THE MI-3 ASSEMBLER REFERENCE MANUAL

R. W. Cornelli

DECEMBER 1963

ESD RECORD COPY

RETURN TO
SCIENTIFIC & TECHNICAL INFORMATION DIVISION
(ESTL) BUILDING 1211

Prepared for

DIRECTORATE OF PLANNING AND TECHNOLOGY
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 700A
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts

Contract F19(628)-68-C-0365

This document has been approved for public release and sale; its distribution is unlimited.
When U.S. Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.
THE MI-3 ASSEMBLER REFERENCE MANUAL

R. W. Cornelli

DECEMBER 1969

Prepared for

DIRECTORATE OF PLANNING AND TECHNOLOGY
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 700A
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract F19(628)-68-C-0365

This document has been approved for public release and sale; its distribution is unlimited.
FOREWORD

This report describes a one pass assembler built by the MITRE Corporation for a family of microprogrammable computers. It is in partial fulfillment of Project 7120 under Contract No. F19(628)-68-C-0365. It was prepared under the cognizance of Mr. Robert W. Cornelli of the MITRE Corporation, Bedford, Massachusetts. The USAF project monitor is Mr. Russell A. Meier.

REVIEW AND APPROVAL

Publication of this technical report does not constitute Air Force approval of the report's findings or conclusions. It is published only for the exchange and stimulation of ideas.

WILLIAM F. HEISLER, Colonel, USAF
Chief, Command Systems Division
Directorate of Planning and Technology
ABSTRACT

MI-3 is a primitive, interactive, one pass assembler which assembles in-core code for a family of microprogrammed computers based on an Interdata 3 (I-3) micromachine.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>SECTION</th>
<th>CONTENTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II</td>
<td>DATA TYPES</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>INTEGERS</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>SYMBOLS</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>S Symbols</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>G and L Symbols</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>REGISTER IDENTIFIERS</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>ASTERISK (*)</td>
<td>3</td>
</tr>
<tr>
<td>III</td>
<td>FORMATS</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>LOC FIELD</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>OP FIELD</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>DATA FIELD</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>COMMENTS FIELD</td>
<td>5</td>
</tr>
<tr>
<td>IV</td>
<td>EXPRESSIONS</td>
<td>5</td>
</tr>
<tr>
<td>V</td>
<td>INSTRUCTIONS</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>OPERATION CODES OF TYPE 1</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>OPERATION CODES OF TYPE 2</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>OPERATION CODES OF TYPE 3</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>OPERATION CODES OF TYPE 4</td>
<td>11</td>
</tr>
<tr>
<td>VI</td>
<td>PSEUDO-OPERATIONS</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>DC (DEFINE CONSTANT)</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>DS (DEFINE STORAGE)</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>END</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>EQU (EQUALS)</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>OPD (OPERATION DEFINITION)</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>ORG (ORIGIN)</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>PUT</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>SID (SET INPUT DEVICE)</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>SOD (SET OUTPUT DEVICE)</td>
<td>17</td>
</tr>
<tr>
<td>VII</td>
<td>IDIOSYNCRASIES</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>MESSAGES</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>INPUT</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>OUTPUT</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>LANGUAGE</td>
<td>19</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS (Concluded)

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECTION VIII</td>
<td>EXAMPLES</td>
<td>20</td>
</tr>
<tr>
<td>APPENDIX I</td>
<td>MNEMONICS AND VALUES</td>
<td>23</td>
</tr>
<tr>
<td>APPENDIX II</td>
<td>FORMAL SYNTAX</td>
<td>26</td>
</tr>
<tr>
<td>APPENDIX III</td>
<td>ALPHABETIC LIST OF MNEMONICS</td>
<td>28</td>
</tr>
<tr>
<td>APPENDIX IV</td>
<td>NUMERIC LIST OF MNEMONICS</td>
<td>32</td>
</tr>
<tr>
<td>APPENDIX V</td>
<td>INDEX</td>
<td>36</td>
</tr>
</tbody>
</table>
SECTION I

INTRODUCTION

MI-3 is a primitive, interactive, one pass assembler which assembles in-core code for a family of microprogrammed computers based on an Interdata 3 (I-3) micromachine.

Versions of MI-3 have been assembled and operate under the Calliope and Venus microprograms. Calliope is a MITRE-produced superset of the I-3 delivered machine. Venus, also MITRE produced, provides multiprogramming capabilities; it includes most instructions implemented in Calliope, plus many others.

MI-3 was written for interim use, to allow time for a more powerful, flexible and useful assembler to be built. As a result, it betrays a number of anomalies and idiosyncracies not normally to be expected of a more finished product. These are described in Section VII.
SECTION II
DATA TYPES

INTEGERS

All integers are represented in the MI-3 Assembly languages in hex (base 16) as a string of hex digits. No facilities are provided for representing integers in either decimal or binary.

SYMBOLS

Three kinds of symbols are implemented in MI-3, known as S, G and L symbols.

S Symbols

An S symbol is formed by writing S followed by two hex digits. These symbols are unusual, in that they may be defined as often as desired. When S symbols are referenced, a B or F must be appended, specifying whether the symbol is defined before (Backward) or ahead (Forward) of the reference.

The occurrence of a backward S symbol in a line of code refers to the closest previous definition of that symbol. The occurrence of a forward S symbol in a line of code refers to the closest following definition of that symbol. S symbol references never refer to the line of code in which they occur.

Thus, for example,

\[
\begin{align*}
S01 & \quad \text{LHI} \quad R6, S01F \\
\vdots & \\
S01 & \quad \text{BAL} \quad R7, S01B
\end{align*}
\]

The S01F in the LHI refers to the S01 on the BAL. The S01B on the BAL refers to the S01 on the LHI.
G and L Symbols

G and L symbols consist of the letter G or L followed by 1 to 5 characters chosen from the alphabet and the digits.

