Technical Report TR-196 August 1972

A TRANSLATOR AND SIMULATOR FOR THE BURROUGHS D MACHINE

by

John Roberts

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER
COLLEGE PARK, MARYLAND

Technical Report TR-196 August 1972

A TRANSLATOR AND SIMULATOR FOR THE BURROUGHS D MACHINE

by

John Roberts

This research was supported in part under Grant NGL-21-002-008 from the National Aeronautics and Space Administration to the Computer Science Center of the University of Maryland.

ACKNOWLEDGEMENTS

The author would like to acknowledge the contribution of Dr. Ashok Agrawala of the University of Maryland for his suggestions on conversational program responses and his guidance in developing this software package.

The author would also like to express his gratitude to Dr. Earl Reigel of the Burroughs Corporation for his permission to reproduce parts of the D Machine technical manuals and for introducing the author to the D Machine.

CONTENTS

I. ABSTRACT

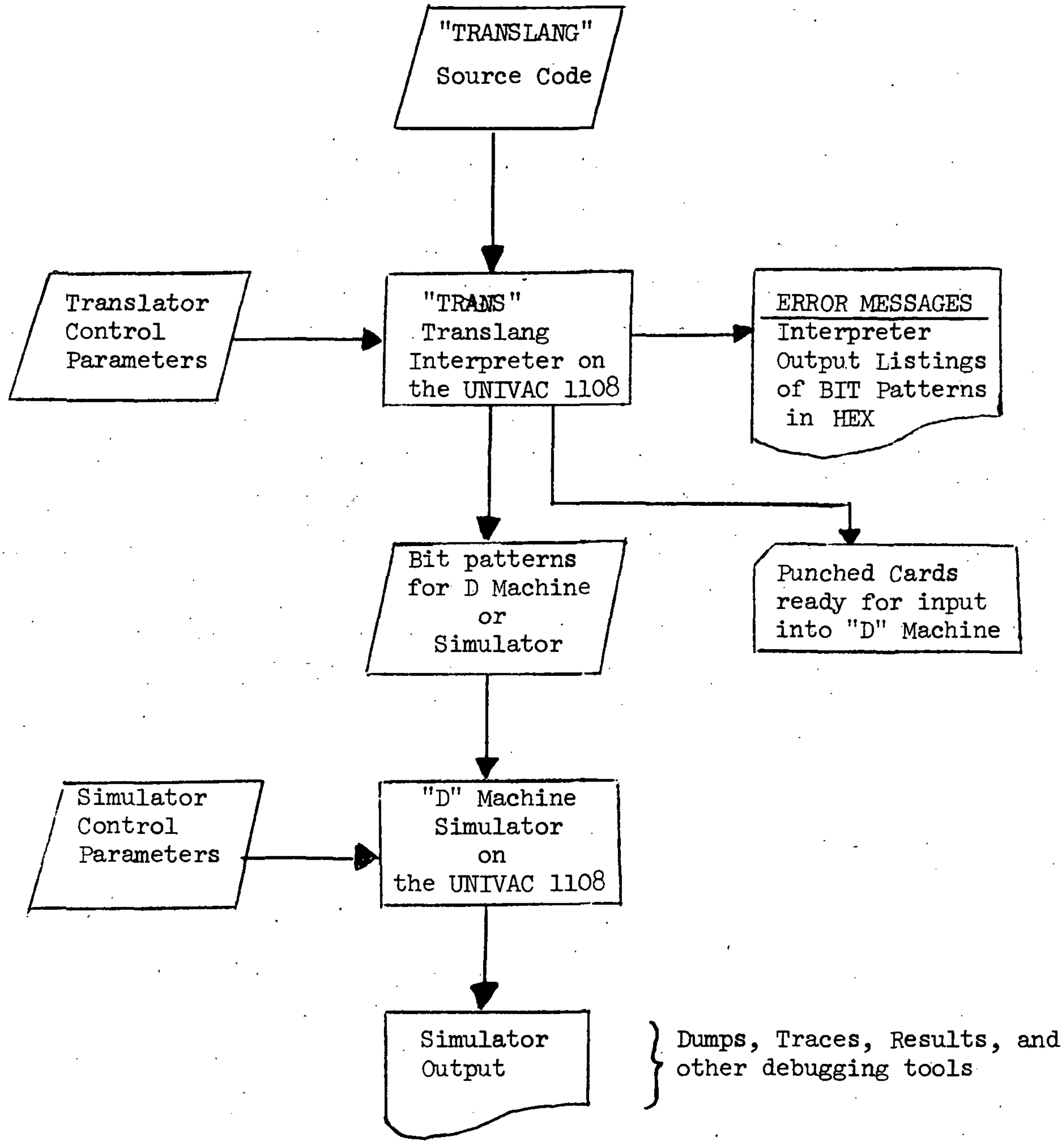
Traditionally software for small microprogrammed computers has been minimal or nonexistent. One solution to this problem is given for the Burroughs Corporation D Machine. The system, which is implemented on the University of Maryland UNIVAC 1108 and shown in the flowchart on the next page, is composed of two programs: a TRANSLANG translator, and a D Machine simulator.

In an effort to fully understand this system, a full description of the D Machine and TRANSLANG are given in this document.

Briefly, however, the D Machine is a small user microprogrammable computer designed to be a versatile building block for such diverse functions as: disk file controllers, I/O controllers, and emulators.

TRANSLANG is an Algol-like language, created by Burroughs, which allows D Machine users to write microprograms in an English-like format as opposed to creating binary bit pattern maps.

The TRANSLANG translator parses TRANSLANG programs into D Machine microinstruction bit patterns which can be executed on the D Machine simulator. In addition to simulation and translation, the two programs also offer several debugging tools, such as: a full set of diagnostic error messages, register dumps, simulated memory dumps, traces on instructions and groups of instructions, and finally, breakpoints.



D MACHINE SOFTWARE PACKAGE

II. D MACHINE

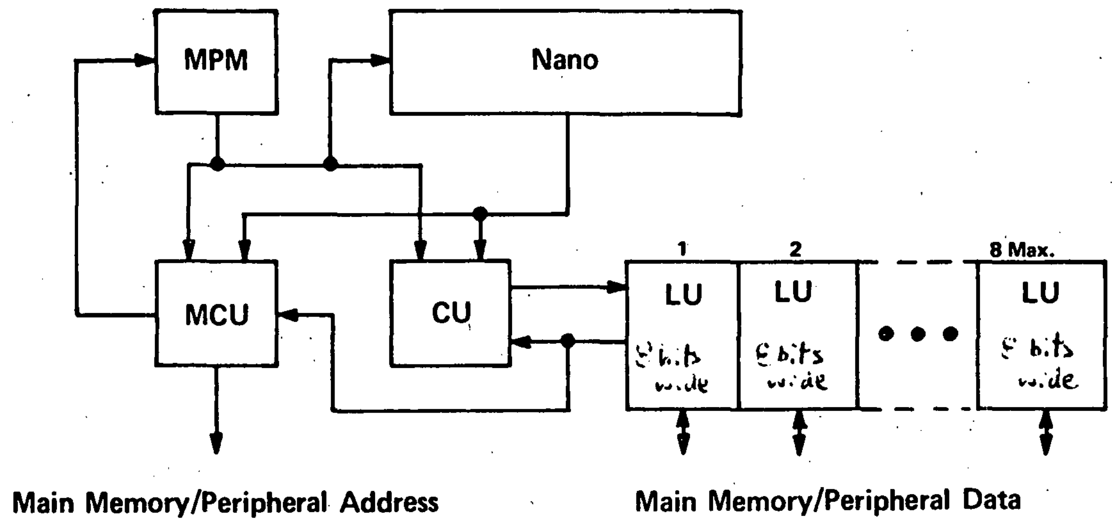
The D Machine consists of four functional units; the Micro Program Memory (MPM or Control Store), the Memory Control Unit (MCU), the Control Unit (CU), and finally the Logic Unit (LU). These elements are shown diagrammatically in Figure 1. All specifications are for the D Machine which is emulated.

The MPM holds microprograms which are to be executed. This memory is 70 bits wide by 1024 words long. The 70 bits are composed of two areas. The first is 16 bits (M memory) and is used to set literal amounts in registers or to define whether the next 54 bits are used. (In some D Machines, the two areas reside in completely separate memories). The 54 bit area (NANO Memory) activates all the gates and paths inside the D Machine except in the case just mentioned.

The MCU generates addresses for both the Main Memory and for the MPM. It contains two 12 bit registers for MPM addressing, three 8 bit registers for Main Memory addressing, one for 8 bit literals (LIT), and one 8 bit counter (CTR). The two for MPM addressing are called the Micro Program Count Register (MPCR), the main program counter, and the Alternate Micro Program Count Register (AMPCR), which is used to hold return addresses. The three for Main Memory addressing are Base Register One (BR1), Base Register Two (BR2) and the Memory Address Register (MAR). Main Memory addressing is accomplished by concatenating BR1 and MAR or BR2 and MAR to form a 16 bit address. For the simulated Main Memory only 1024 words of 32 bits are provided, so only 12 bits of the concatenated Main Memory Addresses are needed.

The CU receives the bits from the NANO Memory and decodes them to provide commands to the Logic Unit. Each instruction can have several conditional parts and thus the CU is responsible for testing

