3420.5-27
REVISION 2

APOLLO GUIDANCE PROGRAM SYMBOLIC LISTING INFORMATION FOR BLOCK 2

NAS 9-8166

20 NOVEMBER 1969

Prepared for
MISSION PLANNING AND ANALYSIS DIVISION
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
MANNED SPACECRAFT CENTER
HOUSTON, TEXAS

WORKING PAPER

ABSTRACT

The information presented in this document on the mechanization of current Block 2 Apollo guidance computer programs is intended for use only as an aid to the understanding of guidance program symbolic listings for both the Command Module and the Lunar Module. The material which is included is oriented towards permitting a user to understand the computations being performed by the program, and to follow the logic associated with the control of a complete guidance program assembly. With the aid of the information in this document, it should be possible to become proficient in determining from a program symbolic listing just what computations are being carried out. This document, however, is insufficient to permit a user, without access to supplemental material, to write a reliable program for the guidance computer.

This document supersedes the Revision 1 issue of 3420.5-27 (dated 27 June 1968), and includes the updating information that is summarized in Appendix B. Major sections of the document are devoted to computer hardware information, machine language instructions, the format and features of the assembly program, the interpretive language, and program performance control. To facilitate using the document for reference, appendices contain a review of computer concepts, a summary of computer inputs and outputs, and alphabetical listings of machine and interpretive operations, registers and tags, and terms.

This document was produced in order to fill the need for a compilation of this material for use by those interested in reviewing a guidance program symbolic listing for Block 2. A number of assumptions had to be made concerning the operating characteristics of the hardware and the intended application of the software, and therefore this document must never be used as definitive information on the guidance computer hardware or programs. If such information should be required, the G&N contractor is the proper source for it, not this document. Since this document has not been approved by NASA, it should never be cited as a reference in official NASA documentation.

CONTENTS

I INTRODUCTION

Under the auspices of TRW Systems MTCP task A-201 ("Support of Apollo Guidance Document Review," J. Garman and J.E. Williams, FS5, task monitors), information on certain of the hardware and software aspects of an Apollo Block 2 primary guidance system program symbolic listing has been assembled into this document. The purpose of this effort was to facilitate the review of Block 2 guidance program symbolic listings by those unfamiliar with the Apollo guidance computer program listing format. This document is Revision 2 of an earlier document on the same subject (originally published on 10 January 1967, with Revision 1 published on 27 June 1968), and completely supersedes these previous documents. Appendix B summarizes some of the significant changes made since the Revision 1 issue.

Several different sources of information were used during the preparation of this document, including:

1. A program assembly listing bearing the heading print:

   GAP: ASSEMBLE REVISION 072 OF AGC PROGRAM COMANCHE
   BY NASA 2021113-071 18:53 OCT. 17,1969

   "COMANCHE" is the term used for the COLOSSUS (manned CSM earth/lunar capability) 2x series of programs: this version is intended for use on Apollo 13, and is also referred to as COLOSSUS 2D.

2. A program assembly listing bearing the heading print:

   GAP: ASSEMBLE REVISION 130 OF AGC PROGRAM LUMINARY
   BY NASA 2021112-081 18:29 NOV. 4,1969

   "LUMINARY" is the term used for the manned LM earth/lunar capability programs: this version is intended for use on Apollo 13, and is also referred to as LUMINARY 1C.

3. Raytheon Apollo Guidance Computer Information Series publications, used for much of the hardware information. Two separate documents were employed, one identified as "Issue 30, Block II Apollo Guidance Computer Subsystem, FR-2-130", updated 25 February 1966; the other was "Issue 32, Block II Machine Instructions, FR-2-132", updated 25 March 1966.
4. MSC LM G&C Data Book, Revision 2, dated 15 July 1967.
5. "Apollo Operations Handbook for CSM, Volume 2 for Apollo 12, CSM 108 and Subsequent," dated 10 October 1969.
6. AGC4 Memo #9, "Block II Instructions," revised 1 June 1967.

7. A GAP assembly program listing dated 17 May 1968.
8. A number of other miscellaneous sources of information, such as other program listings, G&N contractor documentation, etc.