For example:

LSA4
G123LM
LPQRST

Usage of G and L symbols is presently identical. An unimplemented addition would limit the scope (the set of statements over which they are defined) of L symbols so that they become, in a sense, Local. G symbols would not be so limited, and thus would be Global.

REGISTER IDENTIFIERS

A programmer may use a register identifier when he wishes to draw special attention to the fact that a value is to designate a general purpose register (otherwise, an integer will do just as well).

A register identifier is written as the letter R followed by a single hex digit identifying the particular register.

Thus, R6 can be used instead of 6, RD instead of D.

ASTERISK (*)

The * is a symbol which may be used to denote the current value of the location counter. When used in an instruction, its value is the address of the first byte of the instruction, in a DC or DS the address of the first byte assigned.
SECTION III
FORMATS

A program consists of a sequence of lines. Each line contains a LOC (location) field, an OP (operation code) field, a DATA field, and a COMMENTS field. Fields are separated from other fields by one or more spaces:

LOC     OP     DATA     COMMENTS

Embedded spaces may not appear in the LOC, OP or DATA fields.

LOC FIELD

The LOC field is optional on instructions and on the DC and DS pseudo-operations, required on an EQU line, and ignored on all others.

When present, it must start in the first input column, and consist of a G, L or S symbol. In the EQU line, the symbol in the LOC field is assigned the value of the operand of the EQU; in all other cases, the symbol takes on the value of the current location counter (*).

OP FIELD

The OP field contains the name of an instruction or a pseudo-operation. It may be from 1 to 4 alphanumeric characters.

If the LOC field is absent, the OP field may still not start before the second input column. It is terminated by the first blank character.

DATA FIELD

The DATA field, separated from the OP field by one or more spaces, contains the operands for instructions and pseudo-operations.

COMMENTS FIELD

Input columns beyond the DATA field may be used to enter comments into the program, and may contain embedded blanks. The COMMENTS field is separated from the OP field by one or more spaces.
SECTION IV

EXPRESSIONS

The DATA field of all instructions and most pseudo-operations contain expressions. An expression is one or more symbols and/or numbers, connected with the operators + and/or -. Arithmetic may be performed on any combinations of:

- backward $S$ symbols
- register identifiers
- integers
- * (the current location counter)

Arithmetic may not be performed on:

- forward $S$ symbols
- L symbols
- G symbols

When these symbols are used, they must stand alone.

Each component of an expression is considered to be a 16-bit value; 16-bit sums and differences are computed in two's complement arithmetic. When the value of an expression is inserted into a field of an instruction and the field is less than 16 bits wide, the leftmost bits of the value are stripped off.

Examples:

LXYZ
GBQ3E
SAAF
S22B
*
2A
R5
S01B-7++
RB+23
Ø-12
Counter Examples:

L9FC+5
**-CRAB
SAAF-S$1B

(Arithmetic not legal on L symbols,
G symbols and forward S symbols)

-12

( + and - must appear between two
symbols or numbers)
SECTION V

INSTRUCTIONS

Four assembler formats, which assemble into two basic machine formats, are supported for instructions. The machine formats are either 16 or 32 bits long. The first 8 bits of each contains the operation code:

```
   OP  |  R1  |  R2
```

```
   OP  |  R1  |  X2  |  A
```

R1, R2 and X2 are 4-bit fields; A is 16 bits.

<table>
<thead>
<tr>
<th>Type</th>
<th>Data Field Format</th>
<th>Assembled Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R1,R2</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>R1,A(X2) or R1,A</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>R2</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>A(X2) or A</td>
<td>32</td>
</tr>
</tbody>
</table>

Any expression may be used to define the A field. The R1, R2 and X2 fields may be defined using expressions, but not L, G or forward S symbols. Expressions used to define the R1, R2 and X2 fields are evaluated as 16 bits, then truncated to 4 bits.

Except for the always optional X2 field, fields may not be omitted; Ø may be entered instead.

In operations of types 1 and 2, the R1 field usually refers to one of the general registers. In the BTC, BTCR, BFC and BFCR instructions, however, the R1 field is a mask which determines the conditions to be tested. Type 3 and 4 operations represent extended mnemonics for such instructions in which the mask, i.e. the R1 field, is implicit in the mnemonic.
OPERATION CODES OF TYPE 1

An operation code of type 1 requires two operands, R1 and R2, both four bits in length. It is written in the form:

\[ \text{LOC} \quad \text{OP} \quad R1, R2 \]

and is assembled into 16 bits:

<table>
<thead>
<tr>
<th>OP</th>
<th>R1</th>
<th>R2</th>
</tr>
</thead>
</table>

Examples:

- LBR  R6, R5
- BALR S91B+3, *-S3FB

Counter Examples:

- BTCR LABC, R7 (L and G symbols may not be used to define an R1 or R2 field)
- MHR R5, G124
- STBR , RC (operand may not be omitted)
OPERATION CODES OF TYPE 2

An operation code of type 2 requires two operands, R1 (4 bits) and A or A(X2). R1 and X2 are 4 bits long, and A is 16 bits. It is written:

\[
\text{LOC OP R1, A} \\
\text{or} \\
\text{LOC OP R1, A(X2)}
\]

and is assembled into 32 bits:

<table>
<thead>
<tr>
<th>OP</th>
<th>R1</th>
<th>∅</th>
<th>A</th>
</tr>
</thead>
</table>

or

| OP | R1 | X2 | A |

respectively.

Examples:

AHI R6,24
NH X2,CA(R5)
BAL RF, LABEL
XHI RA, LOC(R1+R2-SØ1B)

Counter Examples:

STBS 3, LX(GAX) (G symbol not allowed in X2 field)
WD R4 (missing operand)
OPERATION CODES OF TYPE 3

An operation code of type 3 requires one operand, R2, 4 bits long. It is written in the form:

LOC    OP    R2

and assembles as 16 bits:

```
OP    R2
```

Operations of this type represent extended mnemonics for the instructions BTCR and BFCR, with R1 set implicitly. Those defined are listed in Appendix A.