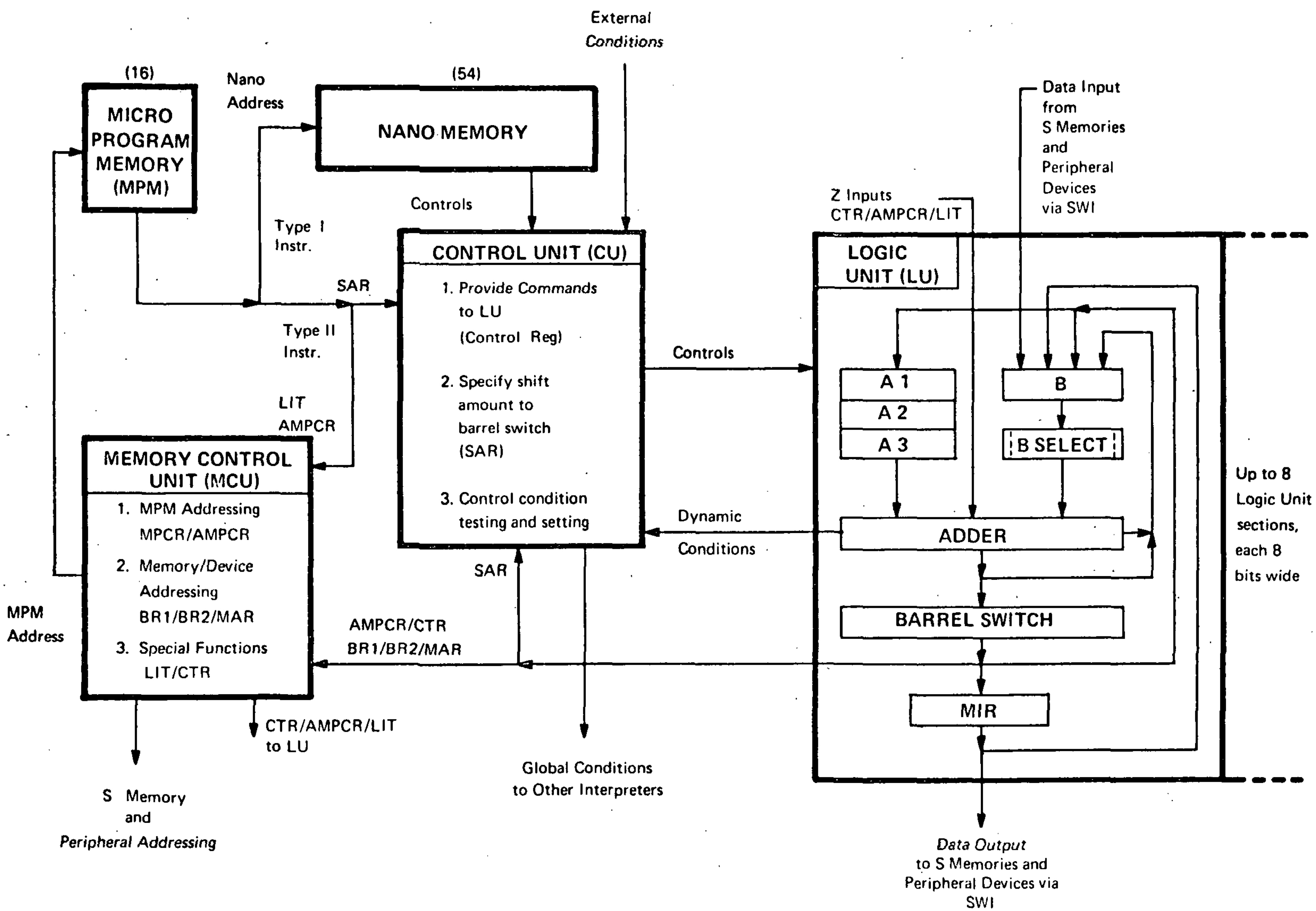
INTERPRETER

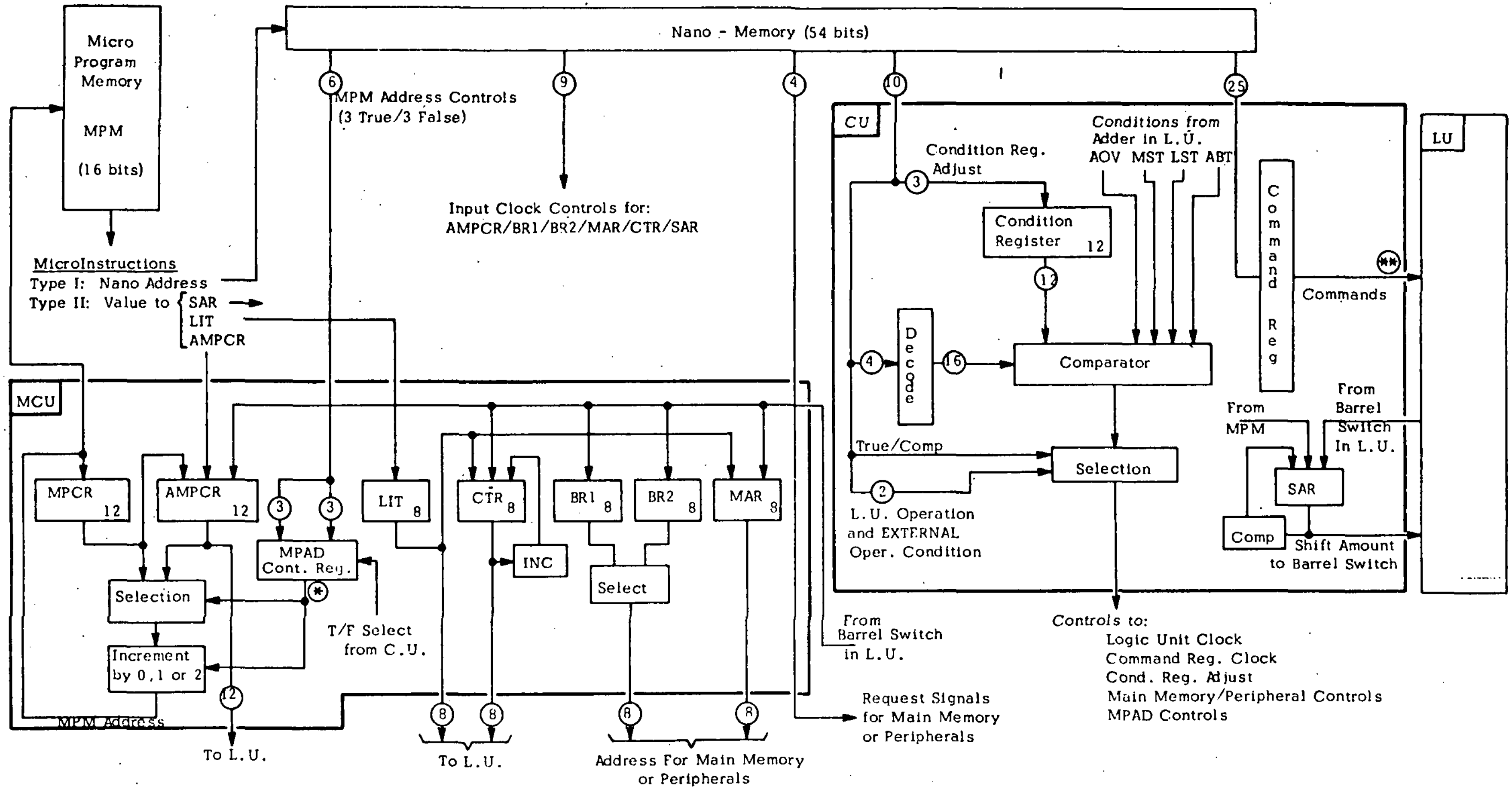


Each Interpreter consists of:

- Three types of logic packages (each containing only 700 gates, approx.) –
  - ▲ Logic Unit (LU)
    - – Registers A1/A2/A3/B/MIR
  - ▲ Control Unit (CU)
    - – Registers SAR/Command/Condition
  - ▲ Memory Control Unit (MCU)
    - – Registers MPCR/AMPCR; BR1/BR2/MAR; LIT/CTR
- Two memory packages (standard available units – R/W or R/O)
  - ▲ Microprogram Memory (MPM)
  - ▲ Nanoinstruction Memory (Nano)

INTERPRETER BLOCK DIAGRAM





(*)

	Select	Incr.	
Wait:	MPCR	0	
Step:	MPCR	1	
Skip:	MPCR	2	
Call:	AMPCR	1	(also saves MPCR Contents in AMPCR (for return))
Jump:	AMPCR	1	
Return:	AMPCR	2	
Exec:	AMPCR	1	(inhibits loading of MPCR, therefore proceeds in sequence)
Save:	MPCR	1	(saves MPCR contents in AMPCR (for looping))

(**)

(1) Selection:
- A Reg (17-19)
- B Reg (20-26)

(2) Adder op & Carry Control (27-31)

(3) Barrel Switch Control (32-33)

(4) Destinations
- A Reg (34-36)
- B Reg (37-40)
- MIR (41)

and setting conditions inside the entire D Machine. Another function of the CU is to provide shift amounts to the LU shifter. This is accomplished by maintaining the 5 bit Shift Amount Register (SAR) to indicate a shift of from 0 to 31 bits.

The last unit is the Logic Unit, and it is this unit which is the most obvious to the microprogrammer. The LU architecture consists of five 32 bit registers, an adder, and a shifter (barrel switch). The A1, A2, and A3 registers make up the left inputs to the adder from the LU and the B register is the only right input from the LU. A microinstruction selects, during each adder cycle, a left and right adder input from those just mentioned or from other registers outside the LU. Once selected, the requested adder operation is performed and then that result is optionally shifted any amount and stored into any of the four registers already mentioned, or into the Memory Input Register (MIR). Data enters the LU from external sources through the B register and leaves through the MIR register. An interesting feature associated with the B register is the B select unit. This logic network allows the user to select the B register as a source for the adder in many variations. The select unit breaks the B register into three parts; most significant bit, least significant bit, and all bits in between. Each of these parts can be selected as True, False, Zero or One. True indicates the part is unchanged. False indicates the one's complement of the part is selected; Zero requests the part to be set to all zeros (or zero) and One, similarly, dictates the part to be set to all ones (or one). This will be utilized in TRANSLANG which would allow one to say BOTT = B, which means B is replaced by the absolute value of B (i.e., the sign bit is set to zero to denote positive).

To understand the D Machine and how it is microprogrammed, it is necessary to examine the control word formats of the MPM Memory and to study the timing of events inside the D Machine. A detailed examination of these two topics is given in Appendix D. It should be noted here that if one writes programs in TRANSLANG he need not know the function of each bit in the Control Memory, but he must be familiar with the timing of events or he will find it impossible to write error free programs.

III. TRANSLANG

TRANSLANG is a Algol-like language created to assist in writing microprograms for the Burrough's D Machine. Its complete syntax is given in Appendix A. The vocabulary of TRANSLANG consists of numerous Key Words (reserved words) which, after proper ordering as prescribed in the syntax, can be translated into microinstructions. A complete list of these Key Words is given in Appendix B.

Each TRANSLANG instruction consists of one line (or card) and corresponds to the set of D Machine functions which are activated in parallel during that machine clock (i.e., one control store word). These functions include register adjustments, I/O, Boolean, logical and computational operations, control transfers and assignments. To aid in control transfer each instruction can begin with a label identifier which can be used to uniquely identify that address throughout a program.

Backus-Naur form (BNF) is used as the metalanguage to define the syntax of TRANSLANG. The complete syntax is broken up into groups and presented in the following order: syntax, semantics, and examples.

In an effort to increase understanding and simplify definitions, the following convention will be used in conjunction with the BNF definitions - "{" and "}" will be used to encompass English language descriptions which might otherwise take more effort to describe in BNF.

Basic Elements

Syntax: <Program> :: = <Body> <End Line>

<Body> :: = <Comment> |<Line>|<Body> <Comment>|<Body> <Line>

<Line> :: = <N Instruction> $ |<Literal Assignment> $

<Comment> :: = COMENT {Any string of <Character>s except $}$

<Empty> :: = {The null string of characters}

<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

<Character> :: = <Letter>|<Digit>|<Single Space>|<Symbol>

<Label> :: = <Letter>|<Label><Letter>|<Label><Digit>

<Single Space>:: = {One horizontal blank position}

<Symbol> :: = |+|-| |=|.|*|(|)|$

<Assignment Op> :: = =

<End Line> :: = END $

Semantics:

A TRANSLANG program consists of a series of instructions followed by an END instruction. Each instruction is translated into one micro-instruction and placed into the simulated control store starting at address 0 and stepping by 1 for each additional instruction (up to 1024). Labels can be used as symbolics for addresses. Whenever a label is encountered it is inserted into a table for resolution in pass 2 of the translator. Labels can be up to 6 characters long.

Comments can be made in two fashions. One is by starting the line with the Key Word comment, which causes no action to be taken on any lines which follow until a line with a "$" appears. After that line translation resumes. The second method is to put comments after the "$" which must end each instruction. Please note that an instruction cannot be continued and must appear on one line only.

Spaces can appear around Key Words but not inside them. Embedded blanks will always cause problems.

Literal Assignment

Syntax:

```
<Literal Assignment> :: = <Literal> = AMPCR|<Literal> = SAR|
                          <Literal> = SAR, <Literal> = <Lit>|
                          <Literal> = <Lit>,<Literal> = SAR|
                          <Literal> = <Lit>
         <Literal> ::   = <Integer>|Comp<Interger>|<Label>|
         <Integer> ::   = <Digit>|<Integer> <Digit>
         <Lit>     ::   = LIT|SLIT
```

Semantics:

These instructions are 16 bit type II instructions as mentioned in the timing section of Appendix D. Each is used for inserting constants into one of three registers: AMPCR, SAR or LIT.

		Width
AMPCR	Alternate Micro Program Count Register	12
SAR	Shift Amount Register	5
LIT	Literal Register	8

The registers may be individually loaded or both the SAR and the LIT may be loaded in the same 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.

If SLIT is the destination register, then the literal value is entered into the LIT registers in the same form as the SAR. The SAR register requires an unusual format as shown in the following table. Zeros are used to fill the 8 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	110X 00	00
29	01	000X 11
30	10	10
31	11	01

X - indicates an unused position.

