVE IT INTO AC3 AND SHIFTL E
MOVOR 0, 0 ; BRING IN CARRY
STA 0, ORTST, 2 ; STORE IN RTST POINTER
DSZ RTST, 2
LDA 0, X17T, 2
JMP 0, 3

RETRN:
LDA 3, ORTST, 2 ; LOAD IT INTO AC2
STA 3, X17T, 2
SUBC 3, 3
STA 3, ORTST, 2
LDA 3, X17T, 2
DSZ RTST, 2 ; GET THE RETURN POINT
INC 3, 3 ; INCREMENT AC3
JMP 0, 3 ; RETURN

GCHAR:
ISZ RTST, 2 ; GET NEXT STORE POSITION
STA 3, ORTST, 2 ; SAVE RETURN ADDRESS

JSR @RTFR, 2
LDA 3, DEL, 2
SUBW 0, 3, SZR ; CHECK FOR PUBLISH
JSR @RTN, 2 ; NOT A PUBLISH
SUBZL 3, 3
STA 3, CPFFL, 2 ; SET PUBLISH FLAG
JSR @RTER, 2
JSR @RTN.2 ; NOT SET, EQUAL COMPARE
JMP C11 ; SET EQUAL COMPARE ZERO OUT TEMPORARY QUINT COUN
A1:
LDA 1,CNT1.2
MOV 1.1,SNR ; EQUAL TO ZERO
JMP .+3 ; YES
MOV L# 1.1,SNR ; NEGATIVE
JMP A2
LDA 1,OPIDT.2
LDA 0,SIX.2
SUB 1,6,SNR ; CHECK TO SEE IF ZERO FILL
JMP +4
LDA 1,BLK.2
STA 1,CHAR1.2
JMP .+3
LDA 1,660.2
STA 1,CHAR1.2
JSR AR0?
JSR @RTER.2
JMP CHECK
A2:
JSR AR0?
JSR @RTER.2
JSR AR0?
JSR @RTER.2
CHECK:
LDA 0,CHAR1.2 ; LOAD CHAR1
LDA 1,CHAR2.2 ; LOAD CHAR2
SUB 0,1,SNR ; CHAR2 = CHAR1
JMP INC3 ; RESULT = 0 (ARG1 = ARG2)
MOV L# 1,5ZC ; MOVE BIT ZERO TO CARRY
JMP INC2 ; BIT0 = 1 (ARG1 > ARG2)
JMP INC1 ; BIT0 = 0 (ARG1 < ARG2)
INC3 ISZ CHKWD.2 ; INCREMENT CHKWD (WILL END UP A 3)
INC2 ISZ CHKWD.2 ; INCREMENT CHKWD (WILL END UP A 2)
INC1 ISZ CHKWD.2 ; INCREMENT CHKWD (WILL END UP A 1)
LDA 3,ROCHK.2 ; LOAD REQUEST CHECK
LDA 1,THREE.2
SUB # 3,1,SNR
JMP OTHER
LDA 1,CHKWD.2 ; LOAD CHECK WORD
SUB 3,1,SNR ; CHKWD = ROCHK
JMP A0 ; RESULT = ZERO
JMP B1 ; FLAG IS SET
C11:
SUB 1,1
STA 1,TDC.2
JMP @RTN.2 ; RETURN
A0:
LDA 1,NFLAG.2 ; LOAD NOT FLAG
MOV 1,1,SNR ; C11CHECK IF FLAG IS SET
JMP C11 ; FLAG IS SET
JSR @RTN.2
B1:
DSZ CNT1.2 ; FLAG NOT SET, DECREMENT CNT1
JMP .+1 ; CNT1 NOT EQUAL ZERO
DSZ CNT2.2 ; DECREMENT CNT2
JMP .+1 ; CNT2 NOT ZERO
JSZ PNT1.2 ; INCREMENT PNT1
JSZ PNT2.2 ; INCREMENT PNT2
JMP A0 ; GO NO NEXT CHARACTER
AR01 ISZ RTST.2 ; GET NEXT STORE POSITION
STA 3,APRTST.2 ; SAVE RETURN ADDRESS
LDA 3,PNT1.2 ; LOAD OPER1 BYTE PTR
STA 3,APYPR.2 ; PASS TO GET BYTE ROUTINE
JSR @GET
JSR @RTER.2
LDA 0.MSK, 2 ;LOAD MASK 177
AND 0.1 ;SAVE RIGHT 7 BITS
STA 1.CHAR1, 2 ;STORE THE CHARACTER
JSR @RTN, 2

ARG2:
ISZ RTST, 2
STA 3.@RTST, 2 ;SAVE RETURN ADDRESS
LDA 3.PNT2, 2 ;LOAD OFFER BYTE POINTER
STA 3.BYPTR, 2 ;PASS TO GET BYTE ROUTINE
JSR @GET ;GET THE BYTE
JSR @RTFR, 2
LDA 0.MSK, 2 ;LOAD MASK 177
AND 0.1 ;SAVE RIGHT 7 BITS
STA 1.CHAR2, 2 ;STORE THE CHARACTER
JSR @RTN, 2

OTHER:
LDA 1.TWO, 2
SUB# 3.1.SZR
JMP ONEA ;IT IS A ONE
LDA 1.CHKND, 2
LDA 0.THRSS, 2
SUB# 1.0.SNR ;CHECK TO SEE IF CHARACTERS ARE EQUAL
JMP B1 ;YES, GET ANOTHER CHARACTER
SUB# 3.1.SZR
JMP A0 ;GREATER THAN CONDITION
JMP NTTEST

ONEA:
LDA 1.CHKND, 2
LDA 0.THRSS, 2
SUB# 1.0.SNR ;CHECK TO SEE IF CHARACTERS ARE EQUAL
JMP B1 ;YES, GET ANOTHER CHARACTER
SUB# 3.1.SZR
JMP A0 ;LESS THAN CONDITION
JMP NTTEST
THESE ROUTINES RUN AN ENTERED PROGRAM

.TITLE RUN
.NREL
.ENT RUN, CONT, STEP, GTONT
.TXTM 1

RUN:
.ISZ RTST, 2
.STA 3, @RTST, 2
.JSR @GCHAP, 2
.JSR @RTER, 2

.PCHAR
.JSR @RTER, 3
.LDA 1, C1, 2
.SUB 0, 1, SNR

.CARTRAGE RETURN?
.JMP RUN1

.JMP RUN1+C

.JSR @MNCP, 2

.GET #

.JMP +2

.NOT #

.JMP RUN+1

.LDA 1, 5
.SUB 0, 1, SNR
.JMP .+3

.ISZ CERFL, 2
.JSR @RTER, 2
.JSR @GCHAP, 2
.JSR @RTER, 2

.PCHAR
.JSR @RTER, 2
.LDA 0, LF, 2

.SYSTEM
.PCHAR
.JSR @RTER, 2

.SUB2L 1, 1
.STA 1, OC, 2
.JSR @PPONP, 2
.JSR @RTER, 2
.JSR @RTN, 2

.RUN1:
.LDA 0, LF, 2

.SYSTEM
.PCHAR
.JSR @RTER, 3

.SUB2L 1, 1
.STA 1, OC, 2

.QUINT # TO RUN AT
.JMP RUN2

.CONT:
.ISZ RTST, 2
.STA 3, @RTST, 2
.JSR @RDSEP, 2

.GET CHAR
.JSR @RTER, 2
.JMP .+1

.NOT C/R
.JMP .+1

.SUB 0, 0
.JMP .+1

.JMP RUN2

.STEP:
.ISZ RTST, 2
.STA 3, @RTST, 2
.JSR @GCHAP, 2
.JSR @RTER, 2

.SYSTEM
.PCHAR
.JSR @RTER, 3
LDA 1, C1, 2
SUB 0, 1, SZR
JMP +5
LDA 0, LF , 2
SYSTEM
PCHAR
JSR @RTER, 3
LDA 1, THO , 2
STA 1, STPFL, 2
JSET STOP FLAG
RUN2:
JSR GTONT
JSR @RTER, 2
LDA 1, OP , 2
LDA 0, 1 , 3
LDA 1, BELL, 2
AND 0, 1
STA 1, OPFLG, 2
MOVZR 0, 0
MOVZR 0, 0
MOVZR 0, 0
LDA 1, OPDSP, 2
ADD 0, 3
LDA 0, 00, 3
STA 0, SP, 2
JSR @EXONP, 2
EXECUTE QUINT
JSR @RTER, 2
LDA 0, STPFL, 2
MOV 0, 0, SNR
STOP OR STEP?
JMP RUN2
MOV 0, 0, SZC
JMP +3
JSR @PRONP, 2
JSR @RTER, 2
SUB 0, 0
STA 0, STPFL, 2
JSR @RTN, 2
GTONT:
ISZ RTST, 2
STA 3, ORTST, 2
LDA 0, QC, 2
QUIT NUMBER
ADC 1, 1
ADD 1, 0
MOVZL 0, 1
MOVZL 1, 1
ADD 1, 0
LDA 1, QNST, 2
PLUS QUINT TABLE STARTING ADDRESS
ADD 1, 0
STA 0, OP, 2
JSR @RTN, 2
.TXTM 0
S:
.TXT /S/
THESE ROUTINES PROCESS THE SET SUBSCRIPT AUXILIARY STATE TABLES

.TITLE SETSB
.NREL
.TXTM 1
.ENT SETSB, SSB11, SSB12, SSB21, SSB22, SSB23

SEBSB: MOV 3, 1
       LDA 2, OP, 2
       LDA 0, 2, 3
       STA 0, SBF1, 2
       LDA 0, 3, 3
       STA 0, SBF2, 2
       LDA 0, 4, 3
       STA 0, SBF3, 2
       MOV 1, 3
       JMP 1, 3

SSB11: SUB 0, 0
       STA 0, SBF1, 2
       JMP 1, 3

SSB12: SUB 0, 0
       STA 0, SBF2, 2
       JMP 1, 3

SSB13: SUB 0, 0
       STA 0, SBF3, 2
       JMP 1, 3

SSB21: ISZ RTST, 2
       STA 3, ORTST, 2
       LDA 0, OP1DT, 2
       MOV 5, 0
       MOVZ 5, 0, 5ZC
       JNE S211

S211: MOV 5, 0, S211
       LDA 3, PACK1, 2
       LDA 1, 3, 3
       STA 1, TEMP2, 2
       LDA 0, OPIDX, 2
       MOV 1, 1
       SUBZ 1, 0, SNC
       JSR ORTER, 2
       LDA 0, OP1DP, 2
       STA 0, TEMP, 2
       LDA 0, OP1DL, 2
       SUB 1, 1
       JSR S212