It should be clearly understood that the information in this document was derived from program symbolic listings, and hence cannot be considered to be an "independent source", particularly when matters such as signal polarities and channel bit assignments are considered. In addition, changes to some of the material presented in this document may well take place in the future (such as the addition of a "rate-aided optics" capability to the CSM), so that where applicable information should be checked against mission-peculiar data prepared based on a specific program assembly.

The material on the following pages should under no circumstances be employed as a definitive description of the guidance hardware or portions of the guidance software. Such information, including that necessary to write (as contrasted with read) guidance computer programs, should be provided only by the G&N contractor through the appropriate MSC channels.

Notation

The notation employed in this document is intended to be consistent with that employed in the previous two issues, as well as with documents which have been produced on the AS-202, AS-204, LM-1, Sundisk, and Colossus programmed guidance equations. For convenience, some of this specialized notation is summarized below (specialized notation is also defined in individual sections to which it applies, such as Section IVA for machine language order codes).

1. Unless otherwise specified, information applies to both the Command Module (CM) and Lunar Module (LM) computers. These abbreviations are used when it is necessary to cite hardware or software differences between the two systems.
2. Unless otherwise specified, material which is provided is intended to be consistent with sources #1 and #2 on page I-1. Material applicable to earlier programs, but no longer valid, is cited only when of historical or potential future application.

3. Bits are numbered from #15 (the sign bit) and #14 (the most significant magnitude bit) down to bit #1 (the least significant bit of the 15-bit computer word).

4. A quantity in capital letters, unless an operation code, generally means the contents of a cell with that tag. The capital letter E, with subscripts of quantities in capital letters, is reserved to mean "the contents of the cell or cells whose tags are in the subscript." Hence E_{TS}, for example, would be the contents of the cell whose address is stored in cell TS. TS alone, of course, would be the contents of the cell with tag TS.

5. A quantity in quotation marks indicates that its address is of interest rather than the contents of that address. No quotation marks are used, however, unless the quantity corresponds to a tagged program step: transfers in Section VIB to other interpretive operations are indicated without quotation marks.

6. Logical branches in the software are indicated by "If" statements, with subsequent equation information indented to indicate the extent of the computations performed should the "If" condition be satisfied.

7. Unless otherwise specified, numbers are given in decimal. The subscript 8 signifies an octal number and the subscript 2 signifies a binary number. Where conventional reference to quantities (such as channel numbers) considers them to be octal, however, the subscript has been omitted.

8. The equation X = ABCD +2 means that X is set to the contents of the address with tag "ABCD" plus 2; X = ABCD+2, however, omitting the space before the sign, means that the cell address used is the cell two memory locations beyond "ABCD". If variables are used as subscripts, however, the first meaning always applies.

9. Perform "XXXX" means transfer to the routine starting at "XXXX" and retain return address information (to permit return after completion of the routine); Proceed to "XXXX", however, merely means transfer to routine starting at "XXXX".

10. The scale factor of a quantity is the power of two by which the number in the computer (considered as a fraction in the range between -1 and +1) must be multiplied to obtain its true value. The scale factor is frequently shown as Bxx, to signify binary as opposed to decimal exponent information. See Appendix A for more details.

11. The equation: "Set AA = BB and BB = AA" means to exchange the contents of the cells with addresses "AA" and "BB".

12. The subscripts dp, tp, vc, x,y, and z mean double precision, triple precision, vector, and vector x-z components respectively.

13. The equation $B = (b_1, b_2)$ means form a double precision number B with most significant half b_1 and least significant half b_2. The equation $\underline{A} = (a_1, a_2, a_3)$ means form a vector from the indicated components.

14. A task (short sequence of computations based upon some time or event criterion) may be entered into a list for subsequent performance in yy seconds (see Section VIIA). The notation for this is: Call "XXXX" in yy seconds, where "XXXX" is the starting address of the waitlist task.

15. A job (computation that is not a task) may be entered into a list for performance when its priority is high enough (see Section VIIB). The notation for this is: Establish "XXXX".

16. A bit is "set" when its value is made a binary one, and is "cleared" or "reset" when its value is made a binary zero. "Set" is also used to mean "force bit value to be as specified".