The successor of a <Literal Assignment> is always a default STEP.

Examples:

```
5 = SAR                        $ SAR set to 001X01
COMP 8 = SAR, 13 = SLIT        $ SAR is 110X00, LIT is 00011X 01
COMP 0 = LIT                   $ LIT is 255
255 = LIT                      $ Same as above
START = AMPCR                  $ Value of address START will go to AMPCR
LOOP - 1 = AMPCR               $ Address LOOP-1 goes to AMPCR
```

N Instruction

Syntax:

```
<N Instruction>          :: = <Unconditional Part> <Conditional Part>|
                              <Label> . <Unconditional Part>
                              <Conditional Part>
<Unconditional Part>     :: = <Component List>|<Empty>
<Component List>         :: = <Component>|<Component List> <Component>
<Component>              :: = <Ext op>|<Logic op>|<Successor>
<Conditional Part>       :: = <If clause> <Cond Comp List> <Else clause>|
                              <If cluase>|<When Clause> <Cond Comp List>|
                              <Empty>
<Cond Comp List>         :: = THEN <Component List>
```

Semantics:

Each N instruction produces a 54 bit microinstruction (Nano) and is called a type I instructions. An N instruction may begin with a Label followed by a "." If this occurs the label is then defined by putting the label and the address of this instruction in a table, such that all references to this label can be resolved with this address.

The following restrictions are imposed:

1. At most one <Ext Op> - either conditional or unconditional
2. At most one <Logic Op> - either conditional or unconditional

3. At most either one conditional successor or one unconditional successor in the <Cond Comp List> and possibly one in the <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 the components in the <Cond Comp List> are executed; otherwise, the <Else Clause> is executed.

Examples: (to be explained later)

Unconditional Part Alone:

```
SET LC1              $
MR2                  $
SET LC2, MW1         $
A2 + B R = A3        $
A1 + B R = A2, BEX, LMAR $
JUMP                 $
DR1, 0 = A2, SKIP    $
```

Conditional Part Alone:

```
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 MST THEN LIT-B-1 = A3, BEX $
MR2, SET LC2, IF LST THEN A1+B = A2 $
A1 OR B = A2, IF LC2 THEN MR1 $
```

Condition

Syntax:

```
<If Clause>    :: = IF <Condition> .
<Condition>    :: = <Not> <Cond>
<Not>          :: = NOT |<Empty>
<Cond>         :: = LST |MST |AOV |ABT |COV |RMI |RDC |LC1 |LC2 |IRQ |EXT |IRQ |
                    SRQ |INT |GC1 |GC2 |
<When Clause>  :: = WHEN <Condition>
<Else Clause>  :: = ELSE <Successor> |<Empty>
```

Semantics:

Each N instruction can perform a test on the Boolean value of one <Cond> or its complement (i.e., NOT <Cond>). 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.

If the <Else Clause> is absent, an ELSE STEP (see <successor>) is substituted by default. A <When Clause> is equivalent to an <If Clause> with the same <Condition> and an <Else Clause> of ELSE WAIT.

Testing of a condition bit causes the bit to be reset. These conditions are:

RMI - Ready MIR bit
Follows memory operations, indicates data in MIR and addresses have been captured and can be changed. Occurs one clock after initialization of memory operation. Required operation before another memory write can occur with full confidence of accuracy.

RDC - Read Complete

Follows memory read, indicates data will be available for entry to B in the next clock.

COV - Counter Overflow

Follows after an INC which caused the CTR to overflow (i.e., exceed 255).

LC1 - Local Condition 1

Tests and resets local Boolean condition bit LC1 (a one bit status flag).

LC2 - Local Condition 2

Tests and resets local Boolean condition bit LC2.

The following four logic unit conditions are not bits which are reset on checking as above, but are dynamic levels and indicate the resultant output from the adder in the phase 3 of the last instruction which had a logic unit operation. These conditions are sustained until execution of another instruction involving the logic unit and may be tested by that instruction. (See Appendix D for timing). Please note the conditions may change due to a type II instruction which alters AMPCR or LIT when either is also specified as an adder input.

AOV - Adder Overflow

Results from an adder operation with carry out of the most significant end of the adder.

LST - Least significant

State of the least significant bit of the adder output.

MST - Most significant

State of the most significant bit of the adder output.

ABT - All Bits True

This condition is true (one) if and only if the adder output is all ones.

The following conditions are legal syntactically but have no meaning in the D Machine simulation due to the limited I/O emulation and other reasons.* The correct code is generated, however, and would run correctly on a D Machine with a full implementation. Similar problems will be mentioned in <Ext Op> in the next section.

IRQ
EXT
INT
SRQ
URQ
GC1
GC2

Examples:

```
IF NOT LC1 THEN A1+B = A2 ELSE SKIP $
WHEN RDC THEN BEX $  Same as below - puts data in B
IF RDC THEN BEX ELSE WAIT $  Same as above
IF LST THEN SET LC1, MR1  $
```

*Such as maintaining compatibility since D Machines come in many sizes and implementations - not all functions are on each.

External Operations

Syntax:

<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|DR1|DR2|DU1|DU2|DW1|DW2|ASR|ASE

<Set Op> ::= SET <Cond Adjust Bit>|RESET GC1| RESET GC2

<Cond Adjust Op> ::= INT|LC1|LC2|GC1|GC2

Semantics:

The external operations are part (or all) of an <N instruction> and therefore a type I instruction. 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 operations MR1, MR2, MW1, and MW2 are the only external operations which have been allowed in the D Machine simulation. All other operations are syntactically correct but have no implementation. If their meaning is important, please refer to reference 1.

MR1 and MR2 are memory read operations. MW1 and MW2 are memory write operations with 1 and 2 referring to the address source registers. Memory addresses are generated by concatenating the 8 bits of the MAR, to form the low order bits, and 8 bits from either BR1 (MR1 and MW1) or BR2 (MR2 and MW2).

The set and reset operations are used to set and reset condition bits. Again, not all operations have been implemented due to the same reasons cited earlier, but they remain syntactically correct.

SET LC1 and SET LC2 are the only operations which are allowed in the D Machine simulation. RESET LC1 or LC2 is not needed since testing resets them automatically.

Examples:

```
SET LC1, MR2    $
MR1, SET LC1    $
MW1             $
```

Logic Operation

Syntax:

```
<Logic Op> :: = <Adder Op> <Inhibit Carry> <Shift Op> <Destination List>
<Adder Op> :: = 0|1|<Monadic>|<Dyadic>|<Empty>
<Monadic>  :: = <Not> <X Select>|<NY Select>
<Not>      :: = NOT|<Empty>
<Dyadic>   :: = <X Select> <Any Op> <Y Select>
                NOT <X Select> <Normal Op> <Y Select>
                <X Select> {<Normal Op>|<Not Y Op>}NOT <Y Select>
                <X Select> + <N Y Select> + 1|<X Select> - <N Y Select> -1
```

Semantics:

The logic operations include the selection of adder inputs, the adder operation, the barrel switch shifting, the destination specifications for the adder and BSW outputs, and the controls for the literal, counter, and shift amount register. The monadic operations are those which have only one explicit input. The selected value or its ones complement may become the adder input depending on the <NOT> function being NOT or <Empty>. The dyadic operations have both an <X Select> and a <Y Select>.

The default <Logic Op> is unconditional 0+0 = . This does cause completion of the prior <Logic Op> in progress in phase 3. (See Appendix D for timing explanations).

