DOCUMENT TRANSMITTAL

PAGE ____ OF ____

CONTROL DOC. None

SYSTEM B1900 PL FILE NO. ____

PROJ. NO. ____ COORD. ____ CHKR. Nicki TO RECORDS 7-25-18 DISTRIBUTION 7-25-18

DOC PFX	DOCUMENT NUMBER	REV	UNIT NUMBER	DL NO.	PL FILE NO.	ENG. RECORDS ONLY						REMARKS
						QTY REV PAGE	COPIES EACH PAGE	SEPIAS	MT REQ'D	CD DK REQ'D		
IPS	2222 2780	A				36	21					
DOCUMENT TRANSMITTAL												

SBP 1085 REV. 9-71

Burroughs Corporation

COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

2222 2780
B1900 DIAGNOSTIC MICRO LANGUAGE

PRODUCT SPECIFICATION

REV LTR	REVISION ISSUE DATE	APPROVED BY	REVISIONS
A	7/24/78	[signature]	Original Issue -- Release Mark VIII.0

RECEIVED
JUL 26 1978
GENERAL MANAGER
SANTA BARBARA PLANT

TABLE OF CONTENTS

INTRODUCTION

The Diagnostic Micro Language (DML) compiler is used to generate B1900 micro instructions. Each source statement (from the beginning of the statement until the semicolon) always generates one microinstruction. All code is assembled in-memory. Symbols must be defined before their use, except for branch targets. Input is free form. Instructions generally have the form of assignments if they move data.

In this document, Bmax is the maximum number of bits in any expression, and Nmax is the corresponding maximum value. Currently,

```
Bmax    =    20
Nmax    =    2 ** 20 - 1 = 1,048,575
```

This manual describes the instructions and opcodes specified in the B1920 Processor Product Specification (P.S. 2212 9209) Revision B of April 11, 1978 and should be used in conjuction with that document. The DML syntax for the microinstructions is described in the same order as those in the B1920 Processor specification. For details on the hardware implementation of each micro, refer to the above product specification.

INPUT AND OUTPUT FILES

DML is executed by: CO cocename DML options

INPUTS

The input file is named CARDS. It is defined to be a card reader with 80 character records. Only columns 1-72 are significant.

OUTPUTS

The outputs are a listing and a code file (if compile to LIBRARY was used in the compile statement). The listing is in file LINE, which is a printer or disk backup file. The code file is given the name on the compile card and is on disk. It has the format of an S-language code file, with an interpreter name of B1900/EMULATOR. It also includes seven files:

1. EM_LINE
   device = PRINTER or BACKUP DISK
   records = 132/1
   buffers = 1
   open_option = OUTPUT/NEW

2. EM_TRACE
   device = PRINTER or BACKUP DISK
   records = 132/1
   buffers = 1
   open_option = OUTPUT/NEW

3. EM_DISK
   device = DISK
   records = 180/1
   buffers = 1
   open_option = OUTPUT/NEW

4. EM_RMTE
   device = REMOTE
   label = UNLABELED
   records = 2250

5. EM_CARDS
   device = CARD
   open_option = INPUT
   records = 96

6. EM_BINTRC
```
device = DISK
records = 3240
buffers = 1
open_option = OUTPUT/NEW
```

7. 2B1900000000000000002
identification file for use by the B1900 emulator.

TOKENS AND TEXT FORMAT

The tokens are:

names -	strings of letters, digits, or _ characters, beginning with a letter.
integer constants -	strings of digits.
bits constants -	2(1) binary digits 2 (0, 1) 2(2) quartal digits 2 (0,1,2,3) 2(3) octal digits 2 (0,1,2,3,4,5,6,7) 2(4) hex digits 2 (0,1,2,3,4,5,6,7,8,9 A,B,C,D,E,F) Default bit group size is 4 (hex).
character constants -	strings of zero or more characters enclosed in quoted pairs. A single quote (') is represented by a double quote ('') in the string.
reserved words -	these are represented in upper case in this document.
special characters -	+ - * / : ; = < > % ( ) := <= >= /= /< /> /* */ . ..

Blanks are significant and may be used to separate tokens. They may not appear within tokens. Commas are treated as blanks.

[ ] are equivalent to ( ).

