64055C

TR 70-8
NOVEMBER 1970

MICROPROGRAMMING MANUAL FOR INTERPRETER BASED SYSTEMS

Burroughs Corporation
Federal and Special Systems Group
Paoli, Pa. 19301

64055C

TR 70-8
NOVEMBER 1970

MICROPROGRAMMING MANUAL FOR INTERPRETER BASED SYSTEMS

Burroughs Corporation

Federal and Special Systems Group

Paoli, Pa. 19301

ABSTRACT

The Burroughs Interpreter Based Systems emphasize two basic concepts: building block structure and soft machine architecture through microprogramming. These two concepts provide great versatility for a wide range of applications. The microprogrammable building block is an Interpreter. Other building blocks include main memory and peripheral devices accessed as ports, port select units for direct connection in a single Interpreter system, or switch interlock modules for shared connection of ports into multiple Interpreter systems.

The microprogrammer is aided in producing microprograms by a Microtranslator that translates symbolic instructions written in TRANSLANG into microinstructions and nanoinstructions for the Interpreter. These characterize an Interpreter for a specific use such as a computer, I/O processor, disk file controller, communications processor, or a combination of such uses.

This manual describes the structure of TRANSLANG through syntax, semantics, and examples. It also discusses the Microtranslator program, and the operating instructions showing how the microprogrammer can exploit the options available to him. A library of microprograms in symbolic form may be referenced during translation, giving limited macro capability. The static count of occurrences of each nanoinstruction during a translation is kept and may be added to a previously specified table in a collection of defined tables of nanoinstructions. An Editor is available for nanotable modification.

This manual includes descriptions of register state changes resulting from executing the microinstructions and nanoinstructions. Interpreter controls and timing are explained. Examples of instructions are given that illustrate coding techniques and conventions. The manual also discusses external operations with main memory and peripheral devices, and the coordination and safe control of multiple Interpreters via a switch interlock and global condition bits. Microprogramming reference cards are provided.

CONTENTS

ILLUSTRATIONS

TABLES

1. INTRODUCTION

The Burroughs Interpreter Based Systems emphasize two basic concepts: building block structure and soft machine architecture through microprogramming. These two concepts provide great versatility for a wide range of applications. The microprogrammable building block is an Interpreter. Other building blocks include main memory and peripheral devices accessed as ports, port select units for direct connection in a single Interpreter system, or switch interlock modules for shared connection of ports into multiple Interpreter systems.

Figure 1-1 is a summary diagram of an Interpreter. The five functional parts are tabulated below.

MCU	Memory Control Unit	Registers for memory addressing.
CU	Control Unit	Registers for conditional control and logic commands.
LU	Logic Unit	Width 1 to 8 bytes; data registers, adder, any shift amount, multiple concurrent processing.
MPM	Micro Program Memory	Microprogram sequences: some words have literals, others have nano addresses.
NM	Nano Memory	Specific controls created for the microprogrammer.

The Interpreter is described in detail in Appendix D. The method for interaction between an Interpreter and other building blocks is also described there.

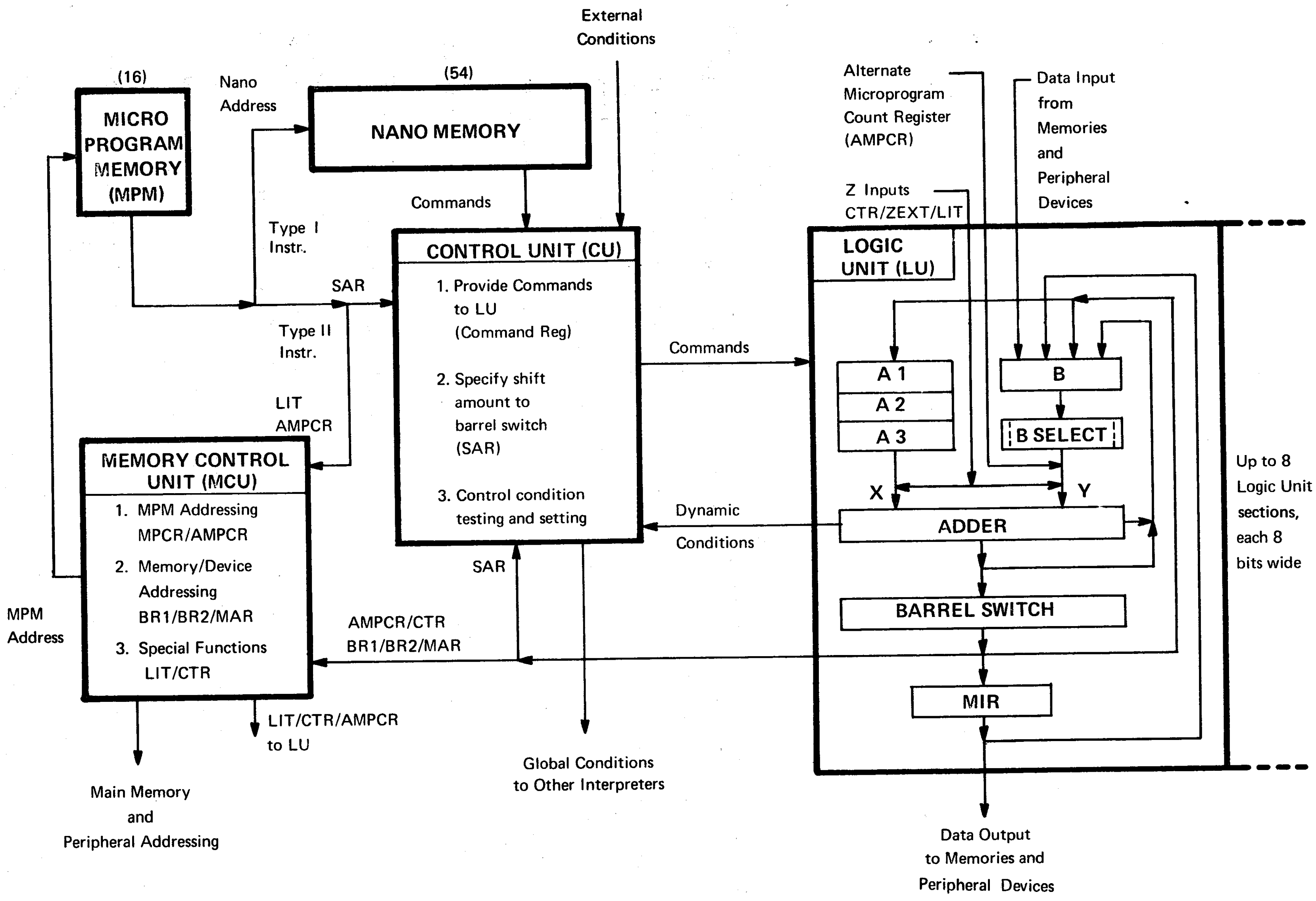

(16)
MICRO PROGRAM MEMORY (MPM)
Nano Address
(54)
NANO MEMORY
External Conditions
Commands
Type I Instr.
SAR
Type II Instr.
LIT AMPCR
CONTROL UNIT (CU)
1. Provide Commands to LU (Command Reg)
2. Specify shift amount to barrel switch (SAR)
3. Control condition testing and setting
MEMORY CONTROL UNIT (MCU)
1. MPM Addressing MPCR/AMPCR
2. Memory/Device Addressing BR1/BR2/MAR
3. Special Functions LIT/CTR
MPM Address
SAR
AMPCR/CTR BR1/BR2/MAR
LIT/CTR/AMPCR to LU
Main Memory and Peripheral Addressing
Global Conditions to Other Interpreters
Commands
Dynamic Conditions
Alternate Microprogram Count Register (AMPCR)
Z Inputs CTR/ZEXT/LIT
Data Input from Memories and Peripheral Devices
LOGIC UNIT (LU)
A 1
A 2
A 3
B
B SELECT
X
Y
ADDER
BARREL SWITCH
MIR
Up to 8 Logic Unit sections, each 8 bits wide
Data Output to Memories and Peripheral Devices

Figure 1-1. Interpreter Block Diagram

This report describes in detail (1) the structure of the language in which microprograms are written and (2) the program which translates instructions written in this language into microinstructions and nanoinstructions that are subsequently placed into the microprogram memory and nanoinstruction memory respectively of an Interpreter.

The following definitions of terms are presented to assist in the understanding of these descriptions:

H-language	High level language, alternately referred to as compiler language or source language, and includes languages such as ALGOL, COBOL, FORTRAN, APL and SNOBOL4.
S-language	This language is equivalent to the assembly (or object) language in conventional machines.
S-instruction	A discrete instruction equivalent to a machine instruction in a conventional machine. An S-instruction may be simple or complex. In some applications an S-instruction may be much more complex than a conventional machine instruction.
S-program	A set of S-instructions which is given a name.
S-memory	The memory in which S-programs are stored for execution. This is generally considered main memory or data memory.
M-language	Microlanguage, the language that is used for developing microprograms.
M-instruction	Microinstruction, a discrete instruction that performs a set of basic functions in parallel. One M-instruction occurs every clock. An M-instruction may contain either a constant (type II) or an address of an N-instruction (type I).
M-program	Microprogram, a set of M-instructions which is given a name.
M-memory	Microprogram memory, the memory in which M-programs reside. Alternatively named MPM.
N-instruction	Nanoinstruction, a set of controls addressed by an M-instruction. These controls cause parallel logical actions.
N-memory	Nanomemory, a decoder or memory which contains N-instructions.

The general function and interrelationships of these terms are indicated by the following:

A commercial user of an Interpreter Based System would write a program in an H-language (e.g., ALGOL) which would then be compiled into an S-language program, which would then be stored in an S-memory for execution. Each S-instruction is executed through the use of an M-program.

The M-program provides basically two functions:

1. Interpretation of the S-instruction including the fetching of the S-instruction. This depends primarily on the format(s) of the S-instruction.

2. Execution of the indicated S-instruction operation as defined by a set of M-instructions for that operation.

The N-instructions are addressed by M-instructions. Usually each N-instruction is addressed by many M-instructions.

Microprogramming is part of the design process for Interpreter Based Systems. It involves a combination of programming concepts (assignment of value to variables, conditional execution, looping, hierarchical design) and logic concepts (Boolean logic functions, time sequencing and concurrent execution subject to partial ordering of events).

Device controller design is an important microprogramming task. The objective is to provide all logic functions for device control through microprogramming. By this means design flexibility is maintained, optional features are easily incorporated, and many special hardware controllers (typically one or more per device) are replaced by common hardware specialized through microprograms. The opportunity for shared use of one Interpreter among several devices, dynamically being interleaved as needed, represents a significant potential for system simplification.

Emulation of an existing processor and/or its I/O channels is another microprogramming task. The objective is to run programs prepared for the emulated machine. To do this, the emulated machine registers are mapped into S-memory and/or the actual registers of an Interpreter. The operation codes accessible to programs become S-instructions. Other S-instructions may be added for I/O commands, depending on how much of the I/O processing is absorbed by the Interpreter.

Proper processor design for interpreting or executing constructs occurring in a particular higher level language is another microprogramming task. The design of the S-language for such processors is open, and opportunities for tradeoffs exist between primitive S-instructions versus S-language macros or procedures. One important processor design is for translating source programs. Another is for producing translator writing systems. A third is for processing complex data structures on one or more Interpreters.

The Microtranslator is a tool to aid the designer in producing M-programs for the Burroughs Interpreter Based Systems. The M-programs in the M-memory associated with a particular Interpreter determine its use. The M-programs are composed mainly of statements of M-language reserved words. These reserved words allow for a convenient description of hardware functions. Appendix B contains a complete glossary of the reserved words.

The Microtranslator translates the symbolic M-instructions into an output to be loaded into the M-memory. A corresponding table of N-instructions is developed which forms the content of the nanomemory. If a table of N-instructions which is on a library of tables is chosen as input, this table may be used as the basis of the M-program and may be expanded as needed by the program.

A listing of the original symbolic M-instructions as well as an edited representation of each N-instruction may be requested. Flags will indicate errors in the M-program that are detected by the translator. Errors detected because of an over-expanded set of N-instructions used in a program will also be noted.

Statistics will be collected on the use of each N-instruction during a translation. This information will be supplied to the user upon request. It will give a list of all the N-instructions and indicate the static count of the occurrences of each of them during the translation process.

M-programs in symbolic form are being developed. These may be referenced during translation and like macros become part of the M-program. The INSERT operation with the name of the function will cause this code to be inserted into the incoming string.

The Microtranslator is written in ALGOL for the B 5500. It is written modularly with each function set up as a procedure call. The program is divided into two major sections (see Figure 1-2). The first section parses the language and produces the input listing and the N-instructions necessary. The second section reviews the generated M-instructions, inserts label address(es), and produces the microinstruction and nanoinstruction outputs.

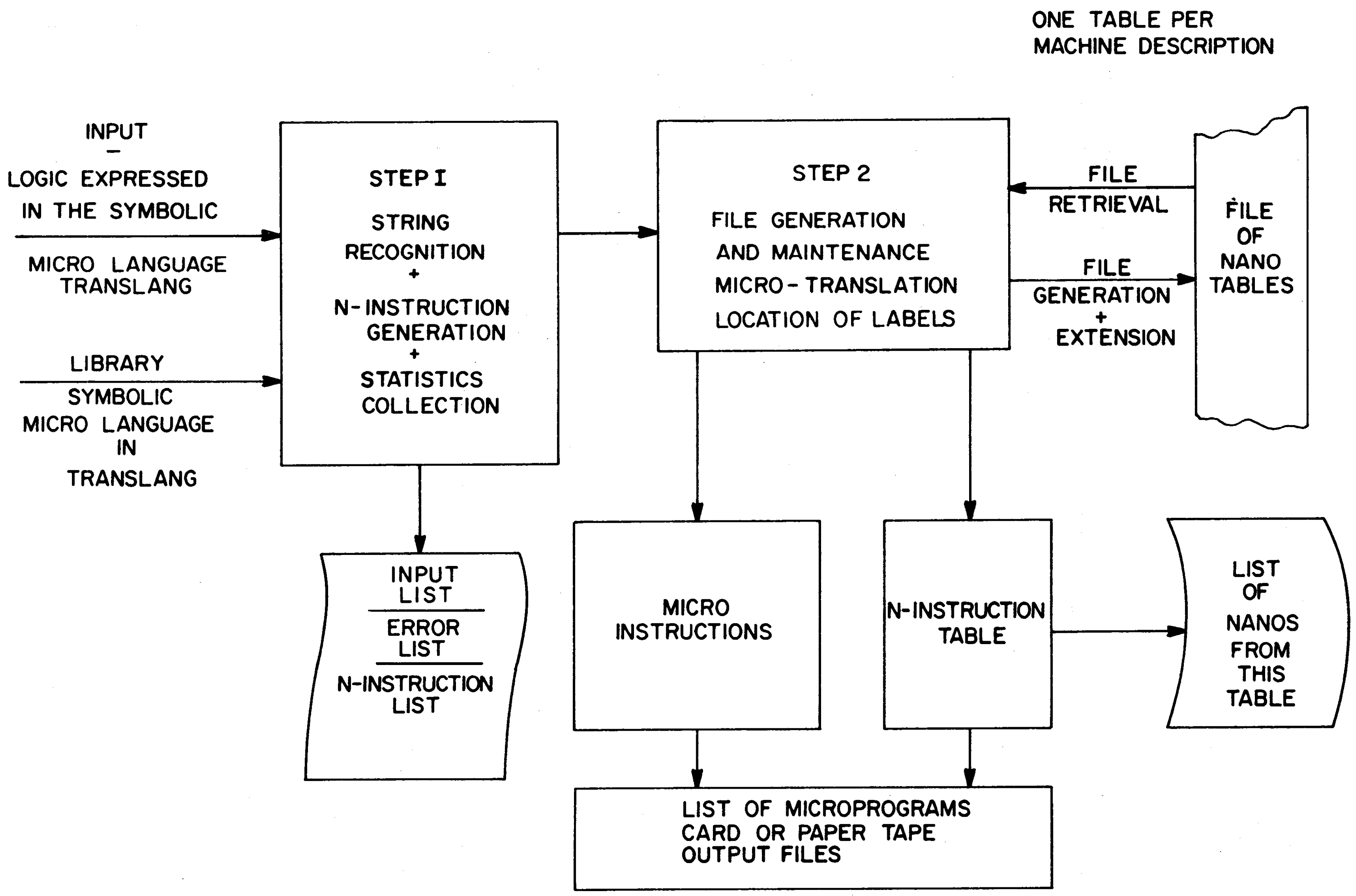

ONE TABLE PER MACHINE DESCRIPTION
INPUT — LOGIC EXPRESSED IN THE SYMBOLIC MICRO LANGUAGE TRANSLANG
LIBRARY SYMBOLIC MICRO LANGUAGE IN TRANSLANG
STEP I
STRING RECOGNITION + N-INSTRUCTION GENERATION + STATISTICS COLLECTION
STEP 2
FILE GENERATION AND MAINTENANCE MICRO-TRANSLATION LOCATION OF LABELS
FILE RETRIEVAL
FILE GENERATION + EXTENSION
FILE OF NANO TABLES
INPUT LIST
ERROR LIST
N-INSTRUCTION LIST
MICRO INSTRUCTIONS
N-INSTRUCTION TABLE
LIST OF NANOS FROM THIS TABLE
LIST OF MICROPROGRAMS CARD OR PAPER TAPE OUTPUT FILES

Figure 1-2. Microtranslator Block Diagram

2. TRANSLATOR DESCRIPTION

The translator was developed as a tool to aid the system designer in developing microprograms in a language using labels and words rather than bits and numbers. It allows him to discover what size table and type of N-instructions have become part of his N-instruction sets and how he might use these sets more efficiently.

Theoretically, the Interpreter may have 2^{54} different N-instructions. However, only a small subset of these will be used by any Interpreter. The translator will indicate the most applicable subset for a particular implementation of an S-language.

As more microprograms are developed a pattern of frequently used subroutines or programs will be developed. These may be placed on a library and used by the translator to be inserted into programs as called. The library programs may reference other library programs thus developing a nested structure of programs. Programs may reference labels located at any level more global than themselves.

The language, TRANSLANG, for the translator has used ALGOL as a model. However, since the system designer must have complete control of all of the Interpreter functions, almost all of the language is composed of reserved words. Reserved words have very specific meaning to the translator and cause specific N-instructions to be developed. TRANSLANG is free form and each instruction may be written in almost any order; however, one and only one instruction may appear on a line or card (72 characters).

When a program is being parsed by the translator, the microprogrammer has the option of listing the program and/or listing the N-instructions constructed or neither of these. The syntax errors will always be printed on the teletype. The programmer has the choice of listing his program on the teletype or on the printer. A complete set of error messages is given in Appendix C.

As the program is being parsed, a table is developed of the generated N-instructions as well as an output file for the microinstructions. The table has one entry per N-instruction and will record each time this instruction was used. Thus, a statistical record is maintained of the number of times an N-instruction has been used during processing. The output file is an M-instruction record of the program a line at a time.

The user is asked what kind of table (description of a decoder) he wants to use for the N-instructions. His options include using an old table or constructing his own new table. If the user wants to use an option requiring a previously formed table, its name is requested. Otherwise a new table is constructed for him.

If an old table is used for a translation, its statistics will be updated by adding the new statistics to the ones already developed. If a new table is developed, its statistics will be only those of the present translation. At the completion of the program, the statistics may be displayed on request. The newly constructed table or the updated table may be saved for use at a future time.

The final output will be a card or teletype paper tape file containing the M-instructions and the N-instructions which define it. The paper tape may be used as input to the APL model, and the punch cards as input to the "D" machine. The M-programs are either pointers to N-instruction addresses or simple literal assignment statements.

All the tables of N-instructions developed are located in a library file of nano-tables. The file has a directory indicating the name and location of each table. These tables may be edited when necessary using the EDITOR/ESOADO program.

3. STRUCTURE OF TRANSLANG

The TRANSlator LANGuage (TRANSLANG) is an assembler for Interpreter M-programs. The complete syntax of TRANSLANG is given in Appendix A. It employs a vocabulary of reserved words and symbols used to develop an M-program and their corresponding table of N-instructions. Reserved words and symbols are grouped as defined in this manual to form microinstructions and programs. The reserved words are summarized in Appendix B.

Each TRANSLANG line corresponds to one microinstruction which is the set of Interpreter functions performed in parallel at each machine clock. The constructs include iterative mechanisms, I/O, Boolean, logical and computational operations, control transfers and assignment functions. In order to provide control points for transfer operations, each instruction may be labeled with a symbolic M-address.

The INSERT function has been included to allow for the use of a macro library of previously debugged M-programs.

The semantic meaning of the constructs is expanded in Appendix D. The control actions and their timing are described, keyed to the TRANSLANG reserved words.

CONVENTIONS IN LANGUAGE DESCRIPTION

Backus-Naur form (BNF) is used as the metalanguage to define the syntax of TRANSLANG. The following BNF symbols are used:

1. ⟨ ⟩ Left and Right Broken brackets are used to bracket the names of syntactic categories.

2. ::= Colon Colon Equal means "is defined as" and separates the name of the syntactic category from its definition.

3. | Bar separates alternative definitions of a syntactic category.

4. { } Left and Right Braces enclose an English language description of a syntactic unit.

5. Juxtaposition of metalanguage symbols, symbols, or reserved words is used to indicate catenation.

Any character or symbol in a metalanguage formula which is not a metalanguage symbol and is not enclosed within matching braces or broken brackets, denotes itself.

BASIC ELEMENTS

```
⟨Letter⟩ ::=                A |B |C |D |E |F |G |H |I |J |K |L |M |N |O |
                            P |Q |R |S |T |U |V |W |X |Y |Z

⟨Digit⟩ ::=                 0 |1 |2 |3 |4 |5 |6 |7 |8 |9

⟨Hex Digit⟩ ::=             ⟨Digit⟩ |A |B |C |D |E |F

⟨Symbol⟩ ::=                , | ; | + | - | : | = | % | " | ( | ) | *

⟨Single Space⟩ ::=          { One horizontal blank position }

⟨Space⟩ ::=                 ⟨Single Space⟩ | ⟨Space⟩ ⟨Single Space⟩

⟨Assignment Op⟩ ::=         =:

⟨Character⟩ ::=             ⟨Letter⟩ | ⟨Digit⟩ | ⟨Single Space⟩ | ⟨Symbol⟩

⟨Comment Character⟩ ::=     ⟨Character⟩ |. | # | & | $ | [ | ] | \ | /

⟨Empty⟩ ::=                 { The null string of characters }

⟨Comment⟩ ::=               { Any sequence of ⟨Comment Characters⟩
                            except ";" };
```

Semantics

TRANSLANG uses a character set of 56 characters including ⟨ single space⟩ , of which 8 are only used in comments. All letters are upper case.

Spaces – No space may appear between the letters of a reserved word or within an ⟨ Assignment Op⟩ ; otherwise, they will be interpreted as two or more elements. Spaces are used as a delimiter to separate reserved words, labels, or integers. Spaces may appear between any two basic components without affecting their meaning, where basic components indicate reserved words, symbols, or labels.

Parentheses – The parentheses are treated as spaces. They are used for the convenience of the microprogrammer to make code more readable. (E.g. instruction elements which are irrelevant to the current instruction but are used only to allow shared use of a nanoinstruction by several M-instructions.) <u>Parentheses do not imply precedence</u>.

Comments – In order to include explanatory material at various points in a program, two conventions exist as defined.

1. COMMENT { any sequence of comment characters except ";" } ;

   The comment statement acts the same as a ";" and may appear anywhere a ";" may occur if within a line of program. As multi-line documentation the ";" terminator indicates that the micro-translator should resume processing code. Always follow a comment statement with a ";".

2. % { any sequence of comment characters until end of line }

   All comment characters after the % in a line of program are ignored by the microtranslator.

Comments are for documentation purposes only. They appear only in the source file, are significant only in listings and do not affect the machine language generated.

The following printing characters are used for control purposes between the B 5500 and teletypewriter. They should not be used in comments

< | > | ? | ' | ! | ↑

The line terminator for all lines from the teletypewriter is the end of line character:

←

This control character is equivalent to the end of a card if card input is used to build a source file. It is not part of the character set processed by the micro-translator.

LITERAL ASSIGNMENT INSTRUCTION

```
⟨Literal Assignment⟩ ::= ⟨Literal⟩ =: AMPCR | ⟨Literal⟩ =: SAR |
                          ⟨Literal⟩ =: SAR; ⟨Literal⟩ =: ⟨Lit⟩ |
                          ⟨Literal⟩ =: ⟨Lit⟩ ; ⟨Literal⟩ =: SAR | ⟨Literal⟩ =:⟨Lit⟩

⟨Literal⟩ ::= ⟨Integer⟩ | COMP ⟨Integer⟩ |
              ⟨Label⟩ | ⟨Label⟩ -1 | "Comment Character"

⟨Integer⟩ ::= ⟨Digit⟩ | ⟨Digit⟩ ⟨Integer⟩
⟨Label⟩ ::= ⟨Letter⟩ | ⟨Label⟩ ⟨Letter⟩ | ⟨Label⟩ ⟨Digit⟩
⟨Lit⟩ ::= LIT | SLIT
```

Semantics

A ⟨Literal Assignment⟩ becomes a type II M-instruction for an Interpreter. This M-instruction contains the literal value(s) and specifies the receiving register(s).

		Width, bits
AMPCR	Alternate Micro Program Count Register	12
SAR	Shift Amount Register	Least integer not less than Log_2 (logic unit bit width)
LIT	Literal Register	8

The registers may be individually loaded or both the SAR and the LIT may be loaded in the same M-instruction. Usually the latter may be used in place of separate instructions to individually load LIT and to load SAR.