Examples:

```
O = CTR   $
A1 AND B011 = A1   $
A2 + NOT CTR R = A2, BEX, CTR, CSAR   $
```

Input Selects

Syntax:

```
<X Select>   :: = 0|A1|A2|A3|CTR|ZEXT|LIT|2|<Empty>
<Y Select>   :: = 0|1|B|B <M> <C> <L>|CTR|ZEXT|LIT|Z|
                  AMPCR|BMAR
<NY 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 a D Machine. Any one of the A registers, or the counter, external source (not implemented in simulation), literal, or Z (also not implemented) may be selected as input to the adder is the <X Select> part of the instruction. The B register is the primary interface for external inputs from the main memory. 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)
C - Central bits of B
L - Least significant bit of B (right most)

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 (no change) of the B part. An F gating will produce the false value (ones complement) of B for that part. There can be no spaces between gatings. If B is specified by itself, then BTTT is assumed. However B is selected as an adder input; its contents are left unchanged.

Whenever any other register is specified as an adder input, it will be right justified in the specified 32 bit input.

Examples:

BTFO	
A1	
AMPCR	
BOTT	Adder input is absolute value of B

Adder and Shift Operators

Syntax:

<Normal Op>	:: = NOR | NRI | NAN | XOR | NIM | JMP | EQV | AND | RIM | OR
<Not Y Op>	:: = + | -
<Any Op>	:: = OAD | AAD | <Normal Op> | <Not Y Op>
<Inhibit Carry>	:: = IC | <Empty>
<Shift Op>	:: = R | L | C | <Empty>

Semantics:

Each <Dyadic> contains two operands (X any Y Select> and an operator. All operators are commutative except for "-", "OAD" and

and "AAD." Commutative operands can appear in any order. The recommended standard order is X Op Y which works for all operators. The following table defines the adder operations.

Commutative Operators	Name	Equivalence
X NOR Y	Nor	$\bar{X}\ \bar{Y}$
X NRI Y	Not Reverse Imply	$\bar{X}$ Y
X AND Y	And	X Y
X NIM Y	Not Imply	X $\bar{Y}$
X XOR Y	Exclusive Or	(X $\bar{Y}$) V ($\bar{X}$ Y)
X EQV Y	Equivalence	(X Y) V ($\bar{X}$ $\bar{Y}$)
X IMP Y	Imply	$\bar{X}$ V Y
X NAN Y	Nand	$\bar{X}$ V $\bar{Y}$
X RIM Y	Reverse Imply	X V $\bar{Y}$
X OR Y	Or (inclusive)	X V Y
X + Y	Add	X plus Y

Non Commutative operators

X - Y	Subtract	X + $\bar{Y}$ + 1
X OAD Y	Or Add	X + (X V Y)
X AAD Y	And Add	X + (X Y)

The carries from 8 bit bytes can be inhibited by specifying IC. Since this was found to be used so infrequently, it is not included in the simulator.

There are four shift operations. One can be selected for each adder operation. The operator specifies shift direction and the SAR register specifies the number of bits. All shifts are completed in the same amount of time (within the same cycle - see Appendix D).

R - Right end-off shift by amount in SAR, filled with left zeros

L - Left end-off shift by complemented 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 R

A2 OR NOT CTR C   Same as A2 RIM CTR C

Destination Operators

Syntax:

<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 will occur at the end of the logic unit operation.

Restrictions

1. At most one each from <Input B>, <Input Ctr>, <Input Mar>, and <Input Sar>.

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.

4. After "=" in the destination list, separate operators with either a comma or blank, but not another "=".

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. The following 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 5 bits
AMPCR	Least significant 12 bits

B, MAR, CTR, SAR and AMPCR registers may have other inputs as shown below:

B Registers Inputs (BC4 and BC8 are not simulated)

B	The barrel switch output is placed into B
BEX	Data from the memory bus is placed into B

BAD	The adder output is placed in the B register (short path to B)
BMI	The MIR contents are placed in B independent of any concurrent change to the MIR
BBE	The barrel switch output ORed with the data from the memory bus is placed into B
BBA	The barrel switch output ORed with the adder output is placed into B
BBI	The barrel swtich output ORed with the MIR contents is placed into B independent of any concurrent change to the MIR.

MAR inputs

LMAR	The literal register content is placed in MAR

CTR inputs

LCTR	The ones complement of the LIT register contents is placed in CTR
INC	Increment CTR by one

SAR input

CSAR	Complement prior content of SAR

If AMPCR is changed by a successor selection of CALL or SAVE these operations take procedence over any specified by the logic unit.

Examples:

= B

= CTR

= A1, BEX, MIR, LCTR, CSAR

Successor

Syntax:

<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 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 the table which follows after the text.

- WAIT - Repeat the instruction in the MAR
- STEP - Step to the next instruction in sequence from MPCR
- SKIP - Skip to the second next instruction in sequence from MPCR
- SAVE - Step as in "STEP" but also save MPCR contents in AMPCR
- CALL - Transfer control to AMPCR+1 address, save current MPCR in AMPCR
- EXEC - Execute instruction in AMPCR+1, then proceed as specified in the executed instruction (MPCR is unchanged)
- JUMP - Transfer control to AMPCR+1
- RETN - Transfer control to AMPCR+2

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.

Successor Command	Successor M Instruction Address	Next Content of MPCR will be	Next Content of AMPCR will be
WAIT	MPCR	MPCR	- (no change)
STEP	MPCR+1	MPCR+1	-
SKIP	MPCR+2	MPCR+2	-
SAVE	MPCR+1	MPCR+1	MPCR
CALL	AMPCR+1	AMPCR+1	MPCR
EXEC	AMPCR+1	MPCR	-
JUMP	AMPCR+1	AMPCR+1	-
RETN	AMPCR+2	AMPCR+2	-

Examples:

WAIT

JUMP

Sample Programs

Two programs in TRANSLANG are given next to aid in the understanding and structure of TRANSLANG. Each is fully documented to aid in its understanding.

EXAMPLE OF MICROPROGRAM
FOR BINARY MULTIPLY

Assumptions

(1) Sign-magnitude number representation

(2) Multiplier in A3; multiplicand in B

(3) Double length product required with resulting most significant part, with sign, in B and least significant part in A3

1. A3 XOR B = , IF LC1 $

2. B_{OTT} = A2, IF MST THEN SET LC1 $

   Comment: Step 1 resets LC1. Steps 1 and conditional part of 2 check signs; if different, LC1 is set.

3. B_{OOO} = B, LCTR $

   Comment: Steps 2 and 3 transfer multiplicant (0 sign) to A2 and clear B.

4. N-2→LIT , 1→SAR $

   Comment: Steps 3 and 4 load the counter with the number (N = magnitude length) to be used in terminating the multiply loop and load the shift amount register with 1.

5. A3 R→A3, SAVE $

   Comment: Begins test at least bit of multiplier and sets up loop.

6. LOOP. IF NOT LST B_{OTT} C = B SKIP ELSE STEP $

7. A2 + B_{OTT}C = B $

8. A3 OR B_{TOO}R = A3, INC, IF NOT COV THEN JUMP ELSE STEP $

   Comment: 6 through 8 - inner loop of multiply (average 2.5 clocks/bit).

9. IF NOT LC1 THEN B_{OTT} = B, SKIP ELSE STEP $

10. B_{1TT} = B $

    Comment: If LC1 = 0, the signs were the same, hence force sign bit of result in B to be a 0.

11. END $

EXAMPLE OF MICROPROGRAM

FOR GENERATION OF FIBONACCI SERIES

Assumptions:

A1 contains starting address for storing of series

A2 contains the number representing the length of the series to be computed

1. A1 = MAR1 $

   Comment: Load starting address of series into address register

2. B_{000} = A3, MIR $

3. B_{001} = B, MW1 $

   Comment: Load initial element of series (0) into A3 and MIR and write it into starting address. Load second element of series (1) into B.

4. A2 = CTR, SAVE $

   Comment: Load counter with length of series; the counter will be incremented for each generation of an element of the series; COV will signify completion. The SAVE sets up the loop.

5. LOOP. IF RMI THEN A1+1 = A1, MAR1, INC, STEP ELSE WAIT $

   Comment: Set up the next address and increment counter

6. A3 + B = MIR $

   Comment: Generate new element in series and place in MIR

7. B → A3, BM1, MW1; IF NOT COV THEN JUMP ELSE STEP $

   Comment: Write new element into next address

   Transfer i - 1 element to A3

   Transfer i element to B

   Test counter overflow for completion (go to LOOP, if not done)

8. END $

IV. TRANSLANG Translator

The TRANSLANG Translator takes programs written in TRANSLANG and generates the microcode for D Machine bit patterns which can run on the D Machine simulator. A full set of error diagnostics is included.

This program is written in RALPH and runs on the UNIVAC 1108. EXEC 8 control cards will not be given as they will vary according to the way the programs are entered in the system.

USE:

Input

(1) The TRANSLANG programs should be created and inserted into a file before running the translator. This file is considered to be on unit number 9 so the user should include a @ USE statement to equate his file with the translators input logical unit number 9.

(2) The Translator asks the user three questions. The answer to these questions (YES or NO) control the amount of output generated. These questions follow:

Question 1 - "NANO AND MPM BIT PATTERN LISTING DESIRED"

If the user wishes to see the 16 bits of the M Memory and the 54 bit Nano memory (if present) as generated for each instruction, reply "YES" otherwise "NO".

Question 2 - "SOURCE INPUT LISTING DESIRED"

If the user wishes to have each statement of his input program listed, reply "YES"; otherwise "NO". If he replies "NO" and an errors are encountered, then all instruction with errors will be listed along with the diagnostic.

Question 3 - "OBJECT OUTPUT LISTING DESIRED"

If the user desires to see the bit patterns generated after an error free syntax pass, reply "YES"; otherwise "NO". If an error occurs and the reply was "YES" the error has precedence and the output listing is suppressed.

The object listing must display up to 80 bits of information in 72 columns. (TTY format) To do this, the bits are displayed in hexadecimal groups of four characters. For example:

```
F003    009    0C40    1000    0800
E001
F004    AC08   00D0    0C00    0000
```

The first 4 characters are the 16 bit M Memory control bits and the next 14 characters give the 54 bits of the nano memory. The last two characters are meaningless except for compatibility and D Machine requirements.

Output

(1) The object binary bit patterns are written out on logical unit #10 for input into the D Machine simulator. Appropriate EXEC 8 commands should be used so that the simulator will be able to reference the object file from logical unit #10. This output occurs whether the object code is listed or not.

(2) Error Messages - The translator has 28 error messages which are always printed along with the instruction in which the error occurred. These error messages are given in Appendix C.

A Sample Translator run follows. Only those lines that are underlined were entered by the user.

```
@XQT

    PLEASE ENTER TRANSLATOR CONTROL PARAMETERS
    NANO AND MPM BINARY BIT PATTERN LISTING DESIRED
    NO
    SOURCE INPUT LISTING DESIRED
    YES
    OBJECT OUTPUT LISTING DESIRED
    YES
    SOURCE PROGRAM FOLLOWS

