MUNTZ

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

APOLLO

GUIDANCE AND NAVIGATION

Approved _____ Date 4/6/65
MILTON B. TRAGESER DIRECTOR
APOLLO GUIDANCE AND NAVIGATION PROGRAM

Approved _____ Date 4/9/65
ROGER B WOODBURY DEPUTY DIRECTOR
INSTRUMENTATION LABORATORY

R-489

USERS GUIDE TO THE BLOCK II AGC/LGC INTERPRETER

by

Charles A Muntz

April 1965

MIT INSTRUMENTATION LABORATORY
CAMBRIDGE 39, MASSACHUSETTS

COPY # 0

ACKNOWLEDGEMENT

This report was prepared under DSR Project 55-238, sponsored by the Manned Spacecraft Center of the National Aeronautics and Space Administration through Contract NAS 9-153..

R-489

USERS GUIDE TO THE BLOCK II A GC/LGC INTERPRETER

ABSTRACT

A description is given of the AGC/LGC (Apollo and LEM Guidance Computers) algebraic interpreter, a language in which Apollo Mission computer programs may be conveniently prepared.

by Charles A. Muntz
April 1965

Table of Contents

I. Introduction

The interpreter is a collection of programs in the Apollo and LEM Guidance Computers which permits the preparation of programs in a convenient, problem-oriented language. The advantages of such a pseudo-language over basic machine language for solving most computational problems are well known. Conventionally, compiler techniques are used to translate equations into basic machine language and the basic language program executed to solve the required problem. One disadvantage of this translation process is its uneconomic use of program storage. The limitations of a space-borne guidance computer indicate the desirability of storage savings at the expense of some execution time; i.e., the computer is necessarily small but considerably faster than the devices it controls. Such a trade-off may be achieved with an interpreter.

In an interpretive system, the translation from problem-oriented language to basic language is carried out in two stages. The first stage, performed by the YUL Assembler for AGC/LGC operation, consists of reducing interpretive language to a compact, encoded form. This intermediate form of the program is stored in a relatively small number of registers in the guidance computer. Final translation is done by the guidance computer at execution time, at the expense of some increase in running time.

II. Multi-Precision Numbers

Most of the variables involved in computations during an Apollo mission require accuracy beyond the 4 digits specified by a single precision number, indicating that the problem-oriented pseudo-language should be centered about multiple-precision computation. As many of the required equations are most economically represented in vector and matrix form, it is highly desirable to include vector-matrix operations in such a pseudo-language. To accomplish these ends and to facilitate operational descriptions in the remaining sections of this report, the following symbols and nomenclature are introduced:

S(X) The single register X is understood to be a fraction, $-1 < S(X) < 1$.

D(X) The pair of registers (X, X+1) has the value $S(X) + 2^{-14} S(X+1)$ where S(X) and S(X+1) may have different signs.

T(X) The trio of registers (X, X+1, X+2) has the value $S(X) + 2^{-14} S(X+1) + 2^{-28} S(X+2)$ with possible sign disagreement as in D(X).

V(X) The six registers (X, X+1, . . . , X+5) form a column vector.

$$\begin{bmatrix} D(X) \\ D(X+2) \\ D(X+4) \end{bmatrix}$$

M(X) The eighteen registers (X, . . . , X+17) are understood to be the following matrix:

$$\begin{pmatrix} D(X) & D(X+2) & D(X+4) \\ D(X+6) & D(X+8) & D(X+10) \\ D(X+12) & D(X+14) & D(X+16) \end{pmatrix}$$

Note that all numbers in the above definitions are understood to be fractions. This convention is preferable for fixed-point arithmetic since the binary point is unaffected by addition, subtraction, multiplication, and division.

III. Interpretive Accumulator, MPAC

In executing interpretive programs, a set of erasable registers is set aside as a pseudo-accumulator. This area is designated MPAC for Multi-Purpose Accumulator. MPAC consists of seven registers with slightly modified array definitions:

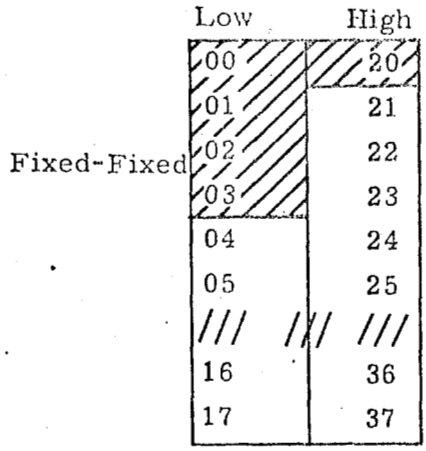
D(MPAC)	Same as D(X) with X = MPAC.
T(MPAC)	Same as T(X) with X = MPAC.
V(MPAC)	Six of the seven registers are understood to be the column vector. $\begin{pmatrix} \text{D(MPAC)} \\ \text{D(MPAC + 3)} \\ \text{D(MPAC + 5)} \end{pmatrix}$ with S(MPAC + 2) irrelevant.
M(MPAC)	Not used. Matrix operations are vector-matrix multiplication and yield vector results.
S(MPAC)	Not used since single precision numbers are not manipulated in MPAC.

An interpretive overflow indicator, OVFIND, is provided for recording overflows in arithmetic operations. This register is zero initially and set to ± 1 if an overflow occurs. Instructions are provided for testing and resetting OVFIND.

IV. Memory Organization

Almost all memory may be used for interpretive variables, constants, and programs. Variables may be stored anywhere in erasable memory except registers $0\text{-}77_8$. An interpretive program will not change E banks; it is assumed that any such computation may be confined to un-switched erasable and one E bank. General erasable from 61_8 to 1377_8 plus the current E bank are together referred to as "local erasable".

Fixed memory is divided into two parts for interpretive considerations.



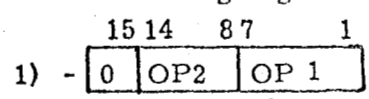
(Cross-hatched area is unavailable to interpretive programs.)

The low memory consists of banks 04-17 and high memory of 21-37, these two portions referred to as half-memories. Programs may be stored anywhere in high or low memory. Any program may branch to any other program; however, programs stored in low memory may only refer to constants stored in low memory and correspondingly, programs in high memory are limited to constants in high memory.

V. Organization of Interpretive Language

The interpretive language itself consists of a number of pseudo operation codes. These operation codes are mostly single address, with no-address codes included where desirable. A more conventional organization would employ pseudo-codes and address constants packed into one word of computer memory, similar to the basic instruction format. However, since the AGC/LGC word length is short, this organization is not practical for this application.

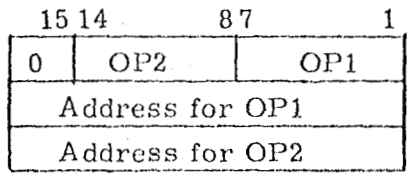
Returning to memory considerations, it is desirable to allow programs and constants anywhere in fixed memory, while variables are of course confined to erasable memory. A full word is required to specify a fixed memory address while only 10 bits are required to locate a local erasable memory location. This constraint on locations of variables leads to the following organization of coded pseudo instructions:



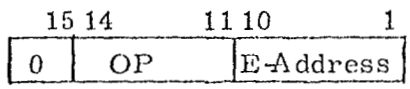
General Instructions

For instructions which address variables, programs, and constants

the above format is used to store two seven-bit operation codes in one word of memory. Address constants are given a full word of memory:



2)



Store Instructions

Instructions which store an accumulator are generally the only ones whose addressing capability may be restricted to erasable memory.

Several methods of address modifications are provided, as will be explained later. The most important of these is furnished by two index registers, X1 and X2. If an address is indexed, the contents of the specified index register are subtracted from the unmodified address and the result used as a net operand address.

The seven-bit operation codes comprising the general op code set are divided into 4 addressing classes according to the last two bits of the operation code:

Suffix	Address
00	OP code requiring no address - i.e., a set of unary instructions. Included here are function codes (for sine, cosine, square root, etc.) and some shift instructions.
01	Arithmetic instructions with non-indexed address. Address may be any location in local erasable, or the half-memory from which the instruction was taken.
10	Branch instructions and index instructions. This set of instructions performs sequence changes and modifies contents of index registers. Addresses may be the same as class 01 except that either half-memory may be referenced.
11	Same as 01 except that an index register is subtracted from the given address to form a net operand address.

VI. Description of Interpretive Operation Codes

The following operations may be used to load the multi-purpose accumulator MPAC.

DLOAD	X	Load MPAC in double precision
TLOAD	X	Load MPAC in triple precision
VLOAD	X	Load MPAC with a vector
SLOAD	X	Load MPAC with a single precision

The address X may be direct or indexed and refer to any local erasable memory register or any fixed memory location in the half-memory from which the instruction was selected. DLOAD, TLOAD and VLOAD load double precision (DP), triple precision (TP), and vector quantities into MPAC, while SLOAD prepares a single precision quantity (counter, etc.) for double precision computation. To assist in subsequent operations which store MPAC, an indication of the type of quantity currently in MPAC is maintained in a register known as MODE, this information being referred to as the store mode of MPAC. The store mode has three states: +0 for double precision, +1 for triple precision, and -1 for vector. Accordingly, DLOAD and SLOAD set the store mode to double precision, TLOAD to triple precision, and VLOAD to vector.

The following store code stores MPAC, leaving the store mode unchanged:

STORE	X	Store MPAC in X

The address X may be anywhere in local erasable and may be direct or indexed. The type of quantity to be stored is specified by the store mode. Similarly, the following store codes combine storing with other useful operations:

STODL	X	Store MPAC in X
	Y	and reload in DP from Y
STOVL	X	Store MPAC in X
	Y	and reload as vector from Y
STCALL	X	Store MPAC in X
	Y	and call the routine at Y

STODL is entirely equivalent to STORE X immediately followed by DLOAD Y, except that it is more compact and faster. Similarly, STOVL is STORE X followed by VLOAD Y. STCALL is not quite as general, accepting only direct addresses, but is otherwise equivalent to STORE X followed by CALL Y (See discussion on sequence changing instructions).

Using combinations of the load and store codes, information transfers may be performed as follows:

$$\text{TOFF} = \text{T}$$
$$\overline{\text{R}}\text{RECT} = \overline{\text{R}}$$

is coded as

```
DLOAD
                T
STOVL           TOFF
                R
STORE           RRECT
```

Of course, the DLOAD could be the last half of a STODL at the end of a previous computation and the STORE could be a STODL or STOVL to begin the next.

The following instructions are available for performing scalar arithmetic:

DAD	X	DP add
DSU	X	DP subtract
BDSU	X	DP substract from
DMP	X	DP multiply
DMPR	X	DP multiply and round
DDV	X	DP divide by
BDDV	X	DP divide into
SIGN	X	DP sign test
TAD	X	TP add

All leave results in MPAC and may address local erasable or the current half of fixed memory with a direct or indexed address (SIGN is an exception and may only address erasable memory). As the arithmetic is fixed-point throughout, almost all of the above operations may result in

overflow; e. g. the sum of . 5 and . 75 is . 25 and overflow. An interpretive overflow indicator, OVFIND, is maintained to communicate this information to interpretive programs. Instructions are available for testing and resetting it (see BOV and BOVB). Usually, the overflow corrected result is left in MPAC (. 25 in the case above). Handling of overflow by each instruction is explained in the operation code summary.