The input is divided into 80 character records. Characters 73-80 are ignored. Tokens are not continued across record boundaries. Comments begin with a % or a /* and terminate at a record boundary. The /* comments also terminate at a */.

RESERVED WORDS

ADJUST
ALL
ALLOCATE
AND
ANY
BACKWARD
BITS (or BIT)
BUFFERED_IO_STATUS
BUSY
BY
CALL
CAPABILITY
CASE
CLEAR
CODE
COMPLEMENT
CONNECT
COUNT
CURRENT_IO_STATUS
DAM
DEBUG
DECIMAL
DISCONNECT
DOWN
DUMP
ELOG
ELSE
EMULATOR
END
EQL
EXIT
FALSE
FIELDS
FINI
FLAGS
FOR
FROM
GEQ
GLOBAL
GO
HALT
IF
IO
INITIALIZE
INTERRUPT
JOIN
LCP
LEADING
LEFT
LEQ
LIMIT
LPW_ACCUMULATOR
MAX
MIN
MINUS
NEGATE
NEQ
NO
NOP
NOT
OF
OFF
ON
OR
PAGE
PARAMETER
PARSE_OUTPUT
POP
PORT
PROCEED
PUSH
RC
READ
REALLOCATE
RECEIVE
REQUESTING
RESET
RETURN
RIGHT
ROTATE
S_MEMORY
SELF
SEND
SET
SHIFT
STATUS
STROBE
SWAP
TERMINATE
THEN
THROUGH (or THRU)
TO
TOKENS
TRACE
TRUE
TRUNCATED
UP
USING
WAIT
WITH
WRITE
XOR

RESERVED NAMES

A	- all 32 bits of the DAM word referenced
AB	- all 64 bits of two adjacent DAM words
B	- all 32 bits of the DAM word following the one referenced
C	- first 12 bits of the DAM word referenced
D	- last 20 bits of the DAM word referenced
F	- first 12 bits of the DAM word following the one referenced
L	- last 20 bits of the DAM word following the one referenced
AMU	- associative memory unit
CACHE	- cache memory (M-memory)
CARRY	- the carry bit
HCR	- halt condition register
IMR	- interrupt mask register
ISV	- interrupt state vector
LL1	- lexic level 1 base in DAM
LL2	- lexic level 2 base in DAM
MELR	- memory error log register
MONITOR	- monitor
PROCESSOR_ID	- processor identification
TIMER	- timer

SYNTAX DESCRIPTION

The syntax is informally described. Most of the instructions are simply described by example. Instead of grouping statements by syntactic similarity (i.e., all IF statements together) they are grouped by individual microinstruction class (i.e., conditional, IO, arithmetic, ...) in the same order as in the B1920 Processor Product Specification. This document is intended to show the programmer how to generate any B1900 micro that is desired as opposed to formally describing a language.

Three methods of describing syntax are used.

1. Most instructions are described by example. All reserved words or names are shown in upper case.

2. Some terms (opd, lit_n, ...) are described in a table or in preceding sections in order to show all possible combinations of instructions. When these terms are used in a description, they will be shown in lower case. Any possible value of the term (as shown in its table) may be substituted for the lower case word.

3. Optional phrases, alternative phrases, and complicated syntax are described using an extended BNF notation or metalanguage. The metalanguage is similar to BNF with these extensions:

   - Brackets [ ] indicate that one (or none) of the enclosed alternatives may be written

   - Braces { } indicate that one of the enclosed alternatives must be written.

   When a metalinguistic symbol is used to represent itself, it is enclosed within single quotation marks; e.g. '['. The metalinguistic symbols are:

   < > ::= | { } [ ] '

   All lower case terms enclosed in < > brackets are expanded in the standard BNF fashion.

PROGRAM SYNTAX

A program is a collection of statements. Each statement ends with a semi-colon (;). The word FINI ends the program.

The terms d1, d2, and d3 can be replaced by any DAM name. They can be the same DAM name if desired. The following rules with regard to DAM names must be obeyed:

- Anywhere d1 (d2, d3) is used in the description d1.A or d1.B may be used. It must be noted that d1.B is generated as d1 + 1. DAM overflow is checked.
- Anywhere d1.C is used, d1.F may be used. It must be noted that d1.F is generated as d1.C + 1. DAM overflow is checked.
- Anywhere d1.D is used, d1.L may be used. The conditions above still apply.
- d1.AB denotes the two adjacent DAM words d1 and d1 + 1.