An ⟨Integer⟩ should be non-negative, in the range of the intended receiving register(s). COMP ⟨Integer⟩, if the receiving register is LIT or AMPCR takes the ones complement of the ⟨Integer⟩, then takes the number of bits indicated above into the receiving register. An ⟨Integer⟩ for the SAR, or LIT via SLIT receives an encoding appropriate to the logic unit width. COMP ⟨Integer⟩, if the receiving register is SAR or LIT via SLIT creates the appropriate word length complement (see Table 3-1) prior to encoding. The encoded value is used in the receiving field. For a SLIT, leading zeros are entered into the more significant end.

The successor of a ⟨Literal Assignment⟩ is implicitly STEP.

Table 3-1. Shift Amounts and Their Complements (for Logic Unit Widths of 64, 56, 48, 40, 32, 24, 16, and 8 Bits)

Logic Unit Width

64 Bits

	SAR		"COMP."	
0	000	000	000	000
1		001	111	111
2		010		110
3		011		101
4		100		100
5		101		011
6		110		010
7		111		001
8	001	*		*
9			110	
10		RA		
11		EB		
12		PO		
13		EV		
14		AE		
15		T		
16	010	*		*
17			101	
18				
19				
20				
21				
22				
23				
24	011	*		*
25			100	
26				
27				
28				
29				
30				
31				
32	100	*		*
33			011	
34				
35				
36				
37				
38				
39				
40	101	*		*
41			010	
42				
43				
44				
45				
46				
47				
48	110	*		*
49			001	
50				
51				
52				
53				
54				
55				
56	111	*		*
57			000	
58				
59				
60				
61				
62				
63				

56 Bits

	SAR		"COMP."	
0	000	000	000	000
1		001	111	110
2		010		101
3		011		100
4		100		011
5		101		010
6		110		001
7	001			
8		*	110	*
9				
10				
11				
12				
13				
14	010			
15		*	101	*
16				
17				
18				
19				
20				
21	011	*		*
22			100	
23				
24				
25				
26				
27				
28	100	*		*
29			011	
30				
31				
32				
33				
34				
35	101	*		*
36			010	
37				
38				
39				
40				
41				
42	110	*		*
43			001	
44				
45				
46				
47				
48				
49	111	*		*
50			000	
51				
52				
53				
54				
55				

48 Bits

	SAR		"COMP."	
0	000	000	000	000
1		001	111	101
2		010		100
3		011		011
4		100		010
5		101		001
6	001	*		*
7			110	
8				
9				
10				
11				
12	010	*		*
13			101	
14				
15				
16				
17				
18	011	*		*
19			100	
20				
21				
22				
23				
24	100	*		*
25			011	
26				
27				
28				
29				
30	101	*		*
31			010	
32				
33				
34				
35				
36	110	*		*
37			001	
38				
39				
40				
41				
42	111	*		*
43			000	
44				
45				
46				
47				

40 Bits

	SAR		"COMP."	
0	000	000	000	000
1		001	111	100
2		010		011
3		011		010
4		100		001
5	001			
6		*	110	*
7				
8				
9				
10	010			
11		*	101	*
12				
13				
14				
15	011	*		*
16			100	
17				
18				
19				
20	100	*		*
21			011	
22				
23				
24				
25	101	*		*
26			010	
27				
28				
29				
30	110	*		*
31			001	
32				
33				
34				
35	111	*		*
36			000	
37				
38				
39				

32 Bits

	SAR		"COMP."	
0	000X	00	000X	00
1		01	111X	11
2		10		10
3		11		01
4	001X	00		00
5		01	110X	11
6		10		10
7		11		01
8	010X	00		00
9		01	101X	11
10		10		10
11		11		01
12	011X	00		00
13		01	100X	11
14		10		10
15		11		01
16	100X	00		00
17		01	011X	11
18		10		10
19		11		01
20	101X	00		00
21		01	010X	11
22		10		10
23		11		01
24	110X	00		00
25		01	001X	11
26		10		10
27		11		01
28	111X	00		00
29		01	000X	11
30		10		10
31		11		01

24 Bits

	SAR		"COMP."	
0	000X	00	000X	00
1		01	111X	10
2		10		01
3	001X	00		00
4		01	110X	10
5		10		01
6	010X	00		00
7		01	101X	10
8		10		01
9	011X	00		00
10		01	100X	10
11		10		01
12	100X	00		00
13		01	011X	10
14		10		01
15	101X	00		00
16		01	010X	10
17		10		01
18	110X	00		00
19		01	001X	10
20		10		01
21	111X	00		00
22		01	000X	10
23		10	000X	01

16 Bits

	SAR		"COMP."	
0	000	XX0	000	XX0
1	000	1	111	1
2	001	0	111	0
3	001	1	110	1
4	010	0	110	0
5	010	1	101	1
6	011	0	101	0
7	011	1	100	1
8	100	0	100	0
9	100	1	011	1
10	101	0	011	0
11	101	1	010	1
12	110	0	010	0
13	110	1	001	1
14	111	0	001	0
15	111	1	000	1

8 Bits

	SAR		"COMP."	
0	000	XXX	000	XXX
1	001		111	
2	010		110	
3	011		101	
4	100		100	
5	101		011	
6	110		010	
7	111		001	

Labels used in a program may be chosen freely except for the reserved words of TRANSLANG. The reserved words are given in Appendix B. A label must start with a letter which can be followed by any combination of letters or digits. No spaces or symbols may appear in a label. A label can be as little as one letter and as long as 15 letters and digits. The same label may not be used to locate more than one instruction in the same program. See the INSERT function description subsequently for allowable nesting of labels when subprograms are inserted. The normal use of a label with a ⟨ Literal Assignment⟩ is as ⟨ Label⟩ -1 since control transfers occur to the indicated location +1 (or +2).

A quoted ⟨ Comment Character⟩ may be entered as a literal and will be converted in the right-justified B 5500 internal 6-bit representation with leading zeros. This is of limited use since the input set of ⟨ Comment Characters⟩ is incomplete, containing only 56 of the possible 64 B 5500 characters. The internal representations may be found in the B 5500 Compatible Algol Reference Manual, Burroughs Form 1038643 9-68.

<u>Examples</u>

```
5=: SAR                        % converted for proper logic unit width
COMP 8 =: SAR; 13=: SLIT       % in one M-instruction
COMP 0 =: LIT                  % same as 255=:LIT
START =: AMPCR                 % JUMP to Start +1; RETN to START + 2
LOOP-1=: AMPCR                 % JUMP to Loop; RETN to LOOP + 1
```

N INSTRUCTION

⟨ N Instruction⟩ ::= ⟨ Unconditional Part⟩ ⟨ Conditional Part⟩

⟨ Unconditional Part⟩ ::= ⟨ Component List⟩

⟨ Component List⟩ ::= ⟨ Component⟩ | ⟨ Component List⟩ ; ⟨ Component⟩ |
⟨ Empty⟩

⟨ Component⟩ ::= ⟨ Ext Op⟩ | ⟨ Logic Op⟩ | ⟨ Successor⟩

⟨ Conditional Part⟩ ::= ⟨ If Clause⟩ ⟨ Cond Comp List⟩ ⟨Else Clause⟩ | ⟨ If Clause⟩ |
⟨ When Clause⟩ ⟨ Cond Comp List⟩ | ⟨ Empty⟩

⟨ Cond Comp List⟩ ::= THEN ⟨ Component List⟩

<u>Semantics</u>

An ⟨ N instruction⟩ becomes a Type I M-instruction containing an address to a nano instruction. If an identical N-instruction already exists the M-address will point to the single copy of the nano instruction. If the N-instruction is new, the address will be to the next unused nano address. Into this nano location are entered the operations indicated in the ⟨N Instruction⟩ .

Restrictions

1. At most one ⟨ Ext Op⟩ - either unconditional or conditional.
2. At most one ⟨ Logic Op⟩ - either unconditional or conditional
3. At most either one unconditional successor, or one conditional successor in the ⟨ Cond Comp List⟩ and possibly one in an ⟨Else Clause⟩.

The ⟨Unconditional Part⟩ is always executed. In the ⟨ Conditional Part⟩ if the condition resulting from the ⟨ If Clause⟩ or ⟨ When Clause⟩ is true then the components in the ⟨ Cond Comp List⟩ are executed, otherwise only the ⟨ Else Clause⟩ is executed.

Examples (to be subsequently explained)

Unconditional Part, Component List:

```
SET GC1

MR2

RESET GC, DR2

A2 AND B001 =: A1

A1 + B IC R =: A2, BEX, LMAR

JUMP

DL1; 0=: A2; SKIP
```

Conditional Part:

```
IF AOV THEN A1 + 1 =: A1 ELSE SKIP

IF NOT ABT THEN SET LC2; SKIP ELSE SAVE

WHEN RDC THEN MR2; BEX, INC
```

N Instruction:

```
WHEN RDC THEN BEX

SET LC1   IF SAI THEN LIT-B-1=:A3, BBE
```

CONDITION

⟨ If Clause⟩ ::=	IF ⟨ Condition⟩
⟨ Condition⟩ ::=	⟨ Not⟩ ⟨ Cond⟩
⟨ Not⟩ ::=	NOT | ⟨ Empty⟩
⟨ Cond⟩ ::=	LST | MST | AOV | ABT | COV | SAI | RDC | EX1 | EX2 | EX3 | ⟨Cond Adjust Bit⟩
⟨ When Clause⟩ ::=	WHEN ⟨ Condition⟩
⟨ Else Clause⟩ ::=	ELSE ⟨ Successor⟩ | ⟨ Empty⟩

Semantics

Each N instruction performs a test on the Boolean value of one ⟨ Cond⟩ or its complement. The Boolean value of the result is ⟨ Condition⟩ . If this value is true the ⟨ Cond Comp List⟩ is executed and the successor from this list is used to determine the next M-instruction. Otherwise the successor in the ⟨ Else Clause⟩ is used to determine the next M-instruction address. See the subsequent discussion of successor.

A ⟨ When Clause⟩ is a synonym for an ⟨ If Clause⟩ with the same ⟨ Condition⟩ and an ⟨ Else Clause⟩ of ELSE WAIT. An empty ⟨ Else Clause⟩ is equivalent to ELSE STEP.

In the absence of an ⟨ If Clause⟩ or ⟨ When Clause⟩ an implied ⟨ If Clause⟩ of IF NOT MST is inserted. This changes no condition bit. It does cause unconditional initiation of a ⟨ Logic Op⟩ and hence completion of the prior ⟨ Logic Op⟩.

With the exception of the two global condition bits,, testing a condition bit causes the bit to be reset. The least and most significant bits out of the adder, the adder overflow, and the adder bit transmit are levels and not condition bits. The conditions that may be tested (Table 3-2) are the following:

SAI	Switch Interlock Accepts Information. Following memory or device operation, indicates that connection to the addressed memory or device is completed through the switch interlock.
RDC	Read Complete, or Requested Device Completes Following memory read or device read by request, indicates that data will be available for entry to B in the next clock. Following device write by request, indicates completion.

COV	Counter Overflow Following or concurrent with increment counter INC, indicates counter is overflowing or has already overflowed from all ones (255) to all zeros.
LC1	Local Condition 1 Tests and resets local Boolean condition bit LC1.
LC2 LC3	Local Conditions 2 and 3 Same as LC1
GC1 GC2	Global Conditions 1 and 2 Tests but does not reset global condition bit GC1. See the description of the set and reset operation for further explanation of global condition bits.
INT	Inter-Interpreter Interrupt Tests and resets the local copy of the inter-Interpreter interrupt.
EX1 EX2 EX3	External Conditions 1, 2, and 3 Test and reset interrupts (usually the OR of interrupts from several devices) from external devices (local copy).

The following four logic unit conditions are dynamic and indicate the result output from the adder in the phase 3 commands from the previous instruction which had logic unit operation, and using the current values of the adder inputs. These conditions are sustained until execution of another instruction involving the logic unit, and may be tested by that instruction. A type II instruction loading the LIT or AMPCR may change the value of an adder input selected in the ⟨X Select⟩ or ⟨Y Select⟩ and hence change the value of any of these conditions.

AOV	Adder Overflow Results from an adder operation with carry out of the most significant end of the adder (see above).
LST	Least significant State of the least significant bit of the adder output (see above).

Table 3-2. Set and Reset of Conditions

BIT	SET	RESET
AOV	Dynamic Adder State - (Overflow)	#
ABT	Dynamic Adder State - (Adder bit transmit)	#
LST	Dynamic Adder State - (Least Significant Bit of Adder Output)	#
MST	Dynamic Adder State - (Most Significant Bit of Adder Output)	#
COV	Overflow when Counter is Incremented	Reset by loading counter or by testing
GC1	SET GC1 providing no other Interpreter has GC1 set, or no higher priority Interpreter is concurrently doing SET GC1	RESET GC
GC2	SET GC2 similar to GC1	RESET GC
INT	Set INT executed in any Interpreter	Reset by testing*
LC1	SET LC1	Reset by testing
LC2	SET LC2	Reset by testing
LC3	SET LC3	Reset by testing
RDC	By memory at completion of memory or device read	Reset by testing
SAI	By switch interlock or PSU when data received from MAR and MIR	Reset by testing
EX1	By requests from devices	Reset by testing*
EX2	By requests from devices	Reset by testing*
EX3	By requests from devices	Reset by testing*

#Recomputed each clock time
*In local Interpreter only

MST	Most significant State of the most significant bit of the adder output (see above).
ABT	Adder bit transmit This condition is true (one) if and only if the adder output is all ones (see above).

Examples

IF NOT LC1

WHEN SAI

ELSE CALL

EXTERNAL OPERATIONS

```
⟨Ext Op⟩ ::=         ⟨Mem Dev Op⟩ | ⟨Set Op⟩ |
                     ⟨Mem Dev Op⟩ , ⟨Set Op⟩ |
                     ⟨Set Op⟩ , ⟨Mem Dev Op⟩ | ⟨Empty⟩

⟨Mem Dev Op⟩ ::= MR1 | MR2 | MW1 | MW2 | DL1 | DL2 | DR1 | DR2 |
                 DW1 | DW2 | DU1 | DU2 | ASR | ASE

⟨Set Op⟩ ::=         SET ⟨Cond Adjust Bit⟩ | RESET GC

⟨Cond Adjust Bit⟩ ::= INT | LC1 | LC2 | LC3 | GC1 | GC2
```

Semantics

The external operations are composed of ⟨N Instruction⟩ functions which if explicity present affect the operations external to the Interpreter logic. An ⟨Ext Op⟩ may be specified as either conditional or unconditional as it appears in at most one of the ⟨Unconditional Part⟩ or ⟨Conditional Part⟩.

The memory or device operations ⟨Mem Dev Op⟩ are used to transfer data between the Interpreter and main S-memory or a peripheral device. Address source registers for those operations are the catenation of either BR1 or BR2 with MAR, indicated respectively by MAR1 or MAR2. The MAR part is less significant. The memory or device operations are described in detail in Appendix D. Coding conventions are suggested in Appendix F. The explicit memory or device operations follow. If none is specified then any memory or device operation in progress is continued and no new operation is initiated. Address or MIR change may terminate the operation.

MR1	Memory Read 1 Read data from S-memory address specified in MAR1
MR2	Memory Read 2 Read data from S-memory address specified in MAR2
MW1	Memory Write 1 Write data from MIR to S-memory address specified in MAR1
MW2	Memory Write 2 Write data from MIR to S-memory address specified in MAR2
ASR	Status Request for highest priority locked device*
ASE	Status Request for highest priority unlocked device*
DL1	Device Lock 1 Request * Reserve the device or memory module named in MAR1 for use by this Interpreter.
DL2	Device Lock 2 Request* Reserve the device or memory module named in MAR2 for use by this Interpreter.
DR1	Device Read 1 Read data from device named in MAR1
DR2	Device Read 2 Read data from device named in MAR2
DW1	Device Write 1 Write data from MIR to the device named in MAR1
DW2	Device Write 2 Write data from MIR to the device named in MAR2
DU1	Device Unlock 1 Release the locked device named in MAR1
DU2	Device Unlock 2 Release the locked device named in MAR2

*Systems with switch interlock use DL1 and DL2; systems with port select unit use ASR and ASE

The set and reset operations are used to set and reset condition bits. The inter-Interpreter interrupt INT, is used for communication (to signal) all Interpreters of a multiprocessing system. The global conditions, GC1 and GC2, are used as Boolean semaphores to guarantee mutual exclusion for critical sections of m-programs and to prevent simultaneous access to shared data. The local condition bits are Boolean variables local to each Interpreter. The INT and local condition bits are reset (within the local Interpreter only) by testing. The explicit test and reset operations follow. If no ⟨Set Op⟩ is present, none is done.

SET INT	Interrupt Interpreters
	Causes the interrupt bit to be set in all Interpreters. Each Interpreter resets its own bit by testing it. Setting occurs after testing should both occur in the same N-instruction.
SET LC1	Set the first local condition bit
	Causes the setting of the LC1 bit in the condition register. Setting occurs after testing should both occur in the same N-instruction, Both set and test of LC1 occur in phase 1.
SET LC2	Set the second local condition bit
	Same as for LC1 replacing LC1 by LC2.
SET LC3	Set third local condition bit
	Same as for LC1 replacing LC1 by LC3.
SET GC1	Set first global condition bit request
	Requests that the GC1 bit in the requesting Interpreter be set if a GC1 bit is not already set in another Interpreter or is not being set simultaneously by a higher priority Interpreter. For all Interpreters in a multiprocessing system at most one will have GC1 set. GC1 is set at the end of the phase after phase 1 if no conflict occurs. A request lasts for one clock.
SET GC2	Set second global condition bit request
	Same as for GC1 replacing GC1 by GC2.
RESET GC	Resets the global condition bits
	Causes GC1 and GC2 to be reset in the issuing Interpreter.

The default ⟨Ext Op⟩ is unconditionally continue the prior ⟨Ext Op⟩ unless already complete.

Examples

MR2

SET LC1

DR2, RESET GC

LOGIC OPERATIONS

```
⟨Logic Op⟩ ::=      ⟨Adder Op⟩ ⟨Inhibit Carry⟩ ⟨Shift Op⟩ ⟨Destination List⟩

⟨Adder Op⟩ ::=      0 | 1 | ⟨Monadic⟩ | ⟨Dyadic⟩ | ⟨Empty⟩

⟨Monadic⟩ ::=       ⟨Not⟩ ⟨X Select⟩ | ⟨N Y Select⟩ |
                    DEC ⟨X Select⟩

⟨Not⟩ ::=           NOT | ⟨Empty⟩

⟨Dyadic⟩ ::=        ⟨X Select⟩ ⟨Commut Op⟩ ⟨ N Y Select ⟩
                    ⟨Y Select⟩ ⟨Commut Op⟩ ⟨X Select⟩ |
                    ⟨X Select⟩ ⟨Non Commut Op⟩ ⟨N Y Select⟩
                    ⟨X Select⟩ +⟨ N Y Select ⟩ + 1
                    ⟨X Select⟩ -⟨ N Y Select ⟩ - 1
```

Semantics

The logical operations are composed of N instruction functions which occur within an Interpreter and affect the logic unit and associated registers.

The logic operations include the selection of adder inputs, the adder operation, the barrel switch operation, the destination specifications for the adder and BSW outputs, and the controls for the literal, counter and SAR registers. The monadic operations are those adder operators for which only one input select is explicit. The selected value or its ones complement may become the adder input depending on the ⟨Not⟩ function being ⟨Empty⟩ or NOT. DEC ⟨X Select⟩ is equivalent to ⟨X Select⟩ -1. The dyadic operations have both an ⟨X Select⟩ and a ⟨Y Select⟩.

The default ⟨Logic Op⟩ is unconditional 0 + 0 =:. This <u>does cause completion</u> of the prior ⟨Logic Op⟩ in progress in phase 3.

Examples

0 =: CTR

A1 AND B011 =: A1

A2 + NOT CTR IC R =: A2, BEX, CTR, CSAR

INPUT SELECTS

⟨X Select⟩ ::=	0 | A1 | A2 | A3 | CTR | ZEXT | LIT | Z | ⟨Empty⟩
⟨Y Select⟩ ::=	0 | 1 | B | B ⟨M⟩⟨C⟩⟨L⟩ | CTR | ZEXT | LIT | Z | AMPCR
⟨N Y Select⟩ ::=	⟨Not⟩ ⟨Y Select⟩
⟨M⟩ ::=	⟨Gating⟩
⟨C⟩ ::=	⟨Gating⟩
⟨L⟩ ::=	⟨Gating⟩
⟨Gating⟩ ::=	0 | 1 | T | F

Semantics

There are three A registers which may be used for data storage within an Interpreter. Any one of the A registers, or the counter, external source, literal or the catenation of these (Z) may be selected as input to the adder in the ⟨X Select⟩ part of the instruction. The B-register is the primary interface for external inputs from main memory or devices. It also serves as input to the adder. The B-register can be partitioned when it is selected as input to the adder. The partitions are as follows:

M	Most significant bit of B (left most bit)
C	Central bits of B (all but the end bits)
L	Least significant bit of B (right most bit)

When selecting the B register as input to the adder, each of the three parts may be independently specified as being either 0, 1, T, or F. A zero gating will cause that part to be all zeros. A one gating will cause that part to be all ones. A T gating will produce the true value of B for that part. An F gating will produce the one complement value of B for that part. The B register and its gating are specified without embedded spaces. If no gating is specified when selecting B, then it is assumed that the true value of B is desired (i.e., BTTT). When the ⟨Y Select⟩ is CTR, ZEXT, LIT, Z, or AMPCR, the center bits are 0 or T. When the center bits are specified as 1 or F, the adder operation is used by TRANSLANG which complements B from that specified. The center bits for the ⟨Y Select⟩ for AAD or OAD may not be F or 1.

Several registers may become input to the adder from either the ⟨X Select⟩ or the ⟨Y Select⟩ or both. These include the counter (CTR) to the most significant byte, an external source (ZEXT) to the center byte(s) where the logic unit is wider than 2 bytes, and the literal (LIT) to the least significant byte. All three of these may be specified at once (Z). The AMPCR can only be a ⟨Y Select⟩ input into the least significant 12 bits. The remaining bits of these register inputs are zeros.

Examples

BTFF

A1

AMPCR

ADDER AND SHIFT OPERATORS

⟨Commut Op⟩ ::= NOR | NRI | NAN | XOR | NIM | IMP | EQV | AND | RIM | OR | +

⟨Non Commut Op⟩ ::= OAD | AAD | -

⟨Inhibit Carries⟩ ::= IC | ⟨Empty⟩

⟨Shift Op⟩ ::= R | L | C | ⟨Empty⟩

Semantics

Each ⟨Dyadic⟩ contains a ⟨Commut Op⟩ or a ⟨Non Commut Op⟩. The set of operators in ⟨Commut Op⟩ contain commutative pairs, hence the x and y inputs may appear in either order, where x refers to the ⟨X Select⟩ and y refers to the ⟨Y select⟩. The ⟨Non Commut Op⟩ are non commutative and must appear in the order x ⟨Non Commut Op⟩ y. The recommended standard order is x operator y which works for all operators.

Operator ⟨Commut Op⟩	Name	Equivalent*	Bitwise Result#
x NOR y	Nor	$\bar{x}\bar{y}$	$x \not\vee y$
x NRI y	Not Reverse Imply	$\bar{x}y$	$x < y$
x AND y	And	xy	$x \wedge y$
x NIM y	Not Imply	$x\bar{y}$	$x > y$
x XOR y	Exclusive Or	$(x\bar{y}) \vee (\bar{x}y)$	$x \neq y$
x EQV y	Equivalence	$(xy) \vee (\bar{x}\bar{y})$	$x = y$
x IMP y	Imply	$\bar{x} \vee y$	$x \leq y$
x NAN y	Nand	$\bar{x} \vee \bar{y}$	$x \not\wedge y$
x RIM y	Reverse Imply	$x \vee \bar{y}$	$x \geq y$
x OR y	Or (inclusive)	$x \vee y$	$x \vee y$
x + y	Add		

*In terms of negation ($\bar{u}$), logical and (uw), and logical inclusive or ($u \vee w$). Precedence is parentheses before negation before and before or.

#The bitwise result r_i for each bit i means: x_i operator y_i =: r_i

Operator	Name	Equivalent*
⟨Non Commut Op⟩		
x - y	Subtract	$x + \bar{y} + 1$
x OAD y	Or Add	$x + (x \vee y)$
x AAD y	And Add	$x + (xy)$

The carries into 8-bit bytes that normally may propagate during an adder operation that includes either add or subtract may be inhibited by including IC if byte arithmetic is desired.

There are four operations causing shifting, one of which is selected each time an adder operator is used.

R	Right end-off shift by amount in SAR, filled with left zeros
L	Left end-off shift by word length complement of amount in SAR, filled with right zeros.
C	Circular right end-around shift by amount in SAR
⟨Empty⟩	No shift

Examples

```
0
NOT LIT
A1 + B + 1 IC R
A2 OR NOT CTR C % same as A2 RIM CTR C
+ B + 1     % same as 0 + B + 1 — note B + 1 is invalid syntactically.
```

DESTINATION OPERATORS

```
⟨Destination List⟩ ::= ⟨Asgn⟩ ⟨Dest⟩ |
                       ⟨Destination List⟩ ⟨Asgn⟩ ⟨Dest⟩|⟨Asgn⟩
⟨Asgn⟩ ::=             , | =:
⟨Dest⟩ ::=             A1 |A2 |A3 |MIR |BR1 |BR2 |AMPCR |
                       ⟨Input B⟩ | ⟨Input Ctr⟩ | ⟨Input Mar⟩| ⟨Input Sar⟩
⟨Input B⟩ ::=          B |BEX |BAD |BC4 |BC8| BMI |BBE |BBA |BBI
⟨Input Ctr⟩ ::=        CTR | LCTR | INC
⟨Input Mar⟩ ::=        MAR | MAR1 | MAR2 | LMAR
⟨Input Sar⟩ ::=        SAR | CSAR
```

Semantics

The destination operators explicitly specify registers in which changes are to occur at the end of a logic unit operation.