Example:

```
BR    RF
```
OPERATION CODES OF TYPE 4

An operation code of type 4 requires one operand, A or A(X2), where A is 16 bits long, and X2 is 4 bits. It is written:

LOC OP A

or

LOC OP A(X2)

and assembles into 32 bits:

\[
\begin{array}{c|c|c}
\text{OP} & \emptyset & A \\
\end{array}
\]

or

\[
\begin{array}{c|c|c}
\text{OP} & X2 & A \\
\end{array}
\]

respectively.

Operations of this type are extended mnemonics for the BTC and BFC basic instructions, using implicit values for the RI field. Those defined are listed in Appendix A.

Examples:

\[
\begin{align*}
\text{BZ} & \quad \text{LABEL} & \quad \text{(same as BFC 3, LABEL)} \\
\text{BO} & \quad S\emptyset 1B-5(R6) & \quad \text{(same as BTC 4, S\emptyset 1B-5(R6))}
\end{align*}
\]
SECTION VI

PSEUDO-OPERATIONS

DC (DEFINE CONSTANT)

A DC line is used to define a single, 16-bit constant:

LOC       DC       EXPRESSION

DS (DEFINE STORAGE)

A DS line is used to reserve a number of bytes in storage:

LOC       DS       EXPRESSION

Symbols used in the expression must have been previously defined.

The line

SØØ       DS       35C

is equivalent to

SØØ       EQU       *
ORG       *+35C
END

An END line is used to indicate the end of the source program and the address of the first memory location to be executed.

END OPERAND

OPERAND may be any expression, but it must be present and defined (it may be $\emptyset$).

Examples:

```
END LSTART
END $\emptyset$
```

Counter Examples:

```
END (operand must be present)
END SA5F (operand undefined)
```

EQU (EQUALS)

An EQU line is used to define a symbol without generating a line of object code:

```
LOC EQU EXPRESSION
```

where LOC is a G, L or S symbol. Symbols used in the expression must have been previously defined.
OPD (OPERATION DEFINITION)

The OPD line may be used to add instruction and pseudo-operation mnemonics to the set recognized by MI-3. In fact, MI-3 itself uses the OPD to define all mnemonics except OPD itself, which is built in. When MI-3 is assembled, a deck of OPD cards is also assembled. For the Venus instruction set, the OPD cards are listed in Appendixes C and D.

OPD is useful for defining new mnemonics for instructions, or synonyms for pseudo-operations. Only additions are possible; old mnemonics may not be deleted, nor may values associated with them be changed, or entirely new pseudo-operations added.

OPD is written:

\[
\text{OPD 'OPNAME', OPCODE, FORMAT}
\]

which is assembled into the operation table as:

<table>
<thead>
<tr>
<th>OPNAME</th>
<th>OPCODE</th>
<th>FORMAT</th>
</tr>
</thead>
</table>

where

OPNAME is a 1 to 4-character alphanumeric string which is the desired mnemonic.

OPCODE is an expression in which all symbols have been previously defined. It specifies the 8-bit operation code to be associated with the name OPNAME for instructions. It must be zero for pseudo-operations.

FORMAT is an expression in which all symbols have been previously defined. It is an 8-bit field; the last 4 bits specify the instruction types 1, 2, 3 or 4 or the pseudo-operation type (see Appendix A).

The first 4 bits of FORMAT are meaningless except for instruction types 3 and 4, in which they specify the value to be used in the R1 field.
Examples:

    OPD 'STBR',22,01    (type 1, OP = 22)
    OPD 'BL',72,84     (type 4, OP = 72, R1 = 8)
    OPD 'OP',00,00     (type 8, OP = 0 for all pseudo-operations)
                      (adds a new name for DC)

Note that since definitions are made directly into core, a definition is permanent until a new copy of the assembler is loaded.

ORG (ORIGIN)

An ORG line is used to set the value of the location counter:

    ORG EXPRESSION

where symbols used in the expression must have been previously defined.

PUT

A PUT line is used when a program is to be assembled into locations other than those from which it will be executed. Thus, MI-3 must assemble the instructions as though they were in the locations from which they will be executed, but it must PUT them in a different place.

The PUT is used most often in one of two situations: to relocate code which will later overlay part of the assembler; and to assemble code with origin zero (for relocation by index) without destroying the low address region of memory.

The PUT is written:

    PUT EXPRESSION

where symbols used in the expression must have been previously defined.
The effect of a PUT line is to define a constant that is added to the assembled address of each line of code to obtain the address in core at which it will be placed. The addition is done modulo \(2^{18}\), causing a wrap-around effect.

Example 1:

```
ORG 2500
PUT 1000
S01 B *
```

S01 and * will be defined as 2500, and the instruction will be stored in 3500.

Example 2:

```
ORG 5500
PUT ED00
B *
```

* is assigned the value 5500, and the instruction stored at location 4200, since ED00 is equivalent to -1300. It could also have been written:

```
PUT 0-1300
```

Example 3:

```
ORG 47E2
S11
EQU *
PUT 4500-S11B
B *
```

assembles into location 4500, with * and S11 defined as 47E2.

Note that in Example 3 the output of the assembly will be placed at 1000 regardless of where it is ORGed.
**S**

**ID (SET INPUT DEVICE)**

A SID line controls the source of symbolic input to MI-3. The only valid possibilities are:

- SID 0  paper tape
- SID 1  keyboard
- SID 2  card reader (initial value)

The operand field may not contain an expression; only the explicit values, $\emptyset$, 1 and 2 may be used.