    LIT = MAR1 $
    1 = LIT $
    MRI $
    WHEN RDC THEN LIT + 1 = MAR1, BEX, STEP $
    MR1, B = A3 $
    WHEN RDC THEN BEX $
    A3 XOR B = , IF LC1 $
    BOTT = A2, IF MST THEN SET LC1 $
    BOOO = B, LCTR $
    1 = SAR, 29 = LIT $
    A3 R = A3 , SAVE $
    IF NOT LST THEN BOTT C = B, SKIP ELSE STEP $
    A2 + BOTT C = B $
    A3 OR BTOO R = A3, INC IF NOT COV THEN JUMP ELSE STEP $
    IF NOT LC1 THEN BOTT = B, SKIP ELSE STEP $
    B1TT = B $
    STEP $
    END $
```

THE TOTAL NUMBER OF ERRORS = 0

OUTPUT READY FOR SIMULATOR INPUT

ADD	MPM				NANO
0	F000	0009	0140	002C	0000
1	E001				
2	F001	0009	0000	0000	0800
3	F002	AC08	0146	0C2C	0000
4	F003	0009	0C40	1000	0800
5	F004	AC08	0000	0C00	0000
6	F005	2009	EC4C	0000	0000
7	F006	4BC9	0440	2000	0000
8	F007	0009	0000	0B01	0000
9	811E				
10	F008	0012	E000	9000	0000
11	F009	5419	0441	8B02	0000
12	F00A	0009	C441	8B02	0000
13	F00B	8029	E81C	9000	0000
14	F00C	2419	0440	0B00	0000
15	F00D	0009	1C40	0B00	0000
16	F00E	0009	0000	0000	0000
17	4000				

V. D MACHINE SIMULATOR

The D Machine simulator takes the microprograms generated by the TRANSLANG Translator and executes them just like the D Machine described throughout this document. In addition to being able to execute these programs, the simulator is a valuable tool for use in debugging microprograms. A user can choose his output in many different formats and can trace, break and dump the simulated memory of 1024 words.

The simulator is also written in RALPH for execution on the UNIVAC 1108. Again EXEC 8 control cards are not given.

Use:

Input

(1) The simulator requires the program which is to be executed reside on logical unit #10, and be of the same format as that created by the TRANSLANG translator. After creating such a file in the translator the user need only insert the proper EXEC 8 control card to allow the simulator to access this file.

(2) Responses to simulator control questions -
The user has no control of how the simulator will execute a given microprogram, but he does have many options on the type of output he wants generated.

Let us examine a sample run to study the options available. Simulator responses are in quotes and user replies are underlined.

@XQT

"PLEASE ENTER THE SIMULATION CONTROL PARAMETERS"

Question 1 - "MAXIMUM NUMBER OF CLOCKS (1-99999999) to SIMULATE = "

99

The user requests the maximum number of clock cycles to simulate. The simulator simulates each clock and all those events inside a clock cycle. It takes one clock per instruction to execute programs if overlapping is included (See Appendix D).

Question 2 - "THE NUMBER OF CLOCKS BETWEEN OUTPUT POINTS = "

99

The user can dictate how much and how often output occurs. This question is concerned with the frequency. In this example, the user wants output only every 99 clocks, which is only at the end of the program since the example will require only 11 clocks to run.

Question 3 - "S MEMORY DUMP REQUESTED AT END OR BREAKPOINT"

YES

The simulator has a 1024 word simulated memory for storing data. This is the only storage medium available outside of the registers in the D Machine. If one desires to look at results of a program stored in memory, or intermediate results, respond YES.

Question 4 - "PRINTED OUTPUT EXPLANATION DESIRED"

YES

The user who is unfamiliar with output options and control will want to respond "YES" to have the following explanation printed out:

"PRINTED OUTPUT IS CONTROLLED BY LINE NO. ID
LINE ID = 1 - ADDRESSES AND CLOCK, 2 = A1, A2, A3, 3 = B, MIR
4 = SAR, LIT, AMPCR, CTR, 5 = BR1, BR2, MAR, BMAR,
6 = CONDITIONS"

This explanation tells the user that from 1 to 6 lines of output can be printed every time output is desired. Each line of output is controlled by a line i.d. which identifies that line and the registers and or addresses associated with it. Conditions are printed out as LST = 1, LC2 = 0, etc. for all the conditions simulated. BMAR is the last concatenation of a base register and the MAR to generate a memory address. Addresses in Line 1 refer to the two phases being executed every clock. The addresses are the Control Store address for each phase. CLOCK is the current clock number being executed. Output always occurs at the end of the clock specified.

If the user responds to Question 4 with NO, the above lines will not appear. In both cases, the next question is as follows:

Question 5 - "ENTER THE NUMBER OF OUTPUT LINES DESIRED"
4

This question seeks the total number of lines that should be printed (regardless of which ones) at each output point. Any number from 1 to 6 is acceptable.

Question 6 - "ENTER THE LINE NUMBER ID, S SEPARATED BY COMMAS"
1, 2, 3, 4

The user should now enter 4 (always the same number as answered in Question 5) line number identifications separated by commas. This uniquely identifies the lines of output desired. In this example, the first four are selected, but any four could have been specified. (Remember four is only the example and not the rule). If 6 were requested in Question 5, then there would be no need for Question 6 and it would not occur.

Question 7 - "BEGIN OUTPUT AT ADDRESS"

<u>0</u>

Question 8 - "END OUTPUT AT ADDRESS"

<u>9</u>

Question 7 and 8 are used to define the output address limits. If the beginning address is the first program address and the end address is greater than or equal to the last instruction address, then output can occur at any clock according to the other output parameters. If output was desired only in one section of a program the answers to Questions 7 and 8 could be set to bracket this block and thus could trace the simulation only in this block except upon termination which always causes the printing of all registers.

Question 9 - "TRACE REQUESTED"

<u>NO</u>

In addition to the output options already mentioned, one may specify a particular address and get output every time that address is executed. In the example, this was not requested, but if it had the following message would be printed:

Question 10 - "ENTER TRACE ADDRESS" (not in example because of NO in 9)

<u>address</u>

The address to trace on would then be entered.

Question 11 - "BREAKPOINT REQUESTED"
<u>YES</u>

Another output option available is the breakpoint, which allows the users to stop the simulation whenever an instruction is executed with an address greater than or equal to this address (which is entered to answer Question 12 if YES is the answer to eleven). When the simulation stops, output is generated according to line id,s as entered earlier. A memory dump can then be taken if requested by answer 3. Finally, you may request a new breakpoint before the simulation continues.

Question 12 - "ENTER BREAKPOINT ADDRESS"
<u>7</u>

This is the breakpoint address requested by question 11.

Question 13 - "SOURCE PROGRAM LISTING DESIRED"
<u>YES</u>

If the user would like to list the program being simulated, reply <u>YES</u> and the following line will be printed, followed by the program listing in the same format as object listings in the Translator.

"THE PROGRAM BEING SIMULATED FOLLOWS"

Program Listing

Question 14 - "IS THERE ANY S MEMORY INPUT"

<u>YES</u>

If the user wishes to enter data into the simulated memory before the execution begins, reply YES; otherwise, NO. If the answer is YES, the following message is printed:

"ENTER MEMORY INPUT INFORMATION
MEMORY IS ALL ZERO TO START
ENTER VALUES IN CONSECUTIVE BLOCKS AS REQUESTED BELOW"

This message instructs the user that the simulated memory is initially cleared to zero and that input will be in consecutive blocks. (i.e., Address (i), i+1, i+2 then address (j), j+1, j+2, j+3, etc. - see below).

Question 15 - "WILL ANY MEMORY INPUT BE FROM A FILE"

<u>NO</u>

Users can insert data into the simulated memory from two sources: TTY keyboard or from a file of data already created. If a data file has already been created, answer YES and the following question will be printed. All input must be from one or the other.

Question 16 - "LOGICAL UNIT NUMBER FOR MEMORY INPUT IS"

<u>12</u>

In this example, a data file exists on logical unit number 12. Any legal unit number is acceptable. The user should include appropriate EXEC 8 control cards to equate a file to this logical unit number. Data should appear in this file just as if it were going to be keyed in on the TTY in response to the simulator questions mentioned below.

Question 17 - "STARTING S MEMORY ADDRESS = "

<u>1</u>

The user should enter the first S Memory address where the first data point of this block should be placed.

Question 17 - "FINAL S MEMORY ADDRESS FOR THIS BLOCK = "

<u>2</u>

Enter the final simulated memory address where data will be entered in consecutive locations. In this example, the block is from S Memory location one to S Memory location two (i.e., two data values must be entered). Up to 1024 values could be entered in one clock. Block addresses do not have to be sequential. For a contrasting example, we could enter data from locations 9 to 25 then 3 to 5 and finally 50 - 56. Questions 17 and 18 are repeated as many times as requested by the answer to Question 20.

If Question 15 was answered with NO, the following question will be repeated for each data point requested by questions 17 and 18 (i.e., answer 18 - answer 17 + 1 repetitions).

Question 19 - "SET S MEMORY (i) = "

<u>data</u>

Respond with a positive integer $\leq 2^{32}-1$. To enter negative numbers, users must enter the ones complement (32 bits) of the positive number. (i.e., -1 is represented by the decimal integer equal to $2^{32}-2$).

This question is not present if answer 15 is YES.

Question 20 - "IS THERE ANY MORE MEMORY INPUT"

<u>NO</u>

If more blocks of data are to be entered, reply YES and Questions 17, 18, 19 and 20 will be repeated again. In our example, the reply was NO which ends S Memory input and also all data input which allows the simulation to begin. The following message confirms the NO response:

"END OF DATA INPUT - SIMULATION BEGINS"

With the end of data input the simulation begins. Output as requested by answers to the simulator questions is generated in the format as shown in the sample run which follows.

RUN TIME QUESTIONS

If the user has not requested any dumps, traces, or breakpoints, the only run time question will occur after all the register dumps have occurred (i.e., the simulation has exceeded the maximum clocks or the end microinstruction - 4000 was executed). This question follows the end of simulation run message:

"END OF SIMULATION RUN"

Question R-1 - "DO YOU WISH TO RESTART SIMULATOR"

<u>NO</u>

This response would end simulator execution, but a YES would cause an entire new run starting with Question 1. This question always ends a simulation run regardless of output options.

Memory Dumps

If the answer to Question 3 was YES, then a memory dump will occur at each breakpoint and upon termination of the simulation run. At this point, the following message is printed:

"MEMORY DUMP REQUESTED
ENTER VALUES AS DONE IN MEMORY INPUT"

Memory is dumped in consecutive blocks by selectively specifying the beginning and end addresses of these blocks as done in memory input. These questions follow.

Question R-2 - "STARTING S MEMORY ADDRESS = "
i

Answer with the beginning block address as in memory input.

Question R-3 - "FINAL S MEMORY ADDRESS = "
j

Answer with the final block address as in memory input.

After these two questions, the following message is printed followed by the contents of the requested S Memory words. (6 integer values per line).

"S MEMORY (i) TO S MEMORY (j) = "

j-i+1 data values with 6 per line

After the block of data is listed, the following question is listed.

Question R-4 - "DO YOU WISH TO DUMP MORE S MEMORY"
NO

If no more memory is to be dumped, reply NO and the next question will be either R-5 or R-1 depending on whether this ends the simulation or its is merely a breakpoint.

If YES is the reply then Questions R-2 and R-3 are repeated.

BREAKPOINTS

Question R-5 - "NEW BREAKPOINT REQUESTED"
YES

This questions is listed in the case where a breakpoint has occurred. (This is indicated by a register dump, several line feeds, then this message

"BREAKPOINT"

followed by several more line feeds. At this point, memory dump questions are listed and data printed if a memory dump was requested. If no dump was requested or after the dump is completed, Question R-5 occurs).

If a new breakpoint address is desired, answer YES and the following question will be listed. If NO, processing continues from this point on with no more breakpoints.

Question R-6 = "ENTER BREAKPOINT ADDRESS (0 IMPLIES RESTART)"
<u>83</u>

If answer R-5 was YES, one can enter a new breakpoint address or zero. If zero is entered the simulation will start over from Question 1 with all previous answers, registers and memory values cleared.

<u>General Comments</u>

1. All D Machine registers are cleared to start or upon restarting a simulation.

2. Some conditions will never appear to be cleared or set if listed every clock during a particular portion of code using these conditions (RMI and RDC). These conditions work but due to when they occur and when the output occurs, they appear listed incorrectly. This is not the case (please refer to Appendix D) however and no alarm is necessary since the proper code will be executed.

3. INT condition is listed but not implemented.

4. If syntactically correct, code is passed to the simulator with unimplemented code points, error messages will result. (Refer to TRANSLANG semantics for unimplemented code points).

Sample Program Run of the D Machine Simulator

The program being simulated is a simple program which reads in memory locations 1 and 2 and adds them together with results stored in A2.

```
 PLEASE ENTER THE SIMULATION CONTROL PARAMETERS

 MAXIMUM NUMBER OF CLOCKS (1-99999999) TO SIMULATE =
99
 THE NUMBER OF CLOCKS BETWEEN OUTPUT POINTS =
99
 S MEMORY DUMP REQUESTED AT END OR BREAKPOINT
YES
 PRINTED OUTPUT CONTROL EXPLANATION DESIRED
YES
 PRINTED OUTPUT IS CONTROLLED BY LINE NO. ID
 LINE ID = 1-ADDRESSES AND CLOCK, 2=A1,A2,A3, 3=B,MIR
 4=SAR,LIT,AMPCR,CTR, 5=BR1,BR2,MAR,EMAR, 6=CONDITIONS


 ENTER THE NUMBER OF OUTPUT LINES DESIRED
4
 ENTER THE LINE NUMBER ID,S SEPARATED BY COMMAS
1,2,3,4
 BEGIN OUTPUT AT ADDRESS =
0
 END OUTPUT AT ADDRESS =
9
 TRACE REQUESTED
NO
 BREAKPOINT REQUESTED
YES
 ENTER BREAKPOINT ADDRESS
7
 SOURCE PROGRAM LISTING DESIRED
YES


 THE PROGRAM BEING SIMULATED FOLLOWS


     0 F000 0009 0140 002C 0000
     1 E001
     2 F001 0009 0000 0000 0800
     3 F002 AC08 0146 0C2C 0000
     4 F003 0009 0C40 1000 0800
     5 F004 AC08 0000 0C00 0000
     6 F005 0009 EC40 2000 0000
     7 F006 0009 0000 0000 0000
     8 4000
 IS THERE ANY S MEMORY INPUT
YES
```

```
 ENTER MEMORY INPUT INFORMATION
 MEMORY IS ALL ZERO TO START
 ENTER VALUES IN' CONSECUTIVE BLOCKS AS REQUESTED BELOW
 WILL ANY MEMORY INPUT BE FROM A FILE
NO
 STARTING S MEMORY ADDRESS =
1
 FINAL S MEMORY ADDRESS FOR THIS BLOCK =
2
 SET S MEMORY (      1) =
12
 SET S MEMORY (      2) =
3
 IS THERE ANY MORE MEMORY INPUT
NO


 END OF DATA INPUT - SIMULATION BEGINS


P(1) ADDR =      7    P(3) ADDR =      6    CLOCK =          10
A1 =           0        A2 =            15      A3 =              12
B =            3       MIR =             0
SAR = 0        LIT =     1       AMPCR =     0       CTR =        0



BREAKPOINT



 MEMORY DUMP REQUESTED
 ENTER VALUES AS DONE IN MEMORY INPUT


 STARTING S MEMORY ADDRESS =
1
 FINAL S MEMORY ADDRESS FOR THIS BLOCK =
2


 S MEMORY (     1)  TO S MEMORY (     2) =


        12                   3
 DO YOU WISH TO DUMP MORE S MEMORY
NO
 NEW BREAKPOINT REQUESTED
NO
```

END OF SIMULATION - REGISTERS CONTAIN

```
P(1) ADDR. =     7     P(3) ADDR. =     7    CLOCK =          11
A1 =         0       A2 =         15      A3 =          12       B =          3
MIR =         0        SAR = 0     LIT =   1      CTR =  0        AMPCR =   0
BR1 =  0     BR2 =     0     MAR =  2       BMAR =     2
LC1=0 LC2=0 LST=1 ABT=0 ABV=0 COV=0 PMI=1 RDC=0 INT=0
```

MEMORY DUMP REQUESTED
ENTER VALUES AS DONE IN MEMORY INPUT

STARTING S MEMORY ADDRESS =
1
FINAL S MEMORY ADDRESS FOR THIS BLOCK =
2

S MEMORY (1) TO S MEMORY (2) =