Instructions may be labeled for use as a target in a branch instruction. A label is a name followed by a colon (:). Labels may be referenced before they are declared, however since DNL prints the code as it reads the input, the branch target will appear as zeros in the code listing. This is fixed in the code file when the label is encountered.

An expression is defined as:

```
<expression> ::=   <product> [ [ + | - ] <product> ]

<product>    ::=   <negation> [ [* | / ] <negation> ]

<negation>   ::=   [ + | - ] <primitive>

<primitive>  ::=   name |
                   integer constant |
                   bit constant |
                   character constant |
                   (<expression>)
```

Examples of expressions :

```
5 + 3 * 2(1)0101012

DAM_43 + 10

"A" + 2FF2

127 * (42 + 25*8)
```

In the following description lit_n represents a literal with a maximum length of n bits. The literal may be an <expression> whose value is less than 2 ** n.

SYMBOL DEFINITIONS SYNTAX

```
<definition>   ::=   name = [<type>]  ( <expression> |
                                       name [(+ 1 -) <expression>];

<type>         ::=   GLOBAL |
                     LOCAL |
                     PARAMETER
```

An empty type implies that the name defined has the same type as the object it is defined as. Each type imposes its range of possible values on the expression.

<type>	range
GLOBAL	0 - 95
LOCAL	0 - 23
PARAMETER	-1 - -8
numeric (blank type)	0 - Nmax

Some Residual Control (RC) Registers may be redefined. These are: CARRY, MCR, IMR, ISV, LL1, LL2, MELR, MONITOR, PROCESSOR_ID, S, TIMER. An RC redefined can be used anywhere the original RC is used. Any name defined as GLOBAL, LOCAL, PARAMETER is a DAM name and can be used anywhere d1, d2, or d3 is shown in the description. All numeric defines can be used as constants and literals. A name must be defined before its use and a program defined name cannot be redefined. There are no string defines or macros; only numeric defines.

Examples of defines:

```
G_DAM_40                =        GLOBAL 40;
G_DAM_43                =        GLOBAL 40 + 3;
G_DAM_50                =        G_DAM_40 + 10
L_DAM_1                 =        LOCAL 1;
P_DAM_2                 =        PARAMETER -2;
TOP_DAM_STACK           =        S;
LEXIC_LEVEL_1           =        LL1;
LIT_8                   =        2(1)11111111;
ALLOCATE_AMOUNT         =        15 + 4 * 8;
```

INSTRUCTION SYNTAX

CLASS 1 == ARITHMETIC AND LOGICAL OPERATORS

opd	action
+	binary add without carry
-	binary subtract without carry
PLUS	binary add with carry
MINUS	binary subtract with carry
PLUS DECIMAL	binary coded decimal add with carry
MINUS DECIMAL	binary coded decimal subtract with carry
AND	logical and
OR	logical or
XOR	logical xor
MAX	maximum
MIN	minimum
AND NOT	logical and with the complement of the operand

op20

The above excluding PLUS, MINUS, PLUS DECIMAL, and MINUS DECIMAL. All operations are 20 bits.

op12

The above excluding PLUS, MINUS, PLUS DECIMAL, and MINUS DECIMAL. All operations are 12 bits.

Dyadic

```
d1   :=  SELF opd d2;

d1.D  :=  SELF op20 d2.D;

d1.C  :=  SELF op12 d2;

d1.C  :=  SELF op12 d2.C;
```

Reverse Dyadic

```
d1   :=  d2 opd SELF;

d1.D  :=  d2.D op20 SELF;

d1.C  :=  d2 op12 SELF;

d1.C  :=  d2.C op12 SELF;
```

Literal Dyadic

```
d1   :=  SELF opd lit_13;

d1.D  :=  SELF op20 lit_13;

d1.C  :=  SELF op12 lit_13;

d1    :=  lit_13 opd SELF;
```

Literal Adjust

```
d1   :=  d2 + lit_10;

d1   :=  d2 - lit_10;

d1   :=  lit_10 - d2;
```