**S**

**OD (SET OUTPUT DEVICE)**

A SOD line controls the printing of a listing and the output device on which it is to be printed. The only possibilities are:

- SOD 1  teletype
- SOD 2  printer (initial value)
- SOD 3  no print

The operand field may not contain an expression; only the explicit values 1, 2 and 3 may be used.
SECTION VII
IDIOSYNCRASIES

Some of the idiosyncrasies of MI-3 are discussed briefly below. It is possible that as time goes on changes may be made to MI-3 which eliminate or change some of them.

MESSAGES

There are only four messages built into MI-3. All four are forced to the operator's teletype. Two of them announce the beginning and ending of an assembly. The third recognizes a system failure which has occurred in the form of an illegal instruction during the assembly. The fourth, consisting of the words ERROR IN FOLLOWING LINE, represents MI-3's total capability for diagnosing user errors. At this point, the user must enter a valid line on the teletype which is to replace the one found to be in error.

INPUT

When using the on-line teletype as an input device, MI-3 provides no editing capabilities whatsoever. The usual abilities to cancel a line or to backspace characters are not present.

The specifications state that the end of the DATA field is determined by a space. In fact, it is determined by the first character which MI-3 recognizes as being invalid in a data field. Thus, for example, characters such as & or % will, in fact, terminate the data field without any error indication.

When MI-3 is first entered, it identifies itself by logging a message on the console teletype. If the card reader, the assumed input device, is not ready with cards, MI-3 will wait until it is, with no indication of what it is waiting for. Similarly, MI-3 will wait until the printer, the assumed listing output device, is ready; it will have read the first card. In both cases, readying the device allows operation of MI-3 to proceed without restarting.
OUTPUT

The binary results of the assembly are stored directly into core memory. There is no other computer processable output.

The output listing does not print out the binary values assembled nor their locations in memory. While S symbol values are logged when the S symbol is defined, no indication is given of the value of G or L symbols. Undefined symbols are not listed, nor, for that matter, are defined symbols.

LANGUAGE

The concept of S symbols appears entirely unique to MI-3. The fact that a particular S symbol can appear any number of times in a given assembly is, perhaps, the most unusual aspect. Coupled with this is the need for the programmer to indicate to MI-3 whether the particular S symbol referred to has been defined earlier or will be defined later in the assembly.

Character strings may not be written explicitly. The only way character strings can be specified is to write them as the series of equivalent hex constants in DC statements, or in hex as instruction operands.

All numbers entered by a programmer as part of the program must be entered in hex. No provision is made to handle numbers written in decimal (base 10) form.
SECTION VIII

EXAMPLES

In the examples below, the output of the assembler is shown as a 4 hex digit location, followed by a colon, followed by 4 or 8 hex digits representing the code stored in that location.

Example 1:

```
ORG 2000
B 4(3)
XHR R1, RD
LHI RC, 3DE1
BR FC
```

assembles into:

```
2000:  7403 0004
2004:  C71D
2006:  D8C0 3DE1
200A:  840C
```

Example 2:

```
ORG 1F00
B *
B **+1
B **=*+1
B 4-1-1-1-1
```

assembles into:

```
1F00:  7400 1F00
1F04:  7400 3E08
1F08:  7400 0001
1F0C:  7400 0000
```
Example 3:

<table>
<thead>
<tr>
<th></th>
<th>ORG</th>
<th>3000</th>
</tr>
</thead>
<tbody>
<tr>
<td>S00</td>
<td>B</td>
<td>S00F</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>S00B</td>
</tr>
<tr>
<td>S00</td>
<td>B</td>
<td>S00B</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>SFCF</td>
</tr>
<tr>
<td>SFC</td>
<td>B</td>
<td>S00B</td>
</tr>
</tbody>
</table>

Assembles into:

<table>
<thead>
<tr>
<th></th>
<th>7400</th>
<th>3008</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>7404</td>
<td>3000</td>
</tr>
<tr>
<td>3008</td>
<td>7400</td>
<td>3000</td>
</tr>
<tr>
<td>300C</td>
<td>7400</td>
<td>3014</td>
</tr>
<tr>
<td>3010</td>
<td>7400</td>
<td>3008</td>
</tr>
<tr>
<td>3014</td>
<td>7400</td>
<td>3008</td>
</tr>
</tbody>
</table>

Example 4:

<table>
<thead>
<tr>
<th></th>
<th>ORG</th>
<th>40F0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S08</td>
<td>DC</td>
<td>897FD1C8B</td>
</tr>
<tr>
<td></td>
<td>DC</td>
<td>S07F</td>
</tr>
<tr>
<td></td>
<td>DC</td>
<td>S08B-1</td>
</tr>
<tr>
<td></td>
<td>DC</td>
<td>*</td>
</tr>
<tr>
<td>S07</td>
<td>DC</td>
<td>S08B+*</td>
</tr>
</tbody>
</table>

Assembles into:

<table>
<thead>
<tr>
<th></th>
<th>1C8B</th>
</tr>
</thead>
<tbody>
<tr>
<td>40F0</td>
<td>40F2</td>
</tr>
<tr>
<td>40F4</td>
<td>40F1</td>
</tr>
<tr>
<td>40F6</td>
<td>40F6</td>
</tr>
<tr>
<td>40F8</td>
<td>81EA</td>
</tr>
</tbody>
</table>
Example 5:

```
ORG 44E0
S09 DS 200
S10 DC S09B
S11 DS 4
B *
```

assembles into:

```
46E0: 44E0
46E6: 7400 46E6
```
APPENDIX I

MNEMONICS AND VALUES

The set of mnemonics supplied in the version of MI-3 which runs on the Venus machine is summarized below. As far as the assembler is concerned, mnemonics fall into fifteen types, each identified by a type code. The type determines the format and meaning of the DATA field of instructions, and identifies extended mnemonics and pseudo-operations:

1. 16 bit instructions
2. 32 bit instructions
3. Extended mnemonics for 16 bit instructions (BTCR, BPCR)
4. Extended mnemonics for 32 bit instructions (BTC, BFC)
5. ORG Origin
6. EQU Equals
7. END End
8. DC Define Constant
9. DS Define Storage
A. PUT Put
B. OPD Operation Definition
C. SOD Set Output Device
D. SID Set Input Device
E. LEND Local End (Not implemented)
F. Not assigned
Instructions appear in a 16 x 16 matrix in which the 8-bit operation code is formed by taking the row number in hex followed by the column number, also in hex. For example, CALL is in row 7, column A; its operation code is 7A. Instructions in rows 2, 8 and C are of type 1; those in the other rows, type 2.