S212: LDA 0, SAV1, 2
       JSR @DNSTP, 2

JSR ORTER, 2
LDA 3, PACK1, 2
STA 0, 3, 3
LDA 0, TEMP2, 2
MOV 1, 3
STA 1, SAV1, 2
LDA 1, 3, 3
JSR @TER, 2
LDA 3, MSK4, 2
AND 3, 1
SUBZ 0, 1, SNC
JSR ORTER, 2
LDA 3, SAV1, 2
LDA 0, 2, 3
STA 0, SAV1, 2
JSR @RSSTP, 2
JSR  @RTER, 2
LDA  0, TEMP, 2
LDA  0, SAV1, 2
LDA  1, OPDX, 2  ;# CHAR FROM ARRAY
S212:  JSR  SSF00  ;STORE ACTUAL BYTE PTR
JSR  @RTER, 2
STA  1, OPDP, 2
JSR  @RTN, 2
SS22:  ISZ  RTST, 2
STA  3, @RTST, 2
LDA  0, OPDT, 2
MOVZ  0, 0
MOVZR  0, 0, SZC  ;LEVEL =?1
JMP  S221  ;YES
LDA  3, PACK2, 2  ;SUBSCRIPT IN 4TH WORD OF PACK
LDA  1, TEMP, 2
LDA  0, OPDX, 2  ;# OCCURS
SUBZ  1, 0, SNC
JSR  @RTER, 2  ;ARRAY EXCEEDED
LDA  0, OPDP, 2  ;ARRAY BYTE PTR
LDA  0, OPDL, 2  ;LENGTH OF ARRAY
SUB  1, 1  ;ZERO CHAR TO FRONT
JMP  S222
S221:  LDA  0, SAV2, 2  ;GROUP DATA NAME ENTRY
JSR  @ONSTP, 2  ;GET DATA NAME POINTER
JSR  @RTER, 2
LDA  3, PACK2, 2
LDA  0, 3, 3  ;SUBSCRIPT
STA  0, TEMP, 2
MOV  1, 3
STA  1, SAV1, 2
LDA  1, 3, 3  ;GET NUMBER OF OCCURS
LDA  3, MSK4, 2  ;GET MASK
AND  3, 1  ;SAVE RIGHT BYTE
SUBZ  0, 1, SNC
JSR  @RTER, 2  ;BOUND EXCEEDED
LDA  3, SAV1, 2
LDA  0, 2, 3
STA  0, SAV1, 2
LDA  0, 0, 3
JSR  @RSSTP, 2
JSR  @RTER, 2
STA  0, TEMP, 2
LDA  0, SAV1, 2
LDA  1, OPDX, 2  ;# CHAR FROM ARRAY
S222:  JSR  SSF00  ;STORE ACTUAL BYTE PTR
JSR  @RTER, 2
STA  1, OPDP, 2
JSR  @RTN, 2
SSB23:  ISZ  RTST, 2
STA  3, @RTST, 2
LDA  0, OPDT, 2
MOVZ  0, 0
MOVZR  0, 0, SZC  ;LEVEL =?17
JMP  S231  ;YES
LDA  3, PACK2, 2  ;SUBSCRIPT IN 4TH WORD OF PACK
LDA  1, 3, 3  ;SUBSCRIPT IN 4TH WORD OF PACK
STA  1, TEMP, 2
LDA  0, OPDX, 2  ;# OCCURS
SUBZ 1, 0, SNC
JSR @RTEP, 2 ; ARRAY EXCEEDED
LDA 0, OP20P, 2 ; ARRAY BYTE PTR
STA 0, TEMP, 2 ; LENGTH OF ARRAY
SUB 1, 1 ; ZERO CHAR TO FRONT
JMP S232
S231: LDA 0, SAVJ, 2 ; GROUP DATA NAME ENTRY
JSR @NSTP, 2 ; GET DATA NAME POINTER
JSR @RTEP, 2
LDA 3, PACK2, 2 ; SUBSCRIPT
LDA 0, 3, 3
STA 0, TEMP, 2
MOV 1, 3
STA 1, SAV1, 2
LDA 1, 3, 3 ; GET NUMBER OF OCCURS
LDA 3, MSK4, 2 ; GET MASK
AND 3, 1
SUBZ 0, 1, SNC
JSR @RTEP, 2 ; BOUNDS EXCEEDED
LDA 3, SAV1, 2
LDA 0, 2, 3
STA 0, SAV1, 2 ; GROUP LENGTH
LDA 0, 0, 3
JSR @PSSTP, 2
JSR @RTEP, 2
STA 0, TEMP, 2
LDA 0, SAV1, 2
LDA 1, OP20X, 2 ; # CHAR FROM ARRAY
S232: JSR $5800 ; STORE ACTUAL BYTE PTR
JSR @RTEP, 2
STA 1, OP20P, 2
JSR @RTN, 2
;
SUBROUTINE CALCULATES BYTE POINTER FOR SUBSCRIPT
;
INPUT IS:
;
AC0 TOTAL LENGTH OF ONE ARRAY ELEMENT
AC1 NUMBER OF CHARACTERS FROM START OF (GROUP)
TEMP, 2 (GROUP) ARRAY STARTING POINTER
TEMP2, 2 SUBSCRIPT NUMBER (INTO ARRAY)

$5800: ISZ RTST, 2
STA 3, RTST, 2
SUB 3, 3
DSZ TEMP2, 2
JMP .+2
JMP $02
S01: ADD 0, 3 ; MULTIPLES LENGTH * SUBSCRIPT-1
DSZ TEMP2, 2
JMP $01
S02: ADD 3, 1 ; ADD # CHAR TO START OF ARRAY
LDA 3, TEMP, 2
ADD 3, 1 ; ADD # CHAR FROM START OF CORE
TITL SHIFTLEFT
ENT MOLF, MOZLF
NREL

; CALL TO THIS ROUTINE IS (N, PAC1, PAC2)
MOZLF: ISZ RTST, 2 ; GET NEXT STORE POSITION
STA 3, RTST, 2 ; SAVE RETURN ADDRESS
JSR SETZ
JSR @RTER, 2
LDA 1, @PAC1, 2 ; GET FIRST ARGUMENT
MOVZ 1, 1
JMP MOLF+12 ; CLEAR CARRY DO A SHIFT-LEFT MOVE

MOLF: ISZ RTST, 2 ; GET NEXT STORE POSITION
STA 3, @RTST, 2 ; SAVE RETURN ADDRESS
JSR SETZ
JSR @RTER, 2
JMP +6
DSZ 1, 2 ; DECREMENT LENGTH
JMP +2 ; NOT ZERO
JMP @RTN, 2 ; RETURN
DSZ PAC1, 2 ; GET NEXT ADDRESS
DSZ PAC2, 2 ; GET NEXT STORE POSITION
LDA 1, @PAC1, 2 ; GET FIRST ARGUMENT
MOVL 1, 1 ; SHIFT LEFT
STA 1, @PAC2, 2 ; STORE IT
JMP MOLF+5 ; DO AGAIN

SETZ: ISZ RTST, 2
STA 3, @RTST, 2 ; SAVE RETURN ADDRESS
LDA 1, N, 2 ; GET NUMBER OF WORDS LONG
STA 1, I, 2 ; STORE IT IN I
LDA 3, @PAC1, 2 ; GET ARGUMENT TO BE SHIFTED
ADD 1, 3 ; ADD NUMBER OF WORDS TO BE USED
STA 3, @PAC1, 2 ; STORE BACK IN PAC1
LDA 3, @PAC2, 2 ; GET PLACE TO STORE AFTER MOVE
ADD 1, 3 ; TO GET PROPER PLACEMENT
STA 3, @PAC2, 2 ; STORE IT BACK OUT IN PAC2
DSZ PAC1, 2 ; TO GET PROPER ALIGNMENT
DSZ PAC2, 2 ; FOR STORAGE
JMP @RTN, 2 ; RETURN
CALL TO THIS ROUTINE IS (N, PAC1, PAC2).
SHIFT: ISZ RTST, 2 \ GET NEXT STORE POSITION
STA 3, RTST, 2 \ SAVE RETURN ADDRESS
LDA 3, N - 2 \ GET NUMBER OF WORDS TO BE SHIFTED
STA 3, NT, 2 \ STORE IN NO
AA:  LDA 1, @PAC1, 2 \ GET FIRST WORD TO BE SHIFTED
MOVR 1, 1 \ SHIFT
STA 1, @PAC2, 2 \ STORE IN PAC2
ISZ PAC1, 2 \ GET NEXT WORD TO BE SHIFTED
ISZ PAC2, 2 \ GET NEXT STORE POSITION
DSZ NT, 2 \ DECREMENT NUMBER OF WORDS
JMP AA \ SKIP NEXT STATEMENT IF NOT ZERO
JMP @RTN, 2 \ RETURN
.TITL SIGN
.ENT PLU5. MVLUS
.EXTN GETABP. PUTTV. BL. RRT. PRT. ERT. P
.EXTD PUT. GET