Using the above instructions,

$$x = \frac{a}{b}(.25 - cd)$$

would be coded in the following way:

```
DLOAD       DMP
            C
            D
BDSU        DMP
            .25 DP
            A
DDV
            B
STORE       X
```

The following instructions are available for doing vector arithmetic with a vector in MPAC:

VAD	X	Vector add
VSU	X	Vector subtract
BVSU	X	Vector subtract from
DOT	X	Vector dot product
VXSC	X	Vector times scalar
V/SC	X	Vector divided by scalar
VXV	X	Vector cross product
VPROJ	X	Vector projection
VXM	X	Matrix pre-multiplied by vector
MXV	X	Matrix post-multiplied by vector

As before all may address local erasable and any location in the current half-memory with a direct or indexed address. VAD, VSU,

and BVSU are entirely analogous to their scalar equivalents DAD, DSU, and BDSU. DOT is one of the few non-loading instructions whose execution modifies the store mode: it is changed to double precision corresponding to the result of the scalar product. VXSC and V/SC are unusual in that the store mode determines the type of quantity to be found at X. If the current store mode is double or triple precision, these instructions reference a vector at X, i.e. V(X); if it is a vector they use D(X). This flexibility does not apply to matrix operations since MPAC would then be required to contain a matrix; the address of MXV or VXM always refers to a matrix.

Using the above instructions, the equations

$$a = \bar{y} \cdot \bar{z} + b$$

$$\bar{x} = \frac{1}{a} [\bar{y} + (\bar{z} \cdot \bar{w}) \bar{u}]$$

are coded as follows:

```
VLOAD     DOT
          Y
          Z
DAD
          B
STOVL     A
          Z
DOT       VXSC
          W
          U
VAD       V/SC
          Y
          A
STORE     X
```

The following scalar function codes are available:

SQRT	Square Root
SIN (SINE)	Sine
COS (COSINE)	Cosine
ARCSIN (ASIN)	Arc-sine
ARCCOS (ACOS)	Arc-cosine
DSQ	Square
ROUND	Round
DCOMP	Negate
ABS	Absolute Value

(Alternate spellings are given in parentheses). None of these codes require an address. All perform their function on the scalar in MPAC and leave their result in MPAC.

The functions SIN, COS, ARCSIN, and ARCCOS accept scaled inputs and leave scaled outputs so that the full range in question may be accommodated within the fixed-point fraction (see detailed descriptions). The following illustrates the use of the function codes:

$$a_1 = +\sqrt{b^2 + c}$$

$$a_2 = -\sqrt{b^2 + c}$$

$$x = \frac{\cos^{-1}[2(\bar{y} \circ \bar{z})]}{2\pi}$$

```
DLOAD     DSQ
          B
DAD       SQRT
          C
STORE     A1
DCOMP
STOVL     A2
          Y
DOT       ARCCOS
          Z
STORE     X
```

In addition, the following vector functions are available:

UNIT	Unit vector operation
ABVAL	Vector length
VSQ	Square of vector length
VCOMP	Vector negation

Similar to the scalar functions, these instructions require no address but use the vector in MPAC. ABVAL and VSQ have scalar results and change the store mode to double precision accordingly. UNIT and ABVAL produce additional results as by-products, namely the square of the vector length in one standard location and, in the case of UNIT, the length in another (see detailed operation code description). To accommodate a vector of the form (x, 0, 0) a half-unit vector is left in MPAC by UNIT.

Scaling adjustments are frequently required in fixed-point computation. Operation codes requiring no address are provided for fixed, short length shifts, while another operation code uses a specially decoded address to specify a completely general shift. The short shift codes are as follows:

SR1	Scalar shift right one
SR2	Scalar shift right two
SR3	Scalar shift right three
SR4	Scalar shift right four
SR1R	Scalar shift right one and round
SR2R	Scalar shift right two and round
SR3R	Scalar shift right three and round
SR4R	Scalar shift right four and round
SL1	Scalar shift left one
SL2	Scalar shift left two
SL3	Scalar shift left three
SL4	Scalar shift left four
SL1R	Scalar shift left one and round
SL2R	Scalar shift left two and round
SL3R	Scalar shift left three and round
SL4R	Scalar shift left four and round

VSR1	Vector shift right one
...	...
VSR8	Vector shift right eight
VSL1	Vector shift left one
...	...
VSL8	Vector shift left eight

Those codes provide scalar shift left or right, one to four places, with or without terminal round, or vector shift left or right, one to eight places. Rounding is optional in scalar shifts: round after right shift is desirable unless the quantity is to be retained in triple precision; and round after left shift is only meanful if a triple precision number is in MPAC; e.g. after DOT. Rounding is not optional in vector operations since components are always double precision: vector shift right always rounds components and vector shift left never rounds.

Variables are usually scaled to give maximum precision and still accommodate the required range of values. Suppose position information is required which can be as large as 3×10^8 meters. To prepare this quantity for fixed-point arithmetic we will divide it by an integral power of two so that the maximum value lies between .5 and 1. If $\underline{\tilde{R}}$ is the vector to be stored in the guidance computer, then

$$\underline{R} = \underline{\tilde{R}} \times 2^{29}$$

Suppose that a calculated position increment $\delta\underline{R}$ will never exceed 1.2×10^6 meters. Then, to achieve maximum precision $\delta\underline{R}$ should be stored as

$$\delta\underline{R} = \delta\underline{\tilde{R}} \times 2^{21}$$

To add $\delta\underline{R}$ to $\underline{R}$ as stored in fixed-point we have

$$\underline{R} = \underline{R} + \delta\underline{R}$$

$$\underline{\tilde{R}} \times 2^{29} = \underline{\tilde{R}} \times 2^{29} + \delta\underline{\tilde{R}} \times 2^{21}$$

or

$$\underline{\tilde{R}} = \underline{\tilde{R}} + \delta\underline{\tilde{R}} \times 2^{-8}$$

The interpretive program for performing this operation is:

```
VLOAD       VSR8
            DELR
VAD
            R
STORE       R'
```

While the above codes satisfy most needs, the following general requirements exist and are met by the corresponding operation code:

SR	Scalar shift right	1-41 places
SL	Scalar shift left	1-41 places
SRR	Scalar shift right and round	1-28 places
SLR	Scalar shift left and round	1-13 places
VSR	Vector shift right	1-28 places
VSL	Vector shift left	1-27 places

The above shift capabilities form limits for corresponding direct addresses. If the address (shift count) is indexed, the stored address may lie between $\pm$ 128, but the resultant address should lie within the above limits. If the result of the indexing operation produces a negative result, the specified shift takes place but in the opposite direction.

Often, fixed point division is complicated by wide ranges of variation. Such difficulties may be overcome by adjusting the scaling of the denominator. The instruction

NORM (SLC) X Scalar normalization

shifts the scalar in MPAC left until its magnitude is at least .5, storing the negative of the required number of shifts in the erasable location X. Using this instruction 1/4z may be found with maximum precision as follows:

```
DLOAD     NORM
          Z
          X1
BDDV      SRR*
          .25
          0,1
```

A number of instructions are provided for changing sequence in an interpretive program. (Note: programs may not implicitly cross bank boundaries; an explicit sequence-change instruction must be used for this purpose.) As these instructions are the principal referencers of fixed memory, their addressing range has been extended to cover both half-memories. This prohibits indexed addressing in these instructions, but in this case a more desirable form of address modification is provided: indirect addressing at an arbitrary level. If the address of a sequence-changing instruction refers to erasable, the contents of this location are taken to be the address of the next interpretive instruction. If this address also references erasable, the process is continued until a fixed memory address is encountered. Usually, only one level of indirect address is used.

To execute a simple sequence change, the instruction

```
GOTO      X          Go to X
```

is provided. To call a subroutine which returns to caller

```
CALL      X          Call subroutine
```

may be used. Associated with each interpretive program is a return address register, QPRET. CALL leaves in QPRET the complete address of the next interpretive operation code (See also STCALL). CALL and GOTO are members of a class of operations codes known as "right hand operation codes". If they are in the left hand position of a pair, the right-hand code must be blank. Clearly, if GOTO was in the left-hand position, a right-hand op code would never be executed. In further discussions, no op code is a right-hand code unless specifically stated.

Two instructions are provided for returning from subroutine entered by CALL instructions. If return is desired before any other CALL instructions have been executed, the right-hand no-address instruction

```
RVQ (ITCQ)        Return Via QPRET
```

will execute a GOTO QPRET. If the original return address must be saved while another subroutine is called, the instruction

```
STQ (ITA)    X    Store QPRET
```

stores QPRET in the erasable location X (X must be a direct address). If QPRET was stored in QTEMP, subsequent return may be made with a GOTO QTEMP instruction.

To provide further addressing flexibility for CALL and GOTO the following instructions are provided:

```
CGOTO    X        Computed GOTO
         Y
CCALL    X        Computed CALL
         Y
```

In both instructions, X is the indexed or direct address of an erasable location and Y the address of any location in either fixed half-memory. To locate the operand address for CALL or GOTO, Y + S(X) is formed and the address taken from there. To call a different subroutine for each mode of a program, the following may be used:

```
          CCALL
                MODE
                SUBROADR

          ...

SUBROADR  CADR    MODEZERO
          CADR    MODEONE
          CADR    MODETWO
          ...
```

The address constants are stored out of sequence, of course. QPRET is set to three locations following the CCALL operation code.

The following instructions are provided for changing sequence dependent on the contents of MPAC:

BPL	X	Branch plus
BZE	X	Branch zero
BMN	X	Branch minus
BHIZ	X	Branch high-order zero

As with CALL and GOTO, X may refer to either half-memory, or an address in erasable. In particular, they may be used for early subroutine return via QPRET. Suppose SUBR01 should call SUBR02 if ab-c is positive or return directly if negative.

```
SUBR01      DLOAD       DMP
                        A
                        B
            DSU         BMN
                        C
                        QPRET
            GOTO
                        SUBR02
```

SUBR02 will return to SUBR01's caller via QPRET if ab-c is non-negative.

During the execution of an interpretive program it may be desirable to return to basic language. Two options are provided:

EXIT		Leave interpretive mode
RTB	X	Return to basic at X

EXIT begins executing basic instructions immediately following the last word processed by the interpreter, and is a "right-hand op code" like CALL and GOTO. Suppose a point is reached at which an output bit should be set, and then return to the interpretive mode:

```
            EXIT
            CAF         BITX
            EXTEND
            WOR         CHANNEL
            TC          INTPRET
            (Interpretive coding)
```

RTB may, in effect, be used to construct additional interpretive operation codes, particularly those which require no address. Suppose we desire a routine to load the time counters into MPAC. The following routine could be written:

```
LOADTIME    EXTEND
            DCA     TIME1
            DXCH    MPAC
            CA      ZERO
            TS      MPAC +2
            TS      MODE
            TCF     DANZIG
```

When such a routine is called with an RTB, the exit instruction TCF DANZIG begins the execution of the interpretive instruction following the RTB. Suppose it is desired to go to ACTION if the present time is later than TCRIT:

```
            RTB     DSU
                    LOADTIME
                    TCRIT
            BMN
                    ACTION
```

No indirect addressing feature is provided with RTB: X must be the direct address of a location in either fixed half-memory.