Extended operations are listed separately.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>P</td>
<td>V</td>
<td>STB</td>
<td>POB</td>
<td>STH</td>
<td>PO</td>
<td>SSN</td>
<td></td>
<td>OC</td>
<td>RD</td>
<td>WD</td>
<td>SS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>STBR</td>
<td>POBR</td>
<td>STHR</td>
<td>POR</td>
<td>SSNR</td>
<td></td>
<td>OCR</td>
<td>RDR</td>
<td>WDR</td>
<td>SSR</td>
<td>JOBA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PS</td>
<td>VS</td>
<td>STBS</td>
<td>POBS</td>
<td>STHS</td>
<td>POS</td>
<td>SSNS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>DIE</td>
<td>UNQP</td>
<td>PUC</td>
<td>POC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>BXLE</td>
<td>BAL</td>
<td>BTC</td>
<td>BXH</td>
<td>BFC</td>
<td>SIO</td>
<td>ELI</td>
<td>SET</td>
<td>RSET</td>
<td>RETN</td>
<td>CALL</td>
<td>ICOR</td>
<td>SRHL</td>
<td>SLHL</td>
</tr>
<tr>
<td>7</td>
<td>BXLR</td>
<td>BALR</td>
<td>BTCR</td>
<td>BXHR</td>
<td>BFCR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>PU</td>
<td>PUB</td>
<td>LSN</td>
<td>NH</td>
<td>CLH</td>
<td>OH</td>
<td>XH</td>
<td>LH</td>
<td>LB</td>
<td>AH</td>
<td>SH</td>
<td>MH</td>
<td>DH</td>
<td>ACH</td>
<td>SCH</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>PUR</td>
<td>PUBR</td>
<td>LSNR</td>
<td>NHR</td>
<td>CLHR</td>
<td>OHR</td>
<td>XHR</td>
<td>LHR</td>
<td>LBR</td>
<td>AHR</td>
<td>SHR</td>
<td>MHR</td>
<td>DHR</td>
<td>ACHR</td>
<td>SCHR</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>PUI</td>
<td>PUBI</td>
<td>LSNI</td>
<td>NHI</td>
<td>CLHI</td>
<td>OHI</td>
<td>XHI</td>
<td>LHI</td>
<td>LBI</td>
<td>AIH</td>
<td>SHI</td>
<td>MHI</td>
<td>DHI</td>
<td>ACHI</td>
<td>SCHR</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>PUS</td>
<td>PUBS</td>
<td>LSNP</td>
<td>NHS</td>
<td>CLHS</td>
<td>OHS</td>
<td>XHS</td>
<td>LHS</td>
<td>LBS</td>
<td>AHS</td>
<td>SHS</td>
<td>MHS</td>
<td>DHS</td>
<td>ACHS</td>
<td>SCHS</td>
<td></td>
</tr>
</tbody>
</table>

24
## EXTENDED OPERATIONS

<table>
<thead>
<tr>
<th>Hex</th>
<th>Mnemonic</th>
<th>Equivalent Operation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>720</td>
<td>NOP</td>
<td>BTC $\emptyset$</td>
<td>No operation</td>
</tr>
<tr>
<td>721</td>
<td>BM</td>
<td>BTC 1</td>
<td>Branch on minus</td>
</tr>
<tr>
<td>722</td>
<td>BP</td>
<td>BTC 2</td>
<td>Branch on plus</td>
</tr>
<tr>
<td>723</td>
<td>BNZ</td>
<td>BTC 3</td>
<td>Branch on non-zero</td>
</tr>
<tr>
<td>723</td>
<td>BNE</td>
<td>BTC 3</td>
<td>Branch on not equal</td>
</tr>
<tr>
<td>724</td>
<td>BO</td>
<td>BTC 4</td>
<td>Branch on overflow</td>
</tr>
<tr>
<td>728</td>
<td>BC</td>
<td>BTC 8</td>
<td>Branch on carry</td>
</tr>
<tr>
<td>728</td>
<td>BL</td>
<td>BTC 8</td>
<td>Branch on low</td>
</tr>
<tr>
<td>740</td>
<td>B</td>
<td>BFC $\emptyset$</td>
<td>Branch</td>
</tr>
<tr>
<td>741</td>
<td>BNM</td>
<td>BFC 1</td>
<td>Branch on non-minus</td>
</tr>
<tr>
<td>742</td>
<td>BNP</td>
<td>BFC 2</td>
<td>Branch on non-plus</td>
</tr>
<tr>
<td>743</td>
<td>BZ</td>
<td>BFC 3</td>
<td>Branch on zero</td>
</tr>
<tr>
<td>743</td>
<td>BE</td>
<td>BFC 3</td>
<td>Branch on equal</td>
</tr>
<tr>
<td>748</td>
<td>BNC</td>
<td>BFC 8</td>
<td>Branch on no carry</td>
</tr>
<tr>
<td>748</td>
<td>BNL</td>
<td>BFC 8</td>
<td>Branch on not low</td>
</tr>
<tr>
<td>820</td>
<td>NOPR</td>
<td>BTCR $\emptyset$</td>
<td>No operation</td>
</tr>
<tr>
<td>820</td>
<td>BR</td>
<td>BFCR $\emptyset$</td>
<td>Branch</td>
</tr>
</tbody>
</table>