PLU5:  LDA 1. PLUS. 2  GET A PLUS SIGN
       STA 1. ECHAR. 2  STORE IN EDIT CHARACTER
       JMP TEST
MVUS:  LDA 1. MINUS. 2  GET THE EDIT SIGN
       STA 1. ECHAR. 2  STORE IN EDIT CHARACTER
       JMP TEST  ; FIND OUT WHAT TYPE OF SIGN
TEST:  LDA 1. SIGN. 2  GET THE SIGN
       MOVZR# 1. 1.  SEZ
       JMP SITS  ; SIGN TRAILING SEPARATE TYPE THREE
       MOV 1. 1. SNR
       JMP SILI  ; SIGN LEADING INCLUDED TYPE ZERO
       MOVZR# 1. 1.  SZR
       JMP SITI  ; SIGN TRAILING INCLUDED TYPE TWO
       JMP SISL  ; SIGN LEADING SEPARATE TYPE ONE
SILS:  LDA 1. PNT0. 2  GET BYTE POINTER TO DATA
       MOV 1. 1. SZR  TEST TO SEE IF BYTE POINTER IS SET
       JSR BLNK  JMP TO BLANK FILL
       LDA 1. PNT1. 2  GET DATA
       STA 3. BVPTR. 2  STORE IN BYTE POINTER
       JSR @GET  GO TO GET BYTE ROUTINE
       JMP @RTER. 2
       SUB 0. 0  CLEAR ACA
       SUB# 0. 1. SZR  TEST TO SEE IF SIGN
       JMP .+2  FIELD IS NEGATIVE
       JMP POSIT  FIELD IS POSITIVE
       LDA 1. RCNT. 2  GET REPITITION COUNT
       LDA 1. ONE. 2  GET A ONE
       SUB# 0. 1. SNR  TEST TO SEE IF EQUAL TO ONE
       JMP @ONMIN  ONE MINUS SIGN
NEG:  LDA 1. MINUS. 2  GET A MINUS SIGN
      STA 1. ECHAR. 2  STORE IN EDIT CHARACTER
      JMP @GETBV  GO TO GET BYTE ROUTINE
ONMIN: LDA 1. MINUS. 2  GET A MINUS SIGN
       STA 1. ECHAR. 2  STORE IN EDIT CHARACTER
       JMP @PUTT1
POSIT: LDA 1. PLUS. 2  GET A PLUS SIGN
       STA 0. ECHAR. 2  GET EDIT CHARACTER
       SUB# 0. 1. SNR  TEST TO SEE IF EQUAL
       JMP .+4
       LDA 0. BLK. 2  GET A BLANK FIELD IS NEGATIVE
       STA 0. ECHAR. 2  STORE IN EDIT CHARACTER. EDIT CHARACTER IS NEGATIVE
       JMP @GETBV  GET BYTE ROUTINE
       LDA 1. RCNT. 2  GET REPITITION COUNT
       LDA 1. ONE. 2  GET A ONE
       SUB# 0. 1. SNR  TEST TO SEE IF EQUAL TO ONE
       JMP @ONPLS  ONE PLUS SIGN
       LDA 0. PNT0. 2  GET BYTE POINTER
       MOV# 0. 0. SNR  TEST TO SEE IF EQUAL TO ZERO
       JMP .+2
       JMP .+4
       LDA 1. PLUS. 2  GET A PLUS SIGN
       STA 1. ECHAR. 2  STORE IN EDIT CHARACTER
       JMP @GETBV  GET BYTE ROUTINE
       LDA 1. BLK. 2  STORE IN EDIT CHARACTER
       STA 1. ECHAR. 2  STORE IN EDIT CHARACTER
       STA 1. ECHT. 2  STORE IN EDIT CHARACTER TEMPOSARV
ONPLS:  LDA 1.PLUS,2 ; GET A MINUS SIGN
STA 1.ECHAR,2 ; STORE IN EDIT CHARACTER
JMP @PUTT1 ; GO TO STORE ROUTINE

BLNK:  ISZ RTST,2 ; SAVE RETURN ADDRESS
STA 3.RTST,2
LDA 1.BLK,2 ; GET A BLANK
LDA 3.PNT2,2 ; GET STORE POINTER
STA 3.BYPTR,2 ; STORE IN BYTE POINTER
JSR @PUT ; STORE THE BLANK
JMP @RTER,2

ISZ PNT2,2 ; GET NEXT STORE POSITION
JSZ .+2 ; NOT ZERO
JMP @B1 ; GET ANOTHER EDIT CODE

DSZ RCNT,2 ; DECREMENT REPETITION COUNT
JMP .+2 ; NOT ZERO
JMP .+4
SUB 1.1 ; CLEAR AC1
STA 1.PNT0,2 ; CLEAR ZERO FLAG
JMP AAA ; RESTORE POINTERS
DSZ CNT2,2 ; DECREMENT TOTAL LENGTH
JMP BLANK ; DO AGAIN
JMP @RTN,2 ; RECEIVING FIELD TO SHORT

AAA:  LDA 1.OP1DP,2 ; GET BYTE POINTER TO SENDING FIELD
STA 1.PNT1,2 ; RESTORE BYTE POINTER
LDA 1.OP1DL,2 ; GET THE LENGTH TO SENDING FIELD
STA 1.CNT1,2 ; RESTORE COUNTER
JSR @RTN,2 ; RETURN

SITS:  DSZ CNT2,2 ; DECREMENT TOTAL LENGTH
JMP @RTER,2 ; NOT ZERO SOMETHING IS WRONG
LDA 3.PNT1,2 ; GET THE SIGN
STA 3.BYPTR,2 ; STORE IN BYTE POINTER
JSR @GET ; GO TO BYTE ROUTINE
JMP @RTER,2
MOVW 1.1,SNR ; CHECK TO SEE IF FIELD IS POSITIVE
JMP POT ; FIELD IS BLANK PLUS RESULT
LDA 3.PNT2,2 ; GET BYTE POINTER TO RECEIVING FIELD
STA 3.BYPTR,2 ; STORE IN BYTE POINTER
JSR @PUT ; GO TO STORE CHARACTER ROUTINE
JMP @RTER,2
JMP @RTN,2 ; RETURN TO INTPREPTEP

POT:  LDA 1.MINUS,2 ; TEST TO SEE IF MINUS SIGN
LDA 0.ECHAR,2 ; BRING IN CURRENT EDIT CHARACTER
SUBW 0.1,SNR ; TEST TO SEE IF EQUAL
JMP .+5 ; FIELD IS POSITIVE
LDA 1.BLK,2 ; BRING IN A BLANK
LDA 3.PNT2,2 ; GET POINTER TO RECEIVING FIELD
STA 3.BYPTR,2 ; STORE IN BYTE POINTER
JSR @PUT ; STORE THE CHARACTER
JMP @RTER,2
JMP @RTN,2 ; RETURN TO INTPREPTEP
LDA 1.ECHAR,2 ; GET THE EDIT CHARACTER
JMP .-6 ; PUT THE CHARACTER OUT

SILI:  LDA 0.NEGNO,2 ; GET NEGATIVE NUMBER FLAG
MOVW 0.0,SNR ; TEST TO SEE IF SET
JMP POSIT ; NOT SET NUMBER IS POSITIVE
LDA 1.RCNT,2 ; GET REPETITION COUNT
LDA 1.ONE,2 ; GET A ONE
SUBW 0.1,SNR ; TEST TO SEE IF EQUAL OTO ONE
JMP OMNIN ; ONE MINUS SIGN
JMP NEGA ; ZERO SUPPRESS PUT OUT SIGN
LDA 0,NEONO.2  ;GET NEGATIVE NUMBER FLAG
MOVX 0,0,SHC  ;TEST TO SEE IF SET
JMP POSIT   ;NOT SET NUMBER IS POSITIVE
JMP -10     ;NUMBER IS NEGATIVE
PUTTY: PUTTT
.BI: 01
OTBY: OETBT
.TITL SIZEROUND
.ENT SIZER

.SZ RSRT, 2
STA 3, @RSRT, 2
SUBZL 1, 1
STA 1, MPFLG, 2
JSR @RTN, 2
.TITL STOP
.ENT STOP1, STOP2, KILL

KILL:
JSR @RTER, 2
JSR @RTER, 2
JMP .+1

.SYSTEM THIS IS THE ENTRY POINT TO
.RESET TERMINATE THE PROGRAM ALL
JSR @RTER, 3 PRESENTLY OPENED FILES ARE
.SYSTEM CLOSED, AND CONTROL IS
RTN PASSED BACK TO THE
JSR @RTER, 3 OPERATING SYSTEM

STOP1:
ISZ RTST, 2 GET NEXT STORE POSITION
STA 3, @RTST, 2 SAVE RETURN ADDRESS
ISZ STPFL, 2 SET STOP FLAG
JSR @RTN, 2

STOP2:
ISZ RTST, 2 GET NEXT STORE POSITION
STA 3, @RTST, 2 SAVE RETURN ADDRESS
.SYSTEM GET A CHARACTER FROM THE TTI
JSR @RTER, 3
MOV 3, 2 RESTORE AC2
LDA 1, REMSK, 2 LOAD AC1 WITH BYTE MASK
AND 1, 0 CLEAR THE LEFT BYTE
LDA 3, C1, 2 LOAD AC3 WITH CARRIAGE RETURN
SUB 0, 3, SZR COMPARE
JMP .+2 NOT EQUAL, ECHO THE CHARACTER
JMP STOP3 NOT EQUAL

.SYSTEM
.PCHAR
JSR @RTER, 3
MOV 3, 2 RESTORE AC2
JMP STOP2+2

STOP3:
.SYSTEM
.PCHAR
JSR @RTER, 3
MOV 3, 2 RESTORE AC2
LDA 0, LF, 2 LOAD LINE FEED CHARACTER
.SYSTEM
.PCHAR
JSR @RTER, 3
MOV 3, 2 RESTORE AC2
JMP @RTN, 2 NOT EQUAL, PROCESSING COMPLETE, RETURN
.TITL STORECHANNEL
.ENT STRIT
.NREL

STRIT:  ISZ RTST, 2   GET NEXT STORE POSITION
STA 3, @RTST, 2   SAVE RETURN ADDRESS
LDA 3, OP1DP, 2   LOAD I/O BYTE PTR
LDA 1, CHANL, 2   LOAD CHANNEL NO.
MOVZ 3, 3   CONVERT BYTE PTR. TO WORD PTR
STA 1, 5, 3   STORE CHANNEL NO. IN WORD SIX
JMP @RTN, 2   RETURN, CHANNEL NO. STORED
STATE TABLES

<table>
<thead>
<tr>
<th>TITLE STTB1</th>
</tr>
</thead>
<tbody>
<tr>
<td>NREL</td>
</tr>
<tr>
<td>EXPD OP1DS, OP2DS, OP3DS, OP1DR, OP2DR, OP3DR</td>
</tr>
<tr>
<td>ENT OP1DS, OP2DS, OP3DS, OP1DR, OP2DR, OP3DR</td>
</tr>
<tr>
<td>EXT OP1DS, OP2DS, OP3DS, OP1DR, OP2DR, OP3DR</td>
</tr>
</tbody>
</table>