Several instructions may result in overflow, setting OVFIND to indicate this fact to the interpretive program. These instructions include:

ABVAL	ROUND	SL4	VSL
BDDV	SL	SL4R	VSL1
BDSU	SLR	SR	...
BVSU	SL1	SRR	VSL8
DAD	SL1R	TAD	VSQ
DDV	SL2	UNIT	VXM
DOT	SL2R	VAD	VXV
DSU	SL3	VPROJ	V/SC
MXV	SL3R	VSR	VSU

Two instructions are provided for interrogating the interpretive overflow indicator, OVFIND.

BOV	X	Branch on overflow
BOVB	X	Branch on overflow to basic

If the overflow indicator is off (+0), no operation occurs. If it is on (±1), it is reset; then BOV does a GOTO X and BOVB, an RTB X. At the beginning of every interpretive job, OVFIND is set to zero by the EXECUTIVE.

The need for two-valued indicators arises frequently in Apollo mission programs. The interpreter provides compact storage for these switches and several instructions with which to manipulate and test them. Four erasable locations are reserved in the STATE area for use as sixty interpretive switches, numbered from 0 - 59D, (the set is almost arbitrarily expandable):

STATE	0	. . . .	14
+1	15	. . . .	29
+2	30	. . . .	44
+3	45	. . . .	59

Fourteen two-stage operations are provided. First stage options include:

1) Set switch to 1;
2) Clear switch to 0;
3) Invert switch (0 to 1 - 1 to 0);
4) No operation.

Second stage operations are:

1) Branch if switch initially set (on);
2) Branch if switch initially clear (off);
3) Go to unconditionally;
4) No operation

The fourteen useful combinations are

SET	X	Set switch X
CLEAR	X	Clear switch X
INVERT	X	Invert switch X
SETGO	X	Set switch X
	Y	and GOTO Y
CLRGO	X	Clear switch X
	Y	and GOTO Y
INVGO	X	Invert switch X
	Y	and GOTO Y
BON	X	If switch X is on,
	Y	GOTO Y
BOFF	X	If switch X is off,
	Y	GOTO Y
BONSET	X	Set switch X, and if on
	Y	initially, GOTO Y
BOFSET	X	Set switch X, and if off
	Y	initially, GOTO Y
BONCLR	X	Clear switch X, and if on
	Y	initially, GOTO Y
BOFCLR	X	Clear switch X, and if off
	Y	initially, GOTO Y
BONINV	X	Invert switch X, and if on
	Y	initially, GOTO Y
BOFINV	X	Invert switch X, and if off
	Y	initially, GOTO Y

In all cases but SET, CLEAR, and INVERT, two full words of address are required in addition to the half-word operation code. Note that the indirect addressing features of the sequence changing instruction apply here. Interrupt is inhibited during these instructions so that basic interrupt programs may manipulate switches with basic routines.

As explained earlier, two index registers are provided for address modification, X1 and X2. They also may be used for simple manipulation

of single precision numbers such as computed shift counts, etc. The following instructions are available for loading and storing these registers:

AXT, 1	X	Load address X
AXT, 2	X	directly into index
AXC, 1	X	Load complement of X
AXC, 2	X	directly into index
LXA, 1	X	Load index from erasable
LXA, 2	X	register X
LXC, 1	X	Load index with complement
LXC, 2	X	of erasable register X
SXA, 1	X	Store index in erasable
SXA, 2	X	register X
XCHX, 1	X	Exchange index with
XCHX, 2	X	erasable register X

Note that an index register may be complemented by LXC, 1 X1 or LXC, 2 X2. The "address" X of an AXT or AXC instruction may be any single precision constant: interpretive address, OCT or DEC constant, etc.

Single precision constants may be stored in erasable memory using the following instruction:

SSP	X	Set single precision constant
	Y	Y into erasable location X

Y may be any single precision constant.

The following instructions are provided to modify contents of index registers:

INCR, 1	X	Add X to index register
INCR, 2	X	
XAD, 1	X	Add the contents of erasable
XAD, 2	X	location X to index
XSU, 1	X	Subtract the contents of
XSU, 2	X	erasable location X from index

As with AXT and AXC, INCR may use any single precision constant as an "address".

Index registers may be used for counting with the following instruction:

TIX, 1	X	Count and branch
TIX, 2	X	on index

Associated with each interpretive job are two step registers S1 and S2. They may be used for single or double precision temporary storage but are implicitly used for counting the TIX instruction: if the contents of the specified index may be reduced by the contents of the corresponding step register without producing a result which is zero or less, the index is replaced by the reduced value and a GOTO X is executed (X is indirect if in erasable). Otherwise, no operation occurs. To illustrate the power of TIX, suppose 72 words at WI must be transferred to 72 locations at W. This operation might be coded as follows (X1 will be preserved):

```
        SXA, 1      AXT, 1
                    XTEMP
        DEC         72
        SSP
                    S1
        DEC         6
LOOP    VLOAD*
                    WI +72D, 1
        STORE       W + 72D, 1
        TIX, 1      LXA, 1
                    LOOP
                    XTEMP
```

VII. Computation of Generalized Parenthetical Expressions

Many of the simpler algebraic forms may be directly computed with single-address and no-address instructions. As example is

$$x = \sqrt{\frac{ab - c}{d}}$$

which is coded as

```
DLOAD     DMP
          A
          B
DSU       DDV
          C
          D
SQRT
STORE     X
```

More complicated forms often must be computed in steps, saving and combining partial results in the process. An example of such an expression is

$$x = ab + cd - ef$$

which might be coded as follows:

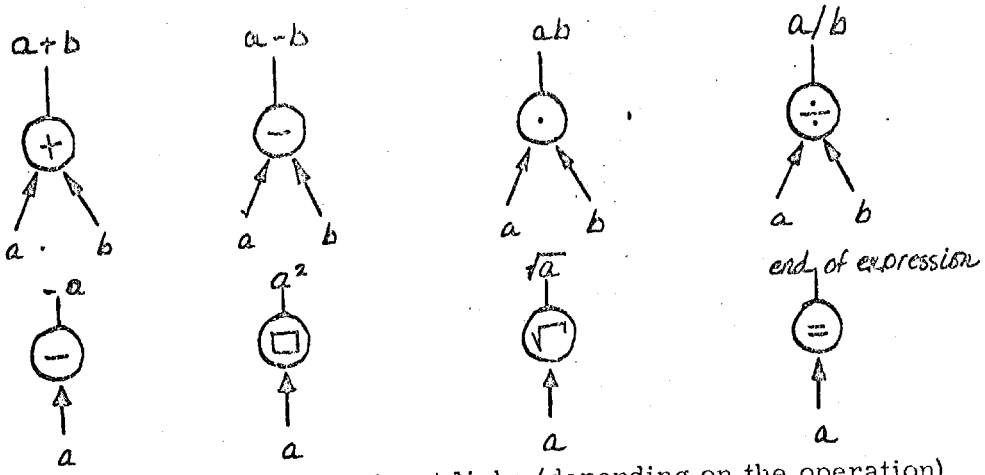
```
DLOAD     DMP
          A
          B
STODL     TEMP
          C
DMP       DAD
          D
          TEMP
STODL     TEMP
          E
DMP       BDSU
          F
          TEMP
STORE     X
```

The contents of register TEMP are only useful in the computation of X and may be discarded afterwards. While the above expression requires the use of only one temporary variable, expressions can be constructed which require the simultaneous use of any number of temporary variables. In particular, two are required for the following:

$$x = \frac{a^2 + b^2}{c^2 + d^2}$$

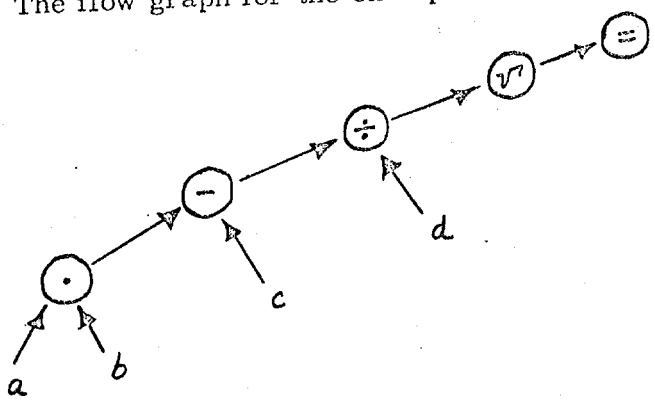
```
DLOAD     DSQ
          A
STODL     TEMP1
          B
DSQ       DAD
          TEMP1
STODL     TEMP1
          C
DSQ
STODL     TEMP2
          D
DSQ       DAD
          TEMP2
BDDV
          TEMP1
STORE     X
```

To give a graphical interpretation of these phenomena, networks of computational flow may be constructed whose topology expresses the parenthetical nature inherent in an expression. Each operation code is mapped on to a node and each node linked to its input variables and to those nodes whose inputs are the result of the operation itself. The former are called input links and the latter, output links. The following symbols are used:



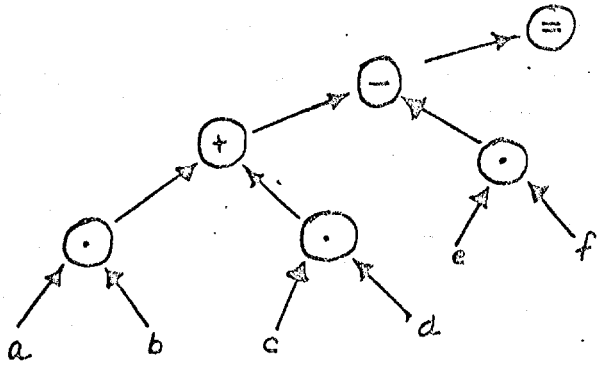
Each node is limited to one or two input links (depending on the operation) but may have as many output links as required.