```
        12              3
```

DO YOU WISH TO DUMP MORE S MEMORY
0?
NO
END OF SIMULATOR RUN
DO YOU WISH TO RESTART SIMULATOR
NO

APPENDIX A

TRANSLANG SYNTAX

⟨Program⟩ :: = ⟨Body⟩ ⟨End Line⟩

⟨Label⟩ :: = ⟨Letter⟩ | ⟨Label⟩ ⟨Letter⟩ | ⟨Label⟩ ⟨Digit⟩

⟨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

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

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

⟨Comment⟩ :: = COMENT {Any sequence of ⟨Characters⟩ except "$"} $

⟨Line⟩ :: = ⟨N Instruction⟩ $ | ⟨Literal Assignment⟩ $

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

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

⟨Symbol⟩ :: = ,|+|-|/|=|.|*|(|)|$

⟨Assignment Op⟩ :: = =

⟨Literal Assignment⟩ :: = ⟨Literal⟩ = AMPCR | ⟨Literal⟩ = SAR |
⟨Literal⟩ = SAR, ⟨Literal⟩ = ⟨LIT⟩ |
⟨Literal⟩ = ⟨LIT⟩, ⟨Literal⟩ = SAR |
⟨Literal⟩ = ⟨LIT⟩

⟨Literal⟩ :: = ⟨Integer⟩ | CØMP ⟨Integer⟩ | ⟨Label⟩ | ⟨Label⟩ -1

⟨Integer⟩ :: = ⟨Digit⟩ | ⟨Integer⟩ ⟨Digit⟩

⟨Lit⟩ :: = LIT | SLIT

⟨N Instruction⟩ :: = ⟨Unconditional Part⟩ ⟨Conditional Part⟩ |
⟨Label⟩ . ⟨Unconditional Part⟩ ⟨Conditional Part⟩

⟨Unconditional Part⟩ :: = ⟨Component List⟩ | ⟨Empty⟩

⟨Component List⟩ :: = ⟨Component⟩ | ⟨Component List⟩ ⟨Component⟩

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

⟨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|DR1|DR2|DW1|DW2|DU1|DU2|ASR|ASE

⟨Set Op⟩ :: = SET ⟨Cond Adjust Bit⟩ | RESET GC1 | RESET GC2

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

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

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

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

⟨Not⟩ :: = NØT | ⟨Empty⟩

⟨X Select⟩ :: = 0|A1|A2|A3|CTR|ZEXT|LIT|Z|⟨Empty⟩

⟨N Y Select⟩ :: = ⟨Not⟩ ⟨Y Select⟩

⟨Y Select⟩ :: = 0|1|B|B ⟨M⟩ ⟨C⟩ ⟨L⟩ | CTR | ZEXT | LIT | Z | AMPCR | BMAR

⟨M⟩ :: = ⟨Gating⟩

⟨C⟩ :: = ⟨Gating⟩

⟨L⟩ :: = ⟨Gating⟩

⟨Gating⟩ :: = 0|T|F|1

⟨Dyadic⟩ :: = ⟨X Select⟩ ⟨Any Op⟩ ⟨Y Select⟩ |
NØT ⟨X Select⟩ ⟨Normal Op⟩ ⟨Y Select⟩ |
⟨X Select⟩ {⟨Normal Op⟩ | ⟨Not Y Op⟩ } NØT ⟨Y Select⟩ |
⟨X Select⟩ + ⟨N Y Select⟩ +1 | ⟨X Select⟩ - ⟨N Y Select⟩ -1

⟨Normal Op⟩ :: = NØR|NRI|NAN|XØR|NIM|IMP|EQV|AND|RIM|ØR

⟨Not Y Op⟩ :: = + | -

⟨Any Op⟩ :: = ØAD|AAD|⟨Normal Op⟩ | ⟨Not Y Op⟩

⟨Inhibit Carry⟩ :: = IC | ⟨Empty⟩

⟨Shift Op⟩ :: = R | L | C | ⟨Empty⟩

⟨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

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

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

⟨If Clause⟩ :: = IF ⟨Condition⟩

⟨Condition⟩ :: = ⟨Not⟩ ⟨Cond⟩

⟨Cond⟩ :: = LST|MST|AØV|ABT|CØV|RMI|RDC|EXT|SRQ|URQ|GC1|GC2|IRQ|INT|LC1|LC2

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

⟨Else Clause⟩ :: = ELSE {A non-Empty ⟨Successor⟩}|⟨Empty⟩

⟨When Clause⟩ :: = WHEN ⟨Condition⟩

⟨End Line⟩ :: = END

APPENDIX B

APPENDIX B

TRANSLANG KEY WORDS

KEY WORDS

The following words are reserved in TRANSLANG and may not be used as labels. No imbedded blanks are allowed.

A1	A1 Register X Select or destination operator.
A2	A2 Register X Select or destination operator.
A3	A3 Register X Select or destination operator.
AAD	And Add logic operator: X AAD Y ← →X+(XY)
ABT	All Bits True or Adder Bit Transmit dynamic condition from phase 3 of prior M-instruction doing Adder Op.
AMPCR	Alternate Microprogram Count Register Y Select or destination operator from barrel switch 12 LS bits.
AND	And logical operator: X AND Y← →XY
AOV	Adder overflow, dynamic condition of previous M-instruction using adder, true if addition results in overflow.
B	B Register Y Select same as BTTT; or To B from barrel switch, destination operator.
BAD	To B from adder, destination operator.
BBA	To B from adder OR barrel switch, destination operator.
BBE	*To B from external bus OR barrel switch, destination operator.*
BBI	To B from prior MIR contents OR barrel switch, destination operator.
BC4*	To B from adder "not 4 bit carry" replicated and shifted, destination operator.
BC8*	To B from adder "not 8 bit carry" replicated and shifted, destination operator.
BEX	To B from external bus, destination operator.

BMI	To B from prior MIR contents, destination operator.
BR1	To Base Register 1 from barrel switch 2nd LS Byte, destination operator.
BR2	To Base Register 2 from barrel switch 2nd LS Byte, destination operator.
C	Circular shift right the entire adder output. Operation takes place in barrel switch.
CALL	Call a procedure: Use AMPCR + 1 as address and new MPCR; old MPCR to AMPCR. Successor.
COMENT	Allows for the inclusion of documentation on a listing.
COMP	Complement as appropriate for destination of literal assignment.
COV	Counter overflow condition bit, reset dominant.
CSAR	Complement SAR, destination operator.
CTR	To counter from ones complement of barrel switch LS Byte, destination operator. X or Y Select: into MS Byte.
DL1*	Device Lock using BR1/MAR for device ident.
DL2*	Device Lock using BR2/MAR for device ident.
DR1*	Device Read using BR1/MAR for device ident.
DR2*	Device Read using BR2/MAR for device ident.
DU1*	Device Unlock using BR1/MAR for device ident.
DU2*	Device Unlock using BR2/MAR for device ident.
DW1*	Device Write using BR1/MAR for device ident.
DW2*	Device Write using BR2/MAR for device ident.
ELSE	Sequential operator prefix to false successor.
END	Bracket word to end a program.
EQV	Equivalence logical operator: $X \text{ EQV } Y \leftarrow\rightarrow XY \vee \overline{X}\overline{Y}$
EXEC	Executes out of sequence: Use AMPCR + 1 as address. Successor.
EXT*	External condition bit externally set, reset by test.
F	False gating of B as part of Y Select.
GC1*	Global condition bit 1: may be set by SET GC1 if presently reset in all Interpreters. Tested without resetting. Used as RESET GC1, resets GC1.

GC2*	Global condition bit 2: may be set by SET GC1 if presently reset in all Interpreters. Tested without resetting. Used as RESET CG2, resets GC2.
IC*	Inhibit carry between bytes.
IF	Starts the conditional part of an instruction.
IMP	Imply logical operator: X IMP Y $\leftrightarrow \overline{X} \vee$ Y
INC	Increment counter destination operator; set COV when overflowing from all ones to all zeros.
INT	Used as SET INT, interrupts all Interpreters. Interrupt Interpreters condition bit: set by any Interpreter, own is reset by testing.