Restrictions:

1. At most one choice from each of ⟨Input B⟩, ⟨Input Ctr⟩, ⟨Input Mar⟩ and ⟨Input Sar⟩ is permitted.
2. If ⟨Input Ctr⟩ is LCTR then ⟨Input Mar⟩ may not be MAR, MAR1 or MAR2.
3. If ⟨Input Mar⟩ is LMAR then ⟨Input Ctr⟩ may not be CTR.

The principal data source is the barrel switch output. It is the only source for loading A1, A2, A3, MIR, BR1 and BR2. It provides one source for loading B, CTR, MAR, SAR and AMPCR. These reserved words are also the register names. The bits used in these transfers are indicated below:

Destination Register	Barrel Switch Output Source Bits
A1	All
A2	All
A3	All
B	All
MIR	All
BR1	2nd least significant byte
BR2	2nd least significant byte
MAR	least significant byte
CTR	least significant byte (ones complement)
SAR	least significant bits
AMPCR	least significant 12 bits

The B, MAR, CTR, SAR and AMPCR registers may have other inputs as well.

B Register – (B)

B	The barrel switch output is placed into B.
BEX	Data from the external source is placed into B.
BAD	The adder output is placed in the B register (short path to B).

BC4	The duplicated complement of the 4-bit carries with zero fill is placed in the B register. *
BC8	The duplicated complement of the 8-bit carries with zero fill is placed in the B register. *
BMI	The MIR content is placed in the B register independent of any concurrent change to the MIR.
BBE	The barrel switch output ORed with the data from the external source is placed in the B register.
BBA	The barrel switch output ORed with the adder output is placed in the B register.
BBI	The barrel switch output ORed with the MIR content is placed in the B register independent of any concurrent change to the MIR.

Memory Address Register — (MAR)

LMAR	The literal register content is placed in MAR

Counter — (CTR)

LCTR	The ones complement of the literal register content is placed in CTR
INC	Increment Counter by 1

Shift Amount Register — (SAR)

CSAR	Complement prior content of SAR

The Alternative Micro Program Count Register AMPCR may during the same clock receive input from the MPCR if the MPAD CTLS register content was CALL or SAVE. The MPCR source takes precedence over the AMPCR specification as a ⟨Dest⟩.

* Form of BC4 and BC8 adder outputs for each 8-bit group:
The carries out of bits 2, 3, 4, 6, 7 and 8 are irrelevant.

Bit	1	2	3	4	5	6	7	8
Carries Out	u	-	-	-	v	-	-	-
BC4	o	o	$\bar{u}$	$\bar{u}$	o	o	$\bar{v}$	$\bar{v}$
BC8	o	o	$\bar{u}$	$\bar{u}$	o	o	o	o

Examples

```
=:B
=:CTR
=:A1, BEX, =:MIR, LCTR, CSAR          % mixed use of , and =:
```

SUCCESSOR

⟨Successor⟩ ::= WAIT | STEP | SKIP | SAVE | CALL | EXEC | JUMP | RETN

Semantics

Each ⟨N instruction⟩ specifies 2 successors explicitly or implicitly, indicating the control to be used for the next M-instruction selection. A ⟨Successor⟩ in the ⟨Unconditional Part⟩ results in the 2 successors being identical. Otherwise one or two successors may appear in the ⟨Conditional Part⟩. The eight choices for each successor are described below and in Table 3-3.

WAIT	Repeat the instruction in the microprogram count register (MPCR).
STEP	Step to the next instruction in sequence from MPCR.
SKIP	Skip to the second next instruction in sequence from MPCR.
SAVE	Step and save current MPCR address in AMPCR.
CALL	Transfer control to AMPCR + 1 address, save current MPCR in AMPCR.
EXEC	Execute instruction in AMPCR + 1, proceed as specified in the executed instruction.
JUMP	Transfer control to AMPCR + 1 address.
RETN	Transfer control to AMPCR + 2 address.

Any successor not explicitly stated is STEP by default. All successors except EXEC place the resulting microprogram address in MPCR.

Each ⟨Literal Assignment⟩ instruction has an implicit successor of STEP.

Table 3-3. Microprogram Memory Addressing

Successor Command	Successor M-instruction Address	Next Content of MPCR will be	Next Content of AMPCR will be
WAIT	MPCR	MPCR	*
STEP	MPCR+1	MPCR+1	*
SKIP	MPCR+2	MPCR+2	*
SAVE	MPCR+1	MPCR+1	MPCR
CALL	AMPCR+1	AMPGR+1	MPCR
EXEC	AMPCR+1	MPCR	*
JUMP	AMPCR+1	AMPCR+1	*
RETN	AMPCR+2	AMPCR+2	*

*Not changed by successor specification

The AMPCR normally contains the address of an alternative instruction (usually label-1). The AMPCR load of the current content of the MPCR from a CALL or SAVE takes precedence over a ⟨Literal Assignment⟩ into AMPCR in the dynamically next M-instruction. It also takes precedence over an explicit ⟨Dest⟩ of AMPCR from the ⟨Logic Op⟩ in progress. In Appendix D see Instruction Phasing and the comments concerning control N [42].

Examples

WAIT

JUMP

PROGRAM STRUCTURE

⟨Program⟩ ::= ⟨Program Name Line⟩ ⟨Body⟩ ⟨End Line⟩

⟨Program Name Line⟩ ::= PROGRAM ⟨Program Name⟩ ⟨Start Address⟩

⟨Program Name⟩ ::= ⟨Label⟩

⟨Start Address⟩ ::= ADR ⟨Hex Address⟩ | ⟨Empty⟩

⟨Hex Address⟩ ::= ⟨Hex Number⟩

⟨Hex Number⟩ ::= ⟨Hex Digit⟩ | ⟨Hex Number⟩⟨Hex Digit⟩

⟨Body⟩ ::= ⟨Comment⟩ | ⟨Line⟩ | ⟨Body⟩ ⟨Line⟩

⟨Line⟩ ::= ⟨Label Constant⟩ | ⟨Start Address⟩ | ⟨Insert⟩ | ⟨Instruction⟩

⟨Label Constant⟩ ::= ⟨Label⟩ * ⟨Integer⟩

⟨Insert⟩ ::= ⟨Label Part⟩ INSERT ⟨File Name⟩ ⟨Start Address⟩

⟨Label Part⟩ ::= ⟨Label⟩ : | ⟨Empty⟩

⟨File Name⟩ ::= ⟨Label⟩

⟨Instruction⟩ ::= ⟨Label Part⟩ ⟨Literal Assignment⟩ |
⟨Label Part⟩ ⟨N Instruction⟩

⟨End Line⟩ ::= END

Semantics

A file containing a source program must have a ⟨File Name⟩ of 6 or less alphanumeric characters. Each record on this file contains 72 data characters (+8 for sequence numbers ignored by the microtranslator). One line of source program is written per record.

The first record is the ⟨Program Name Line⟩. It contains the program internal name and possibly a starting address for a microprogram. The program internal name should be the same as the file name. Only the file name has any external significance. An empty ⟨Start Address⟩ means start with zero for the first M-instruction of the program. A non-empty start address becomes a hexadecimal absolute microprogram address. The body of a program contains one or more lines. Following the body is the ⟨End Line⟩ containing END. Each successive line containing an ⟨Instruction⟩ normally becomes the next M-address. Addresses strictly increase through a program. If a ⟨Start Address⟩ is greater than the next address in the program sequence, M-instructions composed of all zeros are used to fill in the locations between the addresses in the output file. A ⟨Start Address⟩ less than the next address in the program sequence causes an error.

A label is defined for use in two ways. A ⟨Label Constant⟩ permits a ⟨Label⟩ to be declared to be an ⟨Integer⟩. Subsequent use of that label is replaced by the Integer. Use of a ⟨Label Constant⟩ prior to declaration is an error. A label is also defined upon occurrence in a ⟨Label Part⟩ in which case it serves as a symbolic reference to a particular line.

An ⟨Insert⟩ is used to allow a user access to his files outside the program file. When the ⟨Insert⟩ is recognized, the microtranslator extracts from the users files the source program whose ⟨File Name⟩ is given and inserts it at the ⟨Start Address⟩ in the ⟨Insert⟩ if present, otherwise in sequence. A ⟨Start Address⟩ occurring within the body of the inserted program will act as though it were in the main program file. A ⟨Start Address⟩ in the ⟨Program Name Line⟩ of the inserted program is ignored. The inserted program takes the multifile ID name from the program being translated. For example:

BCDADD/JUDY may be inserted into a program named DECVAD/JUDY. There may be seven levels of nesting. A label may be redefined in an inserted subprogram. An inserted program may reference a label in the program which requested it provided the label has not (yet) been defined locally. The most local current definition of a label is used. If labels are not defined during a subprogram the translator assumes they are at a more global level. Labels referenced but never defined result in a warning list of undeclared labels. Caution: Forward jumps within a subprogram to a label that already exists globally will use the global label value. Upon completion of an ⟨Insert⟩ of a subprogram, labels defined in that inserted subprogram disappear. A subsequent backward jump or use of a label constant will use the global value, even though the same label was defined in the subprogram.

Each instruction results in a microprogram word. Any instruction may be labeled as a symbolic reference for control transfer. Although transfer to a ⟨Literal Assignment⟩ is permitted it should be used with caution (see examples 1 and 2 in Appendix F).

Examples

Figure 3-1 shows two programs, the first of which uses the second as a macro. Figure 3-2 is an illustration of a register state map that is useful in indicating the expected register contents at label points of programs.

```
                 DECVAD /JH      LISTED AT 13:35 ON 70327 BYJUDY

PROGRAM DECVAD
COMMENT--BCD DECIMAL VECTOR ADD
   THIS ROUTINE SUMS N (0 LEQ N LSS 256) NON-NEGATIVE BCD DECIMAL
   INTEGERS AT S-MEMORY ADDRESSES A3 THROUGH A3+N-1.
   AT ENTRY TO DECVAD:
      1.  N HAS BEEN LOADED INTO CTR (CTR CONTAINS 255-N)
      2.  AMPCR CONTAINS THE CALL MPAD (MICROPROGRAM ADDRESS)
      3.  BR1 CONTAINS THE S-MACHINE REGISTER BASE.
   THE INSERTED ROUTINE IS USED AS A MACRO.
   AT EXIT FROM DECVAD:
      1.  THE SUM IS IN A2
      2.  IF ADDER OVERFLOW:
      2.1 THE CTR CONTAINS 256-THE NUMBER REMAINING WHEN AOV OCCURRED
      2.2 THE RETURN IS VIA THE CALLERS MPAD+1 WHICH SHOULD CONTAIN
              "AOVERR-1=:AMPCR" POINTING TO THE CALLERS ERROR ROUTINE
      3   IF NO ADDER OVERFLOW:
      3.1 THE SUM IS IN THE S-MACHINE REGISTER "ACCUM" IN S-MEMORY
      3.2 THE RETURN IS TO THE CALLERS MPAD+2. ;
   ACCUM*6                                  % S-ACCUMULATOR (GLOBAL)
   AMPCR=:MIR                               % CALLERS RETURN ADDRESS-1
   A3=:MAR2,INC                             % INITIAL WORD
   DONE-1=:AMPCR                            % AVOID ADD IF 0 OR 1 ADDEND
   0=:A2 IF NOT COV THEN MR2 ELSE JUMP      % 1ST READ
   WHEN RDC THEN A3+1=:A3=:MAR2,BEX,INC     % 2ND ADDRESS, 1ST ADDEND
   B=:A2 IF NOT COV THEN MR2;SAVE ELSE JUMP % 2ND READ; LOOP-1 ELSE DONE
LOOP:WHEN RDC THEN A3+1=:A3=:MAR2,BEX,INC   % NEXT ADDRESS,THIS ADDEND
   B=:A1 IF NOT COV THEN MR2;SKIP           % ADDEND,NEXT READ OR DONE
   DONE-1=:AMPCR                            % SETUP FOR JUMP BELOW
INSERT BCDADD                               % IN LINE SINCE NO ADR GIVEN
AOVER:BMI                                   % FALL THROUGH ON OVERFLOW
   B=:AMPCR                                 % RESTORE CALLERS ADDRESS-1
   NOT CTR=: ;EXEC      % TEST LAST ADDEND; CALLERS "AOVERROR-1=:AMPCR"
   IF ABT THEN JUMP     % AOV ON LAST ADDEND, NO MR2 IN PROGRESS
   WHEN RDC THEN JUMP   % ASSURE RDC RESET FOR NEXT USE
DONE:A2=:MIR,BMI,LMAR                       % SUM, CALLERS ADDRESS-1,
   ACCUM=:LIT                               % DEFINED LIT USED ABOVE
   MW1;B=:AMPCR IF SAI                      % TO S-ACCUMULATOR
   WHEN SAI THEN RETN   % ASSURE WRITE ACCEPTED BEFORE NORMAL RETURN
END
```

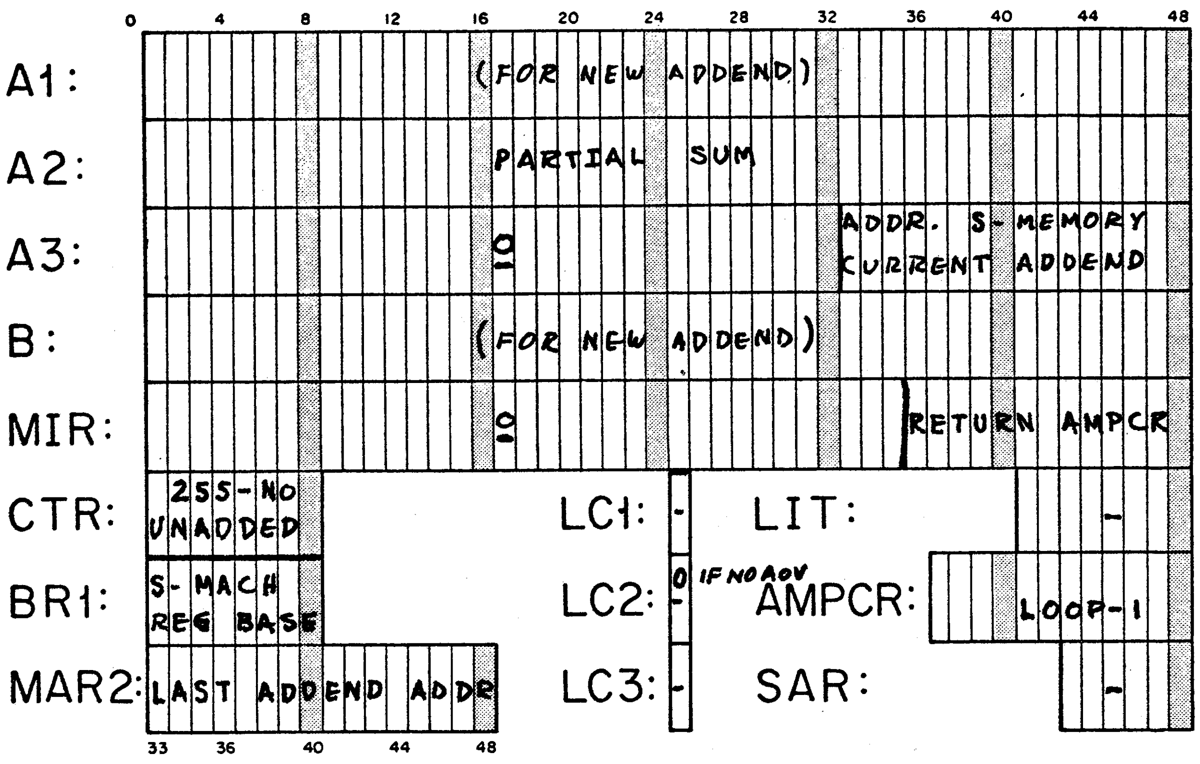
```
                 BCDADD /JH      LISTED AT 14: 6 ON 70327 BYJUDY

PROGRAM BCDADD
COMMENT--SUM THE BCD DECIMAL INTEGERS A1+A2=:A2, AOV IN LC2;
   B001=:SAR,BC4;IF LC2 %HEX ALL THREES TO B
   A1+B=:A1             %CONVERT TO EXCESS 3
   A2+B=:B              %CONVERT TO EXCESS 3
   A1+B C=:A1,BC4,CSAR  %SUM, NOT CARRIES
   A1-B C=:A2;IF AOV THEN SET LC2;STEP ELSE JUMP %-6 FOR DIGITS/NO CARRY
END
```

Figure 3-1. Program Example

PAGE____OF____

48-BIT INTERPRETER REGISTER FORMAT

PROGRAM DECVAD LABEL LOOP DATE ________



A1:	(FOR NEW ADDEND)	
A2:	PARTIAL SUM	
A3:	0	ADDR. S-MEMORY CURRENT ADDEND
B:	(FOR NEW ADDEND)	
MIR:	0	RETURN AMPCR
CTR:	255 - NO UNADDED	
LC1:	-	
LIT:	-	
BR1:	S-MACH REG BASE	
LC2:	0 IF NO AOV	
AMPCR:	LOOP-1	
MAR2:	LAST ADDEND ADDR	
LC3:	-	
SAR:	-	

0
4
8
12
16
20
24
28
32
36
40
44
48
33
36
40
44
48

Figure 3-2. Sample Register Content Form

MICROPROGRAMMING SUMMARY

Each yellow card includes a summary of the syntax described in this section. A prototype line for an N-instruction is shown. It may be described as follows.

A type I line may have the following ordered parts.

1. An optional Label which is terminated by " : " if present

2. A selection of zero or more of Unconditional Ext Op Logic Op or Unconditional Successor. If more than one, they are separated by ": " .

3. An optional conditional part consisting of

3.1 A choice of either IF or WHEN.

3.2 A condition.

3.3 An optional conditional component list prefixed by THEN.

3.3.1 A selection of zero or more of conditional Ext Op Logic Op or True Successor. If more than one, they are separated by ": ".

3.3.2 An optional ELSE False Successor.

Restrictions that indicate the interdependence of controls are indicated on the card. Each of Ext Op, Logic Op, and Successor may be independently unconditional or conditional, but not both. Thus an Unconditional Successor precludes a True Successor, WHEN or ELSE (except for redundant specification of both true and false Successors to be the same). A WHEN has an implied ELSE WAIT, so no other false successor may be specified.

A possible order for the various component reserved words that can be used to form the various parts of an N-instruction is shown below the prototype. Although there is no essential order among Ext Op, Logic Op and Successor, that is the recommended order. Similarly the order is irrelevant within Ext Op and among multiple destinations. Within such a group, a comma is used as a separator.

The card also shows type II Literal Assignment instruction forms and the various pseudo-instructions.

The reverse of the card relates the reserved words to the micro- and nano-controls.

4. TRANSLATOR OPERATING INSTRUCTIONS

The translator is written in compatible ALGOL for the B 5500. It is interactive and uses a Model 33 or 35 Teletypewriter. The user must have an input file of type SEQ whose first record is the word PROGRAM followed by the program name. Figure 4-1 is a sample of the conversation during an execution of the Microtranslator for the example program DECVAD.

The user will request the translator as follows:

```
? ?EXECUTE TRANSL/ADOESO NO Charge-Number;END.Comment ←
```

The translator will be initiated by the system and the system prints:

```
Priority No:TRANSL/ADOESO=Mix-Index BOJ Time FROM Buffer No.
```

When the translator starts, it requests the input file name:

```
PRINT FILENAME : FILEID MULTID
```

The user responds with the input file name and his user code:

```
Input-File-Name   User-Code ←
```

The translator then requests the type of listing expected:

```
LISTING AND NANOS:  NONE=0,  NANOS=1,  PROGRAM=2,  BOTH=3
```

```
??EXECUTE TRANSL/ADDESO NO          ;END.          JULY 11/23←

 5:TRANSL/ADDESO=01 BOJ 1026 FROM 01/12
PRINT FILENAME : FILEID  MULTID
DECVAD JH←
LISTING AND NANOS: NONE=0,NANOS=1,PROGRAM=2,BOTH=3
3←
OUTPUT DEVICE:0=TTY,1=PRINTER
1←
REGISTER SIZE:  16=0,24=1,32=2,40=3,48=4,56=5,64=6
0←
NANOTABLE TYPE: NEW=0,OLD=1.
0←

 LPA OUT PRFILE:TRANSL/ADDESO= 1
MICROPROGRAM LISTING?  NO=0 , YES=1.
1←
STATISTICS?  NO=0  ,  YES =1   .
1←
SAVE NANOTABLE: NO=0 ,YES=1
0←
PUNCH NANOS? NO=0. TAPE=1. CARD=2.
1←
TURN ON TAPEPUNCH OK
←
88%28241I8882888%2827X888887X8&8X;88888X888;&C82;X;88787X8]
84&8X%8;188X888;&8X&4;1818888;4F8248;8884828%282&C18188888]
8X%2827;18827888%282&C18784818%FI;2C17FX8888%28248888;4888]
8T%2824;18881888%28C%X88X88888%R8X4888888888&R8X8888888888]
18%28278888;&288&&824;1888188&&&8F888888888888888888888888]


2888W888W8811812W882W88;W884W88;W88%1812W88&W887W88XW88IW88RW88CW88T]
2818W88EW88FW88WW818F88&W811W812QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ]
```

Figure 4-1. Microtranslator Execution Conversation

The user types

0 ←	if no listing at all is desired
1 ←	for a list of nanos only, including inserted programs
2 ←	for a program list only, including inserted programs
3 ←	for a list of both the program and the corresponding nanos generated

The translator asks where to list the information.

```
OUTPUT DEVICE:0=TTY, 1=PRINTER
```

The user types

0 ←	if the output is wanted on the teletypewriter
1 ←	if the output is wanted on the printer

The listing includes all syntax errors. Syntax errors will always be printed on the teletypewriter. The line numbers on listings refer to the sequence numbers in the input file (which is a regular card file of type BASIC for R/C (Remote Card)).

The translator next requests the size of the machine for which the code is being translated.

```
REGISTER SIZE: 16=0, 24=1, 32=2, 40=3, 48=4, 56=5, 64=6
```

The user types the proper integer according to the width of the machine he is using.

0 ←	for a 16-bit machine
1 ←	for a 24-bit machine
2 ←	for a 32-bit machine
3 ←	for a 40-bit machine
4 ←	for a 48-bit machine
5 ←	for a 56-bit machine
6 ←	for a 64-bit machine

The next request is

```
NANOTABLE TYPE: NEW=0, OLD=1.
```

```
                                                         DECVAD
100 PROGRAM DECVAD
200 COMMENT--BCD DECIMAL VECTOR ADD
300    THIS ROUTINE SUMS N (0 LEQ N LSS 256) NON-NEGATIVE BCD DECIMAL
400    INTEGERS AT S-MEMORY ADDRESSES A3 THROUGH A3+N-1.
500    AT ENTRY TO DECVAD:
600       1. N HAS BEEN LOADED INTO CTR (CTR CONTAINS 255-N)
700       2. AMPCR CONTAINS THE CALL MPAD (MICROPROGRAM ADDRESS)
800       3. BR1 CONTAINS THE S-MACHINE REGISTER BASE.
900    THE INSERTED ROUTINE IS USED AS A MACRO.
1000    AT EXIT FROM DECVAD:
1100       1. THE SUM IS IN A2
1200       2. IF ADDER OVERFLOW:
1300       2.1 THE CTR CONTAINS 256-THE NUMBER REMAINING WHEN AOV OCCURRED
1400       2.2 THE RETURN IS VIA THE CALLERS MPAD+1 WHICH SHOULD CONTAIN
1500              "AOVERR-1=:AMPCR" POINTING TO THE CALLERS ERROR ROUTINE
1600       3   IF NO ADDER OVERFLOW:
1700       3.1 THE SUM IS IN THE S-MACHINE REGISTER "ACCUM" IN S-MEMORY
1800       3.2 THE RETURN IS TO THE CALLERS MPAD+2. ;
1900    ACCUM+6                                     % S-ACCUMULATOR (GLOBAL)
2000    AMPCR=:MIR                                  % CALLERS RETURN ADDRESS-1
     0   2  5 13 16 22 23 26 41
2100    A3=:MAR2,INC                                % INITIAL WORD
     1   2  5 13 16 17 18 19 44 45 46 47
2200    DONE-1=:AMPCR                               % AVOID ADD IF 0 OR 1 ADDEND
     2  110    000 0000 0000
2300    0=:A2 IF NOT COV THEN MR2 ELSE JUMP         % 1ST READ
     3   1  7 13 14 35 53 54
2400    WHEN RDC THEN A3+1=:A3=:MAR2,BEX,INC        % 2ND ADDRESS, 1ST ADDEND
     4   1  3  5  6 13 17 18 19 25 26 36 37 38 44 45 46 47
2500    B=:A2 IF NOT COV THEN MR2;SAVE ELSE JUMP % 2ND READ; LOOP-1 ELSE DONE
     5   1  7 12 14 21 22 26 35 53 54
2600 LOOP:WHEN RDC THEN A3+1=:A3=:MAR2,BEX,INC    % NEXT ADDRESS,THIS ADDEND
     6   1  3  5  6 13 17 18 19 25 26 36 37 38 44 45 46 47
2700    B=:A1 IF NOT COV THEN MR2;SKIP              % ADDEND,NEXT READ OR DONE
     7   1  7 12 13 16 21 22 26 34 53 54
2800    DONE-1=:AMPCR                               % SETUP FOR JUMP BELOW
     8  110    000 0000 0010
2900 INSERT BCDADD                                  % IN LINE SINCE NO ADR GIVEN
3100 COMMENT--SUM THE BCD DECIMAL INTEGERS A1+A2=:A2, AOV IN LC2;
3200    B001=:SAR,BC4;IF LC2 %HEX ALL THREES TO B
     9   3  4  5 13 16 25 26 40 49
3300    A1+B=:A1             %CONVERT TO EXCESS 3
     A   2  5 13 16 17 19 21 22 26 34
3400    A2+B=:B              %CONVERT TO EXCESS 3
     B   2  5 13 16 17 18 21 22 26 37 39 40
3500    A1+B C=:A1,BC4,CSAR  %SUM, NOT CARRIES
     C   2  5 13 16 17 19 21 22 26 32 33 34 40 50
3600    A1-B C=:A2;IF AOV THEN SET LC2;STEP ELSE JUMP %-6 FOR DIGITS/NO CARRY
     D   2  3  4  5  7 10 13 14 17 19 21 22 26 28 29 30 31 32 33 35
4000 AOVFR:BMI                                      % FALL THROUGH ON OVERFLOW
     E   2  5 13 16 37 38 40
4100    B=:AMPCR                                    % RESTORE CALLERS ADDRESS-1
     F   2  5 13 16 21 22 26 42
4200    NOT CTR=: ;EXEC        % TEST LAST ADDEND; CALLERS "AOVERROR-1=:AMPCR"
    10   2  5 11 13 14 16 18 19 31
4300    IF ABT THEN JUMP       % AOV ON LAST ADDEND, NO MR2 IN PROGRESS
    11   2  3  5 11 16
4400    WHEN RDC THEN JUMP     % ASSURE RDC RESET FOR NEXT USE
    12   1  3  5 11
5000 DONE:A2=:MIR,BMI,LMAR                          % SUM, CALLERS ADDRESS-1,
    13   2  5 13 16 17 18 37 38 40 41 45
5100    ACCUM=:LIT                                  % DEFINED LIT USED ABOVE
    14  111    000 0000 0110
5200    MW1;B=:AMPCR IF SAI                         % TO S-ACCUMULATOR
    15   1  4  5 13 16 21 22 26 42 52 53
5300    WHEN SAI THEN RETN     % ASSURE WRITE ACCEPTED BEFORE NORMAL RETURN
    16   1  4  5 11 12 13
```

Figure 4-2. Source Statements and Nanoinstructions

The user types

0 ← if a new nanotable will be created

1 ← if an old nanotable will be used

If an old nanotable is required, its name is requested.