**OPDST:**

| 001 | 002 | 003 | 004 | 005 | 006 | 007 | 008 | 009 | 010 | 011 | 012 | 013 | 014 | 015 | 016 | 017 | 018 | 019 | 020 | 021 | 022 | 023 | 024 | 025 | 026 | 027 | 028 | 0 |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

| RTTO1 = 100000 |

**019:**

<table>
<thead>
<tr>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZER</td>
</tr>
<tr>
<td>RTTO1</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**003:**

<table>
<thead>
<tr>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP6D</td>
</tr>
<tr>
<td>CLOS</td>
</tr>
<tr>
<td>RTTO1</td>
</tr>
</tbody>
</table>

**007:**

| +1 |
| OP4D |
| GOTO1 |
| RTTO1 |

**008:**

| +1 |
| OP5D |
| WRIT |
| RTTO1 |

**009:**

| +1 |
| OP1D |
| AG1 |

240
OP2D
AG2
MOVE
RTTOI
011: .+1
OP6D
OPEN
STRIT
RTTOI
012: .+1
PRFRM1
RTTOI
013: .+1
OP4D
OP5D
READ
RTTOI
015: .+1
STOP1
RTTOI
017: .+1
EXIT1
RTTOI
020: .+1
OP1D
AG1
OP2D
AG2
OP4D
EQUAL
RTTOI
021: .+1
OP1D
AG1
OP2D
AG2
OP4D
LESS
RTTOI
022: .+1
OP1D
AG1
OP2D
AG2
OP4D
GREAT
RTTOI
023: .+1
OP1D
AG1
OP4D
NUMRC
RTTOI
024: .+1
OP1D
AG1
OP4D
ALPHA
RTT01

026:  .+1
OP1D
OP2D
OP3D
LDPK1
LDPK2
ADDPK
M3T01
LDPK3
ADDPK
UNPK2
RTT01

002:  .+1
OP1D
OP2D
OP3D
LDPK1
LDPK2
ADDPK
UNPK2
RTT01

005:  .+1
OP1D
OP2D
OP3D
SFLDP
LDPK2
DVDPK
FLDEC
M2T03
UNPK2
RTT01

018:  .+1
OP1D
OP2D
OP3D
LDPK1
LDPK2
MULT
UNPK2
RTT01

027:  .+1
OP1D
OP2D
OP3D
LDPK1
LDPK2
ADDPK
M3T01
LDPK3
SUPPK
UNPK2
RTT01

016:  .+1
OP1D
OP2D
OP3D
LDPK1
LDPK2
SUBPK
UNPK2
RTTOI

001: . +1
OP1D
APT1
RTTOI

004: . +1
OP1D
DSPLY
RTTOI

OP1DS: LDPK1
SSB11
OP1D
SSB21

OP1DR: . BLK 1
OP2DS: LDPK2
SSB12
OP2D
SSB22

OP2DR: . BLK 1
OP3DS: LDPK3
SSB13
OP3D
SSB23

OP3DR: . BLK 1
025: . +1
OP1D
OP2D
OP3D
SETSB
RTTOI
.
END

; STATE TABLE FOR SUBSCRIPT ON OPERAND 1

; RETURN BRANCH (A5P+1 FOR OPCODE)
THE CALL TO THIS ROUTINE IS (N,PAC1,PAC2,PAC3)

SUBTK:
LDA 1.TEN.2 ;GET AN EIGHT FOR EIGHT WORD SUBTRACT
STA 1.N.2 ;STORE IN N
LDA 1.PAC1.2 ;INITIALIZE ADDRESS
STA 1.PAC1.2 ;FOR SUBTRACT
LDA 1.PAC2.2 ;ROUTINE
STA 1.PAC2.2
LDA 1.PAC3.2
STA 1.PAC3.2

SUBT:
ISZ RSTST.2 ;GET NEXT STORE POSITION
STA 3.RTST.2 ;SAVE RETURN ADDRESS
LDA 3.N.2 ;GET NUMBER OF WORDS TO BE SUBTRACTED
STA 3.1.2 ;STORE IN I
LDA 1.PAC1.2 ;GET FIRST FIELD TO BE SUBTRACT
ADD 3.1 ;GET FIRST WORD
STA 1.PAC1.2 ;STORE RACK OUT
LDA 1.PAC2.2 ;DO THE SAME FOR PAC2
ADD 3.1
STA 1.PAC2.2
LDA 1.PAC3.2 ;DO THE SAME FOR PAC3
ADD 3.1
STA 1.PAC3.2
DSZ PAC1.2 ;TO GET PROPER ALIGNMENT
DSZ PAC2.2 ;FOR SUBTRACTION AND
DSZ PAC3.2 ;FOR STORING

START:
LDA 1.0PAC1.2 ;GET FIRST ARGUMENT
LDA 0.0PAC2.2 ;GET SECOND ARGUMENT
SUBZ 1.0.52C ;SUBTRACT THE TWO
JMP SUBB

SUBA:
DSZ 1.2 ;DECREMENT NUMBER OF WORDS
JMP +2 ;NOT ZERO
JMP @RTN.2 ;RETURN
DSZ PAC1.2 ;TO GET NEXT WORD FOR
DSZ PAC2.2 ;SUBTRACTION AND
DSZ PAC3.2 ;FOR STORING
LDA 1.0PAC1.2 ;GET FIRST ARGUMENT
LDA 0.0PAC2.2 ;GET SECOND ARGUMENT
ADCZ 1.0.52C ;ADD COMPLEMENT
JMP SUBA

SUBB:
STA 0.0PAC3.2 ;STORE ANSWER
DSZ 1.2
JMP +2
JMP @RTN.2
DSZ PAC1.2
DSZ PAC2.2
DSZ PAC3.2
LDA 1.0PAC1.2 ;GET FIRST ARGUMENT
LDA 0.0PAC2.2 ;GET SECOND ARGUMENT
SUBZ 1.0.52C ;SUBTRACT
JMP SUBB
JMP SUBA
**TYPE1:**
- **ISZ:** RTST,2 ;SAVE RETURN ADDRESS
- **STA:** 3,RTST,2
- **LDA:** 3,INDC,2 ;GET INDICATOR
- **LDA:** 0,TABEL ;GET BEGINNING OF TABLE
- **ADD:** 0,3 ;GET ROUTINE
- **LDA:** 3,0,3 ;GET NAME OF ROUTINE
- **JMP:** 0,3

**TABEL:**
- **TAB:** XXX ;ERROR RETURN NO INDICATOR ZERO

**TYPE0:**
- **LDA:** 0,DEC1,2 ;GET ASSUMED DECIMAL PT
- **MOVH:** 0,0,5R ;CHECK TO SEE IF EQUAL
- **JMP:** ORTH,2 ;NO, RUN TIME ERROR

**TYPE2:**
- **SURZL:** 1,1 ;GENERATE A ONE
- **STA:** 1,FLAAL,2 ;STORE IN EDIT FLAG
- **JSP:** ORTN,2

**TYPE3:**
- **JSP:** @LDP1 ;YES COMP TO COMP MOVE
- **JMP:** ORTEP,2
- **LDA:** 1,PACK1,2 ;GET SENDING FIELD
- **STA:** 1,PAC1,2 ;STORE IN PAC1
- **LDA:** 1,PACK2,2 ;GET RECEIVING FIELD
- **STA:** 1,PAC2,2 ;STORE IN PAC2
- **JSP:** @MMOV,2 ;GO TO MATHMOVE ROUTINE
- **JMP:** ORTER,2
- **JSP:** @UNP1 ;UNPACK COMP
- **JMP:** ORTER,2
- **JMP:** ORTN,2

**TYPE4:**
- **JSP:** @LDP1 ;YES BINARY TO ASCII MOVE
- **JMP:** ORTER,2
- **LDA:** 1,PACK1,2 ;SET ADDRESS FOR MATHMOVE ROUTINE
- **STA:** 1,PAC1,2
- **LDA:** 1,PACK2,2
- **STA:** 1,PAC2,2
- **JSP:** @MMOV,2
- **JMP:** ORTEP,2
- **JSP:** @UNP1
- **JMP:** ORTEP,2
- **JMP:** ORTN,2

**TYPE5:**
- **JSP:** @LDP1 ;YES ASCII TO BINARY MOVE
- **JMP:** ORTER,2
- **LDA:** 1,PACK1,2 ;INITIALIZE ADDRESS'S FOR MATHMOVE
- **STA:** 1,PAC1,2
- **LDA:** 1,PACK2,2
- **STA:** 1,PAC2,2
- **JSP:** @MMOV,2
- **JMP:** ORTER,2
- **JSP:** @UNP1 ;THE UNPACK ROUTINE
- **JMP:** ORTER,2
- **JMP:** ORTN,2

**TYPE6:**
- **LDA:** 0,DP2NL,2 ;GET LENGTH OF SENDING FIELD
LDA 1, DEC1, 2  GET DECIMAL POINT  
SUB 1,0  COMPUTE 1 OF DIGITS LEFT OF DEC PT.  
STA 0, DIGIT, 2  STORE IT  
LDA 0, OP2DL, 2  HAVE COMPUTE RECEIVING LENGTH LEFT  
LDA 1, DEC2, 2  D.P. POSITION  
SUB 1,0  COMPUTE 1 OF DIGITS LEFT OF D.P.  
STA 0, RDIGIT, 2  STORE REMAINING DIGIT  
END:  
LDA 1, DIGIT, 2  GET SENDING DIGIT  
LDA 0, RDIGIT, 2  GET RECEIVING DIGIT  
SUBZ 1,0, SNC  WHAT IS LEFT  
JMP BAD  NEG RESULT WONT FIT  
MOV# 0,0, S2P  
JMP .44  
STA 0, CNT0, 2  
STA 0, PNT0, 2  
JSR @RTN, 2  
SUBZL 3,3  GENERATE A ZERO  
STA 3, PNT0, 2  SET BYTE PTR TO ZERO  
STA 0, CNT0, 2  STORE RESULT IN CNT1  
JMP @RTN, 2  
BAD:  
LDA 3, CNT1, 2  GET CNT1  
ADD 0, 3  ADD RESULT TO CNT1, 2  
STA 3, CNT1, 2  
LDA 3, PNT1, 2  GET BYTE PTR  
SUB 0, 3  SUBTRACT RESULT, RUNTIME ERROR  
STA 3, PNT1, 2  
JSR @RTN, 2  
TYPE7:  
SUBZL 1,1  
STA 1, EFLAG, 2  
LDA 0, OP1DL, 2  
LDA 1, DEC1, 2  
SUB 1,0  
STA 0, DIGIT, 2  
LDA 1, DEC2, 2  
LDA 3, OP2DE, 2  GET BYTE POINTER TO EDIT STRING  
MOVR 3,3  CONVERT TO WORD POINTER  
LDA 3, 0, 3  GET LENGTH OF FILL CHARACTER  
SUB 1, 3  TO GET DECIMAL POSITION TO LEFT OF DECIMAL POINT  
SUB 1, 1  TO GET DECIMAL POSITION TO LEFT OF DECIMAL POINT  
STA 3, RDIGIT, 2  
ISZ OP2DE, 2  
ISZ OP2DE, 2  
JMP END  
LDP1: LDPK1  
LDP2: LDPK2  
UNP1: UNPK1
INITIALIZE UNPACK ROUTINE