D3 Dyadic

opt	action
+	binary add without carry
-	binary subtract without carry
AND	logical subtract without carry
OR	logical or
XOR	logical xor
MAX	maximum
MIN	minimum

```
d1 :=  d3 opt d2;
```

Monadic

opm	action
blank	identify - returns unaltered operand
NEGATE	returns 2's complement of operand
NOT	returns 1's complement of operand

```
d1 := opm d2;

d1.D := opm d2.D;

d1.C := opm d2;
```

Truncate

```
d1 := d2 TRUNCATED to d3.D BITS;
```

Count Leading Zeros

```
d1 := COUNT LEADING ZEROS OF d2;
```

Extract

```
d1 := d2.<subfield>;

<subfield> ::=

"[" bit number of start of subfield : length of subfield"]"  |

"["          " "          FOR          " "     "]"  |

"[" bit number of start of subfield . bit number of end of subfield"]" |

"["          " "          THROUGH      " "          "]" |

"["          " "          THRU         " "          "]"
```

and

bit number of start of subfield is 0 - 63,
length of subfield is 1 - 32,
bit number of end of subfield is 1 - 63.

A subfield must be within one DAM word (i.e., [30:6] is invalid). The DAM word d2 + 1 is used if bit number of start of subfield is greater than 31.

Insert

```
d1.<subfield>  :=   d2;

<subfield> - see Extract
```

Single Bit Manipulate

opscc	action
SET	bit is set to 1
CLEAR	bit is cleared to 0
COMPLEMENT	bit is complemented

```
d1 := opscc  d2.'[' bit number ']';

opscc d2.'[' bit number ']';
```

Bit number must be 0-63; if it is greater than 31, d2+1 is used. In the second form for this instruction, d1 and d2 are the same DAM word.

CLASS 2 == DAM FETCH/STORE OPERATORS

Fetch Indirect

```
d1 := DAM (d2 [+ offset]);
```

Offset is an 8 bit literal.

Store Absolute

```
DAM (d1 [+ offset]) := d2;
```

Store Lit_16

```
d1 :=  lit_16;
d1.D  :=  lit_16;
d1.C  :=  lit_16;
d1.<subfield>  :=  lit_16;
```

See Extract for an explanation of <subfield>. The subfield used must be equivalent to [0 : 16] (the upper half of d1).

Store Double

```
d1.AB := d2.AB;
```

Store.C

```
d1.C := d2.C;
```

Dam Swap

```
d1 := d2 SWAP;
```

CLASS 3 == SCALE/SHIFT/ROTATE OPERATORS

Scale Static

```
d1 := SELF (+1-) SHIFT d2 (LEFT | RIGHT) BY d3 BITS;
```

Scale Reverse Static

```
d1 := d2 (+1-) SHIFT SELF (LEFT | RIGHT) BY lit_5 BITS;
```

Scale Reverse Dynamic

```
d1 := d2 (+1-) SHIFT SELF (LEFT | RIGHT) BY d3 BITS;
```

Single Shift/Rotate Static

```
d1 := SHIFT d2 (LEFT | RIGHT) BY lit_5 BITS;
d1 := ROTATE d2 LEFT BY lit_5 BITS;
d1.D := SHIFT d2.D (LEFT | RIGHT) BY lit_5 BITS;
d1.D := ROTATE d2.D LEFT BY lit_5 BITS;
```

Single Shift/Rotate Dynamic

```
d1 :=  SHIFT d2 (LEFT | RIGHT) BY d3 BITS;
d1 :=  ROTATE d2 LEFT BY d3 BITS;
d1.D := SHIFT d2.D (LEFT | RIGHT) BY d3 BITS;
d1.D := ROTATE d2.D LEFT BY d3 BITS;
```

Double Shift/Rotate Static

```
SHIFT d1 JOIN d2 (LEFT | RIGHT) BY lit_5 BITS;
ROTATE d1 JOIN d2 LEFT BY lit_5 BITS;
```

Double Shift/Rotate Dynamic

```
SHIFT d1 JOIN d2 (LEFT | RIGHT) BY d3 BITS;

ROTATE d1 JOIN d2 LEFT BY d3 BITS;
```

CLASS 4 == RESIDUAL CONTROL OPERATORS

Dynamic Monitor Alter

```
MONITOR := d1;
```

Static Monitor Alter

```
MONITOR := lit_8 [[AND] FLAGS := lit_16];
```

Push RCW

```
PUSH RETURN FROM d1 JOIN d2;
```

Pop RCW

```
POP RETURN TO d1 JOIN d2;
```

Allocate

```
S := SELF + allocate amount;

ALLOCATE allocate amount;
```

Allocate amount is a 4 bit literal.