NANOTABLE NAME: XXXXXX

The user must type in the name of his table (e.g.)

Nanotable-File-Id ←

Note that the Multifile-Id must be the same as the user code.

The translator will start processing the input file, listing the program and the generated nanos as requested. Figures 4-2 through 4-4 are printer listings. Corresponding listings may be obtained on the teletypewriter. Less columns are available for comments. The sequence numbers are printed first, taking some of the 72 columns available for lines of source text.

Figure 4-2 shows the source statements with their corresponding nanos. Each source statement is preceded by its sequence number. (The card image would have this number in columns 73-80.) Each N-instruction (type I) is also represented by the corresponding hexadecimal nanoaddress and the bits set to one (in the range 1 to 55). Each literal assignment (type II) instruction is represented by its microinstruction. Forward label references have a link to the prior line number also containing that label reference or zero if the first reference. (See sequence numbers 2200 and 2800. Backward label references have the proper location indicated.)

After the possible initial listing, the translator again requests information.

MICROPROGRAM LISTING? NO=0, YES=1.

The user types

0 ← if no listing is necessary

1 ← if he wants a listing of his microprogram and corresponding nanoinstructions. The device to be used for output is the same device as before.

```
                                                          DECVAD
0   NANO ADDRESS=  0  1111 0000000000
     2  5 13 16 22 23  26 41
1   NANO ADDRESS=  1  1111 0000000001
     2  5 13 16 17 18  19 44 45 46 47
2   AMPCR=   12        110 00000010010
3   NANO ADDRESS=  2  1111 0000000010
     1  7 13 14 35 53  54
4   NANO ADDRESS=  3  1111 0000000011
     1  3  5  6 13 17  18 19 25 26 36 37  38 44 45 46 47
5   NANO ADDRESS=  4  1111 0000000100
     1  7 12 14 21 22  26 35 53 54
6   NANO ADDRESS=  3  1111 0000000011
     1  3  5  6 13 17  18 19 25 26 36 37  38 44 45 46 47
7   NANO ADDRESS=  5  1111 0000000101
     1  7 12 13 16 21  22 26 34 53 54
8   AMPCR=   12        110 00000010010
9   NANO ADDRESS=  6  1111 0000000110
     3  4  5 13 16 25  26 40 49
A   NANO ADDRESS=  7  1111 0000000111
     2  5 13 16 17 19  21 22 26 34
B   NANO ADDRESS=  8  1111 0000001000
     2  5 13 16 17 18  21 22 26 37 39 40
C   NANO ADDRESS=  9  1111 0000001001
     2  5 13 16 17 19  21 22 26 32 33 34  40 50
D   NANO ADDRESS=  A  1111 0000001010
     2  3  4  5  7 10  13 14 17 19 21 22  26 28 29 30 31 32 33 35
E   NANO ADDRESS=  B  1111 0000001011
     2  5 13 16 37 38  40
F   NANO ADDRESS=  C  1111 0000001100
     2  5 13 16 21 22  26 42
10  NANO ADDRESS=  D  1111 0000001101
     2  5 11 13 14 16  18 19 31
11  NANO ADDRESS=  E  1111 0000001110
     2  3  5 11 16
12  NANO ADDRESS=  F  1111 0000001111
     1  3  5 11
13  NANO ADDRESS= 10  1111 0000010000
     2  5 13 16 17 18  37 38 40 41 45
14  LIT=     6         111 00000000110
15  NANO ADDRESS= 11  1111 0000010001
     1  4  5 13 16 21  22 26 42 52 53
16  NANO ADDRESS= 12  1111 0000010010
     1  4  5 11 12 13
```

Figure 4-3. Microprogram Listing

Figure 4-3 is the listing of the microprogram produced for the example program. For each M-instruction is shown the M-address and the content. If the content is a nanoaddress the hexadecimal address is given followed by the bit configuration. If the content is a literal assignment, the content is as indicated. Note that the forward label references have been fixed at this time (lines 2 and 8 now point to DONE-1 in line 12). The example used a 16-bit logic unit; consequently only a 14-bit microprogram word was required.

The translator next allows the user to print the nanotable constructed, and its statistics.

STATISTICS? NO=0 , YES=1

The user types

1 ← if he wants this output

0 ← otherwise.

Figure 4-4 illustrates this output. The first column indicates the number of microinstructions pointing to this N-instruction. The second column is the N-instruction hexadecimal address. The remaining columns are a binary printout of the N-instruction, bits 1 through 55. Note that N-instruction 3 was used twice.

```
                                                              DECVAD
STATISTICS                     NANOINSTRUCTIONS
    1        0100100000 0010010000 0110010000 0000000000 1000000000 0000
    1     1  0100100000 0010011110 0000000000 0000000000 0001111000 0000
    1     2  1000001000 0011000000 0000000000 0000100000 0000000000 0011
    2     3  1010110000 0010001110 0000110000 0000011100 0001111000 0000
    1     4  1000001000 0101000000 1100010000 0000100000 0000000000 0011
    1     5  1000001000 0110010000 1100010000 0001000000 0000000000 0011
    1     6  0011100000 0010010000 0000110000 0000000001 0000000010 0000
    1     7  0100100000 0010011010 1100010000 0001000000 0000000000 0000
    1     8  0100100000 0010011100 1100010000 0000001011 0000000000 0000
    1     9  0100100000 0010011010 1100010000 0111000001 0000000001 0000
    1     A  0111101001 0011001010 1100010111 1110100000 0000000000 0000
    1     B  0100100000 0010010000 0000000000 0000001101 0000000000 0000
    1     C  0100100000 0010010000 1100010000 0000000000 0100000000 0000
    1     D  0100100000 1011010110 0000000000 1000000000 0000000000 0000
    1     E  0110100000 1000010000 0000000000 0000000000 0000000000 0000
    1     F  1010100000 1000000000 0000000000 0000000000 0000000000 0000
    1    10  0100100000 0010011100 0000000000 0000001101 1000100000 0000
    1    11  1001100000 0010010000 1100010000 0000000000 0100000000 0110
    1    12  1001100000 1110000000 0000000000 0000000000 0000000000 0000
END OF STATISTICS
```

Figure 4-4. Nanoinstructions Using Statistics

Next the translator asks if the nanotable should be saved.

SAVE NANOTABLE: NO=0 , YES=1

The user types

0 ← if the nanotable is not to be saved

1 ← if the nanotable is to be saved

If the answer is 1 and if the nanotable is an old one, the following question is asked.

NEW NAME DESIRED: NO=0 , YES=1

The user types

0 ← if the old name is to be used

1 ← if a new name is desired

If the old name is still used, the old table will be updated. Otherwise a new table must be included, and a name for it is requested.

TYPE NANOTABLE NAME: XXXXXX

After the user provides the desired name, the table is saved under this name for reuse at some future run.

If this file is to be saved for use on the Interpreter or on the APL Interpreter model, an output file must be generated. The translator requests the type output needed for running, as follows:

PUNCH NANOS? NO=0. TAPE=1. CARD=2.

The user types

0 ← if no output file is desired

1 ← if an output file is desired punched on paper tape at the teletypewriter terminal as input for the APL Interpreter model. (The BD:Machine, An APL Model for Micro Instruction Execution in Interpreter Based Systems, H. W. Bingham, Burroughs TR 70-3, April 30, 1970.)

2 ← if a punched card file is desired which may be loaded directly into the Interpreter card reader.

At the conclusion of the program the processor time, I/O time, and elapsed time are printed in 1/60 of a second increments.

5. AN EDITOR FOR NANOTABLES

The microtranslator produces tables of nanoinstructions which are kept in a library. Any one of these tables may be used as the basis of an M-program and may be expanded as needed by the program.

Each nanotable is named by its creator and may be used and modified by other programs. A nanotable name can be up to 6 letters long. It contains the nanos used by the programs using the table and a count indicating how often each nano has been used.

A program has been developed to allow a user to modify his own nanotable when necessary. This is the editor for nanotables. It is written in compatible ALGOL on the B 5500 system. It performs the following functions.

A. Lists the nanotables in the library directory
B. Deletes a nanotable from the library
C. Renames a nanotable
D. Lists a nanotable
E. Changes a nano in a nanotable
F. Deletes a nano in a nanotable
G. Prints a nano

OPERATING INSTRUCTIONS

The user will request the editor as follows

```
?? EXECUTE EDITOR/ADOESO NO Charge-Number;END. Comment ←
```

The Editor will be initiated by the system and the system prints

```
Priority No:EDITOR/ADOESO=Mix-Index BOJ Time FROM Buffer No.
```

The program requests a user option as follows

```
OPTIONS: Ø=CHANGE, 1=RENAME, 2=DELETE, 3=LIST, 4=DIRECT, 5=END
```

The options are as follows

Type "0" if changes are to be made to a table.
Type "1" if the table is to be renamed.
Type "2" if the table is to be deleted.
Type "3" if the table is to be printed.
Type "4" if the directory of the library is to be printed.
Type "5" if the program is to be completed.
Type "9" if the table is to be printed starting at a given address until it is completed.

If a 4 is selected the library directory is printed as follows

```
4 ←
D825
N2TETL
STACK
OPTIONS: Ø=CHANGE, 1=RENAME, 2=DELETE, 3=LIST, 4=DIRECT, 5=END
```

If the name of a table is to be changed, the directory name of the table is changed and the old name is deleted from the directory.

```
1 ←
TABLE NAME:
STACK ←
NEW TABLE NAME:
TESTER ←
OPTIONS: Ø=CHANGE, 1=RENAME, 2=DELETE, 3=LIST, 4=DIRECT, 5=END
4 ←
D825
N2TETL
TESTER
```

A table may be completely deleted from the library as follows:

```
OPTIONS: Ø=CHANGE, 1=RENAME, 2=DELETE, 3=LIST, 4=DIRECT, 5=END
2 ←
TABLE NAME:
TESTER ←
OPTIONS: Ø=CHANGE, 1=RENAME, 2=DELETE, 3=LIST, 4=DIRECT, 5=END
4 ←
D825
N2TETL
```

A response to options of a "5" will terminate the Editor routine and cause the changes made to be recorded in the nanotable library.

```
5 ←
END EDITNT 2.ø SEC.
```

The "3" and the "9" options allow a user to list his nanotable. The name of the table to be listed is requested. The statistics of the nano and the nano address will be listed together with the numbers of the bits that are set to one in this nano.

```
3 ←
TABLE NAME:
STACK ←
CNT=   4   ø  2  5  13  16  3ø  45  46
CNT=   2   1  2  5  11  12  14  15  3ø  51  52  53
CNT=   4   2  1  3   5   6  13  30  38  39  4ø
CNT=   2   3  3  5  12  13  15  16  19  21  23  3ø  34
CNT=   2   4  3  5  13  16  18  3ø  45  46
CNT=   2   5  2  5  13  16  18  19  2ø  21  22  3ø  31  34  41  51  52
CNT=   2   6  2  3   5   6   7   8   9  1ø  13  16  3ø  45  46
CNT=   2   7  1  3   5   6   7  11  12  3ø  38  39  4ø  51  52  53
CNT=   2   8  2  5  12  13  15  16  19  21  23  3ø  34  46  48
CNT=   2   9  2  5  13  16  18  3ø  46  48
CNT=   2  10  2  5  13  16  18  23  24  3ø  34  45  46  47
CNT=   2  11  1  5   6   7   8   9  1ø  13  16  17  18  3ø  41
CNT=   2  12  2  5  13  16  3ø  45  46  51  52  53
CNT=   2  13  2  5  13  16  3ø  51  52
CNT=   1  14  1  3   5   6  13  18  3ø  41
OPTIONS: ø=CHANGE, 1=RENAME, 2= DELETE, 3=LIST, 4=DIRECT, 5=END
```

The "0" options allows a user to change the nanos in his nanotable. When a nanotable is to be changed, the name of the table is requested. The user is asked for the address of the nano to be changed, and must type in the nano address as follows:

```
ø ←
TABLE NAME:
STACK ←
ADDRESS OF NANO =
1←
CNT=  2   1   2   5   13   16   3ø   51   52
OPTIONS:  ø=REPLACE, 1=DELETE, 2=SKIP, 3=END CORRECTIONS
```

The nano is then typed out and a new choice of options is requested for changing, deleting, skipping, or ending these corrections.

A "skip" (2) will just cause the request of a new nano address. An "end corrections" (3) will cause an exit to the outer set of options.

A "0" or replace option will request the number of ones in the new nano. The bit positions that are ones must then be typed each followed by a comma (,) as follows:

```
2, 5, 13, 16, 3∅, 51, ← *
```

*Do not forget any commas including the ending one.

```
ADDRESS OF NANO =
13 ←
CNT=    2   13    2   5   13   16   3∅   51   52
OPTIONS: ∅=REPLACE, 1=DELETE, 2=SKIP, 3=END CORRECTIONS
∅ ←
HOW MANY NANO POSITIONS ARE ONES?
6 ←
TYPE NANO BIT POSITIONS THAT ARE ONES
2, 5, 13, 16, 3∅, 51, ←
ADDRESS OF NANO =
13 ←
CNT     ∅    13    2   5   13   16   3∅   51
OPTIONS: ∅=REPLACE, 1=DELETE, 2=SKIP, 3=END CORRECTIONS
3 ←
OPTIONS: ∅=CHANGE, 1=RENAME, 2=DELETE, 3=LIST, 4=DIRECT, 5=END
5 ←
```

The delete option causes a nano to be deleted from the table. If the last nano is deleted the table just becomes one entry shorter. However, if the nano deleted is not the last nano in the table, the last nano is put into that address and the table becomes one entry shorter. Note that programs using this table must be retranslated.

```
ADDRESS OF NANO =
14 ←
CNT=    1   14    1   3   5   6   13   18   3∅   41
OPTIONS: ∅=REPLACE, 1=DELETE, 2=SKIP, 3=END CORRECTIONS
1 ←
ADDRESS OF NANO =
14 ←
ADDRESS TOO LARGE FOR THIS NANOTABLE
```

APPENDIX A

TRANSLANG SYNTAX

	Reference Page
⟨Program⟩ ::= ⟨Program Name Line⟩⟨Body⟩⟨End Line⟩	3-21
⟨Program Name Line⟩ ::= PROGRAM ⟨Program Name⟩⟨Start Address⟩	3-21
⟨Program Name⟩ ::= ⟨Label⟩	3-21
⟨Label⟩ ::= ⟨Letter⟩ | ⟨Label⟩⟨Letter⟩ | ⟨Label⟩⟨Digit⟩	3-6
⟨Letter⟩ ::= A| B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W| X|Y|Z	3-2
⟨Digit⟩ ::= 0|1|2|3|4|5|6|7|8|9	3-2
⟨Start Address⟩ ::= ADR ⟨Hex Address⟩ | ⟨Empty⟩	3-21
⟨Hex Address⟩ ::= ⟨Hex Number⟩	3-21
⟨Hex Number⟩ ::= ⟨Hex Digit⟩ | ⟨Hex Number⟩⟨Hex Digit⟩	3-21
⟨Hex Digit⟩ ::= ⟨Digit⟩ | A | B | C | D | E | F	3-2
⟨Empty⟩ ::= { The null string of characters }	3-2
⟨Body⟩ ::=⟨Comment⟩ | ⟨Line⟩ | ⟨Body⟩⟨Line⟩	3-22
⟨Comment⟩ ::= COMMENT {Any sequence of ⟨Comment Characters⟩ except ";" };	3-3
⟨Comment Character⟩ ::=⟨Character⟩ | • |# |& |$ | [|] | \ | /	3-2

```
                                                                          Reference
                                                                            Page

⟨Character⟩ ::= ⟨Letter⟩ | ⟨Digit⟩ | ⟨Single Space⟩ | ⟨Symbol⟩                3-2

⟨Single Space⟩ ::= { One horizontal blank position }                          3-3

⟨Symbol⟩ ::= , | ; | + | - | : | = | % | " | ( | ) | *                        3-2

⟨Assignment Op⟩ ::=  =:                                                       3-2

⟨Literal Assignment⟩ ::= ⟨Literal⟩ =:AMPCR|⟨Literal⟩ =:SAR |                  3-4
                         ⟨Literal⟩ =:SAR; ⟨Literal⟩ =: ⟨LIT⟩ |
                         ⟨Literal⟩ =:⟨LIT⟩ ; ⟨ Literal⟩ =:SAR |
                         ⟨Literal⟩ =: ⟨LIT⟩

⟨Literal⟩ ::=⟨Integer⟩ | COMP ⟨Integer⟩ | ⟨Label⟩ | ⟨Label⟩ -1 |
             "⟨Comment Character⟩ "                                           3-4

⟨Integer⟩ ::= ⟨Digit⟩ | ⟨Digit⟩ ⟨Integer⟩                                      3-4

⟨Lit⟩ ::= LIT | SLIT                                                          3-4

⟨N Instruction⟩ ::= ⟨Unconditional Part⟩ ⟨Conditional Part⟩                   3-6

⟨Unconditional Part⟩ ::= ⟨Component List⟩                                     3-6

⟨Component List⟩ ::= ⟨Component⟩ | ⟨Component List⟩ ;⟨Component⟩ |
                     ⟨Empty⟩                                                  3-6

⟨Component⟩ ::=⟨Ext Op⟩ | ⟨Logic Op⟩ | ⟨Successor⟩                            3-6

⟨Ext Op⟩ ::= ⟨Mem Dev Op⟩ | ⟨Set Op⟩ | ⟨Mem Dev Op⟩ , ⟨Set Op⟩ |              3-11
             ⟨Set Op⟩ , ⟨Mem Dev Op⟩ | ⟨Empty⟩

⟨Mem Dev Op⟩ ::= MR1 | MR2 | MW1 | MW2 | DL1 | DL2 | DR1 | DR2 |               3-11
                 DW1 | DW2 | DU1 | DU2 | ASR | ASE

⟨Set Op⟩ ::= SET ⟨ Cond Adjust Bit⟩ | RESET GC                                3-13

⟨Cond Adjust Bit ⟩ ::= INT  LC1  LC2  LC3  GC1  GC2                           3-11

⟨Logic Op⟩ ::= ⟨Adder Op⟩ ⟨Inhibit Carry⟩ ⟨Shift Op⟩ ⟨Destination List⟩       3-14

⟨Adder Op⟩ ::= 0 | 1 | ⟨Monadic⟩ | ⟨Dyadic⟩ | ⟨Empty⟩                          3-14

⟨Monadic⟩ ::= ⟨Not⟩⟨X Select⟩ | ⟨N Y Select ⟩ | DEC ⟨ X Select ⟩               3-14
```

```
                                                                   Reference
                                                                     Page

⟨Not⟩ ::= NOT|⟨Empty⟩                                                3-14
⟨X Select⟩ ::= 0|A1|A2|A3|CTR|ZEXT|LIT|Z|⟨Empty⟩                     3-15
⟨N Y Select⟩ ::= ⟨Not⟩ ⟨Y Select⟩                                    3-15
⟨Y Select⟩ ::= 0|1|B|B ⟨M⟩⟨C⟩ ⟨L⟩|CTR|ZEXT|LIT|Z|AMPCR               3-15
⟨M⟩ ::= ⟨Gating⟩                                                     3-15

⟨C⟩ ::= ⟨Gating⟩                                                     3-15

⟨L⟩ ::=⟨Gating⟩                                                      3-15

⟨Gating⟩ ::= 0|T|F|1                                                 3-15

⟨Dyadic⟩ ::=  ⟨X Select⟩⟨Commut Op⟩⟨N Y Select⟩                       3-14
              ⟨Y Select⟩⟨Commut Op⟩⟨X Select⟩|
              ⟨X Select⟩⟨Non Commut Op⟩⟨N Y Select⟩
              ⟨X Select⟩ +⟨N Y Select⟩ +1
              ⟨X Select⟩ -⟨N Y Select⟩ -1

⟨Commut Op⟩ ::=  NOR|NRI|NAN|XOR|NIM|IMP|EQV|                        3-16
                 AND|RIM|OR|+

⟨Non Commut Op⟩ ::= OAD|ADD|-                                        3-17

⟨Inhibit Carries⟩ ::= IC|⟨Empty⟩                                     3-16

⟨Shift Op⟩ ::= R|L|C|⟨Empty⟩                                         3-17

⟨Destination List⟩ ::=  ⟨Asgn⟩⟨Dest⟩|                                3-17
                        ⟨Destination List⟩ ⟨Asgn⟩ ⟨Dest⟩|⟨Asgn⟩

⟨Asgn⟩ ::=  , | =:                                                   3-17

⟨Dest⟩ ::= A1|A2|A3|MIR|BR1|BR2|AMPCR|⟨Input B⟩|                     3-17
           ⟨Input Ctr⟩| ⟨Input Mar⟩ | ⟨Input Sar⟩

⟨InputB⟩ ::= B|BEX|BAD|BC4|BC8|BMI|BBE|BBA|BBI                       3-17

⟨Input Ctr⟩ ::= CTR|LCTR|INC                                         3-17

⟨Input Mar⟩ ::= MAR|MAR1|MAR2|LMAR                                   3-17

⟨Input Sar⟩ ::= SAR|CSAR                                             3-17
```

	Reference Page
⟨Successor⟩ ::= WAIT | STEP | SKIP | SAVE | CALL | EXEC | JUMP | RETN	3-20
⟨Conditional Part⟩ ::= ⟨If Clause⟩⟨Cond Comp List⟩⟨Else Clause⟩ | ⟨If Clause⟩ | ⟨When Clause⟩⟨Cond Comp List⟩ | ⟨Empty⟩	3-6
⟨If Clause⟩ ::= IF ⟨Condition⟩	3-8
⟨Condition⟩ ::= ⟨Not⟩⟨Cond⟩	3-8
⟨Cond⟩ ::= LST | MST | AOV | ABT | COV | SAI | RDC | EX1 | EX2 | EX3 | ⟨Cond Adjust Bit⟩	3-8
⟨Cond Comp List⟩ ::= THEN ⟨Component List⟩	3-6
⟨Else Clause⟩ ::= ELSE ⟨Successor⟩ | ⟨Empty⟩	3-8
⟨When Clause⟩ ::= WHEN ⟨Condition⟩	3-8
⟨End Line⟩ ::= END	3-22

APPENDIX B

RESERVED WORDS AND TERMINAL CHARACTERS

RESERVED WORDS

The following words are reserved in TRANSLANG and may not be used as labels. Some entries have usage limitations:

* Obsolete words not contained in Section 3 but are reserved to support earlier versions of the language.

# Words only used in the microtranslator for port select unit systems. Synonyms producing the same nano-codes are given for the microtranslator for switch interlock systems.

@ Words only used in the microtranslator for switch interlock systems. Synonyms producing the same nano-codes are given for the microtranslator for port select unit systems.

The synonyms do <u>not</u> imply the identical actions.