FOR PACK1

FOR PACK2

GET NEXT STORE POSITION
SAVE RETURN ADDRESS

CLEAR ACO

CHECK TO SEE IF EQUAL TO ZERO
YES A COMP TO COMP MOVE

GET DECIMAL POSITION
STORE IN M

EQUAL TO ZERO
NOT EQUAL TO ZERO

M IS EQUAL TO ZERO
GO TO MULTIPLY BY 10 ROUTINE

DO AGAIN

GET RECEIVING FIELD LENGTH
GET BYTE POINTER
ADD LENGTH TO BYTEPOINTER TO GET END
OF FIELD STORE BACK OUT

GET THE NUMBER OF WORDS LONG
STA 1, PACT3.2 ; STORE POSITION OUT
SUB 0, 0 ; CLEAR AC0
LDA 1, PACT3.2 ; GET THE REMAINDER
MOV 1, 1 ; SWAP BYTES
STA 0, PACT3.2 ; STORE A ZERO IN THAT POSITION
MOV2 1, 1, S2C ; SHIFT LEFT CHECK CARRY
INC 0, 0 ; CARRY IS SET
LDA 3, REMSK, 2 ; GET BYTE MASK 377
AND 3, 1 ; SAVE RIGHT BYTE
ADD 0, 0 ; ADD OVER FLOW IF ANY
MOV2 1, 0 ; MULTIPLY BY TEN
MOV2 0, 0
ADD2 0, 1 ; SWAP BYTES AGAIN
MOV2 1, 1, S2C ; CHECK TO SEE IF CARRY IS SET
INC 1, 1 ; YES
AND 1, 1 ; SAVE RIGHT BYTE
LDA 0, C60, 2 ; GET AN ASCII 60
ADD 0, 1 ; CHANGE TO A NUMBER
STA 1, CHAR, 2 ; STORE IT
LDA 3, X37, 2 ; TEST TO SEE IF FLAG IS SET
MOV 3, 3, SNR
JMP TEST 1 ; NOT SET GO TO TEST FOR SIGN

SET:
LDA 3, OPOP, 2 ; GET TEMP BYTE POINTER
STA 2, BVPTP, 2 ; STORE IN BYTE POINTER
LDA 1, CHAR, 2 ; GET THE CHARACTER
JSR OPUS ; STORE IT
JSR @RTU, 2 ; RETURN
DSZ OPOP, 2 ; DECREMENT BYTE POINTER
DSZ ODP, 2 ; DECREMENT LENGTH
JMP +42
JMP @RTU, 2 ; RETURN
SUB2L 3, 1 ; GENERATE A ONE
LDA 1, ODP, 2 ; GET LENGTH
SUB2 3, 1, SZR ; TEST TO SEE IF EQUAL TO ONE
JMP AA
LDA 1, EFLAG, 2 ; GET FLAG
MOV 1, 1, SNR ; TEST TO SEE IF SET
JMP AA ; ZERO NOT SET
JMP SLI+2

CHANG:
SUB2L 1, 1 ; GENERATE A ONE
STA 1, NEG, 2 ; STORE IN NEGATIVE NUMBER FLAG
LDA 1, PACT, 2 ; GET FIELD TO BE UNPACKED
STA 1, PAC1, 2 ; STORE ADDRESS IN TEMPORARY
LDA 1, PAC0, 2 ; ZERO: OUT TEMPORARY PACK FIELD
STA 1, PAC0, 2
JSR @ZERO, 2 ; ZERO OUT ROUTINE
JSR @RTU, 2 ; ERROR RETURN NOT POSSIBLE ERROR
LDA 1, PAC0, 2 ; INITIALIZE ADDRESS FOR SUBTRACT ROUTINE
STA 1, PAC0, 2
LDA 1, PAC0, 2
STA 1, PAC0, 2
JSR @SUBT, 2 ; GO TO SUBTRACT ROUTINE
JSR @RTU, 2 ; ERROR RETURN NOT POSSIBLE ERROR
LDA 1, PAC0, 2 ; GET ANSWER
STA 1, PAC0, 2 ; AND MOVE BACK TO
LDA 1, PAC0, 2 ; ORGINA PACK FIELD
STA 1, PAC0, 2
JSR @MOV, 2 ; MATHEVC ROUTINE
JSR @RTU, 2 ; ERROR RETURN NOT POSSIBLE ERROR
JMP UNPK +5 ; CONTINUE ON
COMP:
  LDA 1.FIVE, 2 GET A FIVE
  STA 1.N, 2 STORE IN N
  SUBZ 1, 1 GENERATE A ZERO
  STA 1.MM, 2 STORE IN M
  SUBZL 1, 1 GENERATE A ONE
  STA 1.I, 2 STORE IN I
  LDA 1.PACTL, 2 GET FIRST ADDRESS
  STA 1.PACTL, 2 STORE IT

AE:
  LDA 1.8PACTL, 2 GET FIRST ANSWER
  MOV# 1.1, SNR TEST TO SEE IF EQUAL TO ZERO
  JMP AG YES
  ISZ 1, 2 NO

AG:
  LDA 1.PACTL, 2 INITIALIZE FOR
  STA 1.PACTL, 2 MATHMOVE ROUTINE
  LDA 1.OPDCP, 2 GET ADDRESS OF DATA
  MOVZR# 1, 1 CONVERT TO WORD ADDRESS
  STA 1.PAC2, 2
  JSR @MMOV, 2
  JSR @RTSR, 2
  LDA 1.MM, 2 GET M
  STA 1.OPDC, 2 STORE IN DECIMAL POINTER
  JMP @RTN, 2 RETURN

TEST:
  SUBZL 3, 3 CLEAR ACI
  STA 3.K3T, 2 STORE THE ONE IN FLAG
  LDA 3.SIGN, 2 TEST TO SEE IF SIGN
  MOV 3, 3, SNR TEST TO SEE IF ZERO
  JMP SET ZERO NO SIGN
  NEG 3, 3 DECREMENT ACI
  COM 3, 3

  MOVZR# 3, 3, SEZ
  JMP SITS SIGN TRAILING SEPARATE TYPE FOUR
  MOV 3, 3, SNR
  JMP SILL SIGN LEADING INCLUDED TYPE ONE
  MOVZR# 3, 3, SZR
  JMP SITI SIGN TRAILING INCLUDED TYPE THREE
  JMP SILS SIGN LEADING SEPARATE TYPE TWO

SITS:
  LDA 3.NEGNO, 2 TEST TO SEE IF NEGATIVE NUMBER
  MOV 3, 3, SNR
  JMP +7 ZERO PLUS NUMBER
  LDA 1.MINUS, 2 GET A MINUS SIGN
  LDA 3.OPDP, 2 GET BYTE POINTER
  STA 3.BPSTR, 2 STORE IN BYTE POINTER
  JSR @PUT STORE IT
  JSR @RTSR, 2 ERROR RETURN
  DSZ OPDP, 2
  DSZ CNT2, 2
  JMP SET

SII:
  SUBZL 3, 3 GENERATE A ONE
  STA 3.EFLAG, 2 STORE IN EFLAG
  LDA 3.NEGNO, 2 TEST TO SEE IF NEGATIVE NUMBER
  MOV 3, 3, SNR
  JMP SET
  LDA 0.C31, 2 GET AN EDIT CONVERSION CHARACTER
ADD 0,1 ; CONVERT TO A LETTER
STA 1,CHAP1,2 ; STORE IT
JMP SET

SITI:
LDA 3,NEGNO,2 ; TEST TO SEE IF NEGATIVE NUMBER
MOV 3,3,SNR
JMP SET
LDA 0,C31,2 ; GET A CONVERSION CHARACTER
ADD 0,1 ; CONVERT TO LETTER
STA 1,CHAP1,2 ; STORE IT
JMP SET

SILS:
LDA 3,NEGNO,2 ; TEST TO SEE IF NEGATIVE NUMBER
MOV 3,3,SNR
JMP +6
LDA 0,OP2DP,2 ; GET BYTE POINTER
STA 0,RVPTR,2 ; STORE IT IN BYTE POINTER
LDA 1,MINUS,2 ; GET A MINUS SIGN
JSR @PUT ; STORE IT
JSR @RTER,2
DSZ ODD,2 ; DECREMENT TOTAL LENGTH
JMP SET
JSR @RTER,2
TITL  VALID
ENT  VALID
NREL

JF TO
JR 0 CO 91 X2 03 SE4 XE5 Z6 R7
JN

CO:  3 4 0 0 0 0 0 0
C1:  5 6 1 1 7 2 0 0
X2:  0 6 1 1 1 2 0 0
G3:  0 1 1 1 0 0 0 0
SE4:  0 0 1 0 0 1 0 0
XE5:  0 0 1 0 0 1 0 0
Z6:  0 6 1 1 0 0 0 0
R7:  0 0 1 1 0 0 0 0