In the hex equivalent, the first two digits are the operation code of the basic instruction; the last digit is the RI field. The operations based on BFC and BTC are type 4; those based on BTCR or BFCR are of type 3.
APPENDIX II

FORMAL SYNTAX

BASIC DEFINITIONS

A lower case b is used to denote a single blank character.

```plaintext
<space> :: = b | <space> b
<hex digit> :: = 0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F
<integer> :: = <hex digit> | <integer> <hex digit>
<S symbol> :: = S <hex digit> <hex digit>
<backward local reference> :: = <S symbol> B
<forward local reference> :: = <S symbol> F
<alphabetic> :: = A|B|...|Z
<digit> :: = 0|1|2|3|4|5|6|7|8|9
<alphanumeric> :: = <alphabetic> | <numeric>
<L symbol> :: = L <up to 5 alphanumeric>
<G symbol> :: = G <up to 5 alphanumeric>
<symbol> :: = <S symbol> | <L symbol> | <G symbol>
<loc field> :: = <symbol> | <empty>
<register identifier> :: = R <hex digit>
```

EXPRESSIONS

```plaintext
<expression> :: = <forward local reference> |
<proper expression> | <L symbol> | <G symbol>
<defined value> :: = <proper expression> |
<previously defined L symbol> |
<previously defined G symbol>
<additive operator> :: = + | -
<term> :: = <integer> | <backward local reference> |
<register identifier> | *
<proper expression> :: = <term> | <proper expression>
<additive operator> <term>
```
LINES

<line> ::= <loc field> <space> <basic line> | <ORG line> |
       <END line> | <DC line> | <DS line> | <EQU line> |
       <OPD line> | <PUT line> | <SID line> | <SOD line>

<basic line> ::= = <op code of type 1> <space> <data field of type 1> |
               <op code of type 2> <space> <data field of type 2> |
               <op code of type 3> <space> <data field of type 3> |
               <op code of type 4> <space> <data field of type 4>

<data field of type 1> ::= = <4 bit operand>,<4 bit operand>
<data field of type 2> ::= = <4 bit operand>,<16 bit operand> |
                          <4 bit operand>,<16 bit operand> (<4 bit operand>)
<data field of type 3> ::= = <4 bit operand>
<data field of type 4> ::= = <16 bit operand> |
                          <16 bit operand> (<4 bit operand>)

<4 bit operand> ::= = <proper expression>
<16 bit operand> ::= = <expression>
<DC line> ::= = <loc field> <space> DC <space> <expression>
<DS line> ::= = <loc field> <space> DS <space> <defined value>
<END line> ::= = <space> END <space> <defined value>
<EQU line> ::= = <symbol> <space> EQU <space> <defined value>
<OPD line> ::= = <space> OPD <space> 'up tp 4 alphanumeric' ,
               <defined value>,<defined value>
<ORG line> ::= = <space> ORG <space> <defined value>
<PUT line> ::= = <space> PUT <space> <proper expression>
<SID line> ::= = <space> SID <space> <input device>
<input device> ::= = 0|1|2
<SOD line> ::= = <space> SOD <space> <output control>
<output control> ::= = 1|2|3
### APPENDIX III

#### ALPHABETIC LIST OF MNEMONICS

<table>
<thead>
<tr>
<th>OPD</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAM</td>
<td>ACH</td>
<td>RE</td>
</tr>
</tbody>
</table>
DIVIDE HALFWORD
DIVIDE HALFWORD
DIVIDE HALFWORD
DIVIDE HALFWORD
DIVIDE HALFWORD
JOB SUICIDE
DEFINE STORAGE
ENTER LEVEL 1
END
EQUAL
CHECK FOR PAGE IN CORE
FETCH JOB AREA LOCATION
LOAD BYTE
LOAD BYTE
LOAD BYTE
LOAD BYTE
LOAD BYTE
LOAD BYTE
LOAD HALFWORD
LOAD HALFWORD IMMEDIATE
LOAD HALFWORD
LOAD HALFWORD
LOAD HALFWORD
LOAD HALFWORD
LOAD STREAM NAME
LOCAL STREAM NAME
LOCAL STREAM NAME
LOCAL STREAM NAME
LOCAL STREAM NAME
LOCAL STREAM NAME
MULTIPLY HALFWORD
MULTIPLY HALFWORD
MULTIPLY HALFWORD
MULTIPLY HALFWORD
MULTIPLY HALFWORD
MULTIPLY HALFWORD
MULTIPLY HALFWORD
MULTIPLY HALFWORD
AND HALFWORD
AND HALFWORD
AND HALFWORD
AND HALFWORD
AND HALFWORD
NO OPERATION
NO OPERATION
OUTPUT COMMAND
OUTPUT COMMAND
OR HALFWORD
OR HALFWORD
OR HALFWORD
OR HALFWORD
OR HALFWORD
OR HALFWORD
ORIGIN
P OF SEMAPHORE
POP FROM STACK
POP BYTE FROM STACK
POP BYTE FROM STACK
POP BYTE FROM STACK
POP FROM CONTROL STACK
POP FROM STACK
POP FROM STACK
P OF SEMAPHORE
PUSH HALFWORD INTO STACK
PUSH BYTE INTO STACK
PUSH BYTE INTO STACK
PUSH BYTE INTO STACK
PUSH BYTE INTO STACK
PUSH BYTE INTO STACK
PUSH BYTE INTO STACK
PUSH BYTE INTO STACK
PUSH INTO CONTROL STACK
PUSH HALFWORD INTO STACK
PUSH FROM PROGRAM
PUSH HALFWORD INTO STACK
PUSH HALFWORD INTO STACK
PUT
READ DATA
READ DATA
SUBROUTINE RETURN
ROTATE LEFTWARD HALFWORD
RESET CONDITION/ON REGISTER
SUBTRACT WITH CARRY HALFWORD
SUBTRACT WITH CARRY HALFWORD
SUBTRACT WITH CARRY HALFWORD
SUBTRACT WITH CARRY HALFWORD
SUBTRACT WITH CARRY HALFWORD
SUBTRACT WITH CARRY HALFWORD
SUBTRACT WITH CARRY HALFWORD
SET CONDITION/ON REGISTER
SUBTRACT HALFWORD
SUBTRACT HALFWORD
SUBTRACT HALFWORD
SUBTRACT HALFWORD
SUBTRACT HALFWORD
SYSTEM INPUT DEVICE
START I/O CHANNEL
SHIFT LEFT LOGICAL
SYSTEM OUTPUT DEVICE
SHIFT RIGHT ARITHMETIC
SHIFT RIGHT LOGICAL
SENSE STATUS
STORE STREAM NAME
STORE STREAM NAME
STORE STREAM NAME
STORE STREAM NAME
SENSE STATUS
STORE BYTE
STORE BYTE
STORE BYTE
STORE HALFWORD
STORE HALFWORD
STORE HALFWORD
UNQUEUE WHEN DISK SWAP COMPLETE
V OF SEMAPHORE
V OF SEMAPHORE
WRITE DATA
WRITE DATA
EXCLUSIVE OR HALFWORD
EXCLUSIVE OR HALFWORD
EXCLUSIVE OR HALFWORD
EXCLUSIVE OR HALFWORD
EXCLUSIVE OR HALFWORD
EXCLUSIVE OR HALFWORD
APPENDIX IV