IRQ*	Interrupt from locked but unselected device (can be status or data interrupt).
JUMP	Jump to address in AMPCR + 1 and put that address in MPCR. Successor.
L	Left shift end off the entire adder output, right fill with zeros. Operation takes place in barrel switch.
LC1	Local condition bit 1: may be set, or tested which resets.
LC2	Local condition bit 2: may be set, or tested which resets.
LCTR	Ones complement of the literal register contents will be placed in the counter and COV reset. Destination operator.
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.
LMAR	Literal register contents will be placed in MAR. Destination operator.
LST	Least significant bit of adder output, dynamic condition from phase 3 of previous M-instruction doing adder op.
MAR	Memory address register destination operator: from barrel switch LS Byte.
MAR1	Memory address 1 destination operator: same as BR1, MAR
MAR2	Memory address 2 destination operator: same as BR2, MAR

MIR	Memory Information Register destination operator from barrel switch.
MR1	Read from memory address BR1/MAR mem dev op.
MR2	Read from memory address BR2/MAR mem dev op.
MST	Most significant bit of adder output, dynamic condition from phase 3 of previous M-instruction doing adder op.
MW1	Write the content of MIR to memory address BR1/MAR mem dev op.
MW2	Write the content of MIR to memory address BR2/MAR mem dev op.
NAN	Not And logical operator: X NAN Y$\leftarrow\rightarrow\bar{X} \vee \bar{Y}$
NIM	Not Imply logical operator: X NIM Y$\leftarrow\rightarrow X\bar{Y}$
NOR	Nor logical operator: X NOR Y$\leftarrow\rightarrow\bar{X}\bar{Y}$
NOT	Complement monadic or condition operator NOT X$\leftarrow\rightarrow\bar{X}$ Complement Y select for commutative operators.
NRI	Not Reverse Imply logical operator: X NRI Y$\leftarrow\rightarrow\bar{X} \vee Y$
OAD	Or Add logical operator: X OAD Y$\leftarrow\rightarrow X + (X \vee Y)$
OR	Or logical operator: X OR Y$\leftarrow\rightarrow X \vee Y$
R	Right shift end off the entire adder output, left fill with zeros. Operation takes place in barrel switch.
RDC	Read complete bit: set when external data is ready for input to B, reset by testing.
RESET	Reset the condition bit specified.
RETN	Return: use AMPCR + 2 as address and as new content for MPCR. Successor.
RIM	Reverse Imply logical operator: X RIM Y$\leftarrow\rightarrow X \vee \bar{Y}$
RMI	Ready MIR bit: set externally when data has been received from MIR. Reset by testing.

SAR	Shift Amount Register destination operator from LS bits of barrel switch or from literal assignment.
SAVE	Save the MPCR in AMPCR: use MPCR + 1 as M-address and as next MPCR. Successor.
SET	Set the conditional bit specified: either LC1 or LC2.
SKIP	Skip the next instruction; use MPCR + 2 as M-address and as next MPCR. Successor.
SLIT	Literal assignment in SAR converted form.
SRQ*	Solicited request bit. Set externally, reset by testing.
STEP	Step to next instruction: use MPCR +1 as M-address and as next MPCR. Successor.
T	True gating for B register.
THEN	Starts the true alternative of conditional instruction.
URQ*	Unsolicited request bit. Set externally, reset by testing.
WAIT	Wait for condition M-address is MPCR; MPCR and AMPCR unchanged. Successor.
WHEN	Starts a conditional instruction, has an implicit ELSE WAIT.
XOR	Exclusive Or logical operator: $X \text{ XOR } Y \leftrightarrow X\bar{Y} \vee \bar{X}Y$
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.
ZEXT*	Middle bytes of machine larger than 2 bytes as X select and/or Y select.

* Denotes Key Word whose implementation (D Machine action) has not been included in the simulator. They are included in the syntax to allow completeness.

APPENDIX C

TRANSLANG Translator Error Messages

ERROR - ILLEGAL SET OR RESET MODIFIER

ERROR - NOT FOLLOWED BY ILLEGAL KEY WORD

ERROR - UNRECOGNIZABLE FORMAT

ERROR - UNRECOGNIZABLE TYPE 2 FORMAT

ERROR - NO END DELIMITER

ERROR - NO SUCCESSOR FOUND AFTER ELSE

ERROR - NO CONDITION FOLLOWING IF

ERROR - CONDITION NOT FOLLOWED BY THEN

ERROR - THEN FOLLOWED BY ILLEGAL OPERATION

ERROR - ILLEGAL FORMAT FOR LOGICAL OP

ERROR - ILLEGAL OR MISSING Y SELECT

ERROR - ILLEGAL ADDER OPERATION

ERROR - ILLEGAL NOT ADDER OP COMBINATION

ERROR - ADDER OPERATION EQUAL SIGN MISSING

ERROR - LCTR AND MAR CANNOT OCCUR TOGETHER

ERROR - LMAR AND CTR CANNOT OCCUR TOGETHER

ERROR - IMPROPER DELIMITER

ERROR - MISSING EQUAL SIGN IN TYPE 2 INSTRUCTION

ERROR - ILLEGAL KEY WORD AFTER SAR

ERROR - MISSING KEY WORD SAR AFTER LIT,

ERROR - UNABLE TO RESOLVE LABEL "label"

ERROR - ILLEGAL TYPE 2 INSTRUCTION

ERROR - SAR LABEL VALUE GREATER THAN 31

ERROR - LIT LABEL VALUE GREATER THAN 255

ERROR - AMPCR LABEL VALUE BIGGER THAN 4095

ERROR - ILLEGAL LITERAL ASSIGNMENT

ERROR - NUMBER GREATER THAN 4096

ERROR - IMPROPER LABEL - MORE THAN 6 CHARACTERS

APPENDIX D

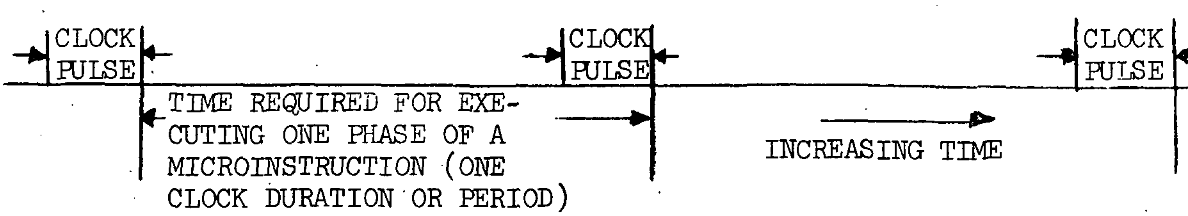
APPENDIX D

D MACHINE (INTERPRETER) INSTRUCTION PHASING AND CONTROLS

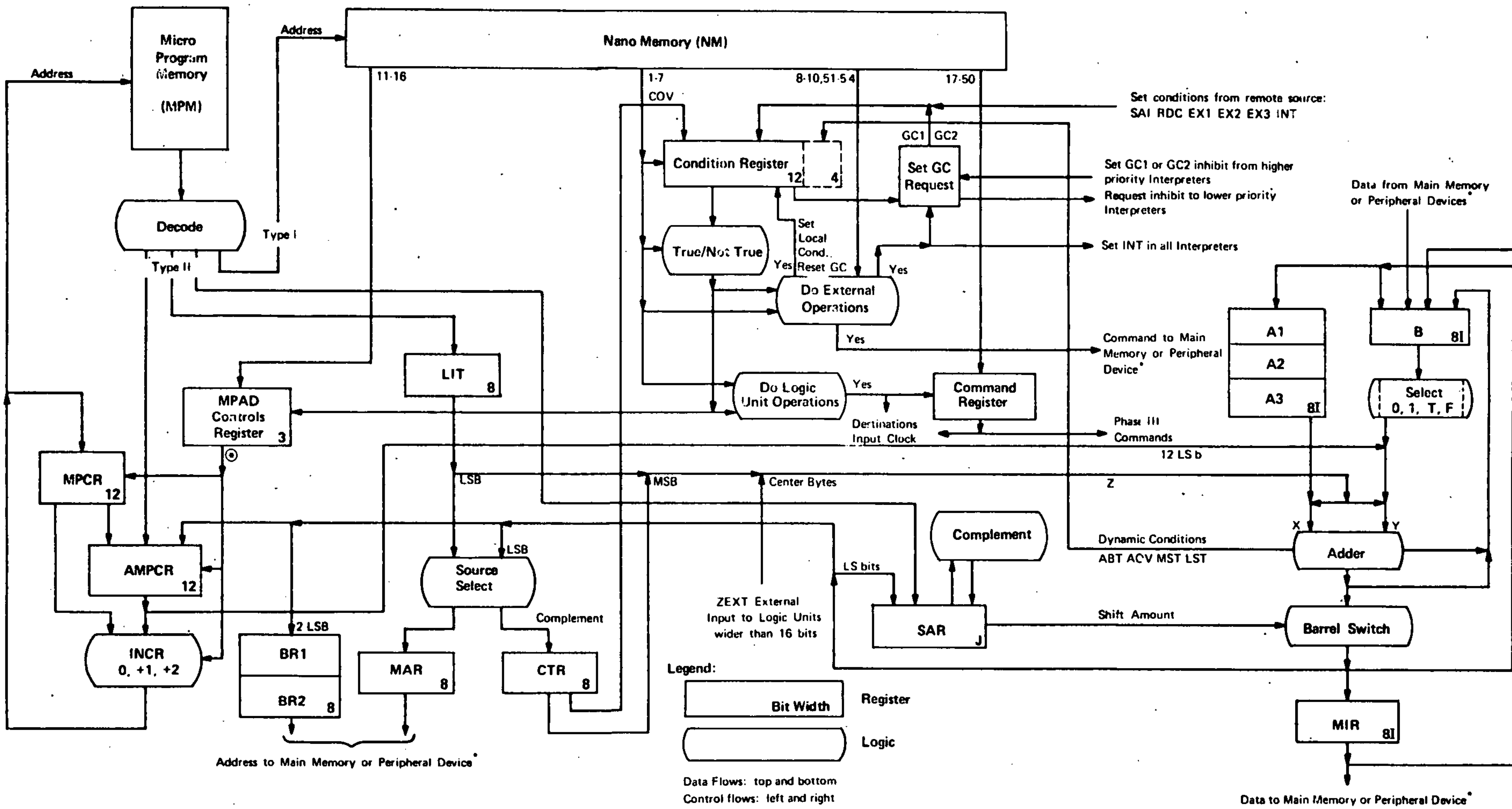
Time phasing of instructions is described in this appendix, both in terms of the partial order of occurrence of the various events and in terms of a sequence of instructions. Controls available in the Interpreter that can be generated by TRANSLANG are reflected in the contents of the words of either 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 operations.

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.



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



⊙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 $\geq 3 + \log_2 I$

Figure D-1. Interpreter Data and Control Flow phases allows starting the execution of a new microinstruction each time a new "clock duration" period starts.

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 continuously 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	$B_{3,2} \rightarrow$	B_3	$D_{3,2} \rightarrow$	D_3	F_3

Clock 0 1 2 3 4 5 6 7

Increasing Time →

Figure D-2. Example of Phased Execution of Microinstructions

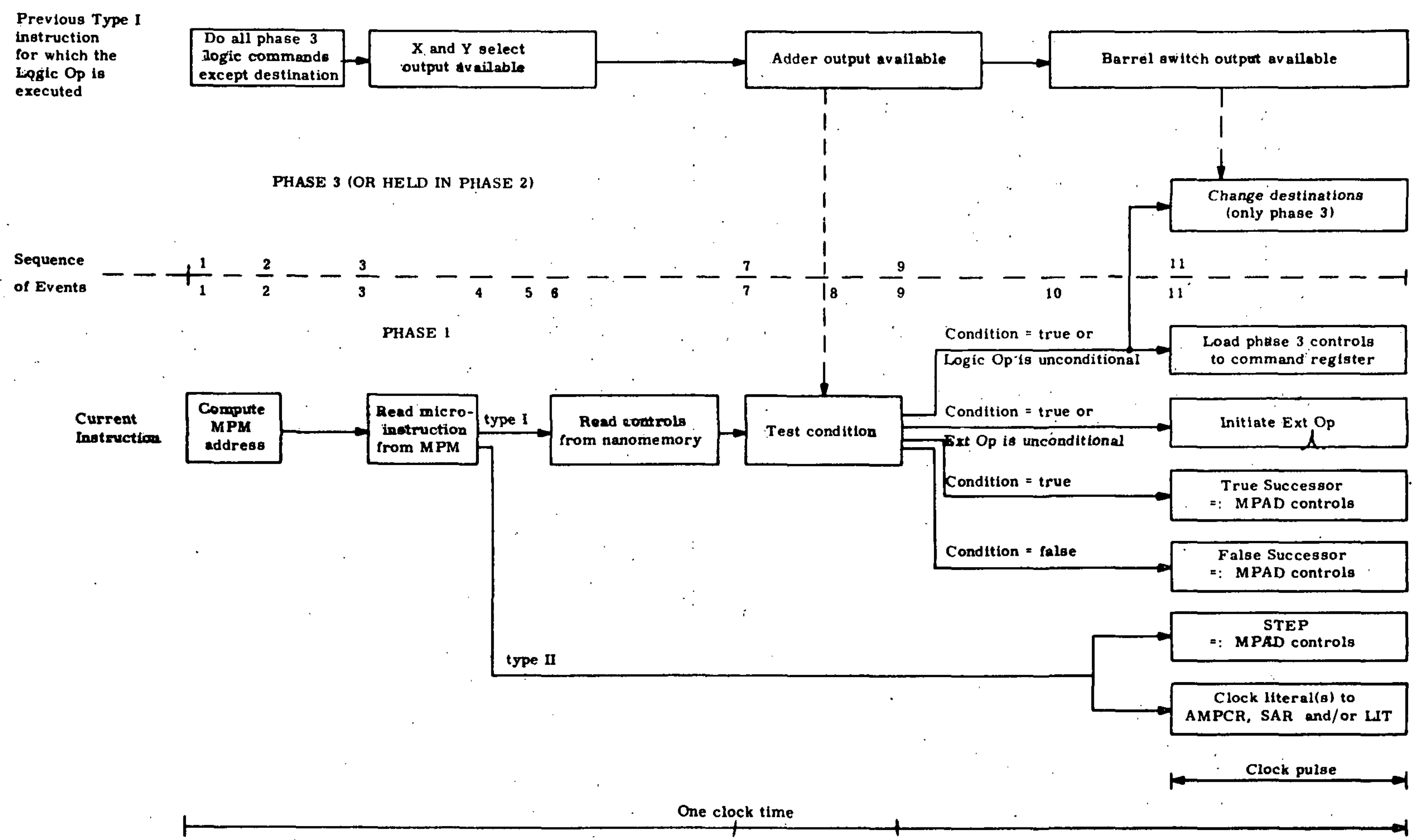


Figure D-3. Instruction Time Flow and Decision Diagram

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 microinstruction. 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.