VALID:  ISZ RTST.2 GET NEXT STORE POSITION
STA 3 ORST.2 SAVE RETURN ADDRESS
LDA 3 OP1DT.2 GET TYPE OF SENDING FIELD
LDA 1 EMSK3.2
AND 1.3
LDA 0 THREE.2
STA 0 XST.2
MOVZR 3.3
DSZ XST.2
JMP -2
STA 3 SIGN.2 STORE SIGN
LDA 3 OP1DT.2 GET TYPE
LDA 1 EMSK3.2 GET MASK 7
AND 1.3 GET TYPE
LDA 1 TABLE GET TABLE DISPLACEMENT
ADD 1.3
LDA 3 0 3
STA 3 TAB.2
LDA 3 OP2DT.2 GET TYPE OF RECEIVING FIELD
LDA 0 TAB.2 GET DISPLACEMENT
ADD 0.3 GET POSITION IN TABLE
LDA 1 TABLE GET BEGINNING OF TABLE
ADD 1.3
LDA 3 0 3
STA 3 INDIC.2
MOV 3 0 SZR SEE IF INDICATOR IS ZERO
JMP @RTH.2 AND IT IS NOT, RETURN
JMP @RTER.2 YES RUN TIME ERROR

TABLE:  TAB
TAB:  0
  10
  20
  30
  40
  50
  60
  70

TABEL:  TAC
TAC:  3
  4
  0
  0
  0
  0

251
DSZ  CNT1.2  DECREMENT # LINES TO ADVANCE
JMP  .-6  NOT ZERO, ADVANCE AGAIN
LDA  1,SKPIT,2  LOAD SKPIT FLAG
MOV  1,1,SZR  #CHECK STATUS
JMP  @RTN,2  IF ON, PROCESSING COMPLETE, RETURN
JMP  WRITS  #OFF, JUMP TO WRITE DATA LINE

WRIT7:  LDA  0,FF1  LOAD FORM FFED CHAR,
LDA  2,CHANL,2  LOAD THE CHANNEL NUMBER
.SYSTEM
.WRL  77
.JSR  @RTER,3  RESTORE AC2
MOV  3,2  #RESTORE AC2
LDA  1,SKPIT,2  LOAD SKPIT FLAG
MOV  1,1,SZR  #CHECK STATUS
JMP  @RTN,2  IF ON, PROCESSING COMPLETE, RETURN
JMP  WRITS  #OFF, JUMP TO WRITE DATA LINE
.TXTH  0

LF1:  +1#2
TXT  /<12><15>/

FF1:  +1#2
TXT  /<14><15>/
CALL TO THIS ROUTINE IS (N, PACZ)

ZEROA:
  ISZ RTST, 2 GET NEXT STORE POSITION
  STA 3, RTST, 2 SAVE RETURN ADDRESS
  LDA 1, N, 2 GET NUMBER OF WORDS TO
  STA 1, NUM, 2 ZERO-OUT STORE IN NUM

AA:
  SUBZ 0, 0 CLEAR AC0
  STA 0, @PACZ, 2 STORE ZERO-OUT
  DSZ NUM, 2 DECREMENT COUNTER
  JMP +2 COUNTER NOT ZEPO
  JMP @RTN, 2 RETURN

ISZ PACZ, 2 GET NEXT ZERO-OUT POSITION
JMP AA DO AGAIN
.TITL ZEROSUPPRESSION
.ENT GETBT, COMMA, ZEE, FCVAR, DOLLAR, AA, PUTCH, GETBY, PUTT, ASTK, PTT, PUTTT
.EXTD PUT, GET
.EXTN @1, SNG, AAB
LESS

ASTK:
SUB 1, 1
STA 1, NFLAG, 2 ; CLEAR NFLAG
LDA 1, RCTN, 2 ; GET REPTITION COUNT
LDA 1, ONE, 2 ; GET A ONE
SUB# 0, 1, SNR ; TEST TO SEE IF EQUAL TO ONE
JMP ONAST ; EQUAL TO ONE
LDA 0, PNT0, 2 ; GET COUNT FOR ASTERISK FILL
MOV# 0, 0, SNR ; TEST TO SEE IF ZERO
JMP +2
JMP +4
LDA 1, AST, 2 ; GET AN ASTERISK
STA 1, ECHAR, 2 ; STORE IN EDIT CHARACTER
JMP PUTT+3 ; ZERO GO TO GET BYTE
LDA 1, AST, 2 ; GET AN ASTERISK
STA 1, ECHAR, 2 ; STORE IN EDIT CHARACTER
STA 1, ECHT, 2 ; STORE IN EDIT CHARACTER TEMPORARY
PUTT JMP PUTT

ONAST:
LDA 1, AST, 2 ; GET A ASTERISK
STA 1, ECHAR, 2 ; STORE IN EDIT CHARACTER
DSZ CNT2, 2 ; DECREMENT TOTAL LENGTH
JMP @RTN, 2 ; RUNTIME ERROR

PUTT:
DSZ CNT2, 2 ; DECREMENT TOTAL LENGTH
JMP .+2 ; NOT ZERO
JMP @RTER, 2 ; ZERO SOMETHING WRONG
DSZ RCTN, 2
JMP .+2
JMP 0, @1

PUTTT:
LDA 1, ECHAR, 2 ; GET EDIT CHARACTER
LDA 3, PNT2, 2 ; GET BYTE POINTER TO RECEIVING
STA 3, BVPTR, 2 ; STORE IN BYTE POINTER
JSR @PUT ; STORE BYTE ROUTINE
JMP @RTER, 2
ISZ PNT2, 2 ; GET NEXT STORE POSITION
DSZ CNT0, 2 ; DECREMENT FILL COUNTER
JMP .+6
LDA 1, ECHT, 2 ; GET EDIT CHARACTER TEMPORARY
STA 1, ECHAR, 2 ; STORE IN EDIT CHARACTER
SUB 1, 1 ; CLEAR AC1
STA 1, PNT0, 2
JMP AA
LDA 1, CNT0, 2
MOVL# 1, 1, SZO ; CHECK TO SEE IF NEGATIVE
JMP .-3 ; CARRY IS SET
JMP PUTT ; GET ANOTHER ASTERISK

GETBY:
DSZ CNT2, 2 ; DECREMENT TOTAL LENGTH
JMP .+2 ; NOT ZERO
JMP @RTN, 2
JSR @SNG, 2 ; GET A BYTE POINTER
JMP @RTER, 2
LDA 0, C60, 2 ; GET AN ASCII ZERO
SUB# 0, 1, SNR ; TEST TO SEE IF A ZERO
JMP PUTT ; GO TO A PUTCHARACTER ROUTINE
LDA 1, ECHAR, 2 ; GET EDIT CHARACTER
LDA 3, C1, 2 ; TEST TO SEE IF FILL CHARACTER

256
SUB# 3, SNR
JMP PUTC+3
LDA 3, PNT2, 2  ; GET STORE POSITION
STA 3, BVPTR, 2  ; STORE IN BYTE POINTER
JSR @PUT  ; STORE IT
JMP @RTER, 2
ISZ PNT1, 2  ; GET NEXT PIECE OF DATA
ISZ PNT2, 2  ; GET NEXT STORE POSITION
DSZ RCNT, 2  ; DECREMENT REPIETION COUNT
JMP +2  ; NOT ZERO
JMP @B1  ; GET ANOTHER EDIT FORMAT
DSZ CNT1, 2  ; DECREMENT BEGIN LENGTH
JMP +2  ; NOT ZERO
JMP @ARB  ; GET ANOTHER EDIT CODE
JMP GETBY  ; GET ANOTHER BYTE

AAB: AAB
R1: B1
SNG1: SNR
PUTCH: DSZ CNT2, 2  ; DECREMENT TOTAL LENGTH
JMP +2  ; NOT ZERO
JMP @RTN, 2  ; ZERO SOMETHING WRONG
DSZ RCNT, 2  ; DECREMENT REPIETION COUNT
JMP +2
JMP @B1  ; GET ANOTHER EDIT CHARACTER

PTT: LDA 3, PNT1, 2  ; GET DATA
STA 3, BVPTR, 2  ; STORE IN BYTE POINTER
JSR @GET  ; GO TO A BYTE ROUTINE
JMP @RTER, 2
ISZ PNT1, 2
LDA 3, PNT2, 2  ; GET STORE POSITION
STA 3, BVPTR, 2  ; STORE IN BYTE POINTER
JSR @PUT  ; GET NEXT STORE POSITION
JMP @RTER, 2
ISZ PNT2, 2  ; GET NEXT STORE POSITION
DSZ CNT1, 2  ; DECREMENT ENDING LENGTH
JMP +2  ; NOT ZERO
JMP @ARB  ; GET ANOTHER EDIT CODE
JMP PUTC  ; GET ANOTHER CHARACTER
AA: LDA 1, NFLAG, 2  ; TEST TO SEE IF DOLLAR EDIT
MOVR 1, SNC  ; SEE IF SET
JMP GETBY  ; GO TO PUTCCHARACTER OUT ROUTINE
JMP GETBT  ; DOLLAR EDIT

DOLAR: LDA 1, RCNT, 2  ; GET REPIETION COUNT
LDA 0, ONE, 2  ; TEST TO SEE IF EQUAL TO ONE
SUB# 0, 1, SNR
JMP ONDOL  ; EQUAL TO ONE
LDA 0, PNTA, 2  ; GET BYTE POINTER
MOVR 0, 0, SNR  ; TEST TO SEE IF ZERO
JMP +2
JMP +4
LDA 1, DOL, 2  ; GET A DOLLAR SIGN
STA 1, ECHAR, 2  ; STORE IN EDIT CHARACTER
JMP GETBT  ; GO TO GET BYTE ROUTINE
LDA 1, BLK, 2  ; GET A BLANK
STA 1, ECHAR, 2  ; STORE IN EDIT CHARACTER
LDA 1, DOL, 2  ; GET A DOLLAR SIGN
STA 1, ECHT, 2  ; STORE IN EDIT CHARACTER TEMPORARY
SUBZL 1, 1  ; GENERATE A ONE
STA 1, NFLAG, 2  ; STORE IN NFLAG FOR DOLLAR EDIT FLAG
JMP PUTC  ; GO TO STORE ROUTINE