Reallocate

```
S := LL2 + allocate amount;

REALLOCATE allocate amount;
```

Alter

```
LL1 := d1;

LL2 := LL1 + d1;

CARRY := d1;

ISV := RESET INTERRUPT d1;

TIMER := d1;

MCR := d1;
```

IMR Swap

```
d1 := IMR THEN IMR := d2;

d1 := IMR SWAP;
```

In the second form of this instruction d1 and d2 are the same DAM word.

Fetch

```
d1 := S - LL1;

d1 := LL1;

d1 := LL2 - LL1;

d1 := CARRY;

d1 := ISV;

d1 := TIMER;

d1 := MCR;

d1 := PROCESOR_ID;

d1 := MELR;
```

Clear M-Memory

```
CLEAR CACHE;
```

Clear AMU

```
CLEAR AMU;
```

Clear AMU Selective

```
CLEAR AMU (d1.C);
```

Write AMU

```
WRITE AMU(d2.C) FROM d1 JOIN d3;
```

CLASS 5 == UNCONDITIONAL SEQUENCING OPERATORS

A location can be either a statement label or an <expression>. Both are relative displacements and can be forward or backward (+ or -) no greater than the maximum branch displacement. <expression> is included only for very short branches in order to eliminate the need for a label.

Call

```
CALL location [PUSH RETURN TO location]
              [ALLOCATE allocate amount];
```

Allocate amount is a 4 bit literal.

The maximum displacement for the CALL location is 4095. The maximum displacement for the return location is 63.

The two optional phrases may be in either order.

Local Call

```
CALL LOCAL location [PUSH RETURN TO location]
                   [ALLOCATE allocate amount];
```

See CALL for more information.

Proceed

```
PROCEED TO location [ALLOCATE allocate amount];
```

The maximum branch displacement is 4095.

Proceed Indirect

```
CAPABILITY PROCEED TO d1;
```

Branch

```
GO TO location [ALLOCATE allocate amount];
```

The maximum branch displacement is 4095.

Branch Indirect

```
CAPABILITY GO TO d1;
```

Exit

```
EXIT [MINUS];
```

Exit Interrupt

```
EXIT [MINUS] RESET INTERRUPT d1 THEN IMR := d2;
```

Case with Limit

```
CASE d1 LIMIT lit_13;
```

Programmatic Interrupt

```
INTERRUPT;
```

Halt

```
HALT [lit_8];
```

CLASS 6 == CONDITIONAL SEQUENCE OPERATORS

A location can be either a statement label or an <expression>. Both are relative displacements and can be forward or backward (+ or -) no greater than the maximum branch displacement. <expression> is included only for very short branches in order to eliminate the need for a label.

Bit Branch

IF [NOT] d1.'[' bit number ']' [TRUE | FALSE] THEN GO TO location;

Bit number must be 0-63; if it is greater than 31 d1+1 is used.

The maximum branch displacement is 4095.

If the [TRUE | FALSE] portion is omitted, then TRUE is assumed.

Bit Branch with Bit Manipulate

opscc	action
SET	bit is unconditionally set to 1
CLEAR	bit is unconditionally cleared to 0
COMPLEMENT	bit is unconditionally complemented

IF [NOT] d1.'[' bit number ']' [TRUE | FALSE] THEN opscc [AND] THEN GO TO location;

The maximum branch displacement is 63.

Zero Field Branch

IF [NOT] d1 <eql relation> 0 [TRUE | FALSE] THEN GO TO location;

IF [NOT] d1.D <eql relation> 0 [TRUE | FALSE] THEN GO TO location;

<eql relation> ::= = | EQL | /= | NEQ

The maximum branch displacement is 4095.