The flow graph for the example of direct computation is as follows:



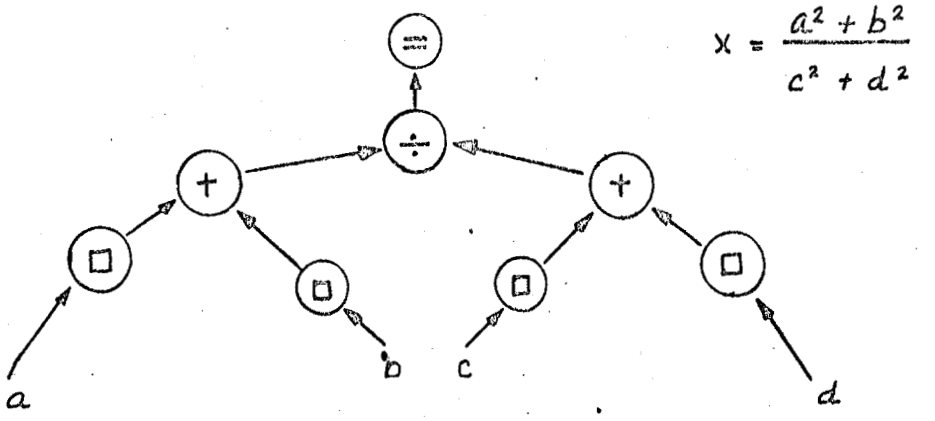
$$X = \sqrt{\frac{ab - c}{d}}$$

An expression is directly computable if and only if for every two input node, at least one of the input links comes from a variable instead of another node. The addition and subtraction nodes in the following graph fail to meet this requirement in the example of single temporary variable requirements:



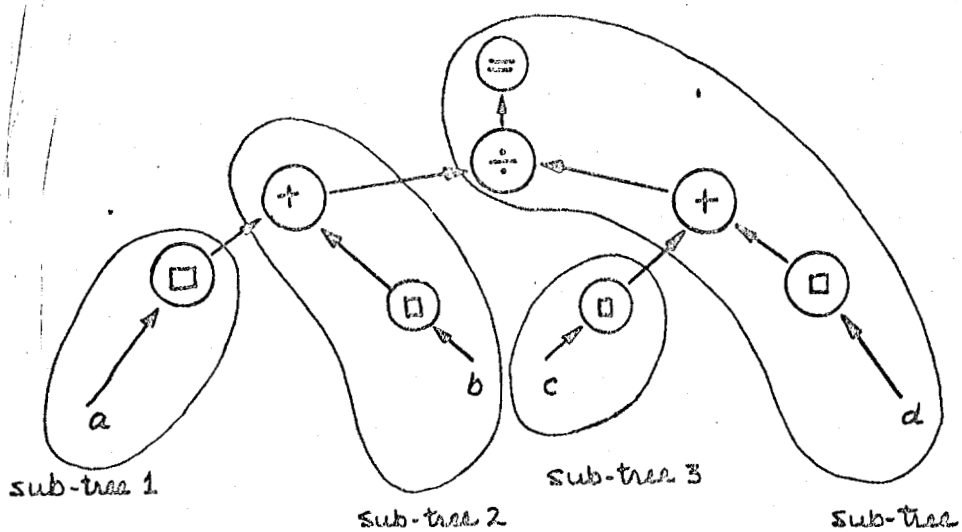
$$X = ab + cd - ef$$

The graph of the requirement for two simultaneous temporaries is as follows:



Two temporaries are required because both input links at the divide node come from nodes, both of whose inputs both come from other nodes. In this tree-like fashion, networks requiring an arbitrary number of simultaneous temporaries may be constructed.

The flow graph representation of such algebraic forms suggests a systematic method for their computation. Start at a variable at the bottom of the flow graph, say at the far, left-hand side. By our definition, any single input nodes are directly computable, as are two input nodes whose other input does not come from another node. If a node is encountered which is an exception to this case, store the present results in temporary storage and, considering the graph of operations leading to that node as a sub-tree, evaluate it in the same manner. When the original node is reached, combine the two inputs and procede up the tree until the entire expression has been evaluated. This division into sub-trees illustrates the process for the two-temporary example:



This systematic approach suggests that required temporary storage assignments might be handled by the interpreter itself, offering additional convenience to the user. In addition, memory savings may be realized by using implied address techniques to specify temporary storage. A pattern for meeting these temporary storage requirements is immediately provided by a push-down list structure. Such a list may contain an arbitrary number of items with the characteristic that the last quantity to be entered is the first to be withdrawn. Entering an item in the list is referred to as a "push down" and withdrawing the last item entered as a "push up". This process might be applied to preceding examples as follows:

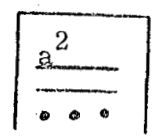
A. x = ab + cd - ed

Operation	Push down list after operation
1) Form ab and push down	
2) Form cd and add ab from the pushdown list	
3) Push down ab + cd and form ef	

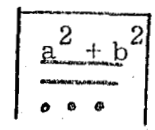
4) Subtract result from push-down list and store

B. $x = \dfrac{a^2 + b^2}{c^2 + d^2}$

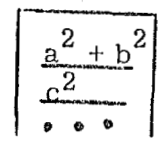
1) Form a^2 and push it down



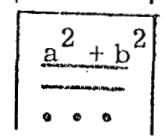
2) Form b^2, add from push-down list, and push that down



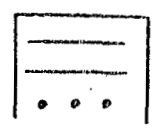
3) Form c^2 and push that down



4) Form d^2 and add from push-down list



5) Divide into push-down list and store result



Two basic types of operations are required:

1) A means for entering quantities into the push-down list. These may be special instructions which then require no address word to reference the push-down list. They must be capable of entering double or triple precision scalars or vectors.

2) Means by which instructions may reference the push-down list instead of an erasable or fixed memory location. This facility must be available to most of the single-address arithmetic instructions.

Push down capabilities are provided by the following instructions:

PUSH		Push down MPAC
PDDL	X	Push down MPAC and re-load with D(X)
PDVL	X	Push down MPAC and re-load with V(X).

To signal to an instruction the fact that it should push up for its operand, we will adopt the economical convention that if no address is supplied to an op code that requires an operand, the operand is taken from the push-down list. Thus,

$$x = a(bc + de)$$

would be coded as follows:

DLOAD	DMP
	B
	C
PDDL	DMP
	D
	E
DAD	
DMP	
	A
STORE	X

DAD pushes up for its operand since DMP is recognized as an op code instead of an address; it uses bc which was entered by PDDL.

This push up technique requires differentiation between op code words and address words. This requirement is fulfiled by pairs of general operation codes and addresses of arithmetic instructions; pairs of general op codes have bit 15 = 1 and these operand addresses have bit 15 = 0 since they are confined to one-half of fixed memory. Unfortunately, no distinction may be made between store codes and operand

addresses. Such conflicts (which do not arise too frequently in practice) are resolved by a special no-address right hand operation code:

STADR Recognize store code

This code may be used to compute

$$x = ab + cd$$

```
DLOAD   DMP
        A
        B
PDDL    DMP
        C
        D
DAD     STADR
STORE   X
```

STADR causes the YUL system to store the STORE instruction complemented. During execution, DAD pushes up and then STADR picks up the store code, complements it, and executes it.

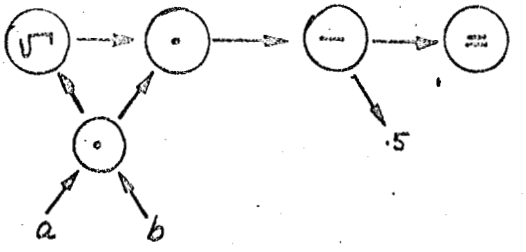
The following instructions will take their argument from the push-down list if no address (i.e., a vacuous address) is given:

BDDV	DOT	VLOAD
BDSU	DSU	VPROJ
BVSU	PDDL	VSU
DAD	PDVL	VXSC
DDV	SIGN	VXV
DLOAD	TAD	V/SC
DMP	TLOAD	
DMPR	VAD	

In addition, STODL and STOVL will push up if no load address is given.

In the previous examples, the flow diagrams were in what might be called normal form; i.e., such that no node had more than one output link. Such situations are entirely covered by PDDL and PDVL with the push up feature of arithmetic instructions. When a small expression appears several times in a larger expression, the desire to compute the smaller expression only once leads to non-normal flow graphs. Consider

$$x = (ab)^{3/2} - .5 = ab\sqrt{ab} - .5$$

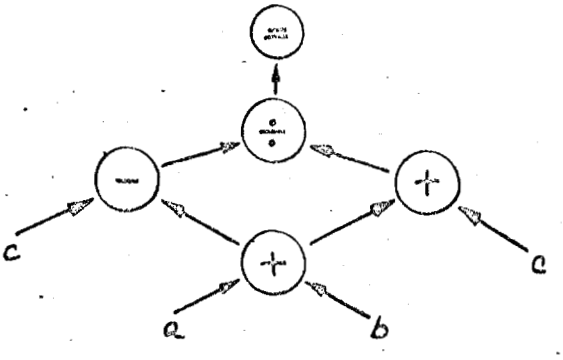


The PUSH instruction is designed to accommodate cases such as this: whenever the result of any node is needed by more than one other node, it may be retained in the push-down list for future use without disturbing the present computational flow. This example is coded as follows:

DLOAD	DMP
	A
	B
PUSH	SQRT
DMP	
DSU	
	.5DP
STORE	X

When two different computations must be formed with the same intermediate result before that result can be discarded, a different situation is encountered:

$$x = \frac{a + b - c}{a + b + c}$$



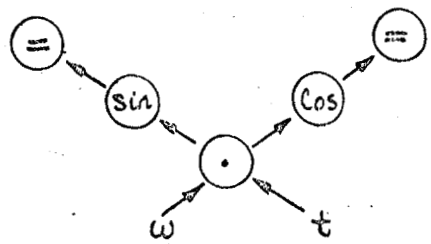
```
DLOAD       DAD
                A
                B
PUSH        DAD
                C
PDDL
DSU         DDV
                C
STADR
STORE       X
```

PDDL (and PDVL) with no address given may be thought of as an exchange with the last quantity to be pushed down. Any type of quantity (vector, etc) may be pushed down even though a scalar is pushed up in the case of PDDL, and similarly PDVL may exchange a scalar for a vector.

When one expression is common to several equations, STODL and STOVL may be used in a similar fashion:

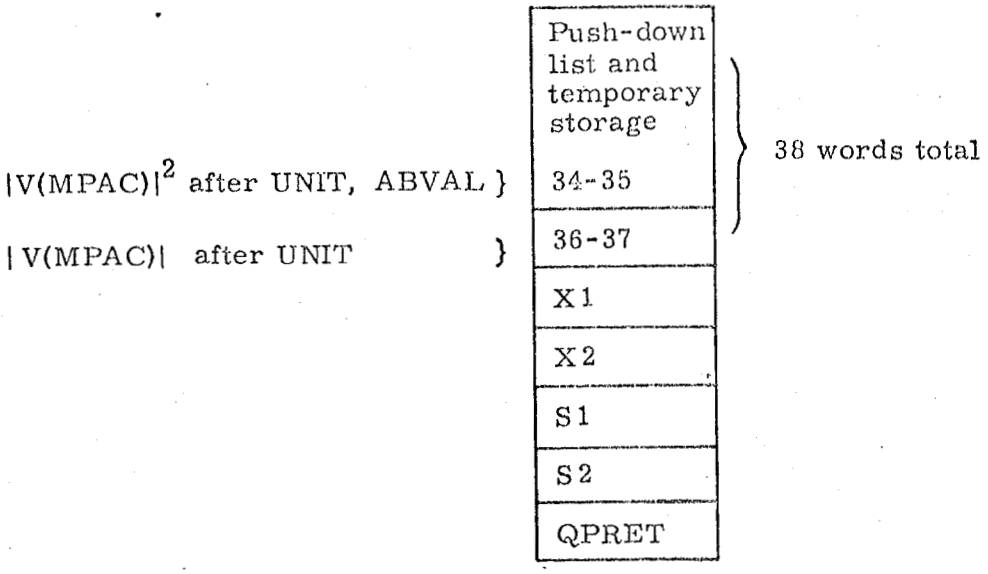
CWT = COS(WT)