ONDOL: LDA 1, DOL, 2  ; GET A DOLLAR SIGN
STA 1.ECHT.2 ;STORE IN EDIT CHARACTER TEMPORARY
STA 1.ECHT.2 ;STORE IN EDIT CHARACTER TEMPORARY
JMP PUTTY

PUCAR: DSZ PNTZ2.2 ;GET LAST STORE POSITION
LDA 1.ECHT.2 ;GET A DOLLAR SIGN
LDA 1.PNTZ2.2 ;GET STORE POSITION
STA 3.BYTPZ1.2 ;STORE IN BYTE POINTER
JSR @PUT1 ;STORE IT
JMP @TERP.2

ISZ PNTZ2.2 ;GET NEXT STORE POSITION
JMP PUTC+1 ;GO TO PUTCHARACTER ROUTINE

ZEE: LDA 0.PNTO.2 ;GET BYTE POINTER TO ARGUMENT ONE
MOV 0.0 SNR ;TEST TO SEE IF ZERO
JMP +2
JMP +4
LDA 1.BLK.2 ;GET THE EDIT CODE
STA 1.BLK.2 ;STORE THE CHARACTER
JMP GETBY+1

LDA 1.BLK.2 ;GET A BLANK
STA 1.ECHT.2 ;STORE IN EDIT CHARACTER
STA 1.ECHT.2 ;STORE IN EDIT CHARACTER TEMPORARY
JMP PUTT ;GO TO A PUT CHARACTER ROUTINE

COMMA: DSZ PNTZ2.2 ;DECREMENT STORE POSITION
LDA 3.PNTZ2.2 ;GET BYTE POINTER TO LAST STORE POSITION
STA 3.BYTPZ1.2 ;STORE IN BYTE POINTER
JSR @GET ;GET LAST BYTE STORED
JMP @TERP.2

ISZ PNTZ2.2 ;PUT BACK TO PROPER STORE POSITION
LDA 0.CEO.2 ;GET A MASK OF 60
AND 0.1 ;TEST TO SEE IF NUMERIC
SUB 0.1 ;SNR ;A NUMBER
LDA 3.PNTZ2.2 ;GET STORE POSITION
STA 3.BYTPZ1.2 ;STORE IN BYTE POINTER
JSR @PUT ;STORE LAST CODE OUT
JMP @TERP.2

DSZ CNTZ2.2 ;DECREMENT LENGTH
JMP @RNF.2 ;RETURN TO GET ANOTHER EDIT CODE
JMP @RNF.2 ;RETURN TO GET ANOTHER EDIT CODE
LDA 1.COL.2 ;GET A COMMA

STA 1.ECHT.2 ;STORE IN EDIT CHARACTER
JMP PUTT1 ;PUT THE CHARACTER OUT

GETBT: DSZ CNTZ2.2 ;DECREMENT TOTAL LENGTH
JMP +2 ;NOT ZERO
JMP @RNF.2 ;ZERO SOMETHING WRONG
JSR @SNGL ;GET A BYTE POINTER
JMP @TERP.2

LDA 0.CEO.2 ;GET AN ASCII ZERO
SUB 0.1 ;SNR ;TEST IF A ZERO
JMP PUCAP ;GO TO A PUTCHARACTER ROUTINE
LDA 1.BLK.2 ;GET A BLANK
LDA 3.PNTZ2.2 ;GET STORE POSITION
STA 3.BYTPZ1.2 ;STORE IN BYTE POINTER
JSR @PUT ;STORE IT
JMP @TERP.2

ISZ PNTZ2.2 ;GET NEXT PIECE OF DATA
ISZ PNTZ2.2 ;GET NEXT STORE POSITION
DSZ RCTZ2.2 ;DECREMENT REPETITION COUNT
JMP +3 ;NOT ZERO
ISZ RCTZ2.2 ;PUT A ONE INTO REPETITION COUNT
JMP @PUTT1 ;GO TO STORE ROUTINE
DSZ     CNT1, 2     DECREMENT BEFIN LENGTH
JMP    .+2     NOT ZERO.
JMP    @ AAB     GET ANOTHER EDIT CODE
JMP    GETBT     GET ANOTHER BYTE
PUTT1:  PUTT
APPENDIX D

INSTRUCTION REGISTER AND MICRO-PROCESSOR
Introduction

The Model 80 Micro-processor expects all of the instructions it will emulate to be in the format of the Instruction Register. The micro-decoder will then inspect the Instruction Register (IR) and generate micro-code based upon the operands found to be present.

Instruction Register

The format of the Instruction Register is presented below:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>7</th>
<th>8</th>
<th>11</th>
<th>12</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP-CODE</td>
<td>YD</td>
<td>YS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure D.1

Instruction Register

With respect to figure D.1, specifying YD as an A operand causes the YD field (IR 8:11) to be placed on bits 12:15 of the A bus. Bits 0:11 of the A bus will be set to zeroes. (See Figure D.2 - Model 80 Block Processor). 6

Specifying YSI in the B field will cause the YS field (IR 12:15) to be placed on bits 12:15 of the B bus. Bits 0:11 of the B bus will be set to zeroes.

Specifying the keyword "NULL" in the A or B field will cause the corresponding operand bus to be set to zeroes. If NULL is specified in the S field, the S bus data will be loaded into MR7. Specifying YDPI will cause the odd member of the even-odd pair of General Registers, one of whose number is in the YD field, to be selected.
Specifying YDI in the S field will cause bits 12:15 of the B bus to be loaded into the YD field of the IR.

The Micro-processor

When the INTERDATA emulates the instruction of a given machine, the micro-program will read the next user instruction which is to be performed from main memory. The instruction decoding circuit will then direct the processor to a uniquely identifiable micro-subroutine that is designed to perform the instruction. This subroutine may consist of from one to several micro-instructions which may be necessary to accomplish the given task. When emulation of the user instruction is completed, the micro-program will then read the next user instruction from main memory, closing the emulation loop. Because of the rather elementary level of the micro-code, the actual instructions will be directed to various parts of the hardware and registers (see figure D.2). The names for these specific areas tend to be long and unwieldily and a series of abbreviations will be used in their place as defined below:

- LOC - Location Counter. This contains the address of the Users' Instruction.
- MAR - Memory Address Register. Contains the contents of LOC, i.e., the address of the Users’ Instruction.
- IR - Instruction Register. Contains the contents of memory specified by MAR.
- MDR - Memory Data Register. The next sequential halfword (16 bits) in main memory.

Table D.1
Operand Abbreviations
Figure D.2

Model 80 Processor Block Diagram
YD - IR (8:11)
YS - IR (12:15)
YX - Constant used the same as YS when YS is equal to zeroes.

Table D.1 (cont.)
Operand Abbreviations

When referencing the instruction word fields the following abbreviations are used:

A - Selects register to be used as first operand.
B - Selects register to be used as second operand.
S - Selects register to receive the result.
F - Specifies function of addressed module.
E - Enables the setting of the Condition Code
C - If set, transfer is conditional.
X - Execute.
I - Second operand is address of data.
D - Decode next user instruction.
K - F field extension.

Table D.2
Abbreviated Instruction Word Fields

There are in addition to the sixteen general registers abbreviated by the letters "GRn" where n is the digit 0-15, eight micro-registers which are abbreviated by the letters "MRn" where n is the digit 0-7. Both general registers and micro-registers are addressable by the micro-program.
When referencing the Memory Control field, the following abbreviations are used:

**IR** - Instruction Read  
**DR** - Data Read  
**DW** - Data Write  
**I2** - Increment MAR by two  
**I4** - Increment MAR and LOC by four  
**JAM** - Copy LOC to MAR

**Table D.3**  
Memory Control Abbreviations

**Micro-instruction Formats**

Model 80 INTERDATA micro-instructions can be presented in one of six formats designated as Address Link, Register Link, Register to Register Transfer, Register to Register Control, Register to Register Immediate, and Register Write. The formats are shown below.
| Address Link |
|--------------|------------------|------------------|------------------|
| 0 2 3 4 5 6 7 10 11 15 16 19 20 30 31 |
| **IOO I X E D MC** |
| **LINK REGISTER F ADDRESS ** |
| **F** |

| Register Link |
|------------------|------------------|------------------|
| 0 2 3 4 5 6 7 10 11 15 16 19 20 24 25 31 |
| **IOO 0 X E D MC** |
| **LINK F B ADDRESS MASK** |

| Register to Register Transfer |
|-------------------------------|------------------|------------------|------------------|
| 0 2 3 4 5 6 10 11 15 16 19 20 24 25 30 31 |
| **MOD 00 I S A F B PAGE ADDRESS C** |

| Register to Register Control |
|-------------------------------|------------------|------------------|
| 0 2 3 4 5 6 10 11 15 16 19 20 24 25 26 27 28 31 |
| **MOD 01 I S A F B K E D MC** |

| Register to Register Immediate |
|-------------------------------|------------------|------------------|------------------|
| 0 2 3 4 5 6 10 11 15 16 19 20 31 |
| **MOD 10 I S A F DATA** |

| Register Write |
|------------------|------------------|------------------|
| 0 2 3 5 6 10 11 15 16 19 20 24 25 26 27 28 31 |
| **111 111 NULL A F B K E D MC** |

Figure D.3

Micro-instruction Formats
If the C bit on the Register to Register Transfer is set, then transfer only occurs if no predefined signal is returned from the addressed module. For the ALU, the signal is Carry, which means no transfer will occur if a carry is generated.

The X bit in the Address Link and Register Link instructions distinguishes Execute and Link instructions from the Branch and Link instructions.

If the I bit is set, then the operand developed on the B bus is taken as a Control Store address. The halfword contents of the addressed Control Store location are placed on the B bus and the instruction continues with the indirect data.

The D bit enables the Privilege/Illegal ROM and the Op-code-to-Address Translator. Unless a branch is taken or an interrupt occurs, the next instruction is taken from the address dictated by the Op-code-to-Address Translator.

The K bit is used as an extension of the F field allowing more than 16 functions to be performed by the addressed module.

The Memory Control (MC) field controls memory access and MAR and LOC activities. The Read Only Location Center (RLC) contains the address of the next micro-instruction to be fetched and executed.

An explanation of each of the basic instruction used in the micro-coding of this project is presented below in a diagrammatic form.