Zero Subfield Branch

IF [NOT] d2.<subfield> <eql relation> 0 [TRUE ! FALSE]
THEN GO TO location;

<subfield> - see Extract

<eql relation> - see Zero Field Branch

The maximum branch displacement is 63.

Literal Relational Branch

IF [NOT] d3 <relation> lit_7 [TRUE ! FALSE] THEN GO TO location;

IF [NOT] d3.D <relation> lit_7 [TRUE ! FALSE] THEN GO TO location;

<relation> ::= = ! EQL !
< ! LSS !
> ! GTR !
<= ! LEQ ! /> !
>= ! GEQ ! /<

The maximum brach displacement is 63.

Relational Branch

IF [NOT] d3 <relation> d2 [TRUE ! FALSE] THEN GO TO location;

IF [NOT] d3.D <relation> d2.D [TRUE ! FALSE] THEN GO TO location;

IF [NOT] d3.C <relation> d2.C [TRUE ! FALSE] THEN GO TO location;

<relation> - see Literal Relational Branch

The maximum branch displacement is 63.

None All Branch

IF [NOT] (ANY ! ALL) d3 OF d2 [TRUE ! FALSE] THEN GO TO location;

The maximum branch displacement is 63.

Bit Call

IF [NOT] d1."[" bit number "]" [TRUE ! FALSE] THEN CALL location;

Bit number must be 0-63; if it is greater than 31, d1 + 1 is used.

The maximum call displacement is 4095.

Zero Field Call

IF [NOT] d1 <eql relation> 0 [TRUE ! FALSE] THEN CALL location;

IF [NOT] d1.C <eql relation> 0 [TRUE ! FALSE] THEN CALL location;

IF [NOT] d1.D <eql relation> 0 [TRUE ! FALSE] THEN CALL location;

<eql relation> ::= = ! EQL ! /= ! NEQ

The maximum call displacement is 4095.

Relational Call

IF [NOT] d3 <relation> d2 [TRUE ! FALSE] THEN CALL location;

IF [NOT] d3.D <relation> d2.D [TRUE ! FALSE] THEN CALL location;

IF [NOT] d3.C <relation> d2.C [TRUE ! FALSE] THEN CALL location;

```
<relation>  ::=  = ! EQL !
                 < ! LSS !
                 > ! GTR !
                 <= ! LEQ ! />
                 >= ! GEQ ! /<
```

The maximum call displacement is 63.

None All Call

IF [NOT] (ANY ! ALL) d3 OF d2 (TRUE ! FALSE) THEN CALL location;

The maximum call displacement is 63.

CLASS 7 == S-MEMORY OPERATORS

Read Memory

READ lit_6 BITS USING d2 to d1 <adjust forward part>;

READ BACKWARD lit_6 BITS USING d2 to d1 <adjust backward part>;

READ [d3 BITS] USING d2 TO d1 <adjust forward part>;

READ BACKWARD [d3 BITS] USING d2 TO d1 <adjust backward part>;

<adjust forward part> ::= [ADJUST [D [UP]]
[L [UP | DOWN}]]

<adjust backward part> ::= [ADJUST [D [DOWN]]
[L [UP | DOWN}]]

The "D" (displacement) and "L" (length) phrases, if present, may be in either order.

If "d3 BITS" is omitted, then d2+1 is used.

Write Memory

WRITE lit_6 BITS USING d2 FROM d1 <adjust forward part>;

WRITE BACKWARD lit_6 BITS USING d2 FROM d1 <adjust backward part>;

WRITE [d3 BITS] USING d2 FROM d1 <adjust forward part>;

WRITE BACKWARD [d3 BITS] USING d2 FROM d1 <adjust backward part>;

<adjust forward part> - see Read Memory

<adjust backward part> - see Read Memory

If "d3 BITS" is omitted, then d2+1 is used for d3.

Swap Memory Forward Dynamic

SWAP [d3 BITS] USING d2 WITH d1 <adjust forward part>;

<adjust foward part> - see Read Memory

If "d3 BITS" is omitted, then d2+1 is used for d3.

Interrupt Port

```
INTERRUPT PORT d1;
```

CLASS 8 == I/O OPERATORS

A location can be either a statement label or an <expression>. Both are relative displacements and can be foward or backward (+ or -) no greater than the maximum branch displacement. <expression> is included only for very short branches in order to eliminate the need for a label.