SWT = SIN(WT)




```
DLOAD       DMP
                W
                T
PUSH        SIN
STODL       SWT
COS
STORE       CWT
```

Frequently, it is desirable to address a quantity saved in the above fashion more than once. This is facilitated by providing another method of addressing the area the interpreter uses for the push-down list. The push-down list, index registers, step registers, and return address are contained in a 43 word work area assigned to each interpretive job:



Five such areas are available to accommodate up to five interpretive jobs in partial stages of completion. Registers in this area are not directly addressable since any of the five may be in use. Addresses $0\text{-}43_{10}$ are reserved for directly addressing a job's work area and at execution time, are interpreted to be relative to the beginning of the work area currently in use. The push-down list occupies locations $0\text{-}37_{10}$; however, any such registers not used by push-down operations may be used for quasi-long-term storage by using direct addresses in that range. Push-down list manipulations use a movable pointer, PUSHLOC which is initialized at register 0 by the EXECUTIVE. A push-down operation stores MPAC at locations beginning at the address in PUSHLOC, advancing PUSHLOC to the first word after the stored array. Push-up operations regress the pointer until it points to the first word in the operand, and this value supplied as the operand address for instruction execution. The

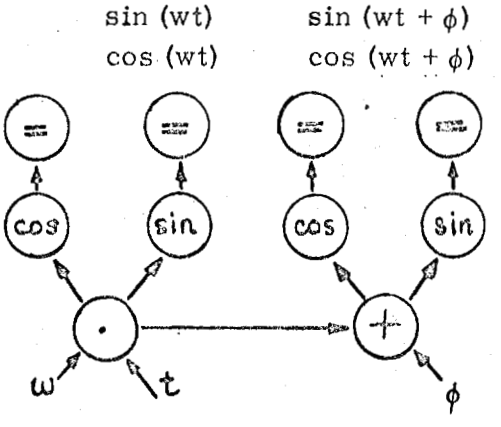
first quantity pushed down may be found in register 0, etc. Quantities written into the work area by push-down operations remain intact until they are written over.

If at any time it is desired explicitly to change PUSHLOC, the instruction

	SETPD	X	Set PUSHLOC

will set PUSHLOC to any local erasable address as specified by X. Of course to avoid confusion, the value of X should be restricted so that operations are confined always within the work area.

Returning to the original example of sines and cosines, suppose we desire:

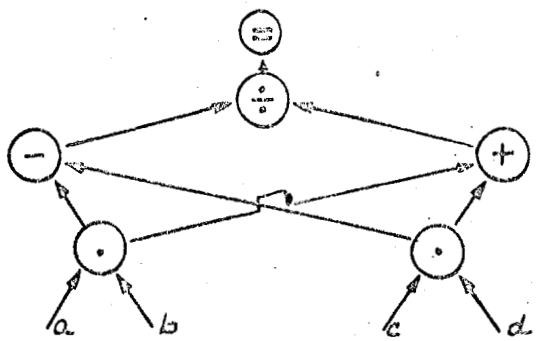


DLOAD	DMP
	W
	T
PUSH	SIN
STODL	SINWT
	0
COS	
STODL	COSWT
DAD	PUSH
	PHI
SIN	
STODL	SINWT + P
COS	
STORE	COSWT + P

It was assumed that the push-down pointer was initially at 0; it is left at 0 as well with wt + ϕ in 0.

This alternate addressing mode may be used to accommodate highly unusual algebraic forms whose economized non-normal graphs do not yield to push-down list strategies. An example is the following:

$$x = \frac{ab - cd}{ab + cd}$$



Assume the push-down pointer is at 0 initially. This computation may be coded as follows:

DLOAD	DMP
	A
	B
PDDL	DMP
	C
	D
PUSH	DAD
	0
PDDL	
BDSU	
	0
DDV	STADR
STORE	X

Note that there is a net change of 2 in the position of the push-down pointer.

The push-down list may also assist in vector definition operations. The no-address instruction

```
VDEF        Vector define
```

assumes the X component of the desired vector is D(MPAC), pushes up for the Y component and pushes up again for Z, changing the store mode to vector. If

$$\overline{V} = (-a,\ 0,\ b)$$

then $\overline{V}$ can be defined as follows:

```
DLOAD       PDDL
            B
            DPZERO
PDDL        DCOMP
            A
VDEF
```

Scalar equations have been used in the previous discussion for simplicity. The push-down list structure may also be used for vector computaions, as the following example illustrates:

$$\overline{u}_\theta = \overline{u}_x \cos(\theta) + \overline{u}_y \sin(\theta)$$

```
DLOAD       SIN
            THETA
VXSC        PDDL
            UY
            THETA
COS         VXSC
            UX
VAD         STADR
STORE       UTHETA
```

VIII. Sample Interpretive Program

The following interpretive subroutine finds the roots of the quadratic $ax^2 + bx + c$. The quadratic formula has been put in the following form more appropriate to fixed point arithmetic where r_1 and r_2 are the two roots:

$$r_1 = \frac{\frac{-b}{2} + \sqrt{(\frac{b}{2})^2 - ac}}{a}$$

$$r_2 = \frac{\frac{-b}{2} - \sqrt{(\frac{b}{2})^2 - ac}}{a}$$

Assume initially that a, b, and c are in D(A), D(B), and D(C), and that the push-down pointer is at 00D. This subroutine is called as follows:

```
CALL
        QROOTS
   (returns here)

or

(OP)    CALL
(Addresses for left hand OP)
        QROOTS
   (returns here)
```

QROOTS	DLOAD	SR1	LOAD codes are usually only needed at the beginning of subroutines. More frequently STODL may be used.
		B	
	DCOMP	PUSH	-b/2 replaces D(00D)
	DSQ	PDDL	$b^2/4$ replaces D(02D)
		A	
	DMP	BDSU	BDSU pushes up to form
		C	$b^2/4 - ac$.
	SQRT	PUSH	$\sqrt{b^2/4 - ac}$ replaces D(02D).
	DAD	DDV	
		00D	-b/2
		A	
	STODL	ROOT1	Completes ROOT1 and pushes up for $\sqrt{b^2/4 - ac}$.
	BDSU		Pushes up -b/2
	DDV		
		A	
	STORE	ROOT2	
	RVQ		

This returns to caller with r_1 as D(ROOT1) and r_2 as D(ROOT2) and D(MPAC).

IX. Detailed Description of Operation Codes with Probable Average Execution Times.

A. Store, Load, and Push-Down Instructions.

STORE	X	Store MPAC	.62 m.s.

D(MPAC), T(MPAC) or V(MPAC) replace D(X), T(X) or V(X), respectively. X may be indexed or direct.

STODL	X	Store MPAC	
	Y	and re-load in DP	1.24 m.s.

D(MPAC), T(MPAC) or V(MPAC) replace D(X), T(X) or V(X). (D(Y), 0) become T(MPAC) setting the store mode to DP. X may be indexed, or direct and Y indexed, direct or vacuous (push-up).

STOVL	X	Store MPAC	
	Y	and re-load as Vector	1.43 m.s.

Same as STODL except V(X) become V(MPAC) and store mode is set to vector.

STCALL	X	Store MPAC	
	Y	and CALL a Routine	1.40 m.s.

D(MPAC), T(MPAC), or V(MPAC) replace D(X), T(X) or V(X), leaving the store mode unaltered. Call the routine at Y, leaving a return address (of the location after the second address) in QPRET. Both addresses must be direct.

DLOAD	X	Load MPAC in DP	.64 m.s.

(D(X), 0) become T(MPAC), setting the store mode to DP. Address may be direct, indexed or vacuous.

TLOAD	X	Load MPAC in TP	.77 m.s.

Same as DLOAD except T(X) become T(MPAC) and store mode is set to TP.

VLOAD	X	Load MPAC with a Vector	.91 m.s.

Same as DLOAD except V(X) become V(MPAC) and store mode is set to vector.

SLOAD	X	Load MPAC in Single Precision	.74 m.s.

Same as DLOAD except (S(X), 0,0) become T(MPAC). X may not be vacuous.

PDDL	X	Push Down and load MPAC in DP	.91 m.s.

D(MPAC), T(MPAC) or V(MPAC) are pushed down; (D(X),0) become T(MPAC) with the store mode set to DP. X may be direct, indexed, or vacuous.

PDVL	X	Push Down and load MPAC with a vector	1.14 m.s.

Same as PDDL except V(X) become V(MPAC) and the store mode is set to vector.

PUSH		Push Down	.55 m.s.

D(MPAC), T(MPAC) or V(MPAC) are pushed down.

SETPD	X	Set Push-down Pointer	.58 m.s.

Set the Push-down Pointer PUSHLOC to X, where X is in local erasable memory. X must be direct.

B. Scalar Arithmetic Operations - All addresses may be direct, indexed, or vacuous.

DAD X DP Add .66 m.s.

D(MPAC) + D(X) replace D(MPAC). Set OVFIND on overflow, and leave the over-flow corrected result in MPAC.

DSU X DP Subtract .66 m.s.

D(MPAC) - D(X) replace D(MPAC). Set OVFIND on overflow, and overflow-correct the result.

BDSU X DP Subtract From .74 m.s.

D(X) - D(MPAC) replace D(MPAC). Set OVFIND on overflow, and overflow-correct the result.

DMP X DP Multiply 1.13 m.s.

D(X) times D(MPAC) replace T(MPAC).

DMPR X DP Multiply and Round 1.29 m.s.

D(MPAC) D(X) = P is formed and rounded to DP so that (P,0) replace T(MPAC).

DDV X DP Divide By 2.48 m.s.

If $|D(MPAC)| < |D(X)|$, the DP quotient Q=D(MPAC) / D(X) is formed and (Q,0) replace T(MPAC). Overflow indication is set if required. $\pm .999999$ replace D(MPAC) in this case.

BDDV X DP Divide Into 2.50 m.s.

Same as DDV except Q = D(X) / D(MPAC) if $|D(X)| < |D(MPAC)|$.

SIGN X DP Sign Test .70 m.s.

X must be in erasable memory. If $D(X) \geq 0$, no operation occurs. Otherwise if store mode is DP or TP, - T(MPAC) replace T(MPAC); if store mode is vector, - V(MPAC) replace V(MPAC).

TAD	X	TP Add	.75 m.s.

T(MPAC) + T(X) replace T(MPAC). OVFIND is set on overflow, with the overflow corrected result left in MPAC.

C. Vector Arithmetic Operations.

All addresses may be direct, indexed, and any but MXV and VXM may have vacuous addresses.

VAD	X	Vector Add	.92 m.s.

V(MPAC) + V(X) replace V(MPAC). Set OVFIND on overflow in any component, leaving the overflow-corrected result.

VSU	X	Vector Subtract	.92 m.s.

V(MPAC) - V(X) replace V(MPAC). Set OVFIND on overflow in any component, leaving an overflow-corrected result.

BVSU	X	Vector Subtract From	1.17 m.s.

V(X) - V(MPAC) replace V(MPAC). Set OVFIND on overflow of any component, leaving an overflow-corrected result.

DOT	X	Vector Dot Product	3.08 m.s.