**ADD**

A, AI: \((S) \leftarrow (A) + Be\)

then: \((RLC 10:15) \leftarrow \text{PAGE ADDRESS if } C = 0 \text{ or Carry } = 0\)

\((RLC 4:15) \leftarrow (RLC 4:15) + 1 \text{ if } C = 1 \text{ or Carry } = 1\)

Figure D.4

Diagrammatic Instruction Flow

267
SUBTRACT

S, SI: \( (S) \leftarrow (A) - B \)

then: \( (RLC\ 10:15) \leftarrow \text{PAGE ADDRESS if } C = 0 \text{ or } \text{Carry} = 0 \)

\( (RLC\ 4:15) \leftarrow (RLC\ 4:15) + 1 \text{ if } C = 1 \text{ or } \text{Carry} = 1 \)

AND

N, NI: \( (S) \leftarrow (A) \text{ AND } B \)

then: \( (RLC\ 10:15) \leftarrow \text{PAGE ADDRESS} \)

EXCLUSIVE OR

X, XI: \( (S) \leftarrow (A) \text{ XOR } B \)

then: \( (RLC\ 10:15) \leftarrow \text{PAGE ADDRESS} \)

BRANCH and LINK

Tested Condition True:

\( (LINK) \leftarrow (RLC\ 4:15) + 1 \)

\( (RLC\ 5:15) \leftarrow \text{ADDRESS 20:30} \quad (RLC\ 4) \leftarrow \text{ADDRESS 31 (ADDRESS LINK)} \)

\( (RLC\ 4:15) \leftarrow (B) \text{ and MASK (REGISTER LINK)} \)

Tested Condition False:

\( (LINK) \leftarrow (RLC\ 4:15) + 1 \)

\( (RLC\ 4:15) \leftarrow (RLC\ 4:15) + 1 \)

OR

0, 0I: \( (S) \leftarrow (A) \text{ OR } B \)

then: \( (RLC\ 10:15) \leftarrow \text{PAGE ADDRESS} \)

LOAD

L, LI: \( (S) \leftarrow \text{BE} \)

LX: \( (S) \leftarrow \text{Be} \)

then: \( (RLC\ 10:15) \leftarrow \text{PAGE ADDRESS} \)

Figure D.4 (Cont.)

Diagrammatic Instruction Flow
**EXECUTE and LINK**

Tested Condition True:

(LINK)+(RLC 4:15) + 1

do instruction at ADDRESS (ADDRESS LINK)

do instruction at (B) and MASK (REGISTER LINK)

(RLC 4:15)←→(RLC 4:15) + 1

Tested Condition False:

(LINK)←→(RLC 4:15) + 1

(RLC 4:15)←→(RLC 4:15) + 1

Figure D.4 (Cont.)

Diagrammatic Instruction Flow
APPENDIX E

ASSEMBLER OP-CODES AND MICRO-CODED EMULATION
Shift Left Logical

Assembler: SLLS R1,N

Description: The content of the first operand is shifted left the number of positions specified by the second operand. High order bits shifted out of position 0 are shifted through the Carry Bit of the PSW (PSW12) and then lost. Zeroes are shifted into the low order bit position.

Micro-code: SLLS SL YD,YD,YS1,IR2,E,D

Description: See figure D.4

Shift Right Logical

Assembler: SRLS R1,N

Description: The content of the first operand is shifted right the number of bit positions specified by the second operand. Low order bits shifted out of position 15 are shifted through the Carry Bit of the PSW (PSW12) and then lost. Zeroes are shifted into position 0.

Micro-code: SRLS SR YD,YD,YS1,IR2,E,D

Description: See figure D.4

Load Halfword

Assembler: LHR R1,R2

LHI R1,Be

LH R1,Be

Description: The second operand is loaded into the General Register specified by R1.

Micro-code: LH A MAR,MDR,YX,DR4

0 YD,NULL,MDR,IRJ,E,D

LHI A MRO,MDR,YX,IR4

0 YD,NULL,MRO,D,E

271
LHR 0 YD,NULL,YS(IR2,E,D

Description: See figure D.4

Store Halfword

Assembler: STH R1,Be

Description: The 16-bit first operand is stored in the memory location specified by the second operand. The first operand is unchanged.

Micro-code: STH A MAR,MDR,YX,I4

A YD,LOC

L LOC,MRO,IRJ,D

Description: See figure D.4

Add Halfword

Assembler: AHR R1,R2

AHI R1,Be

AIS R1,N

Description: The second operand is added algebraically to the contents of the General Register specified by R1. The Add Immediate Short (AIS) instruction, causes the four-bit second operand N to be added to the contents of the General Register specified by R1. The result replaces the contents of R1. The Add Halfword Immediate (AHI) adds the one-byte immediate second operand to the contents of the General Register as specified by the R1 first operand.

Micro-code: AHR A YD,YD,YS,IR2,E,D

AHI A MRO,MDR,YX,I4

A YD,YD,MRO,E,D

AIS A YD,YD,ysi,IR2,E,D

Description: See figure D.4

272
Subtract Halfword

Assembler: SHR R1,R2
            SHI R1,Be
            SIS R1,N

Description: The second operand is subtracted from the General Register specified by R1. The difference is contained in R1. The second operand is unchanged. The Subtract Immediate Short (SIS) instruction causes the four-bit second operand N to be subtracted from the contents of the General Register specified by R1. The Subtract Halfword Immediate (SHI) instruction subtracts the immediate one-byte second operand from the General Register specified by R1.

Micro-code: SHR S YD,YD,YS,IR2,E,D
            SHI A MRO,MDR,YX,IR4
            S YD,YD,MRO,E,D
            SIS S YD,YD,YS1,IR2,E,D

Description: See figure D.4

Compare Logical Halfword

Assembler: CLHR R1,R2
            CLH R1,Be
            CLHI R1,Be

Description: The first operand specified by R1 is compared logically to the 16-bit second operand. The result is indicated by the setting of the Condition Codes (PSW 12:15). Both operands remain unchanged.

Micro-code: CLHR S NUL,YD,YS,IR2,E,D
            CLH A MAR,MDR,YX,DR4
            CLHI S NUL,YD,MDR,IRJ,E,D
CLHI A MRO,MDR,YX,IR4
S MRO,YD,MRO,D,E
NUL EQU '17'

Description: See figure D.4

**Compare Halfword**

Assembler: CHR R1,R2

Description: The first operand specified by R1 is compared to the 16-bit second operand. The comparison is algebraic, taking into account the sign and magnitude of each number. The result is indicated by the setting of the Condition Code (PSW 12:15). Both operands remain unchanged.

Micro-code: CHR BAL CHR1(MR7),IR2
CHR1 A MDR,NULL,YS
X MRO,YD,MDR
BALNL CLH1(MR7)
ASRI MRO,YD,1
OI MRO,MRO,MRO,IRJ,E,D

Description: See figure D.4

**Exclusive OR Halfword**

Assembler: XHR R1,R2
XHI R1,Be

Description: The logical difference of the 16-bit second operand and the General Register specified by R1, replaces the contents of R1. The 16-bit difference is formed on a bit-by-bit basis.
Micro-code: XHR X YD,YD,YS,IR2,E,D
          XHI A MRO,MDR,YX,IR4
          X YD,YD,MRO,E,D

Description: See figure D.4

AND Halfword

Assembler:     NHR   R1,R2

Description: The logical product of the 16-bit second operand and the content of the General Register specified by R1, replaces the contents of R1. The 16-bit product is formed on a bit-by-bit basis.

Micro-code: NHR N YD,YD,YS,IR2,E,D

Description: See figure D.4

Branch and Link

Assembler:     BALR  R1,R2
                BAL   R1,Be

Description: The address of the next sequential instruction is saved in the General Register specified by R1, and an unconditional branch is executed to the 16-bit address specified by the second operand.

Micro-code: BALR L ADR,YS
           AI YD,LOC,2
           L LOC,ADR
           A MRO,MRO,NULL,IRJ,D
           BAL A MRO,MDR,YX,I4
           L YD,LOC
           L LOC,MRO,IRJ,D

Description: See figure D.4
Extended Mnemonics

Extended mnemonics used in the assembly language in previous chapters will be translated into either a BTC or a BFC with the appropriate mask being substituted, i.e.,

- BZ Be assembles BFC 3,Be
- B Be assembles BTC 0,Be
- BNC Be assembles BFC 8,Be
- BC Be assembles BTC 8,Be
- BNE Be assembles BFC 3,Be
- BE Be assembles BFC 3,Be
- BNZ Be assembles BFC 3,Be

Description: The Condition code of the PSW is tested for the condition specified by the mask field. For the BFC, if all conditions tested for are found false, then the branch is executed to the 16-bit address specified (Be). For the BTC, he conditions are tested and if any of the conditions are found to be true, then the branch is executed to the 16-bit address specified (Be).

Micro-code: BTC A MRO,MDR,YX,I4

BALNF BRX(MR7),D
BFC A MRO,MDR,YX,IR4
BALF BRX(MR7),D
BRX L LOC,MRO,IRJ,D

Description: See figure D.4
THE DESIGN OF AN INTERDATA IMPLEMENTATION
OF THE U. S. NAVY MINI-COBOL

by

TERRY WAYNE ANDERSON-ROVIA

B. S., LOYOLA UNIVERSITY (CHICAGO), 1964

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1976
This report describes the transfer of the U. S. Navy MINI-COBOL compiler from a NOVA to an INTERDATA mini-computer. The principles utilized in this report may be used to transfer the MINI-COBOL compiler to any mini-computer which uses a micro-coded instruction set with minor modification to the FORTRAN interpreter as designed by the Navy. Also presented is a description of the MODEL 80 INTERDATA microprocessor and the micro-instruction formats which should enable the reader to better understand the micro-code presented as emulated NOVA instructions. A description of the mnemonics used in the micro-code is given to facilitate the reading of the emulation code. Formats of the NOVA instructions which will be emulated are presented. Specifications of the Fetch and Decode modules found in the emulator are given along with the micro-code generated for each NOVA instruction. A sample algorithm is given including a step-by-step trace table of two typical NOVA instructions. Also included are suggestions for modifications of the MINI-COBOL interpreter, specifications of the MINI-COBOL language, a description of the compiler system, and listings of the source code for all modules used in the system.