Send/Receive Instructions

```
<send receive> ::=    <io action>; |

                      IF (CURRENT_IO_STATUS | BUFFERED_IO_STATUS)
                      <eql relation> lit_4 THEN GO TO location; |

                      IF (CURRENT_IO_STATUS | BUFFERED_IO_STATUS)
                      <eql relation> lit_4 THEN <io action>
                      (AND | ELSE) GO TO location;

<io action> ::=       SEND d2    |
                      SEND d2 [WITH] TERMINATE   |
                      RECEIVE d2 |
                      RECEIVE d2 [WITH] TERMINATE  |
                      RECEIVE d2 [WITH] NO STROBE  |
                      STROBE IO   |
                      STROBE IO [WITH] TERMINATE

<eql relation> ::=    = | EQL | /= | NEQ
```

The following table shows the setting of the send-receive variants for each possible instruction. If a column is blank, then either the variant is not applicable or it will be set as previously shown in the table.

instruction	sio	ndx	e	cso	trm	b/l
SEND d2			0	1	0	0
SEND d2 WITH TERMINATE			0	1	1	0
RECEIVE d2	1	0	0	1	0	0
RECEIVE d2 WITH TERMINATE	1	0	0	1	1	0
RECEIVE d2 WITH NO STROBE	0	0	0	1	0	0
STROBE IO	1	1	0	0	0	0
STROBE IO WITH TERMINATE	1	1	0	0	1	0
IF ... THEN GO TO location (=)	0	1	1	0	0	
IF ... THEN GO TO location (/=)	0	1	0	0	0	
IF CURRENT_IO_STATUS ...						0
IF BUFFERED_IO_STATUS ...						1
... <io_action> AND GO TO ...			1	0		
... <io_action> ELSE GO TO ...			0	0		

Get Lpw Accumulator

```
d1 := LPW_ACCUMULATOR [THEN INITIALIZE LPW_ACCUMULATOR];
```

Connect

```
CONNECT TO d1;
```

Disconnect

```
DISCONNECT;
```

Reconnect

```
d1 := REQUESTING LCP;
```

Get Status

```
d1 := STATUS [[AND] [WAIT] [THEN CLEAR ELOG]];
```

Clear LCP

```
CLEAR LCP;
```

Busy

```
BUSY;
```

PSEUDO AND OPTIONS INSTRUCTIONS

PSEUDO INSTRUCTIONS

Null

A null instruction (semicolon) emits no code.

Code

```
CODE lit_16 lit_16;
```

The expressions are output to the code file.

Dump

```
<dump inst> ::= DUMP (REGS !
                      d1 THROUGH d2 !
                      RC !
                      residual control register name !
                      d1 !
                      S_MEMORY d1) ;
```

Dump is a special instruction to the B1900 Emulator.

REGS - dumps everything except S-memory.

d1 through d2 - dumps all DAM words between and including d1 to d2.

RC - dumps all residual control registers.

residual control register name - dumps only that register.

S_MEMORY d1 - d1 is a canonical descriptor that describes the memory area to be dumped.

Accept

```
ACCEPT d1;
```

This a command to the B1900 Emulator which is passed on to the MCP as an accept. d1 is a canonical descriptor that describes the memory area at which to store the accept data.

Display

```
DISPLAY d1;
```

This is a command to the B1900 Emulator which is passed on to the MCP as a display. d1 is a canonical descriptor that describes the memory area to be displayed.

OPTIONS INSTRUCTIONS

Page

PAGE;

This instruction ejects the listing page.

List

[NO] LIST;

List turns the listing on or off. Default is on.

Debug

[NO] DEBUG [CODE | TOKENS | PARSE_OUTPUT | MONITOR | FIELDS] ;

Debug sets options to produce debugging output.

CODE - prints all code, not just first 64 bits/line.

TOKENS - prints each token scanned.

PARSE_OUTPUT - prints parser output.

MONITOR - monitors all procedures that were within the scope of a MONITOR.OFF statement during compilation of DML.

FIELDS - prints the name, length, and value (decimal, hex, binary) of each field for each generated micro instruction.

If no option is mentioned with the DEBUG statement, all are implied.

INDEX