		Reference Page
A *	Zero 0 as X Select. Use ⟨ Empty ⟩.	
A0 *	Zero 0 as X Select. Use ⟨ Empty ⟩.	
A1	A1 Register X Select or destination operator.	3-15,3-17,3-18
A2	A2 Register X Select or destination operator.	3-15,3-17,3-18

		Reference Page
A3	A3 Register X Select or destination operator.	3-15,3-17,3-18
AAD	And Add logic operator: X AAD Y← →X+(XY)	3-16,3-17
ABT	All Bits True or Adder Bit Transmit dynamic condition from phase 3 of prior M-instruction doing Adder Op.	3-8,3-11
ADD *	Addition logic operator: use X + Y	
ADL *	Add + 1 logic operator: use X + Y + 1	
ADR	Starting address for microsequence.	3-21
AMPCR	Alternate Microprogram Count Register Y Select or destination operator from barrel switch 12 LS bits.	3-4,3-15,3-17,3-18,3-21
AND	And logical operator: X AND Y← →XY	3-16
AOV	Adder overflow, dynamic condition of previous M-instruction using adder, true if addition results in overflow.	3-8,3-9
ASE #	Status request for highest priority unlocked device. Synonym is DL2.	3-11,3-12
ASR #	Status request for highest priority locked device. Synonym is DL1.	3-11,3-12
B	B Register Y Select same as BTTT; or To B from barrel switch, destination operator.	3-15,3-17,3-18
BAD	To B from adder, destination operator.	3-17,3-18
BBA	To B from adder OR barrel switch, destination operator.	3-17,3-19
BBE	To B from external bus OR barrel switch, destination operator.	3-17,3-19
BBI	To B from prior MIR contents OR barrel switch, destination operator.	3-17,3-19
BC4	To B from adder "not 4 bit carry" replicated and shifted, destination operator.	3-17,3-19
BC8	To B from adder "not 8 bit carry" replicated and shifted, destination operator.	3-17,3-19
BEX	To B from external bus, destination operator.	3-17,3-18
BMI	To B from prior MIR contents, destination operator.	3-17,3-19
BR1	To Base Register 1 from barrel switch 2nd LS Byte, destination operator.	3-17,3-18

		Reference Page
BR2	To Base Register 2 from barrel switch 2nd LS Byte, destination operator.	3-17,3-18
BSW *	To B from barrel switch, destination operator; use B instead.	
C	Circular shift right the entire adder output. Operation takes place in barrel switch.	3-15,3-16,3-17
CAD *	Character add by carry inhibit between 8 bit characters (bytes). Use IC instead.	
CALL	Call a procedure: Use AMPCR + 1 as address, and new MPCR; old MPCR to AMPCR. Successor.	3-19,3-20,3-21
COMMENT	Allows for the inclusion of documentation on a listing.	3-3
COMP	Complement as appropriate for destination of literal assignment.	3-4
COV	Counter overflow condition bit, reset dominant.	3-8,3-9
CSAR	Complement SAR,destination operator.	3-17,3-19,3-20
CTR	To counter from ones complement of barrel switch LS Byte, destination operator. X or Y Select: into MS Byte.	3-15,3-17,3-18,3-19
DEC *	Decrement by one the specified X select register; use X select - 1.	3-14
DL1 @	Device Lock using BR1/MAR for device ident. Synonym is ASR.	3-11,3-12
DL2 @	Device Lock using BR2/MAR for device ident. Synonym is ASE.	3-11,3-12
DR1	Device Read using BR1/MAR for device ident.	3-11,3-12
DR2	Device Read using BR2/MAR for device ident.	3-11,3-12
DU1	Device Unlock using BR1/MAR for device ident.	3-11,3-12
DU2	Device Unlock using BR2/MAR for device ident.	3-11,3-12
DW1	Device Write using BR1/MAR for device ident.	3-11,3-12
DW2	Device Write using BR2/MAR for device ident.	3-11,3-12
ELSE	Sequential operator prefix to false successor.	3-8
END	Bracket word to end a program.	3-22
EQV	Equivalence logical operator: X EQV $Y \leftrightarrow XY \vee \bar{X}\bar{Y}$	3-16
EXEC	Executes out of sequence: Use AMPCR + 1 as address. Successor.	3-20,3-21

		Reference Page
EXT *	External condition bit externally set, reset by test. Synonym is EX1.	
EX1 @	External condition bit 1 externally set, reset by test. Synonym is EXT.	3-8, 3-9
EX2 @	External condition bit 2 externally set, reset by test. Synonym is SRQ.	3-8, 3-9
EX3 @	External condition bit 3 externally set, reset by test. Synonym is URQ.	3-8, 3-9
F	False gating of B as part of Y Select.	3-15
GC @	Global conditions used with RESET to reset both GC1 and GC2. Synonym is reset GC2.	3-11, 3-13
GC1	Global condition bit 1: may be set by SET GC1 if presently reset in all Interpreters. Tested without resetting.	3-9, 3-11, 3-13
#	Used as RESET GC1, resets GC1. Synonym is SET LC3.	
GC2	Global condition bit 2: may be set by SET GC1 if presently reset in all Interpreters. Tested without resetting.	3-9, 3-11, 3-13
#	Used as RESET GC2, resets GC2. Synonym is RESET GC.	
IC	Inhibit carry between bytes.	3-16
IF	Starts the conditional part of an instruction.	3-8
IMP	Imply logical operator: X IMP Y$\leftarrow\rightarrow\bar{X} \vee Y$	3-16
INC	Increment counter destination operator; set COV when overflowing from all ones to all zeros.	3-17, 3-19
INSERT	Take a copy of the selected program from the library file and insert it in the program.	3-22
INT	Used as SET INT, interrupts all Interpreters. Interrupt Interpreters condition bit: set by any Interpreter, own is reset by testing.	3-9, 3-11, 3-12
IRQ #	Interrupt from locked but unselected device (can be status or data interrupt). Synonym is LC3.	
JUMP	Jump to address in AMPCR + 1 and put that address in MPCR. Successor.	3-20, 3-21
L	Left shift end off the entire adder output, right fill with zeros. Operation takes place in barrel switch.	3-15, 3-16, 3-17

LC1	Local condition bit 1: may be set, or tested which resets.	3-9,3-11,3-13
LC2	Local condition bit 2: may be set, or tested which resets.	3-9,3-11,3-13
LC3 @	Local condition bit 3: may be set, or tested which resets. Synonym for SET LC3 is RESET GC1.	3-9,3-11,3-13
LCTR	Ones complement of the literal register contents will be placed in the counter and COV reset. Destination operator.	3-17,3-18,3-19
LIT	Literal register: may be loaded by a literal assignment. May be source for X select or Y select LS byte, the MAR, and/or CTR.	3-4,3-15
LMAR	Literal register contents will be placed in MAR. Destination operator.	3-17,3-18,3-19
LST	Least significant bit of adder output, dynamic condition from phase 3 of previous M-instruction doing adder op.	3-8,3-9
MAR	Memory address register destination operator: from barrel switch LS Byte.	3-17,3-18,3-19
MAR1 @	Memory address 1 destination operator: same as BR1, MAR	3-17,3-18
MAR2 @	Memory address 2 destination operator: same as BR2, MAR	3-17,3-18
MIR	Memory Information Register destination operator from barrel switch.	3-17,3-18
MR1	Read from memory address BR1/MAR mem dev op.	3-11,3-12
MR2	Read from memory address BR2/MAR mem dev op.	3-11,3-12
MST	Most significant bit of adder output, dynamic condition from phase 3 of previous M-instruction doing adder op.	3-8,3-11
MW1	Write the content of MIR to memory address BR1/MAR mem dev op.	3-11,3-12
MW2	Write the content of MIR to memory address BR2/MAR mem dev op.	3-11,3-12
NAN	Not And logical operator: X NAN $Y \leftarrow\rightarrow \overline{X} \vee \overline{Y}$	3-16
NIM	Not Imply logical operator: X NIM $Y \leftarrow\rightarrow X\overline{Y}$	3-16
NOR	Nor logical operator: X NOR $Y \leftarrow\rightarrow \overline{X}\overline{Y}$	3-16
NOT	Complement monadic or condition operator NOT $X \leftarrow\rightarrow \overline{X}$ Complement Y select for commutative operators	3-8,3-14,3-16

		Reference Page
NRI	Not Reverse Imply logical operator: X NRI Y $\leftrightarrow \overline{X} \vee Y$	3-16
OAD	Or Add logical operator: X OAD Y $\leftrightarrow$ X + (X $\vee$ Y)	3-16, 3-17
OR	Or logical operator: X OR Y $\leftrightarrow$ X $\vee$ Y	3-16
PROGRAM	Bracket word beginning a program.	3-21
R	Right shift end off the entire adder output, left fill with zeros. Operation takes place in barrel switch.	3-16, 3-17
RDC	Read complete bit: set when external data is ready for input to B, reset by testing.	3-8
RESET	Reset the condition bit specified.	3-11, 3-13
RETN	Return: use AMPCR + 2 as address and as new content for MPCR. Successor.	3-20, 3-21
RIM	Reverse Imply logical operator: X RIM Y $\leftrightarrow X \vee \overline{Y}$	3-16
RMI #	Ready MIR bit: set externally when data has been received from MIR. Reset by testing. Synonym is SAI.	
SAI @	Switch Interlock accepts information bit. Set when switch interlock accepts information, reset by testing.	3-8
SAR	Shift Amount Register destination operator from LS bits of barrel switch or from literal assignment.	3-4, 3-17, 3-18, 3-19, 3-20
SAVE	Save the MPCR in AMPCR: use MPCR + 1 as M-address and as next MPCR. Successor.	3-19, 3-20, 3-21
SET	Set the conditional bit specified: either LC1, LC2, (or LC3@), INT, or request setting of GC1 or GC2.	3-11, 3-12
SKIP	Skip the next instruction; use MPCR + 2 as M-address and as next MPCR. Successor.	3-20, 3-21
SLIT	Literal assignment in SAR converted form.	3-4
SRQ #	Solicited request bit. Set externally, reset by testing. Synonym is EX2.	
STEP	Step to next instruction: use MPCR + 1 as M-address and as next MPCR. Successor.	3-20, 3-21
SUB #	Subtract logic operator: Use X - Y	
SUBL #	Subtract -1 logic operator: Use X - Y - 1	
T	True gating for B register	3-15

		Reference Page
THEN	Starts the true alternative of conditional instruction.	3-6
URQ #	Unsolicited request bit. Set externally, reset by testing. Synonym is EX3.	
WAIT	Wait for condition M-address is MPCR; MPCR and AMPCR unchanged. Successor.	3-20, 3-21
WHEN @	Starts a conditional instruction, has an implicit ELSE WAIT.	3-8
XOR	Exclusive Or logical operator: X XOR Y $\leftarrow\rightarrow X \cdot \overline{Y} \vee \overline{X}\ Y$	3-16
Z	CTR in MS Byte, (ZEXT in middle bytes of machine larger than 2 bytes), LIT in LS Byte as X select and/or Y select.	3-15
ZEXT	Middle bytes of machine larger than 2 bytes as X select and/or Y select.	3-15

Some reserved words have been deleted that were defined in the original description of the microtranslator.

A Microtranslator for the Interpreter Based Systems, S. Zucker, Burroughs TR 70-1, Revision A, February 10, 1970.

These include:

C ZERO, DLR, DLW, DRF, DUR, DUW, DWT, RMA, SK1, SK2, TRY1, TRY2, TRY3, TRY4, and TRY5.

TERMINAL CHARACTERS

		Reference Page
,	Secondary assignment operator for destination operators.	3-2
;	Delimiter. Use is mandatory after a comment statement.	3-2
:	Terminator of label part of instruction or insert.	3-2
=:	Assignment operator for literal assignments or destination list.	3-2
+	Commut operator.	3-2
-	Non commut operator.	3-2
*	Label constant separator.	3-2
(	Prefix delimiter for redundant part of instruction.	3-2
)	Suffix delimiter for redundant part of instruction.	3-2
%	Line terminator and inline comment prefix.	3-3
"	Bracket about comment character used as a literal.	3-3

APPENDIX C
ERROR MESSAGES

The first section of the Microtranslator parses the input file, a line at a time, and produces a listing of the file, N-instructions, and error messages. The error messages indicate that errors were made in the syntax or semantics of an instruction. They will be printed out on the teletypewriter in the following format giving the error number and the line number of the instruction as follows:

```
****ERROR NUMBER NNN IN LINE LLL****
```

Where NNN is the error number and LLL is the sequence number of the instruction in the input file.

When an error is located in an instruction, the error message is printed on the teletype and the next instruction in sequence is passed. If the program is not syntactically correct, the second section of the Microtranslator is not started.

Error Number	Definition
1	Label too large
2	CTR and MAR Conflict
3	Duplicate MAR
4	Duplicate B destination
5	Missing comma
6	Missing semicolon

Error Number	Definition
7	Incorrect destination designator
8	Symbol undefined
9	Duplicate logical operator
10	Logic operator error
11	Colon Equal missing or misplaced
12	Duplicate Z select
13	Duplicate A select
14	Duplicate B select
15	B Gating error
16	Duplicate counter operations
17	More than one set operation
18	Reset error
19	Memory device error
20	Duplicate shift operation
21	Duplicate test condition
22	Duplicate successors
23	Successor error
24	Successor after ELSE error
25	Duplicate label
26	Literal used not in a literal assignment instruction
27	Condition error
28	Misplaced THEN
29	Misplaced ELSE
30	Misplaced integer
31	Integer too large
32	Too many quoted characters
33	Wrong register for receiving a literal
34	Undefined input mistaken for label, or misspelled reserved word
35	Address wanted for insert program less than current address, or misspelled reserved word
36	Reset not followed by proper identifier

Error Number	Definition
37	Set not followed by proper identifier
38	Undeclared label
40	Stack, operation removed, AMPCR goes directly to adder.
41	NOT error — "NOT" misused
61	Named insert program not on library
62	No END on file
63	Address error — present address > insert address

If a nanotable name is requested which has never been saved before, NO SUCH NANOTABLE is printed and a new name requested.

If a new nanotable is given a name already in use, DUPLICATE NANOTABLE NAME ERROR is printed and a new name is requested.

If labels have been used in a program without being declared, the following print-out occurs upon conclusion of the listings.

```
LAST ADR  LABELS NOT FOUND

   2          STR
  3B          SERROR
   4          Y10
```

The address is the hexadecimal micro address of the last instruction using the label in a program.

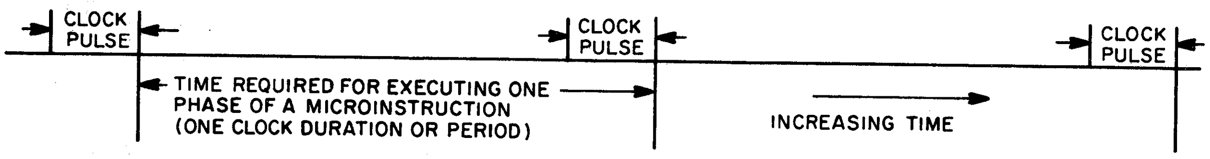
APPENDIX D

INTERPRETER INSTRUCTION PHASING AND CONTROLS

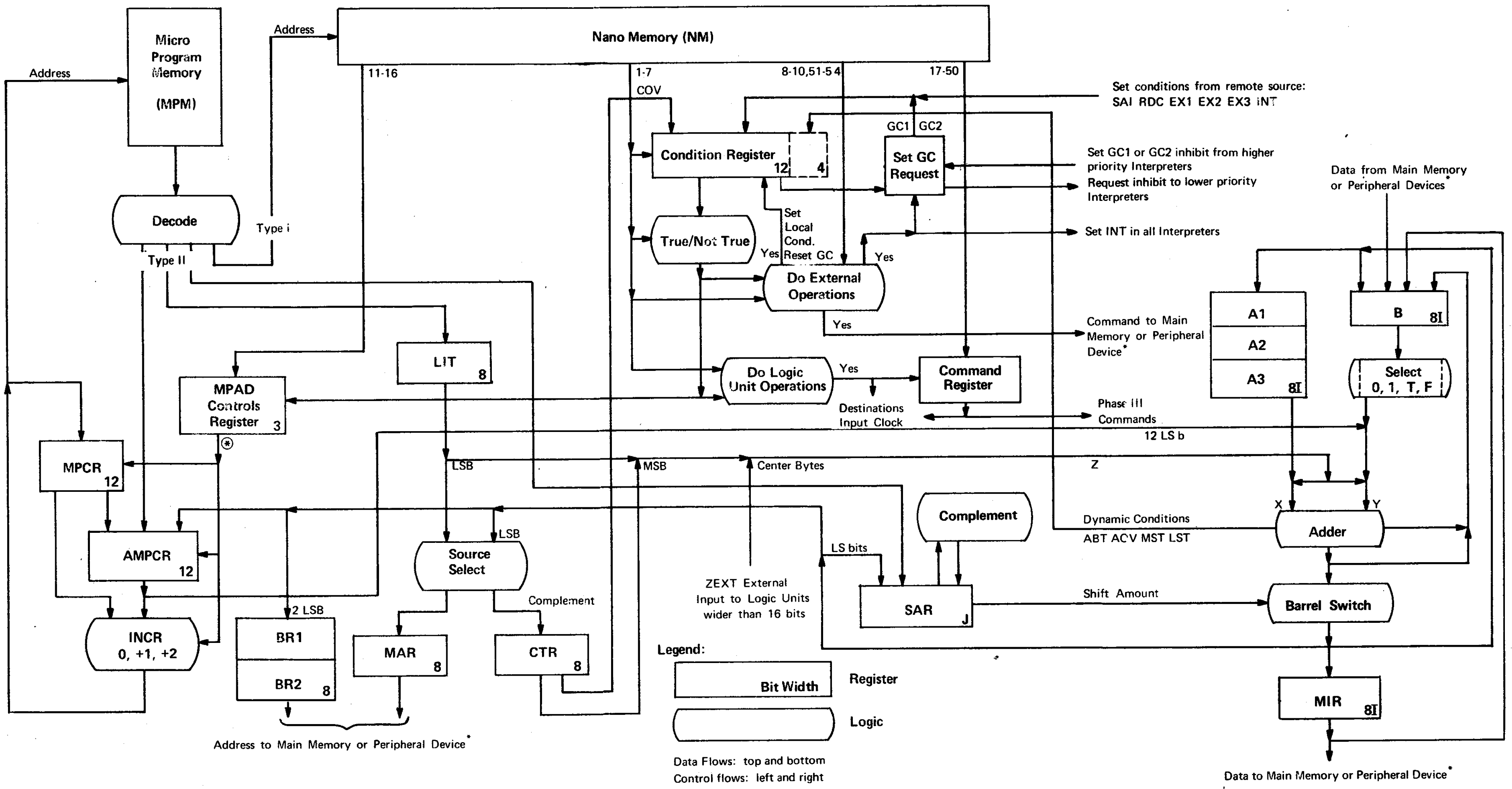
Time phasing of instructions is described in this appendix, both in terms of the partial order of occurrence of the various events and in terms of a sequence of instructions. Controls available in the Interpreter that can be generated by TRANSLANG are reflected in the contents of the words of either the M-memory or the decoder (N-memory). Figure D-1 illustrates the data and control flow among the registers of an Interpreter. The meaning of the content of microprogram words and nanomemory words is detailed. This appendix concludes with a description of memory and device operations as used with a switch interlock.

INSTRUCTION PHASING

The execution of a microinstruction requires one or more sequential time periods, called phase 1, phase 2, and phase 3. The constant interval of time from the end of one clock pulse to the end of the next is the measure of a phase.


CLOCK PULSE
CLOCK PULSE
CLOCK PULSE
TIME REQUIRED FOR EXECUTING ONE PHASE OF A MICROINSTRUCTION (ONE CLOCK DURATION OR PERIOD)
INCREASING TIME

Some microinstructions only have a phase 1, some have both phase 1 and phase 3, and some have phase 1, phase 2, and phase 3. Phases of successive microinstructions usually overlap, so that phase 1 of a current microinstruction is being executed while phase 2 or 3 of a prior microinstruction is also being executed. This overlapping of microinstruction phases allows starting the execution of a new microinstruction each time a new "clock duration" period starts.


Micro Program Memory (MPM)
Address
Nano Memory (NM)
Address
11-16
1-7
COV
8-10,51-54
17-50
Set conditions from remote source: SAI RDC EX1 EX2 EX3 INT
Condition Register 12 4
GC1 GC2
Set GC Request
Set GC1 or GC2 inhibit from higher priority Interpreters
Request inhibit to lower priority Interpreters
Set INT in all Interpreters
Decode
Type i
Type II
True/Not True
Set Local Cond. Reset GC
Yes
Do External Operations
Yes
Yes
Command to Main Memory or Peripheral Device*
LIT 8
MPAD Controls Register 3
Do Logic Unit Operations
Yes
Command Register
Destinations Input Clock
Phase III Commands
Data from Main Memory or Peripheral Devices*
A1
A2
A3 8I
B 8I
Select 0, 1, T, F
12 LS b
MPCR 12
LSB
MSB
Center Bytes
Z
AMPCR 12
LSB
Source Select
Complement
LS bits
Dynamic Conditions ABT ACV MST LST
X
Y
Adder
ZEXT External Input to Logic Units wider than 16 bits
SAR J
Shift Amount
Barrel Switch
INCR 0, +1, +2
2 LSB
BR1
BR2 8
MAR 8
Complement
CTR 8
Legend:
Bit Width
Register
Logic
Data Flows: top and bottom
Control flows: left and right
MIR 8I
Address to Main Memory or Peripheral Device*
Data to Main Memory or Peripheral Device*

⊛SUCCESSOR	Select	Incr.	
WAIT	MPCR	0	
STEP	MPCR	1	Provided for Type II and unless otherwise specified, Type I
SKIP	MPCR	2	
CALL	AMPCR	1	Also saves MPCR contents in AMPCR (for return)
JUMP	AMPCR	1	
RETN	AMPCR	2	
EXEC	AMPCR	1	Inhibits loading of MPCR
SAVE	MPCR	1	Saves MPCR contents in AMPCR (for looping)

REGISTER NAMES

MPCR	Microprogram Count Register
AMPCR	Alternate Microprogram Count Register
BR1	Base Register 1
BR2	Base Register 2
MAR	Memory Address Register
CTR	Counter
SAR	Shift Amount Register
A1	A1 Register
A2	A2 Register
A3	A3 Register

NOTES

* Communication with Main Memory or Peripheral Devices is via Port Select Unit or Switching Interlock

Abbreviations:

LSB	=	Least Significant Byte
MSB	=	Most Significant Byte
2 LSB	=	Second LSB
12 LSb	=	12 Least Significant bits
I	=	Byte width of Logic Unit
J	=	Least integer ≥ $3 + \log_2 I$

Figure D-1. Interpreter Data and Control Flow

A microinstruction may contain either a constant (type II microinstruction) or the address of a nanoinstruction (type I microinstruction). For a type II microinstruction, phase 1 provides sufficient time to execute the instruction (complete the STEP successor and literal assignment), and no additional phases are required. For a type I microinstruction, the events taking place in each of the three phases are identified below.

Phase 1:	Condition test, (conditional) external operation execution, (conditional) logic operation initiation after completion of prior logic operation, and successor microprogram address control.
Phase 2:	Holding phase for logic operation phase 3 controls.
Phase 3:	Completion phase for performing logic unit operations and changing destination registers specified in the logic operation.

If a type I microinstruction does not require the initiation of a logic operation (the condition fails and the logic operation was conditional), the execution is completed in phase 1. Otherwise phase 3 is initiated at the end of the clock duration period by loading the command register concurrently with changing the destination registers for the phase 3 also in process from a prior type I microinstruction. Registers change state during the time a clock pulse is actually present (at the end of a clock duration period).

Phase 3 completes the execution of a logic unit operation. The commands for phase 3 in the command register have two parts: logic specification and destination specification. The logic specification commands apply continously and are taken directly from the command register. The destination specification commands are always executed at the same clock pulse time as the phase 1 initiating a new logic operation.

Phase 2 is a holding phase, the existence of which depends only on subsequent microinstructions. A one-clock duration period hold is created by each subsequent type II microinstruction, or by each type I microinstruction for which the conditional logic unit operation is not to be executed. A phase 2 is created from the original phase 3, which is extended into the next clock duration period as the new phase 3. During phase 2 the original phase 3 logic specification commands continue to apply. Thus the current contents of the selected adder source registers are used to develop adder outputs. The dynamic conditions AOV, ABT, MST, and LST from these adder outputs are available to be tested in a concurrent phase 1.

The phased execution of a sequence of microinstructions is suggested in Figure D-2, with each microinstruction being symbolically represented by a capital letter. Subscripts indicate phase. The subscript 3, 2 indicates a phase 2 that was formerly an "original" phase 3.

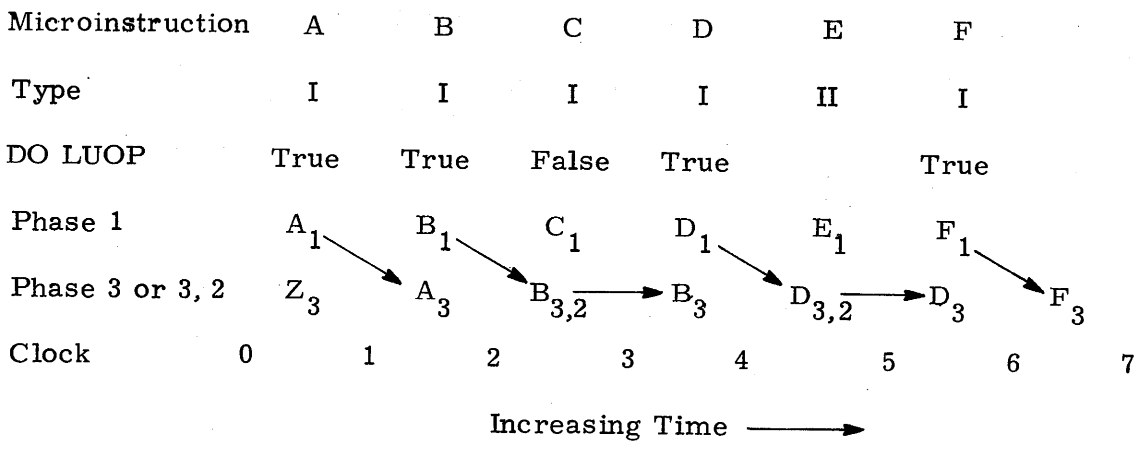

Microinstruction A B C D E F
Type I I I I II I
DO LUOP True True False True True
Phase 1 A_1 B_1 C_1 D_1 E_1 F_1
Phase 3 or 3, 2 Z_3 A_3 $B_{3,2}$ B_3 $D_{3,2}$ D_3 F_3
Clock 0 1 2 3 4 5 6 7
Increasing Time

Figure D-2. Example of Phased Execution of Microinstructions

The phase 3 or 3, 2 in progress is determined by the logic unit operation (LUOP). As indicated in Figure D-2, the old phase 3 is completed and the next clock initiates the new phase 3 (microinstructions A, B, D, and F) if the LUOP is true for type I microinstructions. The old phase 3 is extended (it turned out to be a phase 2) if LUOP is false (microinstruction C creates a phase 2 for B), or if it is type II (microinstruction E), creating a phase 2 for D. This latter case shows how a change by a type II microinstruction can affect the result of a prior type I microinstruction. The destinations for D do not get their new values until the end of phase 1 for F at clock 6, and thus are enabled to use register values that come into existence after clock 5 has expired.