V(MPAC)· V(X) replace T(MPAC), setting the store mode to DP. Set OVFIND if overflow occurs, leaving an overflow-corrected result.

VXSC	X	Vector Times Scalar	3.27 m.s.

If the initial store mode is Vector, each component of V(MPAC) is multiplied by D(X), the rounded products replacing their respective X components of V(MPAC). If the initial store mode is DP or TP, change it to Vector, and each component of V(X) is multiplied by D(MPAC) to form V(MPAC) as above.

V/SC	X	Vector Divided by Scalar	5.39 m.s.

If the initial store mode is Vector, each component of V(MPAC) is divided by D(X), the DP quotients replacing their respective components of V(MPAC). If the initial store mode is DP or TP, it is changed to Vector, and each component of V(X) is divided by D(MPAC) to form V(MPAC). If overflow occurs in any component, the operation is terminated with OVFIND set and unspecified results in MPAC.

VXV	X	Vector Cross Product	4.98 m.s.

V(MPAC) * V(X) replace V(MPAC). Set OVFIND if overflow occurs, leaving an overflow-corrected result.

VPROJ	X	Vector Projection	5.75 m.s.

$\left[V(MPAC) \cdot V(X)\right] V(X)$ replace V(MPAC). Set OVFIND on overflow, and leave the result obtained with overflow-corrected $\left[V(MPAC) \cdot V(X)\right]$.

VXM	X	Matrix Pre-Multiplication by Vector	8.98 m.s.

$(V(MPAC)^T M(X))^T$ replace V(MPAC). Set OVFIND on overflow, leaving an overflow-corrected result.

MXV	X	Matrix Post-Multiplication by Vector	8.97 m.s.

M(X) V(MPAC) replace V(MPAC). Set OVFIND on overflow, leaving an overflow-corrected result.

D. Scalar Functions.

SQRT		DP Square Root	1.94 m.s.

SQRT (D(MPAC)) replace T(MPAC); i.e. the initial contents of MPAC are normalized, the DP square root of the normalized number computed, and that result unnormalized so that MPAC + 2 has marginal signifigance. Receipt of an argument less than -10^{-4} causes an abort.

SIN (SINE)	DP Sine	5.63 m.s.

.5 (Sin (2π D(MPAC)) replace T(MPAC).

COS(COSINE)	DP Cosine	5.80 m.s.

.5 (Cos (2π D(MPAC)) replace T(MPAC).

ARCSIN (ASIN)	DP Arc-sine	9.26 m.s.

(1(2π)Arc-sine (2D(MPAC)) replace T(MPAC). This is the inverse of the SIN function. Receipt of an argument greater than .5001 in magnitude causes an abort.

ARCCOS (ACOS)	DP Arc-Cosine	9.12 m.s.

(1/2π) Arc-Cosine (2D(MPAC) replace T(MPAC). This is the inverse of COS. Receipt of an argument whose magnitude is greater than .5001 causes an abort.

DSQ	DP Square	.76 m.s.

D(MPAC) times D(MPAC) replace T(MPAC).

ROUND	Round to DP	.56 m.s.

T(MPAC) are rounded to DP so that (ROUND (T(MPAC)), 0) replace T(MPAC). Set OVFIND if overflow occurs, leaving an overflow-corrected result, +0.

DCOMP	TP Complement	.52 m.s.

-T(MPAC) replace T(MPAC).

ABS	TP Absolute Value	.48 m.s.

$|T(MPAC)|$ replace T(MPAC).

E. Vector Functions.

UNIT	Unit Vector Function	6.46 m.s.

V(MPAC)/2 $|V(MPAC)|$ replace V(MPAC). $|V(MPAC)|^2$ replace D(34D) and $|V(MPAC)|$ replace D(36D). Set OVFIND if $|V(MPAC)| < 2^{-21}$ or $|V(MPAC)| \geq 1$ in which case the result is incorrect.

ABVAL	Vector Length	3.86 m.s.

$|V(MPAC)|$ become T(MPAC), changing the store mode to DP. In addition, $|V(MPAC)|^2$ replace D(34D). The result is zero if $|V(MPAC)| < 2^{-21}$. If $|V(MPAC)| \geq 1$ set OVFIND to indicate unspecified result.

VSQ	Square of Vector Length	2.21 m.s.

$|V(MPAC)|^2$ become T(MPAC), changing the store mode to DP. If $|V(MPAC)| \geq 1$, set OVFIND and leave an overflow-corrected result.

VCOMP	Vector Complement	.63 m.s.

-V(MPAC) replace V(MPAC).

VDEF	Vector Define	.67 m.s.

Push up for V_Y and again for V_Z so that (D(MPAC), V_Y, V_Z) becomes V(MPAC), setting the store mode to vector.

F. Shift Instructions.

1. Short Shifts

SR1	Scalar Shift Right	.85 m.s.
SR2		.85 m.s.
SR3		.85 m.s.
SR4		.85 m.s.

T(MPAC) 2^{-j} replace T(MPAC) (j= 1,2,3,4).

SL1	Scalar Shift Left	.72 m.s.
SL2		.95 m.s.
SL3		1.17 m.s.
SL4		1.39 m.s.

T(MPAC)$\times 2^{+j}$ replace T(MPAC) (j = 1,2,3,4). If significant bits are lost, set OVFIND but leave the overflow-corrected result as T(MPAC).

SR1R	Scalar Shift Right and Round	.99 m.s.
SR2R		.99 m.s.
SR3R		.99 m.s.
SR4R		.99 m.s.

T(MPAC)$\times 2^{-j}$ is rounded to a DP number R and (R,0) replace T(MPAC) (j = 1,2,3,4).

SL1R	Scalar Shift Left and Round	.88 m.s.
SL2R		1.10 m.s.
SL3R		1.32 m.s.
SL4R		1.54 m.s.

T(MPAC)$\times 2^{+j}$ is rounded to a DP number R and (R,0) replace T(MPAC) (j = 1,2,3,4). If overflow occurs, set OVFIND and leave the overflow-corrected result as T(MPAC).

VSR1	Vector Shift Right and Round	2.01 m.s.
VSR2		2.01 m.s.
VSR3		2.01 m.s.
VSR4		2.01 m.s.
VSR5		2.01 m.s.
VSR6		2.01 m.s.
VSR7		2.01 m.s.
VSR8		2.01 m.s.

Each component of V(MPAC) is replaced by the original value multiplied by 2^{-j} and rounded to DP. (j = 1(1)8).

VSL1	Vector Shift Left	.81 m.s.
VSL2		1.18 m.s.
VSL3		1.55 m.s.
VSL4		1.93 m.s.
VSL5		2.30 m.s.
VSL6		2.68 m.s.
VSL7		3.05 m.s.
VSL8		3.43 m.s.

Each component of V(MPAC) is replaced by the original multiplied by 2^{+j} (j = 1(1)8). If overflow occurs in any component, leave the overflow-corrected result and set OVFIND.

2. General Shifts. Addresses may be direct or indexed.

SR	X	General Scalar Shift Right	1.38 m.s. +.23 INTEGER $(\frac{X}{14})$ m.s.

T(MPAC)$\times 2^{-X}$ replace T(MPAC) where $-42 < X < 42$ (X can be negative only if the address was indexed. Address limits are $0 < X < 42$ if direct and $-128 < X_s < 128$ if indexed. X_s is the stored address before index modification; X is the net address in any case. On overflow leave the overflow-corrected result and set OVFIND.

SL	X	General Scalar Shift Left	1.03 m.s. +.22 X m.s.

Same as SR except that T(MPAC)2^X replace T(MPAC).

SRR	X	General Scalar Shift Right and Round	1.52 m.s. + .23 INTEGER (X/14) m.s.

Same as SR except that T(MPAC)$\times 2^{-X}$ is rounded to a DP number R and (R,0) replace T(MPAC). Address limits are $0 < X < 29$ if direct.

SLR	X	General Scalar Shift Left and Round	1.18 m.s. + .22 X m.s.

Same as SL except that T(MPAC)$\times 2^X$ is rounded to a DP number R and (R,0) replace T(MPAC). Direct address limits are $0 < X < 14$.

VSR	X	General Vector Shift Right	2.61 m.s. +.82 INTEGER (X/14) m.s.

Each component of V(MPAC) is replaced by the original value multiplied by 2^{-X} and rounded to DP. If X is an indexed address and the result address negative, do a VSL -X instead. Address limits are $0 < X < 29$ if direct and $-128 < X_s < 128$ if indexed.

VSL	X	General Vector Shift Left	.89 m.s. +.37 X m.s.

Each component of V(MPAC) is replaced by the original component multiplied by 2^{X}. On overflow of any component, leave the overflow-corrected result and set OVFIND. If the address was indexed and the resulting address negative, VSR(-X) instead. Address limits are $0 < X < 28$ if direct.

3. Normalization. Address may be direct or indexed.

NORM(SLC)	X	Scalar Normalize	.88 m.s. +.21 N m.s.

An N is found such that $|T(MPAC)|\, 2^{N} \geq .5$ provided $T(MPAC) \neq 0$. -N replaces S(X) and $T(MPAC) \times 2^{N}$ replace T(MPAC). If T(MPAC) = 0, -0 replaces S(X) and T(MPAC) are unchanged.

G. Branching, Sequence Changing, and Subroutine Linkage Instructions.

All have a direct address except EXIT and RVQ. Any such address except those associated with transition to basic language (RTB and BOVB) is interpreted as indirect if it refers to erasable memory. Any level of indirect addressing is allowed.

GOTO	X	Go To	.77 m.s.

Begin executing interpretive instructions at X. QPRET is undisturbed. GOTO is a right-hand operation code.

CALL	X	Call a Subroutine	.89 m.s.

Begin executing interpretive instructions at X. A return address is left in QPRET. CALL is a right-hand operation code.

CGOTO	X	Computed	.90 m.s.
	Y	GoTo	

The contents of X(X in erasable) are added to address Y (Y in fixed) and the address at Y + S(X) is selected. Begin executing interpretive instructions there unless the address is in erasable, in which case it is interpreted as indirect. CGOTO is a right-hand op code.

CCALL	X	Computed	1.07 m.s.
	Y	Call	

Same as CGOTO except that a return address is left in QPRET in addition. CCALL is a right-hand op code.

RVQ(ITCQ)		Return Via QPRET	.69 m.s.

Begin executing interpretive instructions at the location whose address is in QPRET. This may be used to return from a subroutine which contains no CALL or CCALL instructions. If QPRET contains the address of an erasable register, the address is interpreted as an indirect address. RVQ is a "right-hand op code".

STQ(ITA)	X	Store QPRET	.69 m.s.

S(QPRET) replace S(X) (X in erasable). This may be used to save the return address in subroutines which contain CALL and CCALL instructions. The STQ X in this case is eventually followed by GOTO X to return.

BPL	X	Branch Plus	.65 m.s.+
			.19 m.s. GO

If T(MPAC) $\geq$ 0, do a GOTO X. Otherwise, no operation occurs.

BZE	X	Branch Zero	.65 m.s. +.19 m.s. GO

If T(MPAC) = 0, do a GOTO X. Otherwise, no operation occurs.