NUMERIC LIST OF MNEMONICS

OPD 1 ORG 1 00,05
OPD 1 EQUAL
OPD 1 END
OPD 1 DEFINE CON
OPD 1 DEFINE STORAGE
OPD 1 PUT
OPD 1 SYSTEM OUTPUT DEVICE
OPD 1 SYSTEM INPUT DEVICE
OPD 1 LOCAL END
OPD 1 P OF SEMAPHORE
OPD 1 V OF SEMAPHORE
OPD 1 STORE BYTE
OPD 1 POP BYTE FROM STACK
OPD 1 STORE HALFWORD
OPD 1 POP FROM STACK
OPD 1 STORE STREAM NAME
OPD 1 OUTPUT COMMAND
OPD 1 READ DATA
OPD 1 WRITE DATA
OPD 1 SENSE STATUS
OPD 1 STORE BYTE
OPD 1 POP BYTE FROM STACK
OPD 1 STORE HALFWORD
OPD 1 POP FROM STACK
OPD 1 STORE STREAM NAME
OPD 1 OUTPUT COMMAND
OPD 1 READ DATA
OPD 1 WRITE DATA
OPD 1 SENSE STATUS
OPD 1 FETCH JOB AREA LOCATION
OPD 1 P OF SEMAPHORE
OPD 1 V OF SEMAPHORE
OPD 1 STORE BYTE
OPD 1 POP BYTE FROM STACK
OPD 1 STORE HALFWORD
OPD 1 POP FROM STACK
OPD 1 STORE STREAM NAME
OPD 1 JOB SUICIDE
OPD 1 UNQUEUE WHEN DISK SWAP COMPLETE
OPD 1 PUSH INTO CONTROL STACK
OPD 1 POP FROM CONTROL STACK
OPD 1 BRANCH ON INDEX LOW OR EQUAL
OPD 1 BRANCH AND LINK
OPD 1 BRANCH ON TRUE CONDITION

32
NOP, 72, 04
BRM, 72, 14
BRP, 72, 24
BNF, 72, 34
BNZ, 72, 34
BP, 72, 44
BNP, 72, 44
BP, 72, 84
RL, 72, 84
RXH, 73, 02
PFC, 74, 02
RI, 74, 04
RNM, 74, 24
RN, 74, 24
RF, 74, 34
RZ, 74, 34
RH, 74, 84
RL, 74, 84
CI, 75, 02
FL, 76, 02
SN, 76, 02
SNU, 76, 02
CAL, 74, 02
IC, 78, 02
SM, 78, 02
SLH, 70, 02
SHA, 7E, 02
SPL, 7F, 02
RXL, 80, 01
RAL, 81, 01
RTC, 82, 01
NOR, 82, 03
RXH, 83, 01
REP, 84, 01
RFC, 84, 01
RR, 84, 03
PUSH, 91, 02
PUSH, 92, 02
LSN, R3, 02
NHR, 4, 02
CLH, 55, 02
OH, 56, 02
XH, 57, 02
LH, 58, 02
NO OPERATION
BRANCH ON MINUS
BRANCH ON PLUS
BRANCH ON NOT EQUAL
BRANCH ON NOT ZERO
BRANCH ON OVERFLOW
BRANCH ON CARRY
BRANCH ON LOW
BRANCH ON INDEX HIGH
BRANCH ON FALSE CONDITION
BRANCH UNCONDITIONAL
BRANCH ON NOT MINUS
BRANCH ON NOT PLUS
BRANCH ON EQUAL
BRANCH ON ZERO
BRANCH ON NO CARRY
BRANCH ON NOT LOW
START I/O CHANNEL
ENTER LEVEL 1
SET CONDITION/ON REGISTER
RESET CONDITION/ON REGISTER
SUBROUTINE RETURN
SUBROUTINE CALL
CHECK FOR PAGE IN CORE
SHIFT RIGHT LOGICAL
SHIFT LEFT LOGICAL
SHIFT RIGHT ARITHMETIC
ROTATE LEFTWARD HALFWORD
BRANCH ON INDEX LOW OR EQUAL
BRANCH AND LINK
BRANCH ON TRUE CONDITION
NO OPERATION
BRANCH ON INDEX HIGH
BRANCH EQUAL REGISTER
BRANCH ON FALSE CONDITION
BRANCH UNCONDITIONAL
PUSH HALFWORD INTO STACK
PUSH BYTE INTO STACK
LOAD STREAM NAME
AND HALFWORD
COMPARE LOGICAL HALFWORD
OR HALFWORD
EXCLUSIVE OR HALFWORD
LOAD HALFWORD
LOAD BYTE
ADD HALFWORD
SUBTRACT HALFWORD
MULTIPLY HALFWORD
DIVID HALFWORD
ADD WITH CARRY HALFWORD
SUBTRACT WITH CARRY HALFWORD
PUSH HALFWORD INTO STACK
PUSH BYTE INTO STACK
LOCAL STREAM NAME
AND HALFWORD
COMPARE LOGICAL HALFWORD
OR HALFWORD
EXCLUSIVE OR HALFWORD
LOAD HALFWORD
LOAD BYTE
ADD HALFWORD
SUBTRACT HALFWORD
MULTIPLY HALFWORD
DIVIDE HALFWORD
ADD WITH CARRY HALFWORD
SUBTRACT WITH CARRY HALFWORD
PUSH HALFWORD INTO STACK
PUSH BYTE INTO STACK
LOCAL STREAM NAME
AND HALFWORD
COMPARE LOGICAL HALFWORD
OR HALFWORD
EXCLUSIVE OR HALFWORD IMMEDIATE
LOAD HALFWORD IMMEDIATE
LOAD BYTE
ADD HALFWORD
SUBTRACT HALFWORD
MULTIPLY HALFWORD
DIVIDE HALFWORD
ADD WITH CARRY HALFWORD
SUBTRACT WITH CARRY HALFWORD
PUSH HALFWORD INTO STACK
PUSH BYTE INTO STACK
LOCAL STREAM NAME
AND HALFWORD
COMPARE LOGICAL HALFWORD
OR HALFWORD
EXCLUSIVE OR HALFWORD
LOAD HALFWORD
LOAD BYTE
ADD HALFWORD
SUBTRACT HALFWORD
MULTIPLY HALFWORD
DIVIDE HALFWORD
ADD WITH CARRY HALFWORD
SUBTRACT WITH CARRY HALFWORD
PUSH FROM PROGRAM
PUSH BYTE INTO STACK
LOCAL STREAM NAME
AND HALFWORD
COMPARE LOGICAL HALFWORD
OR HALFWORD
EXCLUSIVE OR HALFWORD
LOAD HALFWORD
LOAD BYTE
ADD HALFWORD
SUBTRACT HALFWORD
MULTIPLY HALFWORD
DIVIDE HALFWORD
ADD WITH CARRY HALFWORD
SUBTRACT WITH CARRY HALFWORD
KWIC INDEX LISTING