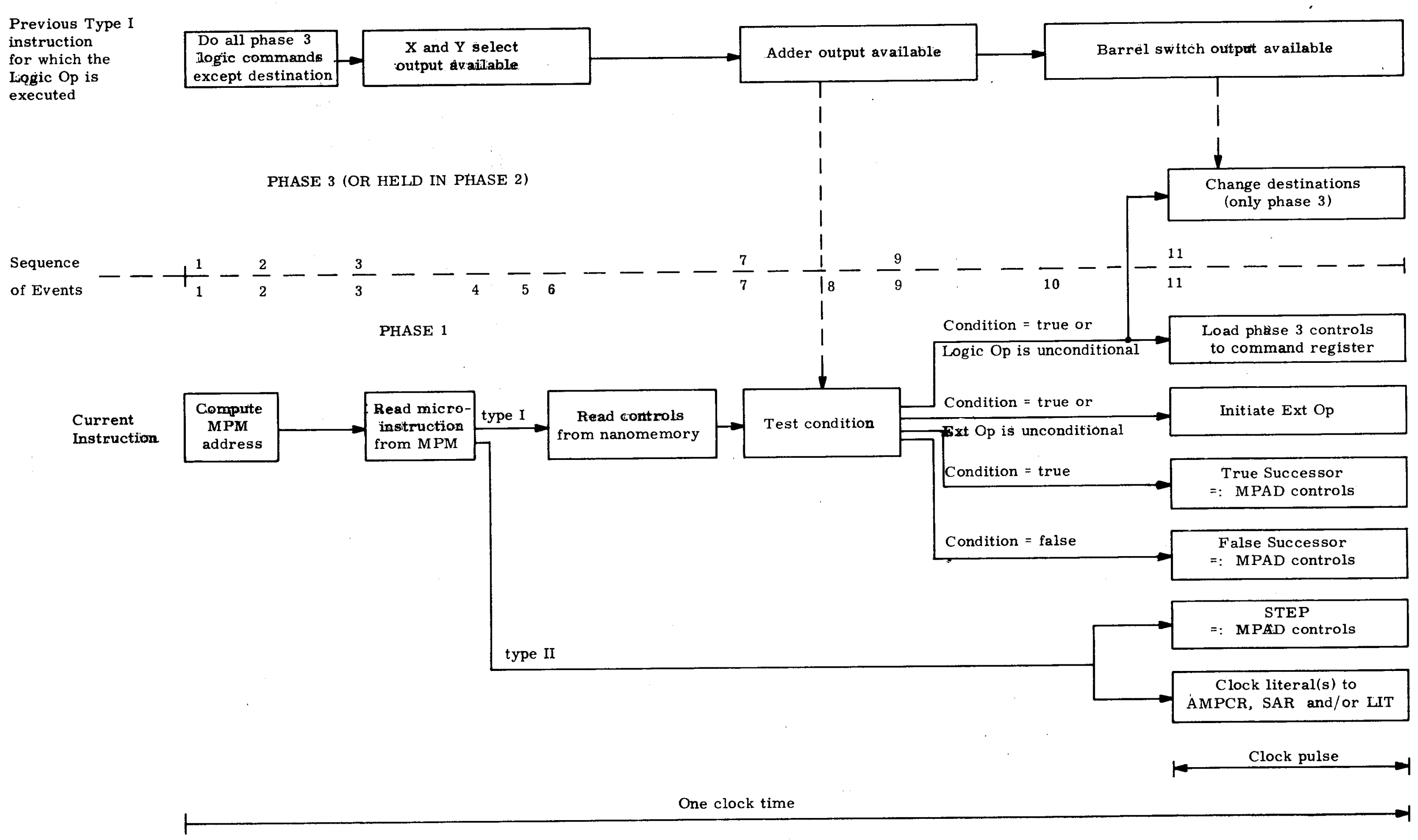
Figure D-3 illustrates instruction phasing and overlap by a time flow and decision diagram. Time flow is from left to right, representing one full clock duration period. The bottom section shows phase 1, the activity of the current micro-instruction. Multiple lines from a box indicate alternatives (not necessarily mutually exclusive). The top section shows the phase 3 commands of a prior microinstruction that are in progress. The phase 3 may turn out to have actually been a hold in phase 2 if the command register does not change. (LUOP was false).

The following events occur in phase 1 in order of ascending numbers. The sequence is logical and is not strictly uniform in time increment.

1. Develop microprogram address MPAD, using MPAD CTLS register content to select either MPCR or AMPCR content, and

2. Select the proper increment amount (+0, +1 or +2).

3. Read out the addressed microprogram word.



Previous Type I instruction for which the Logic Op is executed
Do all phase 3 logic commands except destination
X and Y select output available
Adder output available
Barrel switch output available
PHASE 3 (OR HELD IN PHASE 2)
Change destinations (only phase 3)
Sequence of Events
1
2
3
7
9
11
1
2
3
4
5
6
7
8
9
10
11
PHASE 1
Condition = true or Logic Op is unconditional
Load phase 3 controls to command register
Current Instruction
Compute MPM address
Read micro-instruction from MPM
type I
Read controls from nanomemory
Test condition
Condition = true or Ext Op is unconditional
Initiate Ext Op
Condition = true
True Successor =: MPAD controls
Condition = false
False Successor =: MPAD controls
STEP =: MPAD controls
type II
Clock literal(s) to AMPCR, SAR and/or LIT
Clock pulse
One clock time

Figure D-3. Instruction Time Flow and Decision Diagram

4. Decode the word to determine if microinstruction is type II or type I.

If the microinstruction is type II:

5a. Use low order part of word as literal(s).

11a. STEP successor to MPAD CTLS register.

11b. Clock literal(s) to specified register(s): SAR and/or LIT; AMPCR. (Note that all register changes occur in step 11.)

If the microinstruction is type I:

5b. Use low order part of word as address to nanomemory.

6. Read nanomemory.

7. Decode result.

8. N [1-4] Select condition to test

9. N [5] True/complement condition bit value =: SC

10a. N [6] Do logic unit operations =: LUOP

10b. N [7] Do external operations =: EXTOP

11c. If EXTOP is true then:

   N [8 - 10] enable condition adjust if not 0 0 0, and

   N [51 - 55] enable memory device operation if not 0 0 0 0.

11d. If LUOP is true then:

   complete destination part of phase 3 of prior logic unit operations (bits 34-50) and

   N [17 - 50] decode and load command register.

11e. Successor to MPAD CTLS register:

   N [11 - 13] if SC is true; or

   N [14 - 16] if SC is false

11f. Reset tested condition if appropriate.

The event sequence in phase 2 or 3 starts as follows, where numbers correspond to the sequence for phase 1.

1a. N [17 - 19] Select X input to adder

1b. N [20 - 26] Select Y input to adder

3a. N [27] Inhibit carry

3b. N [28 - 31] Select and do adder operation

7. At this point the dynamic conditions from the adder are available for test in the subsequent instruction now in phase 1 (in its step 8).

9. N [32 - 33] Select shift direction and do shift.

At the end of phase 3 the following events occur:

11g. When LUOP is true (step 10a of same phase 1), any or all of the following independent register changing destination events may occur while the clock pulse is present.

N [34 - 36]	A registers	
N [37 - 40]	B register	
N [41]	MIR	
N [42]	AMPCR	
N [43]	BR1	
N [44]	BR2	
N [45 - 46]	MAR	not totally independent, since they share N [46]
N [46 - 48]	CTR	
N [49 - 50]	SAR	

11h. When LUOP is false, this was a phase 2.

MICROPROGRAM WORD CONTENT

M-word Bits																Instruction	Literal
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Type	Assignment
0	Ø	a	b	SAR				Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	II	k=:SAR
1	0	a	b	SAR				LIT								II	k=:SAR; j=:LIT #
1	1	0	Ø	*	AMPCR											II	i=:AMPCR
1	1	1	0	Ø	Ø	Ø	Ø	LIT								II	j=:LIT #
1	1	1	1	*	N-ADDRESS											I	

Heavy bars indicate possible contraction points for narrower memories, in which case the bits are moved to the left.

Ø indicates unused, 0 supplied by TRANSLANG.
* indicates a field that is right justified if the hardware configuration does not require the entire addressing range, left fill with zeros.
# wherever LIT appears may be replaced by SLIT meaning convert the constant as if it were being loaded into SAR, and left fill with zeros.
a required only for logic units widths greater than 32 bits.
b required only for logic unit widths greater than 16 bits.

The M-memory word may have bit length sufficient for the particular Interpreter hardware configuration. Logic units of 24-bit or 32-bit width need 15-bit M-memory words; wider logic units need 16-bit M-memory words. The design of the M word decoding is flexible. It permits selecting bit groups of the following lengths to form alternatives. One of the columns below will be chosen for each configuration.

Number of Decoded Bits			
2	1	2	1
2	2	2	3
2	3	2	3
3	4	2	3
3	4		3

For the initial 16-bit logic unit configuration, the choices were made as follows:

M-memory word length	=	14 bits
Decoded bit option	=	1, 2, 3, 4, 4

N-WORD CONTENT

The assignment of bits and their meaning in the N-decoder or N-memory is summarized in Table D-1, and is described in detail below. Bit positions in the memory are indicated by integers in boxes. Each box surrounds a field of related bits. The defined alternatives are described, and mnemonics given for each. The mnemonics that directly correspond to TRANSLANG reserved words are identical.

Other mnemonics are provided for descriptive references.

<u>LEGEND</u>:

+	ADD (twos complement)
-	SUBTRACT (twos complement)
v	OR (logical inclusive)
$\overline{N}$	NOT N (ones complement)
--	Don't care, 0 or 1
=:	Assign into
MS	Most significant
LS	Least significant

Table D-1. Nanomemory Decoding

TIMING AND GENERAL ACTION	N–MEMORY BITS	SPECIFIC ACTION
DURING PHASE 1		
Conditional Control	1-4	Condition selection
	5	Condition test (true/complement)
	6	Conditionally update command register from bits 17-50 of nanomemory
	7	Conditionally initiate actions shown below under "at end of Phase I"
AT END OF PHASE 1		
(a) Successor Determination	11-16	Microprogram address (MPAD) controls
(b) External Operations	8-10	Condition adjust (local; global; interrupt Interpreters)
	51-54	Request signals for main memory or peripheral device operations
PHASE 2		
Optional Holding Phase		Dynamic conditions available for test in Phase I
PHASE 3		
Adder Operation Commands	17-19	Adder input X select
	20-26	Adder input Y select
	27	Inhibit carries
	28-31	Adder or logic operation
		Dynamic conditions available for test in concurrent phase 1
	32-33	Shift (right, left, circular) by amount in SAR
AT END OF PHASE 3		
Destination Specification	34-36	Input to A registers (A1, A2, A3) from BSW
	37-40	B register input source selection
	41	MIR input from BSW
	42	AMPCR input from BSW (Input clock commands)
	43	BR1 input from BSW (Input clock commands)
	44	BR2 input from BSW (Input clock commands)
	45-46	MAR input from BSW or LIT (Input clock commands)
	46-48	CTR input from LIT, BSW, or increment CTR (Input clock commands)
	49-50	SAR input from BSW, or complement SAR (Input clock commands)

Phase 1 Controls

Controls N [1 - 7] are used directly from the N-memory and are effective before the end of the first clock (phase 1).

1	2	3	4	CONDITION SELECTION			How Set@	How Reset@	Dominant
0	0	0	0	GC1	Global condition 1	If not set in another Interpreter.	CAJ	CAJ	S
0	0	0	1	GC2	Global condition 2		CAJ	CAJ	S
0	0	1	0	LC1	Local condition 1		CAJ	Test	S
0	0	1	1	LC2	Local condition 2		CAJ	Test	S
0	1	0	0	MST	Adder most significant bit*		-	-	-
0	1	0	1	LST	Adder least significant bit*		-	-	-
0	1	1	0	ABT	Adder bit transmit		-	-	-
0	1	1	1	AOV	Adder overflow		-	-	-
1	0	0	0	COV	Counter overflow		Overflow	Test	R
1	0	0	1	SAI	Switch interlock accepts info		SWI	Test	S
1	0	1	0	RDC	Read complete		SWI	Test	S
1	0	1	1	LC3	Local condition 3#		CAJ	Test	S
1	1	0	0	EX1	External condition 1		EXT	Test	S
1	1	0	1	INT	Interrupt Interpreters**		CAJ	Test	S
1	1	1	0	EX2	External condition 2		EXT	Test	S
1	1	1	1	EX3	External condition 3		EXT	Test	S

*MST and LST in the hardware are true if the value is 0. The Microtranslator complements the programmer-specified test for these so the test is as if the true value were 1, consistent with the other conditions.

**Set from own or another Interpreter. Testing only resets own Interpreter.

@ CAJ is condition adjust N [8 - 10]
SWI is switch interlock
EXT is external source
Test is by inclusion of the selected condition in a type I microinstruction
Dominant if both set and test:
S is set to 1
R is reset to 0

#LC3 is RMA in the DC 2000.

$\boxed{5}$ COMPLEMENT/TRUE CONDITION TEST

0	NOT	Complement value of selected condition =: SC
1		Value of selected condition =: SC

$\boxed{6}$ LOGIC UNIT CONDITIONAL

If LUOP resulting from this control is 0, do not change command register; otherwise at end of this clock, complete the phase 3 for the prior instruction in the command register and replace its content from controls N [17-50].

0	Unconditionally	TRUE=: LUOP
1	Conditionally	SC=: LUOP

$\boxed{7}$ EXTERNAL OPERATIONS CONDITIONAL

If EXTOP resulting from this control is 0, do nothing; otherwise in this clock initiate any specified memory/device operation N [51 - 54] and adjust any specified condition N [8 - 10].

0	Unconditionally	TRUE=: EXTOP
1	Conditionally	=: EXTOP

$\boxed{8 \quad 9 \quad 10}$ CONDITION ADJUST

The indicated action takes place at the end of phase 1 if EXTOP has been determined to be true in this phase 1. Bits are set to true or 1; reset to false or 0.

0	0	0		No action
0	0	1	SET LC2	Set local condition 2
0	1	0	SET GC2	Set global condition 2 request*
0	1	1	RESET GC	Reset global conditions 1 and 2
1	0	0	SET INT	Set interrupt Interpreters in all Interpreters including own
1	0	1	SET LC3	Set local condition 3
1	1	0	SET GC1	Set global condition 1 request*
1	1	1	SET LC1	Set local condition 1

* A request to set a global condition bit is only honored if no higher priority Interpreter has its own correspondingly numbered global condition bit set, or is concurrently requesting to set it. In the second executed instruction following the request to set the global condition bit, a test of that bit will indicate (if set) that the request was successful, otherwise that some other Interpreter is using that global condition. A request lasts for one clock.

MPAD Controls

The MPAD (microprogram address) is determined by the value in the MPAD controls register at the start of phase 1. Depending on the value of SC determined during phase 1, either one of the following two sets of controls is loaded into the MPAD controls register at the end of phase 1. Concurrently, changes to the MPCR and AMPCR occur as indicated by the original content (at the start of phase 1) of the MPAD controls register (and for the AMPCR, possibly a type II or barrel switch output). $MPCR_0$ is the value in the MPCR before the end of phase 1. For type II instructions, MPAD becomes 1 + $MPCR_0$ and MPCR becomes $MPAD_0$--"STEP".

11 12 13 TRUE SUCCESSOR Used if SC=1	MPAD controls register	MPM address will be	Registers Changed at end of Phase 1: MPCR receives value	AMPCR receives value	14 15 16 FALSE SUCCESSOR Used if SC=0
0 0 0	WAIT	MPCR	MPAD		0 0 0
0 0 1	STEP	1+MPCR	MPAD		0 0 1
0 1 0	SAVE	1+MPCR	MPAD	$MPCR_0$*	0 1 0
0 1 1	SKIP	2+MPCR	MPAD		0 1 1
1 0 0	JUMP	1+AMPCR	MPAD		1 0 0
1 0 1	EXEC	1+AMPCR			1 0 1
1 1 0	CALL	1+AMPCR	MPAD	$MPCR_0$*	1 1 0
1 1 1	RETN	2+AMPCR	MPAD		1 1 1

*CALL and SAVE overrride any change to the AMPCR from either a type II instruction or the BSW. The type II overrides the BSW.

Phase 3 Controls

Controls N [17 - 50] are partially decoded and stored in the command register at the end of phase 1 if LUOP is true. Beginning with the next clock (regardless of whether the microinstruction is type I or type II) the controls N [17 - 33] become active causing selection of inputs to the adder, the appropriate adder operation, and kind of shift. These controls continue in effect over one or more clocks until next a type I instruction (at the end of its phase 1) changes the command register. Concurrent with this change, the controls previously in the command register, N [34 - 50] are used to specify the desired set of destination registers to receive new values.

It is thus possible that subsequent type II instructions will cause changes to the result of the logic unit operation specified in the command register. These changes may occur if either the literal register or AMPCR is an input to the adder. These changes may affect the values of the adder dynamic conditions MST, LST, ABT, or AOV. Also if a shift is specified, a change to the SAR will change the amount of the shift and thus change the barrel switch output.

17	18	19	ADDER INPUT X SELECT	
0	0	0		Zeros
0	0	1	LIT	Literal register to LS byte*
0	1	0	ZEXT	Z external connection to center bytes*#
0	1	1	CTR	Counter to MS byte*
1	0	0	Z	CTR v ZEXT v LIT #
1	0	1	A1	A1 register
1	1	0	A2	A2 register
1	1	1	A3	A3 register

* Zeros elsewhere
ZEXT only meaningful for logic unit widths greater than 16 bits.

20	21	22	23	24	25	26	ADDER INPUT Y SELECT	
0	0						B0 --	0 in MS bit
0	1						BT--	B MS bit in MS bit
1	0						BF--	$\overline{B}$ MS bit in MS bit
1	1						B1 --	1 in MS bit
		0	0	0			B-0-	0 in center bits
		1	0	0			B-T-	B in center bits
					0	0	B--0	0 in LS bit
					0	1	B--T	B LS bit in LS bit
					1	0	B--F	$\overline{B}$ LS bit in LS bit
					1	1	B--1	1 in LS bit
comp		1	0	0	comp		B-F-	$
comp		0	0	0	comp		B-1-	$
0	0	0	0	1	0	1	LIT	Literal register to LS byte*
0	0	0	1	0	0	0	ZEXT	Z external connection to center bytes*#
0	1	0	1	1	0	0	CTR	Counter to MS byte*
0	1	1	0	1	0	1	Z	CTR v ZEXT v LIT #
0	0	1	1	0	0	1	AMPCR	AMPCR in least 12 bits*
0	1	1	1	1	0	1	L0	

* Zeros elsewhere

ZEXT only meaningful for logic unit widths greater than 16 bits.

$ Center selection of B gating as F or 1 is achieved by using: the Y complement operator (see Appendix E) and comp for MS bit and LS bit gating (means 0 for 1, B for $\overline{B}$, and vice versa). No complement operators exist for OAD and AAD. These corrections are done by the Microtranslator.

27	INHIBIT CARRIES
0	Allow carries
1	IC̆ Inhibit carries into bytes*

* The behavior of ABT when carries are inhibited is unpredictable. When the carries into bytes are inhibited, the 8-bit carries into the B register are the carries out of a set of 8-bit additions.

28	29	30	31	ADDER OR LOGIC OPERATION (See Appendix E.)	Function	Bitwise Logic	Complement*
0	0	0	0	X + Y			12
0	0	0	1	X NOR Y	$\bar{X}\,\bar{Y}$	$X \not\vee Y$	2
0	0	1	0	X NRI Y	$\bar{X}\,Y$	$X < Y$	1
0	0	1	1	X + Y + 1			15
0	1	0	0	X NAN Y	$\bar{X} \vee \bar{Y}$	$X \not\wedge Y$	8
0	1	0	1	X OAD Y	$X + (X \vee Y)$		none
0	1	1	0	X XOR Y	$(X\,\bar{Y}) \vee (\bar{X}\,Y)$	$X \neq Y$	9
0	1	1	1	X NIM Y	$X\,\bar{Y}$	$X > Y$	11
1	0	0	0	X IMP Y	$\bar{X} \vee Y$	$X \leq Y$	4
1	0	0	1	X EQV Y	$(X\,Y) \vee (\bar{X}\,\bar{Y})$	$X = Y$	6
1	0	1	0	X AAD Y	$X + (X\,Y)$		none
1	0	1	1	X AND Y	$X\,Y$	$X \wedge Y$	7
1	1	0	0	X - Y - 1	$X + \bar{Y}$		0
1	1	0	1	X RIM Y	$X \vee \bar{Y}$	$X \geq Y$	14
1	1	1	0	X OR Y	$X \vee Y$	$X \vee Y$	13
1	1	1	1	X - Y	$X + \bar{Y} + 1$		3

* The complement is the decimal equivalent of the operation for which the Y select is ones complemented.

32 33 SHIFT TYPE SELECTION

The barrel switch (BSW) output is the result of the adder output shifted as indicated by the shift type selection. The shift uses the current content of the shift amount register (SAR) at the start of the last clock of phase 3.

0	0		No shift	LH
0	1	R	Shift right end off, zero fill to left	LH
1	0	L	Shift left end off, zero fill to right*	RH
1	1	C	Shift right circular, all bits	LH V RH

* Actually a right circular shift of the word-length complement of the SAR content with zero fill to the right.

Assume that the shift is to be developed by selection from an ordered set of signals twice the width of the logic unit, with initial value all zeros. Let the two halves of this set be LH and RH, with LH the more significant. The unshifted adder output is aligned to LH. A right shift is performed. The amount of the right shift is that specified in the SAR for R, L, or C; otherwise 0. The resulting shifted adder output is in general now at some intermediate position of the signal set. The last column indicates the single width selection from this signal set used to determine the barrel switch output.

Phase 3 Input Clocks

Results as specified in the command register from bits N [34 - 50] are clocked into selected registers at the end of phase 3. This occurs at the end of phase 1 of the first successor instruction for which LUOP is set to 1 (true).

34	35	36	A REGISTERS INPUT FROM BARREL SWITCH	
0	-	-		A1 unchanged
1	-	-	A1	BSW to A1
-	0	-		A2 unchanged
-	1	-	A2	BSW to A2
-	-	0		A3 unchanged
-	-	1	A3	BSW to A3

37	38	39	40	B REGISTER INPUT SOURCE SELECTION	
0	0	0	0		B unchanged
0	0	0	1	BC4	Complement of 4-bit carries
1	0	0	0	BAD	Adder (unshifted)
1	0	0	1	BC8	Complement of 8-bit carries
1	0	1	0	BBA	BSW $\vee$ Adder
1	0	1	1	B	BSW
1	1	0	0	BEX	External input
1	1	0	1	BMI	Memory Information Register (MIR)
1	1	1	0	BBE	BSW $\vee$ External input
1	1	1	1	BBI	BSW $\vee$ MIR

41	MEMORY INFORMATION REGISTER INPUT	
0		MIR unchanged
1	MIR	BSW to MIR

42	AMPCR INPUT	
0		No change from BSW
1	AMPCR	BSW least bits to AMPCR*

* A conflict in loading AMPCR can occur that will prevent this loading from the BSW. Assume that the phase 3 in progress indicates load AMPCR from BSW. Also assume that the MPAD controls at the same time indicate SAVE or CALL (as a result of the phase 1 prior to the one in progress). Then if the current phase 1 indicates that a new phase 3 should be initiated, the conflict in AMPCR loading is resolved in favor of the old MPCR.

$\boxed{43}$ BR1 INPUT

0		No change
1	BR1	BSW next least byte to BR1

$\boxed{44}$ BR2 INPUT

0		No change
1	BR2	BSW next least byte to BR2

$\boxed{45 \mid 46}$ MAR INPUT

0	-		No change
1	0	LMAR	LIT to MAR
1	1	MAR	BSW least byte to MAR

46 (MAR & COUNTER INPUT SELECT)

0	LIT
1	BSW least byte

$\boxed{46 \mid 47 \quad 48}$ COUNTER INPUT

-	0	0		No change
0	0	1	LCTR	LIT to CTR (ones complement)
1	0	1	CTR	BSW least byte to CTR (ones complement)
-	1	0	INC	Increment CTR (mod 256)

At the end of phase 3, LCTR and CTR reset the COV condition bit, and INC sets the COV upon incrementing from HEX FF to HEX 00 unless the concurrent phase 1 tests COV.

$\boxed{49 \quad 50}$ SAR INPUT

0	0		No change
0	1	CSAR	Complement SAR#
1	0	SAR	BSW least bits*

* The number of bits used is the integer not less than $\log_2$ (logic unit width in bits).

The complement is determined as follows: let S be the shift amount and W be the byte width of the logic unit. Then the most significant 3 bits of the complement are the integer part of S $\div$ W; and the least significant part is the remainder of S $\div$ W, each expressed in binary.

If the phase 3 in progress specifies eventual loading of the SAR from the BSW while a type II instruction attempts to load the SAR, the result to the SAR is the result of the type II.