BMN	X	Branch Minus	.67 m.s. +.19 m.s. GO

If T(MPAC) < 0, do a GOTO X. Otherwise, no operation occurs.

BHIZ	X	Branch High Order Zero	.6 m.s. +.19 m.s. GO

If S(MPAC) = 0, do a GOTO X. Otherwise, no operation occurs.

BOV	X	Branch On Overflow	.58 m.s. +.23 m.s. GO

If OVFIND is set, re-set it to zero and do a GOTO X. Otherwise, no operation occurs.

BOVB	X	Branch On Overflow to Basic	.58 m.s. .16 m.s. GO

If OVFIND is set, reset it to zero and begin executing basic instructions at X. Otherwise no operation occurs. X must be in fixed memory.

RTB	X	Return to Basic	.71 m.s.

Begin executing basic instructions at X. X must be in fixed memory.

EXIT		Exit from Interpreter	.26 m.s.

Begin executing basic instructions after the last op code or address word referenced by the interpreter as follows:

1) If EXIT is a left hand op code, go to the word after the EXIT instruction;

2) If EXIT is a right hand op code, go to the word following the last address used by the left hand op code.

EXIT is a right-hand op code.

H. Switch Instructions

SET	X	Set Switch	1.27 m.s.

Set switch X to 1.

CLEAR	X	Clear Switch	1.25 m.s.

Clear switch X to 0.

INVERT	X	Invert Switch	1.27 m.s.

Invert switch X; i.e., if 0, set to 1; if 1, clear to 0.

SETGO	X	Set Switch	1.54 m.s.
	Y	and Go To	

Set switch X to 1 and do a GOTO Y. SETGO is a right-hand op code.

CLRGO	X	Clear Switch	1.52 m.s.
	Y	and Go To	

Clear switch X to 0 and do a GOTO Y. CLRGO is a right-hand op code.

INVGO	X	Invert Switch	1.54 m.s.
	Y	and Go To	

Invert switch X and do a GOTO Y. INVGO is a right-hand op code.

I. Switch Test Instructions.

BON	X	Branch if	1.26 m.s.
	Y	Switch On	+.23 m.s.

If switch X is set to 1, do a GOTO Y. Otherwise, no operation occurs.

BOFF	X	Branch if	1.27 m.s.
	Y	Switch Off	+.23 m.s. GO

If switch X is cleared to 0, do a GOTO Y. Otherwise, no operation occurs.

BONSET	X	Branch if Switch	1.37 m.s.
	Y	On, Setting Switch	+.23 m.s. GO

Set switch X to 1. If initially set to 1, so a GOTO Y. Otherwise, no further operation occurs.

BOFSET	X	Branch if Switch	1.39 m.s.
	Y	Off, Setting Switch	+.23 m.s. GO

Set switch X to 1. If initially cleared to 0, do a GOTO Y. Otherwise, no further operation occurs.

BONCLR	X	Branch if Switch	1.35 m.s.
	Y	On, Clearing Switch	+.23 m.s. GO

Clear switch X to 0. If initially set to 1, do a GOTO Y. Otherwise, no further operation occurs.

BOFCLR	X	Branch if Switch	1.36 m.s.
	Y	Off, Clearing Switch	+.23 m.s. GO

Clear switch X to 0. If initially cleared to 0, do a GOTO Y. Otherwise, no further operation occurs.

BONINV	X	Branch if Switch	1.37 m.s.
	Y	On, Inverting Switch	+.23 m.s. GO

Invert switch X. If originally set to 1, do a GOTO Y. Otherwise, no further operation occurs.

BOFINV	X	Branch if Switch	1.39 m.s.
	Y	Off, Inverting Switch	+.23 m.s. GO

Invert switch X. If originally cleared to 0, do a GOTO Y. Otherwise, no operation occurs.

J. Index Register Instructions

AXT,1	X	Address to	.75 m.s.
AXT,2	X	Index True	

X replaces S(XT) (T = 1,2)

AXC,1	X	Address to	.76 m.s.
AXC,2	X	Index Complemented	

-X replaces S(XT).

LXA,1	X	Load Index	.78 m.s.
LXA,2	X	from Erasable	

S(X) replaces S(XT).

LXC,1	X	Load Index	.78 m.s.
LXC,2	X	from Erasable Complemented	

-S(X) replaces S(XT).

SXA,1	X	Store Index	.78 m.s.
SXA,2	X	in Erasable	

S(XT) replaces S(X).

XCHX,1	X	Exchange Index	.83 m.s.
XCHX,2	X	with Erasable	

S(XT) replaces S(X) which then replaces S(XT).

INCR,1	X	Increment Index	.76 m.s.
INCR,2	X		

The overflow-corrected sum of S(XT) and X replaces S(XT).

XAD,1	X	Index Register	.77 m.s.
XAD,2	X	Add	

The overflow-corrected sum of S(XT) and S(X) replace S(XT).

XSU,1	X	Index Register Subtract	.78 m.s.
XSU,2	X		

The overflow-corrected difference S(XT) - S(X) replaces S(XT).

TIX,1	X	Transfer on Index	.78 m.s.
TIX,2	X		+.26 m.s. GO

If S(XT) $\leq$ S(ST) (T= 1,2), no operation occurs. Otherwise, S(XT) - S(ST) replaces S(XT) and a GOTO X is executed.

K. Miscellaneous Instructions

SSP	X	Set Single	.67 m.s.
	Y	Precision	

Y replaces S(X). Y may be any constant: arithmetic, logical, address, etc.

STADR		Push Up On	.26 m.s.
		Store Code	

During assembly, the appearance of STADR causes the next store code to be stored complemented. During execution, STADR complements the the next word to be referenced by the interpreter and enters the store code processor. STADR is a right-hand op code.

X. Detailed Timing Summary

The following table gives execution times for all modes of operation of all interpretive instructions. Figures quoted are in milli-seconds, and, unless otherwise stated, exact. For all except the store codes STORE, STODL, STOVL, and STCALL, add .13 m.s. if the operation code is in the left-hand position.

A. Store, Load, and Push-down Instructions

STORE	.55	(1) Add .08 if MPAC TP (2) Add .21 if MPAC vector (3) Add .08 if indexed address
STODL	1.17	(1) Add .08 if MPAC TP (2) Add .21 if MPAC vector (3) Add .08 if store address indexed (4) Add .15 if load address indexed (5) Add .05 if load pushes up
STOVL	1.36	(1) - (5) Same as STODL
STCALL	1.33	(1) - (2) Same as STORE
DLOAD (a, b)	.64	
TLOAD (a, b)	.69	
VLOAD (a, b)	.83	
SLOAD (a)	.66	
PDDL (a,.b)	.76	(1) Add .07 if MPAC TP (2) Add .20 if MPAC V
PDVL (a, b)	.99	(1) Add .08 if MPAC TP (2) Add .20 if MPAC V
PUSH	.48	(1) Add .08 if MPAC TP (2) Add .21 if MPAC V
SETPD	.51	

B. Scalar Arithmetic Instructions (a, b)

DAD	.59	
DSU	.59	
BDSU	.67	
DMP	1.05	
DMPR	1.21	
DDV	2.40	
BDDV	2.42	
SIGN	.61	(1) Add .02 if MPAC V
TAD	.67	

C. Vector Arithmetic Instructions (a)

VAD (b)	.84	
VSU (b)	.84	
BVSU (b)	1.09	
VXSC (b)	2.98	(1) Add .21 if initial MPAC scalar
V/SC (b)	5.15	(1) Add .32 if initial MPAC scalar
DOT (b)	3.00	
VXV (b)	4.90	
VPROJ (b)	5.67	
VXM	8.90	
MXV	8.89	

D. Scalar Functions

SQRT (c)	1.86
SIN	5.55
COS	5.72
ARCSIN (c)	9.18
ARCCOS (c)	9.04
DSQ	.68
ROUND (c)	.48
DCOMP	.44
ABS	.40

E. Vector Functions

UNIT (c)	6.38
ABVAL (c)	3.79
VSQ	2.13
VCOMP	.55
VDEF	.59

F. Shift Instructions

1) Short Shifts

a) Scalars

SR1	.77
SR2	.77
SR3	.77
SR4	.77
SL1	.64
SL2	.87
SL3	1.09
SL4	1.31
SR1R	.91
SR2R	.91
SR3R	.91
SR4R	.91
SL1R	.80
SL2R	1.02
SL3R	1.24
SL4R	1.46

b) Vectors

VSR1	1.93
VSR2	1.93
VSR3	1.93
VSR4	1.93
VSR5	1.93

VSR6	1.93
VSR7	1.93
VSR8	1.93
VSL1	.73
VSL2	1.10
VSL3	1.47
VSL4	1.85
VSL5	2.22
VSL6	2.60
VSL7	2.97
VSL8	3.35

2) General Shifts (a)

a) Scalar

SR	1.30 + .23R
SL	.95 + .22N
SRR	1.44 + .23R
SLR	1.10 + .22N

b) Vector

VSR	2.53 + .82R
VSL	.81 + .37N

3) Normalization (a)

NORM (SLC)	.80 + .21N

R = Number of 14 place shifts required; e.g. SR 19 takes 1.53 m.s.

N = Number of places shifted.

G. Branching, Sequence Changing, and Subroutine Linkage Instructions - "GO" represents additional execution if sequence change performed.

GOTO (d)	.69
CALL (d)	.81
CGOTO (b, d)	.82
CCALL (b, d)	.99

RVQ	.61
STQ	.61
BPL (d)	.57 + .19GO
BZE (d)	.57 + .19GO
BMN (d)	.59 + .19GO
BHIZ (d)	.53 + .19GO
BOV (d)	.50 + .23GO
BOVB	.50 + .16GO
RTB	.63
EXIT	.18

H. Switch Instructions

SET	1.19
CLEAR	1.17
INVERT	1.19
SETGO (d)	1.46
CLRGO (d)	1.44
INVGO (d)	1.46

I. Switch Test Instructions (d)

BON	1.18 + .23GO
BOFF	1.19 + .23GO
BONSET	1.29 + .23GO
BOFSET	1.31 + .23GO
BONCLR	1.27 + .23GO
BOFCLR	1.28 + .23GO
BONINV	1.29 + .23GO
BOFINV	1.31 + .23GO

J. Index Instructions

AXT	.67
AXC	.68
LXA	.70
LXC	.70

SXA	.70
XCHX	.75
INCR	.68
XAD	.69
XSU	.70
TIX (d)	.70 + .26GO

K. Single Precision and Miscellaneous

SSP	.59
STADR	.18

NOTES:

(a) Add .18 m.s. if address is indexed.
(b) Add .04 m.s. if push up called for.
(c) Average time
(d) Add .26 m.s. for each level of indirect addressing.

XI. YUL Assembly Formats

The following is a discussion of the punched-card formats used in preparing interpretive programs, and of the corresponding encoding performed by the YUL assembler. The latter is intended primarily for those who wish to prepare octal corrections in the course of program check-out. The term "ADDRESS" is understood to be an octal or decimal integer, or symbol with or without an octal or decimal modifier. Numbers in parentheses denote card columns at which fields of information begin. If operation OP requires two addresses, the first is labelled OP (1st) and the second OP (2nd).