10/01/79
05:52:36

INDEX

KEYWORD AND TEXT

REGISTER
IDENTIFIERS
IDIOSYNCRASIES
INDEX
INPUT
SD ( SET OUTPUT DEVICE )
INSTRUCTIONS
INTEGERS
INTRODUCTION

G AND L SYMBOLS
LANGUAGE

ALPHABETIC LIST OF MNEMONICS
NUMERIC LIST OF MNEMONICS
LOG FIELD

MESSAGES

ALPHABETIC LIST OF MNEMONICS
NUMERIC LIST OF MNEMONICS
MNEMONICS AND VALUES

NUMERIC LIST OF MNEMONICS

ALPHABETIC LIST OF MNEMONICS
NUMERIC LIST OF MNEMONICS
OPERATION CODES OF TYPE 1
OPERATION CODES OF TYPE 2
OPERATION CODES OF TYPE 3
OPERATION CODES OF TYPE 4
R5 FIELD

OPD ( OPERATION DEFINITION )
OPERATION CODES OF TYPE 1
OPERATION CODES OF TYPE 2
OPERATION CODES OF TYPE 3
OPERATION CODES OF TYPE 4
OPD ( OPERATION DEFINITION )
OPD ( ORIGIN )
OPD ( ORIGIN )
OUTPUT
SD ( SET OUTPUT DEVICE )

37
KWIC INDEX LISTING

10/01/69
05:52:26

--------------- KEYWORD AND TEXT ------------- INDEX

PSUEDO-OPERATIONS
  13
PUT
  16

REGISTER IDENTIFIERS
  3

S SYMBOLS
  2
SID (SET INPUT DEVICE)
  18
SOD (SET OUTPUT DEVICE)
  18
SID (SET INPUT DEVICE)
  18
SOD (SET OUTPUT DEVICE)
  18

DS (DEFINE STORAGE)
  13
S SYMBOLS
  2
SYMBOLS
  2
G AND L SYMBOLS
  2
FORMAL SYNTAX
  29

OPERATION CODES OF TYPE 1
  9
OPERATION CODES OF TYPE 2
  10
OPERATION CODES OF TYPE 3
  11
OPERATION CODES OF TYPE 4
  12
DATA TYPES
  2

MNEMONICS AND VALUES
  25

OPERATION CODES OF TYPE 1
  9
OPERATION CODES OF TYPE 2
  10
OPERATION CODES OF TYPE 3
  11
OPERATION CODES OF TYPE 4
  12

38
THE MI-3 ASSEMBLER REFERENCE MANUAL

N/A

R. W. Cornelli

DECEMBER 1969

F19(628)-68-C-0365

700A

ESD-TR-69-371

MTR-367

This document has been approved for public release and sale; its distribution is unlimited.

N/A

MI-3 is a primitive, interactive, one pass assembler which assembles in-core code for a family of microprogrammed computers based on an Interdata 3 (I-3) micromachine.
<table>
<thead>
<tr>
<th>KEY WORDS</th>
<th>LINK A</th>
<th></th>
<th></th>
<th>LINK B</th>
<th></th>
<th></th>
<th>LINK C</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ROLE</td>
<td>WT</td>
<td></td>
<td>ROLE</td>
<td>WT</td>
<td></td>
<td>ROLE</td>
<td>WT</td>
</tr>
<tr>
<td>MI-3 Assembler</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interdata 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-Core Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microprogrammed Computers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>