51	52	53	54	MEMORY AND DEVICE OPERATIONS

The indicated action is initiated if EXTOP has been determined to be 1 prior to the end of this phase 1.

0	0	0	0		No change
0	0	0	1		--
0	0	1	0	MR1	Memory read using MAR1 as address
0	0	1	1	MR2	Memory read using MAR2 as address
0	1	0	0		--
0	1	0	1		--
0	1	1	0	MW1	Memory write from MIR using MAR1 as address
0	1	1	1	MW2	Memory write from MIR using MAR2 as address
1	0	0	0	*DL1 ASR	Device lock request using MAR1 as address Status request for highest priority locked device
1	0	0	1	*DL2 ASE	Device lock request using MAR2 as address Status request for highest priority unlocked device
1	0	1	0	DR1	Device read using MAR1 as address
1	0	1	1	DR2	Device read using MAR2 as address
1	1	0	0	DU1	Device unlock using MAR1 as address
1	1	0	1	DU2	Device unlock using MAR2 as address
1	1	1	0	DW1	Device write from MIR using MAR1 as address
1	1	1	1	DW2	Device write from MIR using MAR2 as address

*Interpreter based systems with a switching interlock use DL1 and DL2, Interpreter based systems with port select units use ASR and ASE.

Interpreter based systems with a switching interlock use the following condition bits for synchronization of activity requests with memory and devices (see the subsequent discussion):

SAI	Switch Interlock Accepts Information
RDC	Read Complete, or Request of Device Complete (only for devices read from or written to by Interpreter request)

In order to safely use these conditions they must be reset by testing before they may be depended upon.

SWITCH INTERLOCK

The switch interlock connects Interpreters with devices and S-memories. Connection with a device is by reservation for exclusive use by an Interpreter and is maintained until released. Connection with an S-memory module is for the duration of a single data word exchange, but is maintained until some other module is requested or some other Interpreter requests that memory module.

The switch interlock is intended to connect many Interpreters to many devices and to many memory modules. It is important to keep the number of wires in the crosspoints to a minimum. Consequently, a variety of combinations of serial and parallel data transmission paths are allowable. The amount of parallelism depends on the bandwidth necessary to complete the transmission in the number of clocks (usually one) required by the system. The serial transmission rate is significantly higher than the Interpreter's clock, and hence many bits can be transferred in one clock time.

For each Interpreter, the switch interlock provides the option of three buffer shift registers when serial transmission is used.

1. Address data for S-memory (and possibly devices) from the specified MAR1 or MAR2. (XDA).
2. Output data from the MIR. (XDO).
3. Input data for assembly and subsequent acceptance into the B register. (XDI). Data in this register may be repeatedly read non-destructively until the next device or memory operation is initiated (the last read may be concurrent with the next operation).

Use of Base Registers and Memory Address Register

The format for base registers and memory address registers is determined for each system configuration. Some field of this information contains a device memory module number, other fields may include word address or other control information; for example, word address, or a bit to distinguish a device to be read on demand or by request.

In the subsequent discussion and examples, the device identification is assumed to be wholely within a base register so that device requests are insensitive to the MAR content. This is not a necessary restriction. The limitation to 16 bits for addressing may be relaxed in larger systems by extending the addressing by adding additional memory control units. This is not further explored in this report.

Device Operations

The device operations include lock (DL), read (DR), write (DW), and unlock (DU). Each device operation uses as a device identification some part of the value in BR1 or BR2 (and possibly in MAR) as indicated in the operation suffix, e.g., DL1. This identification is unbuffered by the switch interlock; consequently it must be maintained until the device operation is completed, or until some other device or memory action is desired. Any change to the device identification while a device operation is in progress breaks the selected path to or from the Interpreter. Unless the normal completion occurs concurrently, the prior device operation is terminated. Any part of the value in BR1 or BR2 (and MAR) may pass through the switch interlock to the device as required.

An Interpreter must be locked to the device with which a read or write is issued. An Interpreter may be locked to several devices. A device can only be locked to one Interpreter at a time. When an Interpreter is finished using a device, it should be unlocked so that some other Interpreter may access it.

A static priority for Interpreters is assigned to resolve device lock conflicts. By changing an Interpreter image via changing its M-memory and possibly N-memory, dynamic priority adjustment is possible among tasks relative to device access.

Device Groups and Interpreter Access Priority for DL and DU

The switch interlock module for device connection contains a group of 8 ports for connecting devices to up to four Interpreters. System expansion using this module may be in number of Interpreters, in number of device groups, and/or in number of parallel connections. DL, DR, DW, and DU are provided for each port. Each group of 8 ports is independent. Priority order for resolving concurrent requests by Interpreters for DL or DU is fixed within each group. This order may differ from group to group.

Conflicts in DL and DU requests may occur. The DL request from the highest priority requesting Interpreter is honored over a co-occurring request for the same device from any lower priority Interpreter. Concurrent DL requests for different devices in the same group cause the lower priority request to incur a one clock delay in achieving the DL or DU, and in return of SAI for each higher priority request. Consequently DL or DU requests from Interpreters other than the highest priority may be arbitrarily delayed. The earliest confirming SAI response occurs 2 instructions after issue of the DL or DU. If SAI is true, then the DL or DU was successful. If SAI is false, then it means that the DL or DU is not yet successful. The design justification for this potential arbitrary delay is that DL or DU are infrequent events for which arbitrary delay is of little consequence.

Provision for conscious control of this timing is provided (and recommended) by use of global condition bit 2 to protect DL and DU attempts by more than one Interpreter at a time. See the example using global condition bits in Appendix F.

The following discussion assumes that no conflict-causing delay will occur.

Duration of Device Operations

The duration of a DL request depends on its success. If successful the lock occurs concurrent with the following instruction, at the end of which SAI is set true. Thus SAI is available for test in the second following instruction. If false at this time the DL request continues while other work may be in progress so long as neither the device identification changes nor another memory or device operation is initiated. When the previously issued DL is successful, SAI will be returned.

Device reads or writes are only completed with devices locked to the Interpreter issuing the DR or DW. Depending on the device address, a DR or DW may be on demand or by request.

> On Demand. DR and DW provide immediate data exchange. The duration of DR or DW on demand is one instruction after issue. Confirmation of completion may be checked by SAI being true the second instruction after issue. If SAI is false, the device was not locked to the requesting Interpreter.
>
> By Request. DR and DW provide data exchange when the device is ready. The duration of DR or DW by request is determined by the device and is signalled to the requesting Interpreter by the return of RDC – "Request of device complete". As with DR and DW on demand, SAI is returned by the second instruction after issue, and indicates that the device is locked to the requesting Interpreter.

The duration of DU is one instruction after issue unless conflicts from DL or DU requests by other Interpreters occur as indicated above. SAI will be returned only if the device had been locked to the requesting Interpreter. SAI is available for test in the second instruction after issue if no conflicts arise. Any conflict with other DL or DU in the same group can cause delay.

Device Use Sequence

The sequence of device operations necessary for an Interpreter to use a device is as follows:

1. A test of "IF SAI" should be included in some instruction to reset it. This usually can be in the instruction with the unconditional device operation.

2. Device Lock Request: The data in the indicated base register (and possibly MAR) is used as the device identification. On the second following instruction, SAI may be tested.

   2.1 If true, then the device lock was successful.

   2.2 If false, then the device lock was unsuccessful. The request remains in progress while other instructions not changing the device identification or issuing other memory or device operation may be executed. The DL request is terminated by the first of the following actions:

      (a) The Interpreter initiates another memory or device operation.

      (b) The Interpreter changes the device identification.

      (c) The device becomes available and sets SAI. All co-occurring actions are valid. Should (a) and (c) co-occur, SAI refers to the DL in the following instruction and should be tested. Then in the next instruction thereafter SAI refers to the new memory or device operation. Should termination by (b) occur without co-occurrence of (c), the new device identification applies to the DL still in progress, and the path for SAI return is diverted to the newly identified device (if there is one so identified) without reissue of another DL.

3. Once the desired device is locked to the Interpreter, a sequence of one or more data exchanges may be initiated using the following. Assume for simplicity that adequate bandwidth connection is provided so that data transfer is completed in one clock. Otherwise add an appropriate number of clock times to the discussion.

4. Device Write: The data in the indicated base register (and possibly MAR) is used to specify the device, and the data in the MIR provides the information to be written to the device. The second instruction after the device write, SAI may be tested. If true, the Interpreter is locked to the device, the data in the MIR has been accepted by the XDO register, and so the MIR may subsequently be changed. If false, the Interpreter was not locked to the requesting device.

   4.1 On Demand: The device is immediately ready to accept input data from the Interpreter. Consequently the SAI need not be checked, and the MIR or device identification could even be changed in the instruction after the DW.

4.2 By Request: The device provides an RDC when it has completed the requested write. The SAI also indicates that the MIR data has been accepted in the switch interlock. Similar to DL, the request continues until the first of the corresponding 3 actions.

(a) The Interpreter initiates another memory or device operation.

(b) The Interpreter changes the device identification.

(c) The DW is completed and sets RDC. All co-occurring actions are valid. Should (a) and (c) co-occur, SAI refers to the DW in the following instruction and should be tested. In the next following instruction SAI then refers to the new memory or device operation. Should (b) not co-occur with (c), then the DW in progress is diverted to apply to the new device identification without reissue of another DL.

4.3 Separate device identifications are required if the same device is to be read both on demand and by request (some distinguishing bit).

5. Device Read: The data in the specified base register and MAR is used to specify the device. The second instruction after the device read, SAI may be tested. If true, the Interpreter is locked to the device; otherwise not.

5.1 On Demand: The device output register is assumed immediately able to be read on demand (possibly some part of the resulting data indicates validity). The data requested is available for Interpreter access the clock after Phase 1. Thus BEX or BBE may be included in the same instruction as the device read. No conflict or holdup will occur since the Interpreter is locked to the device. The SAI need not be checked, and the device identification may be changed in the instruction after the DR (so long as the address is still not required for a prior memory operation or device read by request).

5.2 By Request: The device provides a RDC after the device read request when it has sent the desired data from its output register. Thus the same instruction that finds RDC true may include BEX. RDC should be reset by testing prior to use for device read by request (usually as part of the prior instruction using BEX).

5.3 Separate device identifications are required if the same device is to be read both on demand and by request.

6. Device Unlock: When use of the device is completed the lock should be terminated by issuing a device unlock. An SAI is returned if the issuing Interpreter was locked to the device. An attempt to unlock a device that is not locked to the Interpreter will not return SAI. SAI is available for test at earliest the second instruction after the device unlock.

Memory Operations

The memory operations include read (MR) and write (MW). Each memory operation uses as a memory address some part of the value in MAR1 and MAR2 (BR1 or BR2 concatenated with MAR). A portion of the address specifies a memory module, with the rest indicating locations within the module.

Memory Groups and Interpreter Access Priority

The switch interlock module for memory connection contains a group of 8 ports for connecting memory modules to two Interpreters. System expansion using this module may be in number of Interpreters, in number of memory groups, and/or in degree of parallel connections. Each port provides MR and MW. Each group of 8 ports is independent. Concurrent access to all memories in a group by different Interpreters is permitted. Interpreters have fixed priority for access to all modules of a group. The fixed priority order may differ from group to group.

Conflicts in access to the same module are resolved in favor of the Interpreter that last accessed the module, otherwise the highest priority requesting Interpreter. Once access is granted it continues until that memory operation is complete. When one access is complete, the highest priority request is honored from those Interpreters then in contention. The Interpreter completing access is not able to compete again for one clock. Thus the two highest priority Interpreters are assured of access. Lower priority Interpreters may have their access rate significantly curtailed.

The switch interlock "remembers" the prior connection of each memory module to some Interpreter. If the next request is also from the remembered Interpreter, the new connection is made with less delay, since no priority resolution need take place.

Memory Use Sequence

The sequence of operations necessary to access S-memory is simple in single Interpreter systems where no conflict in access can exist. In such cases once the address setup is complete (as is the MIR for write), the memory read (or write) can be initiated. After a suitable time the data from memory can be accessed via BEX or BBE. In the presence of conflict potential, the following control sequence should be used. This sequence is recommended for systems without a switch interlock as well.

1. The S-memory address should be in the selected base register and MAR.

2. Memory read

   2.1 A test of RDC should be included in some prior instruction. By convention this should be the previous memory read (or device read or write by request). A test of SAI should be included if address register changes are required before the RDC is returned, or if confirmation of access to the switch interlock is desired.

   2.2 The memory read can occur the instruction after the address is (unconditionally) loaded into MAR1 or MAR2.

   2.3 A SAI is returned when the switch interlock has accepted the address and the memory is connected to the requesting Interpreter through the switch interlock.

   2.4 A group of intervening instructions can be issued, depending on the relative speeds of the Interpreter clock and the S-memory. Once SAI is set and tested, these instructions may change the address registers or even include device read or write operation on demand.

   2.5 A RDC (read complete) signal is returned when data will become available for entry into the Interpreter the following clock.

   2.6 If no intervening device or memory reads occur, BEX may be repeated, each time receiving the data in XDI non-destructively.

3. Memory Write

   3.1 The data to be written should be in MIR.

   3.2 The address should be in the selected base register and MAR.

   3.3 The memory read can occur the instruction after both the address and data have the desired values.

3.4 Return of SAI indicates that the memory is connected and therefore the address and data have been accepted in the XDA and XDO buffer registers respectively, and thus the address registers and MIR may be subsequently changed.

3.5 It is possible that the memory is still in its memory cycle, but if so no other access will be granted to that memory module.

APPENDIX E

ADDER OPERATIONS

The following tables summarize the adder arithmetic and logical operations that may be specified, and the op code choice by TRANSLANG used for N [28-31]. The detailed interpretation of the N fields is contained in Appendix D.

Notes:

1. The X input to the adder is any one of the set indicated in N [17-19]:

   0 | LIT | ZEXT | CTR | Z | A1 | A2 | A3

2. The Y input to the adder is any one of the set indicated in N [20-26] unless noted.

   Bmcl | LIT | ZEXT | CTR | AMPCR | Z

3. The Z input to the adder includes selection from the LIT, ZEXT, CTR, or Z subset of the X or Y input set. Monadic use of a Z input is arbitrarily chosen to be X rather than Y. Dyadic use of a Z input allows a second Z input as part of the Y input set.

4. In Bmcl, the mcl represents the three gating specifiers, each of which may independently be specified as 0, T, F or 1. Note that the complement operator is used if c is F or 1. The normal operator is used if c is either 0 or T.

Table E-1

LOGICAL OPERATIONS

Operation					Adder Op Code, decimal equivalent if ¢ of Bmcl	
Performs	Bitwise	Logic	X Input	Y Input	= T or 0 or if Y is not B	= F or 1
X NOR Y	$X \not\vee Y$	$\overline{X}\,\overline{Y}$	X	Y	1	2
X NRI Y	$X < Y$	$\overline{X}\ Y$	X	Y	2	1
Y NIM X	$Y > X$	$\overline{Z}\ Y$	Z	Y		
X NAND Y	$X \not\wedge Y$	$\overline{X} \vee \overline{Y}$	X	Y	4	8
X XOR Y	$X \neq Y$	$X\overline{Y} \vee \overline{X}Y$	X	Y	6	9
X NIM Y	$X > Y$	$X\overline{Y}$	X	Y	7	11
Y NRI X	$Y < X$	$Z\overline{Y}$	Z	Y		
X IMP Y	$X \leq Y$	$\overline{X} \vee Y$	X	Y	8	4
Y RIM X	$Y \geq X$	$\overline{Z} \vee Y$	Z	Y		
X EQV Y	$X = Y$	$XY \vee \overline{X}\,\overline{Y}$	X	Y	9	6
X AND Y	$X \wedge Y$	XY	X	Y	11	7
X RIM Y	$X \geq Y$	$X \vee \overline{Y}$	X	Y	13	14
Y IMP X	$Y \leq X$	$Z \vee \overline{Y}$	Z	Y		
X OR Y	$X \vee Y$	$X \vee Y$	X	Y	14	13
X			X	0	0	
Y			0	Y	0	1
Z			Z	0	0	
NOT X		$\overline{X}$	X	0	1	
NOT Y		$\overline{Y}$	0	Y	12	0
NOT Z		$\overline{Z}$	Z	0	1	

Table E-2

ARITHMETIC OPERATIONS

Operation	X Input	Y Input	Adder Op Code, decimal equivalent if c of Bmcl = T or 0 or if Y is not B	= F or 1
X + Y	X	Y	0	12
X + Y + 1	X	Y	3	15
X - 1	X	B001	15	
X - Y #	X	Y	15	3
X - Y - 1 $	X	Y	12	0
X OAD Y $X+(X \vee Y)$	X	Y	5	Invalid
X AAD Y $X+(XY)$	X	Y	10	Invalid

# Subtraction is twos complèment: X + YFFF + 1

$ Subtraction is ones complement: X + YFFF

APPENDIX F

CODING TECHNIQUES AND CONVENTIONS

In the following examples some techniques are suggested to standardize coding and take advantage of particular instruction sequences. In these examples ▭ indicates a position within an instruction where a logic unit operation and/or external operation can be used for other work.

1. Order of type I and related type II instructions:

   Type II (literal assignment) load of AMPCR must <u>precede</u> the type I (N-instruction) that uses the AMPCR as a source for next microprogram address (the type I has successor of CALL, EXEC, JUMP, or RETN).

```
label -1 =: AMPCR
   ▭   ;  JUMP          % note the next instruction is at label.
```

   Type II load of SAR, LIT or AMPCR should follow the first instruction that uses it in a logic op. If the LIT is part of the X or Y Select (or if AMPCR is part of the Y Select) of the prior N instruction phase 3 in progress, then the value of the dynamic conditions (MST, LST, AOV, ABT) can change as a result of the type II.

```
J: A1 + LIT C =: A1          % A1 + 6 circular 4 to A1
   4 =: SAR ; 6 =: LIT       % used in J, 4 also possibly in K
K: A1 + LIT C =: B           % will add 3 or 9 to A1
   3 =: LIT                  % and shift 4 or 12 circular depending on
   IF LST THEN 0=:;SKIP      % LST test of A1 + 3. Note conditional logic op.
   12 =: SAR ; 9 =: LIT      % both can affect result from K
   B =:                      % forced Logic Op completes phase 3 of K or K+2
```

2. Label Type I Instructions:

   Generally it is bad practice to label and transfer control to a type II Literal Assignment because of the potential side effect on the phase 3 in progress when the labeled instruction is encountered. When possible the labeled instruction should have an unconditional (or empty) logic unit part to ensure that the prior phase 3 is completed.

3. Select either of two X select sources depending on mask in B:

```
A1 AND B=: MIR          % select A1 where B has one bits
A2 AND NOT B=: BBI      % prior result OR A2 where B has zero bits
```

4. Exchange content of two registers without using a temporary register:

```
A1 XOR B=:A1
A1 XOR B=:B             % exchange A1 and B
A1 XOR B=:A1
```

5. Exchange B-Register and MIR:

```
B=:MIR, BMI
```

6. Exchange A-Register with B-Register with MIR as temporary:

```
A1=:MIR
B=:A1, BMI
```

7. Test next-to-most significant bit of X input:

```
A1 OAD 0 =:             % A1 + A1
IF MST THEN  ▭          % test of next to most significant bit of A1
```

8. Shift masked portions of A-Register left by one:

   Mask in B contains ones wherever the source field(s) are located. Carries are added into field(s) to left of selected field(s).

```
A1 AAD B =:A1
```

9. Double value of X input:

```
A1 OAD 0=:A1            % A1 + A1=:A1
```

10. Replicate Character using SLIT:

Let the character to be replicated be internally represented in the range

0 LEQ character LSS 2^{bytes}

where bytes is one-eighth the number of bits in a word and is initially in some one byte of B with the rest zero. Note that the SLIT conversion provides a 1 only in the third bit from the left of whatever the SAR width is.

```
SET LC1 ; LIT=:A3=:SAR;SAVE
bytes=: SLIT                        % converted for word width
LOOP: B C=: BBA                     % 3 times: unshifted OR shifted amount
A3 OAD 0=:A3=:SAR IF LC1 THEN RETN ELSE CALL
▭                                   % B has replicated character
```

or

```
bytes=:SLIT                         % precedes first use
SET LC1;LIT=:A3=:SAR;SAVE
LOOP: B C =:BBA                     % 3 times
A3 OAD 0=:A3=:SAR IF LC1 THEN JUMP ELSE CALL
▭                                   % B has replicated character
```

11. Convert literal for shift amount

Assume A1 is to be shifted right by 12, or 15 depending on whether A3 MST is 0 or 1 respectively.

```
A3=:
IF MST THEN LIT =:SAR
12=:SAR ; 15=:SLIT                  % by convention after first uses
A1 R=: destination list
```

or

```
A3=:                                % unconditional, no shift or LIT input
12=:SAR ; 15=:SLIT                  % precedes first use
IF MST THEN LIT=:SAR
A1 R=: destination list
```

or

```
A3=:
15=:SAR
A1 R=: IF MST THEN SKIP
12=:SAR                             % on average faster
```

12. Keep shift amount insensitive to logic unit width:

Shift absolute amounts change with word width. Rather than using a literal assignment of a width-dependent amount for shift, it is preferable to only specify width-independent amounts absolutely and create width-dependent shift amounts using one of the following: Assume that a width-independent left shift of 5 bits is desired.

```
    COMP 5=:SAR           % word length complement
or
    5=:SAR                % right shift amount
    CSAR                  % word length complement
```

Assume the SAR is loaded as indicated. The word to be shifted contains the following information in fields of the indicated length, where w is the word width.

content:	i	j	k
field bit width:	5	w-10	5

SAR Loading Form

Shift:	5=:SAR			COMP 5=:SAR		
R=:	0	i	j	0	0	i
L=:	k	0	0	j	k	0
C=:	k	i	j	j	k	i

13. Compare full word numeric between X and Y selects:

The first instruction needn't have a destination. The test is in the second instruction.

Condition to test:	Instruction:	Test true if:
A1 < B	A1 - B	NOT AOV
A1 ≤ B	A1 - B - 1	NOT AOV
A1 ≠ B	A1 EQV B	NOT ABT
A1 = B	A1 EQV B	ABT
A1 ≥ B	A1 - B	AOV
A1 > B	A1 - B - 1	AOV

Note that -B is equivalent to +BFFF + 1
-B-1 is equivalent to +BFFF

Thus the negative sign assumes that numeric representations are twos complement. Subtracts are actually adds of the twos complements. AOV for subtracts occurs when the difference has the same sign as the X Select. No AOV occurs if the difference is the twos complement result.

14. Test condition with cost of two clocks for jump path and one for fall through:

```
     label-1 =: AMPCR
     IF cond THEN ▭ ELSE JUMP
```

15. Test condition with cost of three clocks for jump path and zero for fall through:

```
     IF cond THEN SET LC1; ▭ ;SKIP
     label-1=:AMPCR
     IF LC1 THEN ▭ ELSE JUMP
```

16. Test for exit before loop execution without loading AMPCR inside loop:

Method of leading decisions.

```
     TEST-1=:AMPCR
     ▭     ;CALL     % Jump to TEST before initial iteration and save head of loop
     LOOP: ▭         % no change to AMPCR in loop body
     TEST: IF cond THEN ▭ ;JUMP % Exit by fall through, jump to loop
```

17. Test two terminal conditions for loop end:

a. Either false for exit

```
     ▭     ;SAVE
     LOOP: ▭
          IF NOT condition-1  THEN ▭  ;SKIP
          IF condition-2  THEN ▭  ;JUMP
```

b. Both false for exit

```
          ▭ ;SAVE
     LOOP: ▭
          IF condition-1  THEN ▭  ;JUMP
          IF condition-2  THEN ▭  ;JUMP
```

18. Execute two-instruction loop with multiple exits:

```
AM: IF cond THEN▭;WAIT ELSE suc % The two instructions are at
M: IF cond THEN▭;EXEC ELSE suc % MPCR = M and AMPCR = AM
```

Alternating between AM + 1 and M continues so long as each condition is true. The exit location depends on the else successor, suc, of the instruction first failing its tested condition.

ELSE Successor	WAIT	STEP	SAVE	SKIP	JUMP	EXEC	CALL	RETN
Exit Location	M	M+1	M+1	M+2	AM+1$^{\$}$	AM+1	AM+1$^{\$}$	AM+2

\$ CALL or JUMP causes looping to continue at the single instruction loop at AM+1 with exits depending on its else successor.