General operation codes are submitted in one of the following formats, with the encoded result on the right:

(9)	(18)	(25)		15 14	8 7	1
(SYMBOL)	OP1		-	0	0	CODE 1+1
(SYMBOL)	OP2	OP2	-	0	CODE 2+1	CODE 1+1

The seven bit operation code for any general operation may be found in the index. If OP1 or OP2 has an indexed address, the operation code must have an asterisk suffix. Thus DAD with an indexed address is written DAD*.

Operand addresses (not to be confused with the specially encoded shift and switch bit addresses) are first evaluated by memory type:

Location	Value for addresses limited to one half-memory and erasable	Value for addresses reaching both half-memories and erasable
$0 - 42_{10}$	$0 - 42_{10}$	$0 - 42_{10}$
$100_8 - 1377_8$	$100_8 - 1377_8$	$100_8 - 1377_8$
E3, 1400 - E7, 1777	$1400_8 - 1777_8$	$1400_8 - 1777_8$
04, 2000 - 17, 3777	04, 2000 - 17, 3777	04, 2000 - 17, 3777
21, 2000 - 37, 3777	01, 2000 - 17, 3777	21, 2000 - 37, 3777

Location	Value for addresses limited to erasable	Value for addresses limited to fixed
$0 - 42_{10}$	$0 - 42_{10}$	(Illegal)
$100_8 - 1377_8$	$100_8 - 1377_8$	(Illegal)
E3, 1400 - E7, 1777	1400 - 1777	(Illegal)
04, 2000 - 17, 3777	(Illegal)	04, 2000 - 17, 3777
21, 2000 - 37, 3777	(Illegal)	21, 2000 - 37, 3777

General operation codes whose address is restricted to one half-memory and erasable include:

BDDV	DDV	DOT	PDVL	VAD	VXM
BDSU	DLOAD	DSU	SLOAD	VLOAD	VXSC
BVSU	DMP	MXV	TAD	VPROJ	VXV
DAD	DMPR	PDDL	TLOAD	VSU	V/SC

Non-vacuous addresses for the above are submitted and processed as below:

(9) (18) (25)

	15	14 1
ADDRESS	0	Value + 1
ADDRESS, 1	0	Value + 1
ADDRESS, 2	0	Value + 1

The operation code corresponding to the indexed address must have a suffix asterisk.

General operations whose address may reference all of interpretive program memory (both half-memories and erasable memory) are as follows:

BHIZ	BOFSET (2nd)	BOV	GOTO
BMN	BON	BPL	INVGO(2nd)
BOFF(2nd)	BONCLR(2nd)	BZE	SETGO(2nd)
BOFCLR (2nd)	BONINV(2nd)	CALL	TIX, 1
BOFINV (2nd)	BONSET(2nd)	CLRGO(2nd)	TIX, 2

Address for these operation codes are submitted and processed as follows:

(9) (18) (25)

	15 1
ADDRESS	Value

General operations whose addresses are limited to erasable memory are divided into two classes: those whose address is indexable and those whose address must be direct. The former class is as follows:

CCALL(1st)	NORM	SIGN
CGOTO(1st)	SETPD	SSP(1st)

These are submitted and stored in the following fashion:

(9)	(18)	(25)	15 11 10 1
		ADDRESS	0 | Value + 1
		ADDRESS, 1	0 | Value + 1
		ADDRESS, 2	0 | Value + 1

Those codes which must have a direct address are as follows:

LXA, 1	STQ	XAD, 1	XCHX, 2
LXA, 2	SXA, 1	XAD, 2	XSU, 1
LXC, 1	SXA, 2	XCHX, 1	XSU, 2
LXC, 2			

Their address is submitted and encoded as follows:

(9)	(18)	(25)	15 11 10 1
		ADDRESS	0 | Value

Those general operation codes whose address is confined to fixed memory are the following:

BOVB	CCALL(2nd)	CGOTO(2nd)	RTB

This address is submitted and processed as follows:

(9)	(18)	(25)	15 1
		ADDRESS	Value

Some general operation codes may employ any single precision constant as their "operand address". The are the following:

AXC, 1	AXT, 1	INCR, 1	SSP(2nd)
AXC, 2	AXT, 2	INCR, 2	

They may employ an interpretive address constant referring to any interpretive program memory:

(9)	(18)	(25)	15 1
		ADDRESS	Value

In addition, the following constants may be used: DEC, OCT, CADR, ECADR, FCADR, ADRES and BBCON. These constants are described in YUL system documentation.

General shifts and switch-bit instructions employ specially encoded addresses. For general shifts, the shift count is submitted as an address in the usual manner, understanding that negative indexed addresses are to be put in the form 0 - N, 1 or 0 - N, 2. The following is a table of constants which are added to the shift count obtained by evaluating the given address field (all operations use op code 115 if direct and 117 if indexed):

Code	Value
SL	00200 + count
SR	00600 + count
SLR	01200 + count
SRR	01600 + count
VSL	00200 + count
VSR	00600 + count

Input and resulting address are as follows:

(9) (18) (25)

	15 11	10 1
ADDRESS	0	Value + 1
ADDRESS, 1	0	Value + 1
ADDRESS, 2	0	Value + 1

For example, SR 6 has op code 115 and address 00607; SRR* 0 - 3, 2 has op code 117 and address 76201.

All switch operations follow from op code 162. The first (or only) address contains three fields of information which specify the desired action and switch number: operation type in bits 5 - 8, switch bit position in bits 1 - 4, and switch word number in bits 9 - 10. Operation types set bits 5 - 8 as follows:

BOFF	00340	BONSET	00000
BOFCLR	00240	CLEAR	00260
BOFINV	00140	CLRGO	00220
BOFSET	00040	INVERT	00160
BON	00300	INVGO	00120
BONCLR	00200	SET	00060
BONINV	00100	SETGO	00020

The switch word and bit position information is obtained by dividing the switch number by 15: the quotient is the switch word and the remainder is the switch bit position.

The card on which the user specifies the first address of a switch instruction need only contain the switch number (0 - 59D):

(9)	(18)	(25)	15 11 10 1
		ADDRESS	0 | Value

As an example, CLEAR 46D is encoded as op code 162 and address 01661.

Operation code values for all store code configurations are tabulated below:

STORE	ADDRESS	00000
STORE	ADDRESS, 1	02000
STORE	ADDRESS, 2	04000
STODL	ADDRESS (ADDRESS)	06000
STODL	ADDRESS, 1 (ADDRESS)	10000
STODL	ADDRESS, 2 (ADDRESS)	12000
STODL*	ADDRESS ADDRESS, T	14000
STODL*	ADDRESS, 1 ADDRESS, T	16000
STODL*	ADDRESS, 2 ADDRESS, T	20000
STOVL	ADDRESS (ADDRESS)	22000
STOVL	ADDRESS, 1 (ADDRESS)	24000
STOVL	ADDRESS, 2 (ADDRESS)	26000

STOVL*	ADDRESS	30000
	ADDRESS, T	
STOVL*	ADDRESS, 1	32000
	ADDRESS, T	
STOVL*	ADDRESS, 2	34000
	ADDRESS, T	
STCALL	ADDRESS	36000
	ADDRESS	

Addresses used in the storing portion of the above operations are confined to erasable and have the same ten bit values as operand addresses in that range. The load addresses associated with STODL and STOVL may be vacuous if in parenthesis; otherwise, they are stored just as the address for DLOAD or VLOAD. The call portion of STCALL uses the same address format as CALL. Store codes are received and processed as follows:

(9)	(18)	(25)	15	14 11	10 1
(SYMBOL)	OP	ADDRESS	0	CODE	Value + 1
(SYMBOL)	OP	ADDRESS, 1	0	CODE	Value + 1
(SYMBOL)	OP	ADDRESS, 2	0	CODE	Value + 1

Any accompanying load or call address is processed as explained eariler.

INDEX

Suffix characters require reference to Section XI, YUL Assembly Format:

*	Code to be used with indexed address.
S	See note on switch operations (pp. 63-64).
G	See note on general shifts (p. 63).
SC	See note on store codes (pp. 64-65).

CALL	142	14, 48, 61
CCALL	065, 067*	15, 48, 61, 62
CGOTO	021, 023*	15, 48, 61, 62
CLEAR	162S	19, 50
CLRGO	162S	19, 50, 61
COS	030	10, 43
DAD	161, 163*	7, 17, 29, 40, 61
DCOMP	100	10, 43
DDV	105, 107*	7, 17, 20, 40, 61
DLOAD	031, 033*	6, 20, 38, 61
DMP	171, 173*	7, 29, 40, 61
DMPR	101, 103*	7, 29, 40, 61
DOT	135, 137*	8, 17, 29, 41, 61
DSQ	060	10, 43, 61
DSU	151, 153*	7, 17, 29, 40, 61
EXIT	000	16, 49
GOTO	126	14, 47, 61
INCR, 1	066	20, 52, 62
INCR, 2	062	20, 52, 62
INVERT	162S	19, 50
INVGO	162S	19, 50, 61
LXA, 1	026	20, 52, 62
LXA, 2	022	20, 52, 62
LXC, 1	036	20, 52, 62
LXC, 2	032	20, 52, 62
MXV	055, 057*	8, 17, 42, 61
NORM	075, 077*	13, 47, 61

PDDL	051, 053*	28, 29, 31, 39, 61
PDVL	061, 063*	28, 29, 31, 39, 61
PUSH	170	28, 30, 39
ROUND	070	10, 17, 43
RTB	152	16, 49, 62
RVQ	160	15, 48
SET	162S	19, 50
SETGO	162S	19, 50, 61
SETPD	175	33, 39, 61
SIGN	011, 013*	7, 29, 40, 61
SIN	020	10, 43
SL	115G, 1176*	13, 17, 46
SLOAD	041, 043*	6, 39, 61
SLR	115G, 117G*	13, 17, 46
SL1	024	11, 17, 45
SL1R	004	11, 17, 45
SL2	064	11, 17, 45
SL2R	044	11, 17, 45
SL3	124	11, 17, 45
SL3R	104	11, 17, 45
SL4	164	11, 17, 45
SL4R	144	11, 17, 45
SQRT	010	10, 42
SR	115G, 117G*	13, 17, 46
SRR	115G, 117G*	13, 17, 46
SR1	034	11, 44
SR1R	014	11, 45
SR2	074	11, 44
SR2R	054	11, 45
SR3	134	11, 44
SR3R	114	11, 45
SR4	174	11, 44
SR4R	154	11, 45

VSR6	134	12, 45
VSR7	154	12, 45
VSR8	174	12, 45
VSQ	140	11, 17, 44
VSU	125, 127*	8, 17, 29, 41, 61
VXM	071, 073*	8, 17, 42, 61
VXSC	015, 017*	8, 29, 41, 61
VXV	141, 143*	8, 17, 29, 42, 61
V/SC	035, 037*	8, 17, 29, 42, 61
XAD, 1	106	20, 52, 62
XAD, 2	102	20, 52, 62
XCHX, 1	056	20, 52, 62
XCHX, 2	052	20, 52, 62
XSU, 1	116	20, 53, 62
XSU, 2	112	20, 53, 62