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

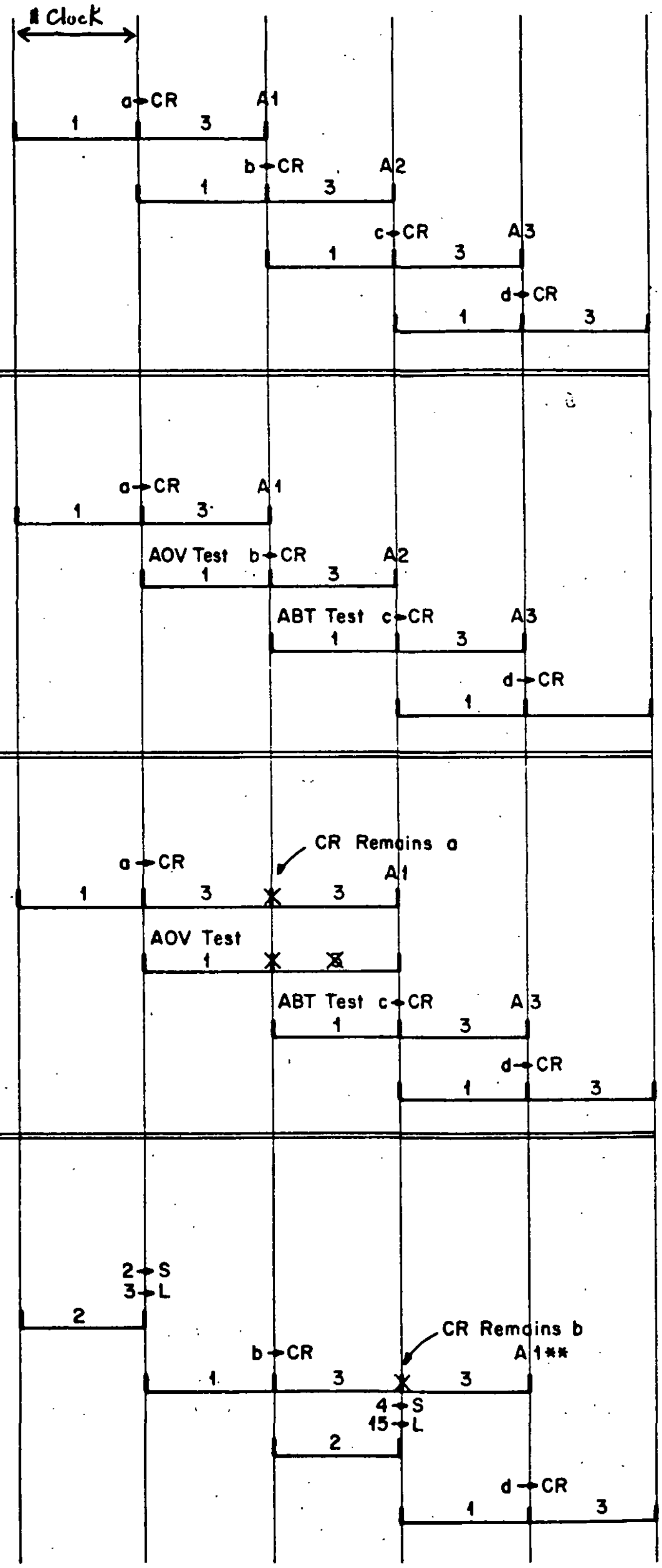
N [45 - 46] MAR } not totally independent, since they share N [46]

N [46 - 48] CTR

N [49 - 50] SAR

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

TIMING EXAMPLES



1. All Type I unconditional instructions

   a. A1 + B → A1

   b. A2 + B → A2

   c. A3 + B → A3

   d. A1 C → A1;

---

2. All type I instructions
   Both AOV and ABT test true

   a. A1 + B → A1

   b. If AOV then A2 + B → A2

   c. If ABT then A3 + B → A3

   d. A1 C → A1;

---

3. All Type I instructions
   AOV tests false; ABT tests true

   a. A1 + B → A1

   b. If AOV then A2 + B → A2

   c. If ABT then A3 + B → A3

   d. A1 C → A1;

---

4. Type I and Type II instructions
   Resulting A1 contains least four bits left justified

   a. 2 → SAR; 3 → LIT

   b. A1 and LIT C → A1

   c. 4 → SAR; 15 → LIT

   d. A1 C → A1;

MICROPROGRAM WORD CONTENT

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

∅ indicates unused, 0 supplied by TRANSLANG translator.

* 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.

N-WORD CONTENT

The assignment of bits and their menaing 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.

LEGEND:

+	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		Not implemented	CAJ	CAJ	S
0	0	0	1		Not implemented	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	RMI	Memory accepts MIR info	Memory	Test	S
1	0	1	0	RDC	Read complete	Memory	Test	S
1	0	1	1		Not implemented			
1	1	0	0		Not implemented			
1	1	0	1		Not implemented			
1	1	1	0		Not implemented			
1	1	1	1		Not implemented			

*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.

@CAJ is condition adjust N [8 - 10]

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

$\boxed{5}$ 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		
0	1	1		Not implemented
1	0	0		
1	0	1		
1	1	0		
1	1	1	SET LC1	Set local condition 1

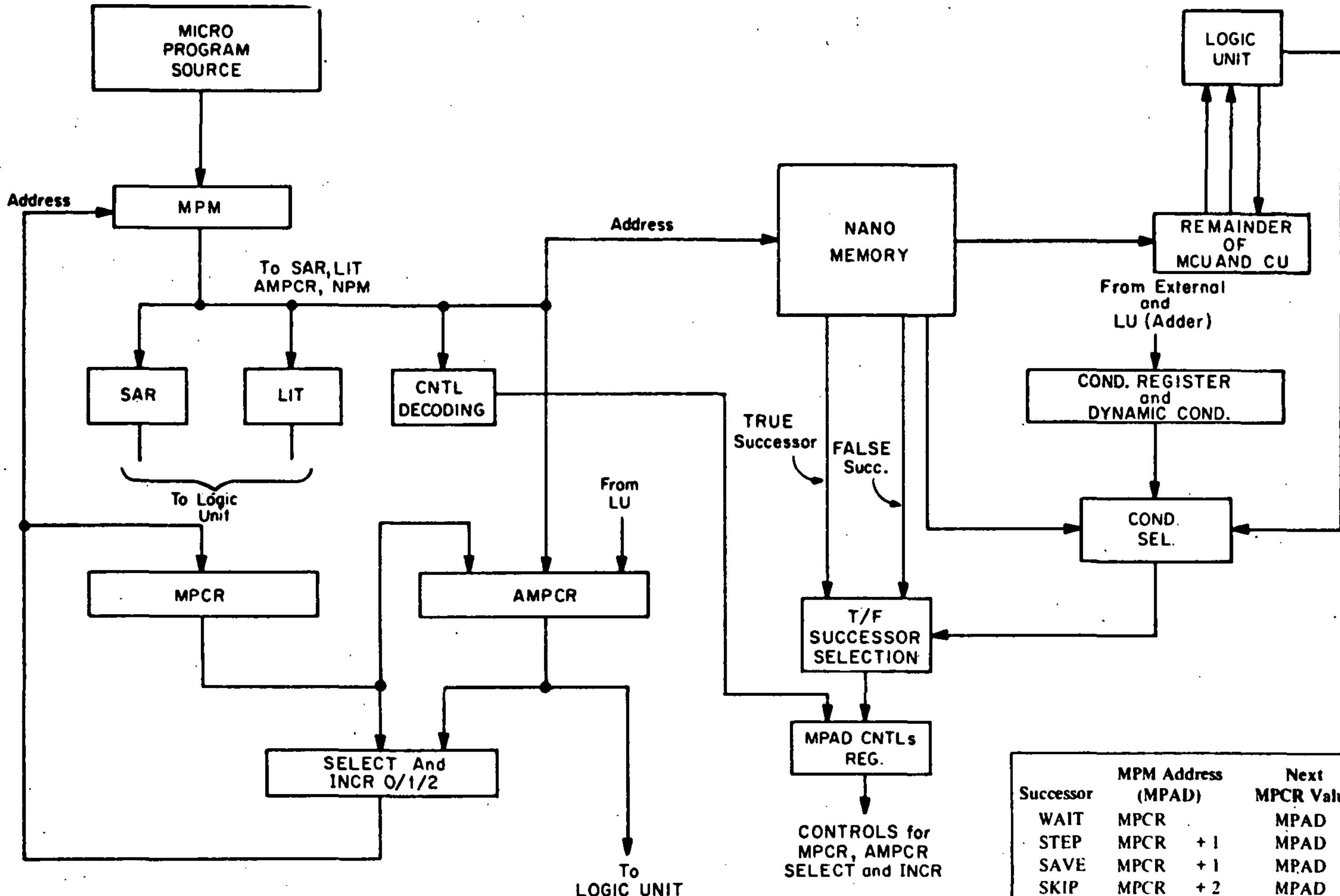
MPAD Controls

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). $MPCR_o$ is the value in the MPCR before the end of phase 1. For type II instructions, MPAD becomes 1 + $MPCR_o$ and MPCR becomes $MPAD_o$--"STEP".

11 12 13			Registers Changed at end of Phase 1		14 15 16
TRUE SUCCESSOR Used if SC=1	MPAD controls register	MPM address will be	MPCR receives value	AMPCR receives value	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	$MPCR_o$*	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	$MPCR_o$*	1 1 0
1 1 1	RETN	2+AMPCR	MPAD		1 1 1

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

MICROPROGRAM INSTRUCTION SEQUENCING



Successor	MPM Address (MPAD)	Next MPCR Value	Next AMPCR Value
WAIT	MPCR	MPAD	*
STEP	MPCR +1	MPAD	*
SAVE	MPCR +1	MPAD	MPCR
SKIP	MPCR +2	MPAD	*
JUMP	AMPCR +1	MPAD	*
EXEC	AMPCR +1	*	*
CALL	AMPCR +1	MPAD	MPCR
RETN	AMPCR +2	MPAD	*

*no change

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		Not implemented
0	1	1	CTR	Counter to MS byte*
1	0	0		Not implemented
1	0	1	A1	A1 register
1	1	0	A2	A2 register
1	1	1	A3	A3 register

* Zeros elsewhere

20	21	22	23	24	25	26	ADDER INPUT Y SELECT	
0	0						BO --	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						Bl --	1 in MS bit
		0	0	0			B-O-	0 in center bits
		1	0	0			B-T-	B in center bits
					0	0	B--O	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		Not implemented
0	1	0	1	1	0	0	CTR	Counter to MS byte*
0	1	1	0	1	0	1		Not implemented
0	0	1	1	0	0	1	AMPCR	AMPCR in least 12 bits*
0	1	1	1	1	0	1	BMAR	Last concatenation of MAR and base register

*Zeros elsewhere

$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.

$\boxed{27}$ INHIBIT CARRIES

0 Allow carries

1 Not implemented

$\boxed{28 \quad 29 \quad 30 \quad 31}$ 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 $\vee$ 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 shifter 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		Not implemented
1	0	0	0	BAD	Adder (unshifted)
1	0	0	1		Not implemented
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

$\boxed{42}$ 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

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

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

$\boxed{46}$ (MAR & COUNTER INPUT SELECT)

0	LIT
1	BSW least byte

46	47	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.

49	50		SAR INPUT
0	0		No change
0	1	CSAR	Complement SAR (See table in syntax for complements)
1	0	SAR	BSW least bits*

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

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		Not implemented
1	0	0	1		Not implemented
1	0	1	0		Not implemented
1	0	1	1		Not implemented
1	1	0	0		Not implemented
1	1	0	1		Not implemented
1	1	1	0		Not implemented
1	1	1	1		Not implemented

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):

RMI	Memory accepts MIR 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.

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 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 in 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.

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 RMI should be included if address register changes are required before the RDC is returned.

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

   2.3 A RMI is returned when the memory has accepted the address and the memory is connected to the requesting Interpreter.

2.4 A group of intervening instructions can be issued. Once RMI 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 following clock.

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 write can occur in the instruction after both the address and data have the desired values.

3.4 Return of RMI indicates that the memory is connected and therefore the address and data have been accepted and thus the address registers and MIR may be subsequently changed.

Bibliography

1. Bingham, Davis, Faber, Fisher, McGonagle, Reigel, Zucker, "Microprogramming Manual for Interpreter Based System," Burroughs Corporation Technical Report TR 70-8, November 1970.

2. "D Machine Users Manual," Burroughs Corporation Technical Report, April 1971.

3. Notes from a short course in "Microprogramming", Continuing Engineering Studies Course 7107, University of Pennsylvania, June 1971.