AM+1 ELSE Successor:	WAIT	STEP	SAVE	SKIP	JUMP	EXEC	CALL	RETN
Exit Location is CALL:	AM+1*	AM+2	AM+2	AM+3	M+1	M+1$^{\#}$	M+1	M+2
Exit Location is JUMP:	AM+1*	AM+2	AM+2	AM+3	AM+1*	AM+1*	AM+1$^{\$}$	AM+2

*Continues one instruction loop. (i.e. infinite loop)

$^{\#}$Executes one instruction out of sequence (may return to one instruction loop)

$^{\$}$Continues one instruction loop but next ELSE will take control to AM+2

19. Exit subroutine to normal or error exit:

a. Error return out-of-line:

In main program

```
L: ▭   ;CALL                  % on subroutine
  E1-1 =:AMPCR                % error exit 1
  ▭                           % normal return
```

In subroutine:

```
IF error-cond THEN EXEC ELSE RETN  % return if no error
JUMP                          % error return (to E1)
```

b. Single instruction error exit in line:

In main program:

```
L: ▭   ;CALL                  % on subroutine

   ▭   ;RETN                  % one line error

   ▭                          % normal return
```

In subroutine:

```
IF error-cond THEN EXEC ELSE RETN
```

c. Multiple error exits, one level of indirection per error test:

In main program:

```
L: ▭   ;CALL

  E1-1=:AMPCR                 %error 1 setup

   ▭                          % normal return
   ---
E1:E2-1=:AMPCR                % Indirect for error 2

   ▭                          % error 1 code

   ---
E2:▭                          % error 2 code
```

In subroutine:

```
IF ercond-1 THEN EXEC ELSE RETN % error else normal

IF ercond-2 THEN EXEC ELSE RETN % errors 1 and 2 else 1

JUMP                            % error 2
```

20. Execute a loop a fixed number of times

a. Execute loop exactly twice:

```
        ▭ ;SAVE              % last instruction before loop
LOOP:   ▭
        ▭                    % body of loop no change to AMPCR
        ▭;CALL               % last instruction of loop
```

b. Execute a loop exactly three times:

```
    SET LC1: ▭  ;SAVE
LOOP: ▭                      % no change to AMPCR or IF LC1
      ▭ IF LC1 THEN JUMP ELSE CALL % last instruction
```

c. Execute loop exactly n times:

```
    n =:CTR                  % 0 ≤ n < 256
    TEST-1=:AMPCR
    ▭   ,INC;CALL            % omitting INC makes range 0 < n ≤ 256
    LOOP: ▭                  % body of loop no change to AMPCR
          ▭
    TEST: IF NOT COV THEN INC; JUMP %entry and last instruction of loop
```

21. Decrement CTR by 1:

```
    -CTR C=:CTR              % MSbyte to LSbyte
    COMP 8=:SAR              % width independent
    % - CTR is twos complement; =: CTR is ones complement
```

22. Count number of zeros in a word:

Initially the word is in B.

```
        0=:CTR;SAVE                 % -1 to CTR, ABT = 0
    LOOP: +B +1 =: MIR IF ABT THEN SKIP
        B=:BBI, INC; JUMP           % replaces rightmost 0 by 1
        ▭                           % CTR contains number of zeros
```

If the number of zeros is generally greater than the number of ones, the routine would be faster if B were initially ones complemented.

23. Increment counter conditionally or unconditionally effect on COV test

In sequence AA phase 3 of the first INC will attempt to set the COV at the same time as it is tested in the next instruction. Since COV is reset dominant it remains reset. In sequence BB the phase 3 which sets the COV is delayed until the end of phase 1 of the third instruction. The test causes the COV to be reset but occurs one clock before the set, so the COV ends up set.

```
AA: 0=:CTR                      % counter receives Hex FF, resets COV
INC; IF NOT COV THEN WAIT       % increments until overflow (2 times)
0=:B                            % COV will be reset after this operation and
                                % counter value will be 1

BB: 0=:CTR                      % counter receives HEX FF, resets COV
IF NOT COV THEN INC;WAIT        % increment until overflow (1 time)
0=: B, INC                      % COV will be set after this operation and
                                % counter value will be 1
```

24. Shift double length end off with zero fill.

a. Shift A1 and B together right by the amount in the SAR.

```
B R=:MIR
A1 L=:BBI                       % B OR MIR to B
A1 R=:A1
```

b. Shift B and A1 together left by the word length complement of SAR.

```
BL=:MIR
A1 R=:BBI
A1 L=:A1
```

25. Field isolation

Right justify the field of an A input. Justified field has leading zeros. Input has m bits preceding field f and n bits following it.

Input:	m	f	n
Result:	m+n zeros		f

```
NOT 0 R=:B                      % prepare mask
m =:SAR
A1 AND B R=:A1                  % extract and right justify
n =:SAR
```

26. Execute from table of literal assignments:

Assume a table containing a sequence of literal assignment (type II) instructions of the same kind. Table address determination is typically:

```
A3 + AMPCR =:AMPCR          % table offset + base below
base -1 =:AMPCR             % table base
```

a. Microprogram memory jump table.

Each table entry is of the form:

```
label -1=:AMPCR
```

Call form:

```
▭       ;EXEC               % puts desired label-1 in AMPCR
▭       ;CALL               % transfer to label with possible return
```

b. Selective mask and/or shift

Each table entry is of the form:

```
c1 =:SAR; c2=:LIT
```

Call form:

```
A1 AND LIT R =: destination-list;EXEC
% EXEC determines SAR and LIT values for the instruction
```

Lit is used in the 8 LSbits with the others zero.

c. AMPCR as 12 bit literal

Each table entry is of the form:

```
c3=:AMPCR                   % 12 bit literal
```

Call form:

```
A1 OR AMPCR=: destination-list;EXEC
% EXEC determines AMPCR value for the instruction
```

AMPCR is used in the 12 LSbits with the others zero.

27. Generate constants in logic unit:

It is faster to generate many constants rather than read them from S-memory.

a. Generated via B selection with gating for Y Select:

Intent	Y Select for Integer	Numeric representation
	twos complement	signed magnitude
-2	-1 -1	+B101 + 1
-1	-1	B101
0	0	0
1	1	1
2	+1+1	+1+1
-B	-B	BFTT
not B	NOT B	
odd B	B00T	B00T
sign B		BT00
abs B		B0TT
neg B		B1TT

b. Generated using the adder:

Hex constant	adder op
0303···03	0=:BC8
3333···33	0=:BC4

c. Literals from microprogram memory:

c1=:LIT	% 8 bit
c2=:AMPCR	% 12 bit

Either of these is available for adder input right justified with zeros in the left hand part.

d. Byte field arbitrarily positioned, possibly complemented:

LIT C=:A1	% ones complement if preceded by NOT
c1=:SAR ; c2=:LIT	
$\boxed{\quad}$	% next logic op completes

e. Special constants by shifting:

```
1   C=:destination              % $2^{n}$=:destination
COMP n =:SAR

1   C=:destination              % bit n=:destination   N > 0
    n=:SAR                      % starting at left with 1

NOT 0  R=:destination           % $2^{n+1}$-1=:destination
COMP n=:SAR
```

28. Form Indivisible Operations P and V for Co-operating Sequential Processes:

The routines P and V described below should be used to ensure exclusive use of shared data or critical sections of code. A lock bit (called a semaphore) is associated with each shared data or critical section. The P operation sets this bit and tests its previous value. The V operation resets the lock bit. The global condition bit GC1 is used to ensure that P and V are indivisible operations.

These operations are briefly described in the first of the following references. Extensive use and generalization is contained in the second reference.

Dijkstra, E.W., The Structure of the "THE"-Multiprogramming System CACM Vol. 11 No. 5 (May 1968) pp. 341-346
______________, Co-operating Sequential Processes, in Programming Languages, Genuys, F. Editor, Academic Press 1968, pp 43-112

a. P Operation:

```
P:  SET GC1 WHEN GC1                 % start critical section
    MR2;CSAR                         % read semaphore word
    WHEN RDC THEN BEX
    B C=:B, CSAR                     % move semaphore to least bit
    +B+1 C=:MIR, BMI                 % set bit if it was reset
    IF LST THEN MW2;SKIP ELSE EXEC
    RESET GC; B=:MIR ; JUMP          % exit unsuccessfully
    WHEN SAI THEN RESET GC ;  B=:MIR ; RETN  % exit normal
    ▭                                % normal return point

%   Call on P, to test and set specified semaphore.
    bit position =:SAR               % numbered from left starting at one
    P-1=:AMPCR
    word address=:MAR2 ;CALL         % call P
    failure label -1=:AMPCR          %return address for failure
```

b. V operation:

```
V:  SET GC1   WHEN GC1              % start critical section
    MR2 ; CSAR                      % read semaphore word
    WHEN RDC THEN BEX
    -B C=:B, CSAR                   % move semaphore to least bit
    -B-1 C=:MIR, BMI                % reset bit if it was set
    IF NOT LST THEN MW2 ; SKIP
    RESET GC ; B=:MIR ; JUMP        % exit without rewriting
    WHEN SAI THEN RESET GC ; B=:MIR ; JUMP % exit normal
```

```
%   Call on V, to reset semaphore after critical section completed.
    bit position =:SAR
    V-1=:AMPCR
    word address =:MAR2 ; CALL
```

29. Use of base registers and MAR:

a. BR1: typically points to the base of a 256 word block of memory in which an instance of the registers of the higher level machine are stored. In this case BR1 may be viewed as the S-machine register base register.

Any particular register is addressed via BR1 and MAR. Each of these registers may be named by a label defined to be equal to the particular integer associated with the MAR portion of its address.

```
ACCUM * 21                  % define accumulator to be register 21₁₀
LMAR, ▭                     % MAR receives accumulator address
ACCUM=:LIT                  % accumulator address
```

BR1 may also be used to contain device identification.

b. BR2: typically is used with MAR as the general address register for memory addressing.

```
B=:MAR2                     % causes address entry to BR2, MAR
```

30. Load MAR and MIR simultaneously for memory write:

Assume BR1 is the S-machine register base register

```
B=:MIR, LMAR
S-register number=:LIT      % MAR receives this number
MW1 IF SAI                  % write uses both MIR and MAR.
```

31. Perform S-memory operations:

a. Memory read:

```
addr=:MAR2

MR2; ▭   % SAI will get set. ▭   cannot be conditional ⟨ Logic Op⟩
     ▭   % sufficient intervening instructions
     ▭   % to at least match the minimum read
     ▭   % access time may be included

WHEN RDC THEN BEX, ▭   % leaves RDC reset for next use.
```

b. Alternative memory read:

```
addr=:MAR2

MR2; ▭   IF SAI % resets SAI. ▭   cannot be conditional ⟨ Logic Op⟩
▭                   % at least one instruction

WHEN SAI THEN ▭  % can change address
▭                       % possible as above

WHEN RDC THEN BEX, ▭   % leaves RDC reset for next use
```

c. Memory write:

```
addr=:MAR2
data=:MIR
MW2; ▭   ;IF SAI % resets SAI. ▭   cannot be conditional⟨Logic Op⟩
▭                   % at least one instruction
WHEN SAI THEN ▭  % the write operation has been completed,
                 % MAR or MIR may be changed.
```

The above sequences of code are proposed because:

1. They work on both the SSI and LSI machines (i.e. with port select units or with switch interlock).

2. The WHEN instructions should not cause delays when there are no conflicts.

3. They permit the earliest possible reuse of the MAR and MIR, and earliest clocking of memory data into B. (It is not necessary to do the BEX as soon as possible, the external data remains available until used). (For the switch interlock the external data may be reaccessed with BEX until another memory or device operation is executed.)

The alternative memory read should be used when it is desirable to reload the MAR or MIR before the read is complete.

Note the restriction on forcing the address (and MIR load) to be completed prior to issuing the MR (or MW). If the logic op were conditionally determined not to be done, the preceding phase 3 would have been a phase 2 instead, the phase 3 extended and the prior address (or MIR load) would have been used. An absent <Logic Op> in a line is assumed to be unconditional.

32. Perform device operations:

a. Lock to device if unlocked, otherwise do other work:

To avoid time uncertainty in completing a lock.

```
DLK:SET GC2 WHEN GC2 THEN DL1

    ▭ % This instruction must be present

    IF NOT SAI THEN DU1; EXEC ELSE SKIP

    ▭ % This instruction must be present

    RESET GC IF SAI THEN JUMP ELSE RETN

% Call for device lock request, DLK

    DLK-1=:AMPCR

    ident=:BR1 ; CALL IF SAI

    already locked -1=:AMPCR        % if locked to other Interpreter

    ▭                               % locked
```

Alternatively device lock requests may be protected by semaphore in which case GC2 is not necessary.

b. Unlock device:

```
DUL: SET GC2 WHEN GC2 THEN DU1

    ▭                % must be present

    RESET GC IF SAI THEN EXEC ELSE RETN

    ▭ JUMP % if not locked to this Interpreter

% Call for device unlock, DUL
     DUL-1=:AMPCR
     Ident=:BR1 ; CALL  IF SAI
     not locked-1=:AMPCR     % if not locked to this Interpreter
     ▭                       % unlocked.
```

c. Device read on demand (transfer time determined by Interpreter):

To a device locked to this Interpreter, taking the data currently in the device output register.

```
ident=:BR1
DR1 ; BEX         % data requested at end of phase 1 is available for
▭                 % end of phase 3.
▭                 % SAI may be tested: true =locked to this Interpreter
                  % and data read is good.
```

d. Device read by request (transfer time determined by device):

To a device locked to this Interpreter.

```
ident=:BR1
DR1 ; ▭
▭
▭                 % SAI may be tested : true = locked to this Interpreter
WHEN RDC THEN BEX     % leaves RDC reset for next use, the
                      % waiting time here is device dependent
```

Instructions between DR1 and RDC test may not include memory or device operations, else the data path will be broken, no data will return and RDC will not occur. A change to BR1 is only safe after RDC is true.

e. Device write on demand (transfer time determined by Interpreter)

To a device locked to this Interpreter.

```
ident=:BR1
data=:MIR
DW1; ▭            % include IF SAI if to be used below;
▭                 % no conditional logic op
▭                 % could change MIR or BR1
                  % could test SAI : true = locked to this Interpreter
```

f. Device write by request (transfer time determined by device) to a device locked to this Interpreter.

```
ident=:BR1
data =:MIR
DW1; ▭ IF SAI          % no conditional logic op
▭                      % MIR=:EXO
IF SAI THEN ▭          % locked to this Interpreter ; can change MIR
WHEN RDC THEN ▭        % request on device complete,
                         can change BR1 or MIR
```

g. Device read on demand embedded in memory read:

At least two clocks must exist after SAI is returned from a memory read before the RDC is returned.

```
address=:MAR2
MR2; ▭ IF SAI          % address to address buffer
ident=:BR1             % at most one instruction
WHEN SAI THEN DR1;BEX  % ident selects input from device to input
                       % buffer to B
B=:A1                  % device input, no conditional logic op, may
                       % change BR1
▭                      % RDC arrives after prior phase 1
WHEN RDC THEN BEX      % memory read result
```

h. Device write on demand embedded in memory read

```
address=:MAR2
MR2; ▭ =:MIR IF SAI   % address to address buffer
ident=:BR1
WHEN SAI THEN DW1      % MIR to output buffer to device
WHEN RDC THEN BEX      % memory read result, may change MIR,
                       % MAR2 or BR1
```

TYPE I (N-INSTRUCTION) LINE PROTOTYPE

```
         ⎛Ext Op            ⎞*  ⎡⎛IF  ⎞              ⎡      ⎛Ext Op        ⎞*                          ⎤⎤
[Label:] ⎜Logic Op          ⎟   ⎢⎜    ⎟  Condition   ⎢THEN  ⎜Logic Op      ⎟  [ELSE False Successor]  ⎥⎥  where
         ⎝Uncond. Successor ⎠   ⎣⎝WHEN⎠              ⎣      ⎝True Successor⎠                          ⎦⎦
```

where

*	indicates zero or more repetititions with ";" as their infix separator
( )	indicates a choice
[ ]	indicates optionally present

with restriction: 1 ≥ ⌈/Ext Op, Logic Op, Uncond Successor + True Successor ⌈ WHEN + ELSE

ie: There is at most one each of Ext Op, Logic Op, Uncond. Successor and True Successor.

If there is an Uncond. Successor there will be no True Successor, WHEN or ELSE.

If there is a WHEN there will be no ELSE.

Condition	Ext. Op.		Logic Op.												Successors
	Mem. Dev. Op.	Set Op.	x Select	Adder Op.	y Select	Inhibit Carry	Shift	Destination							
[NOT] LST	MR1	SET LC1	0	x	[NOT] Bmcl	IC	R	A1	B	MIR	BR1	CTR	SAR	AMPCR	WAIT
MST	MR2	LC2	A1	y	B	--	L	A2	BEX	---	BR2	LCTR	CSAR	---	STEP
AOV	MW1	LC3	A2	x + y	AMPCR		C	A3	BAD		MAR	INC	---		SKIP
ABT	MW2	INT	A3	x + y + 1	0		--	--	BC4		MAR1	---			SAVE
COV	DL1	GC1	CTR	x - y	1				BC8		MAR2				JUMP
LC1	DL2	GC2	ZEXT	x - y - 1	CTR				BMI		LMAR				EXEC
LC2	DR1	RESET GC	LIT	x NRI y	ZEXT				BBE		---				CALL
LC3	DR2	---	Z	x NOR y	LIT				BBA						RETN
SAI	DW1		---	x NIM y	Z				BBI						
RDC	DW2			x AND y					---						
EX1	DU1			x XOR y											
EX2	DU2			x EQV y											
EX3	ASR			x NAN y											
INT	ASE			x IMP y											
GC1				x OR y											
GC2				x RIM y											
				x AAD y											
				x OAD y											
				NOT x											

m	c	l
0	0	0
1	1	1
T	T	T
F	F	F

Type I instructions with SAR or AMPCR destination should not start with 0 or 1.

NUMERIC COMPARE OPERATIONS

Condition:	Instruction:	Test True If:
$x < y$	x - y	NOT AOV
$x \leq y$	x - y - 1	NOT AOV
$x \neq y$	x EQV y	NOT ABT
$x = y$	x EQV y	ABT
$x \geq y$	x - y	AOV
$x > y$	x - y - 1	AOV

TYPE II (LITERAL ASSIGNMENT) INSTRUCTIONS

```
         ⎛[Literal =:  SAR ;]  Literal =:  ⎛LIT ⎞⎞
         ⎜                                 ⎝SLIT⎠⎟
[Label:] ⎜Literal =:  SAR                        ⎟
         ⎝Label  [-1]  =:  AMPCR                 ⎠
```

BOOLEAN OPERATIONS

	$x\,y$ $x\,\bar{y}$ $\bar{x}\,\bar{y}$ $\bar{x}\,y$		
NRI	0 0 0 1	$\bar{x}\,y$	$<$
NOR	0 0 1 0	$\bar{x}\,\bar{y}$	$\nleftarrow$
NIM	0 1 0 0	$x\,y$	$>$
AND	1 0 0 0	$x\,y$	$\wedge$
XOR	0 1 0 1	$x\,\bar{y} \vee \bar{x}\,y$	$\neq$
EQV	1 0 1 0	$x\,y \vee \bar{x}\,\bar{y}$	$=$
NAN	0 1 1 1	$\bar{x} \vee \bar{y}$	$\barwedge$
IMP	1 0 1 1	$\bar{x} \vee y$	$<$
OR	1 1 0 1	$x \vee y$	$\vee$
RIM	1 1 1 0	$x \vee \bar{y}$	$\geq$

Label
Letter $\begin{pmatrix}\text{Digit}\\ \text{Letter}\end{pmatrix}^*$

PSEUDO INSTRUCTIONS

```
⎛⎛PROGRAM            Program Name⎞[ADR Hex Address]⎞
⎜⎝ [Label:]   INSERT  File Name  ⎠                 ⎟
⎜ADR    Hex Address                                ⎟
⎜END                                               ⎟
⎜Label     *      Decimal Integer                  ⎟
⎝COMMENT   Comment   ;                             ⎠
```

Literal
[COMP] $\begin{pmatrix}\text{Label}\\ \text{Decimal Integer}\end{pmatrix}$
Label [-1]

INTERPRETER

MICROPROGRAMMING REFERENCE CARD

MICRO CONTROLS

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	∅	* SAR						∅	∅	∅	∅	∅	∅	∅	∅
1	0	* SAR						LIT							
1	1	0	∅	* AMPCR											
1	1	1	0	∅	∅	∅	∅	LIT							
1	1	1	1	* NANO ADDRESS											

∅ Unused
* Shorter fields are right justified

NANO CONTROLS

Parentheses surround optional lexic units, provided by default.

Brackets contain DC 2000 mnemonics

? Codes not produced by TRANSLANG.

1	2	3	4	Condition Tested Result is Boolean cnd
0	0	0	0	GC1
0	0	0	1	GC2
0	0	1	0	LC1
0	0	1	1	LC2
0	1	0	0	MST
0	1	0	1	LST
0	1	1	0	ABT
0	1	1	1	AOV
1	0	0	0	COV
1	0	0	1	SAI [RMI]
1	0	1	0	RDC
1	0	1	1	LC3 [RMA]
1	1	0	0	EX1 [EXT]
1	1	0	1	INT
1	1	1	0	EX2 [SRQ]
1	1	1	1	EX3 [URQ]

5		FT Condition Value
0	NOT	$\overline{\text{cnd}}$=:SC
1	--	cnd=:SC

6	Logic Unit Conditional
0	Do Unconditionally
1	Do Conditionally if SC

7	Ext Op (MDOP/CAJ) Conditional
0	Do Unconditionally
1	Do Conditionally if SC

8	9	10	Condition Adjust -- CAJ
0	0	0	--
0	0	1	SET LC2
0	1	0	SET GC2
0	1	1	RESET GC
1	0	0	SET INT
1	0	1	SET LC3
1	1	0	SET GC1
1	1	1	SET LC1

11	12	13	Successor	14	15	16
Then Part Used if SC=1			to MPAD Ctls	Else Part Used if SC=0		
0	0	0	WAIT	0	0	0
0	0	1	(STEP)	0	0	1
0	1	0	SAVE	0	1	0
0	1	1	SKIP	0	1	1
1	0	0	JUMP	1	0	0
1	0	1	EXEC	1	0	1
1	1	0	CALL	1	1	0
1	1	1	RETN	1	1	1

17	18	19	Adder X Input
0	0	0	(0)
0	0	1	LIT
0	1	0	ZEXT [EXT]
0	1	1	CTR
1	0	0	Z
1	0	1	A1
1	1	0	A2
1	1	1	A3

20	21	22	23	24	25	26	Adder Y Input
0	0	-	-	-	-	-	B0--
0	1	-	-	-	-	-	BT--
1	0	-	-	-	-	-	BF--
1	1	-	-	-	-	-	B1--
-	-	0	0	0	-	-	B-0-
-	-	1	0	0	-	-	B-T-
-	-	-	-	-	0	0	B--0
-	-	-	-	-	0	1	B--T
-	-	-	-	-	1	0	B--F
-	-	-	-	-	1	1	B--1
Comp		1	0	0	Comp		B-F-*
Comp		0	0	0	Comp		B-1-*
0	0	0	0	1	0	1	LIT
0	0	0	1	0	0	0	ZEXT [EXT]
0	1	0	1	1	0	0	CTR
0	1	1	0	1	0	1	Z
0	0	1	1	0	0	1	AMPCR
0	1	1	1	1	0	1	[L0]
			Others				?

*Use Adder Operation with Complement Y

27		Inhibit Carries into Bytes
0	--	Allow
1	IC	Inhibit

28	29	30	31	Adder Operation			Logic
0	0	0	0	X	+	Y	
0	0	0	1	X	NOR	Y	$\bar{X}\ \bar{Y}$
0	0	1	0	X	NRI	Y	$\bar{X}\ Y$
0	0	1	1	X +	Y	+ 1	
0	1	0	0	X	NAN	Y	$\bar{X} \vee \bar{Y}$
0	1	0	1	X	OAD	Y	$X + (X \vee Y)$
0	1	1	0	X	XOR	Y	$X\bar{Y} \vee \bar{X}Y$
0	1	1	1	X	NIM	Y	$X\bar{Y}$
1	0	0	0	X	IMP	Y	$\bar{X} \vee Y$
1	0	0	1	X	EQV	Y	$XY \vee \bar{X}\bar{Y}$
1	0	1	0	X	AAD	Y	$X + (XY)$
1	0	1	1	X	AND	Y	XY
1	1	0	0	X -	Y	- 1	$X + \bar{Y}$
1	1	0	1	X	RIM	Y	$X \vee \bar{Y}$
1	1	1	0	X	OR	Y	$X \vee Y$
1	1	1	1	X	-	Y	$X + \bar{Y} + 1$

32	33	Shift Type Selection for BSW	
0	0	--	No Shift
0	1	R	Right End Off
1	0	L	Left End Off
1	1	C	Right Circular

34	35	36	A Register Input from BSW	
0	0	0	--	No Change
1	-	-	A1	
-	1	-	A2	
-	-	1	A3	

37	38	39	40	B Register Input Select	
0	0	0	0	--	No Change
0	0	0	1	BC4	Comp 4 Bit Carries
1	0	0	0	BAD	Adder
1	0	0	1	BC8	Comp 8 Bit Carries
1	0	1	0	BBA	BSW v Adder
1	0	1	1	B	BSW
1	1	0	0	BEX	External
1	1	0	1	BMI	MIR
1	1	1	0	BBE	BSW v External
1	1	1	1	BBI	BSW v MIR
Others				?	

41	MIR Input from BSW	
0	--	No Change
1	MIR	

42	AMPCR Input from BSW	
0	--	No Change
1	AMPCR	

43	44	45	46	Mem Dev Address Input	
0	0	0	-	--	No Change
-	-	1	0	LMAR	From LIT
-	-	1	1	MAR	From BSW
-	1	0	-	BR2	From BSW
-	1	1	1	MAR2	From BSW
1	-	0	-	BR1	From BSW
1	-	1	1	MAR1	From BSW

46	47	48	Counter Input	
-	0	0	--	No Change
0	0	1	LCTR	From LIT*
1	0	1	CTR	From BSW*
-	1	0	INC	+1
-	1	1	?	

*Ones Complement

49	50	SAR Input	
0	0	--	No Change
0	1	CSAR	Complement
1	0	SAR	From BSW
1	1	?	

51	52	53	54	Mem Dev Op--MDOP	
0	0	0	0	--	No Change
0	0	1	0	MR1	
0	0	1	1	MR2	
0	1	1	0	MW1	
0	1	1	1	MW2	
1	0	0	0	DL1 [ASR]	
1	0	0	1	DL2 [ASE]	
1	0	1	1	DR1	
1	0	1	1	DR2	
1	1	0	0	DU1	
1	1	0	1	DU2	
1	1	1	0	DW1	
1	1	1	1	DW2	
Others				?	

Burroughs Corporation
DEFENSE, SPACE AND SPECIAL SYSTEMS GROUP
PAOLI, PENNSYLVANIA 19301

67068

Burroughs Corporation

Federal and Special Systems Group

Paoli, Pennsylvania 19301

Printed in U.S. America