17. In some cases an address may be determined from the combination of a bank register and a quantity giving an address within the bank (see Section IIB). In such cases, the bank setting is sometimes shown explicitly and the address in the bank used as described in #4 above.

18. The operation sgn X causes the quantity it affects to be complemented if X is negative (same as multiplication by the quantity $X/|X|$).

19. Where reasonably apparent from the context, explicit scaling is not shown in the equations (and can be assumed to be proper). For example, $MPAC_{tp} = MPAC_{tp} + MPAC+2$ is frequently used for rounding to double precision. MPAC+2 is added to itself and carries propagated.

20. In some cases, loading of registers (such as memory-control cells) that comprise fewer than the 15 bits of the normal computer word length is indicated as if the loading were accomplished by a mask-type procedure, in order to demonstrate the function of individual bits in a word. If hardware design changes were to occur, of course, the indicated masking would no longer be applicable.

II COMPUTER HARDWARE INFORMATION

IIA General Data

The Apollo Block 2 primary guidance computer may be classified as a general-purpose parallel operation binary computer. Various details of its hardware, necessary for proper interpretation of its programs, are given in Section II on the following pages. These details may be summarized to be:

Number System: Fractional binary, with negative numbers generally in ones complement form. Numbers in arithmetic unit operated on in parallel. Angle information is in twos complement form.

Word Length: Sign and 14 magnitude bits of information. Words stored in memory have a sixteenth bit for parity purposes; in the arithmetic unit, the sixteenth bit is used for overflow detection. A limited number of double precision operations are included in the order code.

Error Detection: Odd parity for all cells read from memory.

Erasable Memory: Random access coincident-current ferrite core, destructive readout. Total capacity is 2040 cells, of which 12 are allocated to special functions and 29 to counters. There are 760 cells which are uniquely addressable (including the special function and counter cells); the remaining 1280 cells are addressed (in modules of 256) with erasable memory bank register.

Fixed Memory: Random access core rope, non-destructive readout. Total capacity is 36,864 cells, of which 2048 are uniquely addressable; 22,528 addressed (in modules of 1024) with fixed memory bank register; and 12,288 addressed with both fixed memory bank register and an additional register.

Instruction Format: Three to six bits for operation code, remaining bits for address.

Hardware Registers: Total of 26 may be addressed, of which four are associated with arithmetic unit; four with memory control; ten with computer outputs (channels); and 8 with computer inputs (channels).

Operation Codes: 15 regular machine-language.
19 "extended" machine-language (requires 2 orders).
4 "special" machine-language (hardware functions).
4 shift-register cells for bit shifts.
7 "involuntary" for counter operations.
2 "involuntary" for program interrupts (one can be programmed).
5 "peripheral" for test equipment interface.

Interrupts: 29 for counter control.
11 for program control.

Speed: 23.4 microseconds for single precision addition-type orders.
46.9 microseconds for multiplication (net).
82.0 microseconds for division (net).

Hardware: About 70 pounds weight.
About one cubic foot volume.
About 100 watts power in operation, 10 watts in standby.

The basic source of all timing for the computer is an oscillator which operates at a frequency of 2.048 mc. The output is divided by two to obtain the computer logic-control-pulse rate of 1.024 mc (or a period of 0.9765625 microseconds). A set of computer logic-control-pulses that occur simultaneously is termed an "action": 12 actions make up (usually) a "subinstruction", which takes place in 11.71875 microseconds. Since this is the time for a complete memory cycle, the time interval is

called a "Memory Cycle Time", or "MCT". All instructions take an integral number of MCT's to perform.

The computer clock output of 1.024 mc is applied to a ring counter, which gives an output after dividing by ten at 102.4 kc. This signal is applied to a 33-stage binary counter, whose various output frequencies (3200 pps, 100 pps, 0.78125 pps, etc.), both in phase and out of phase, are used to provide various timing signals for computer functions. In addition, the most significant 28 stages of this counter are available as input channels 03 and 04, and can be used to permit restoration of the computer erasable memory record of time since launch after a period of low power (standby) operation.

An odd parity bit, making the sum of binary ones in the memory cell, including itself, an odd value, is included with all fixed memory cells, and is generated when erasable memory cells are loaded. Readout of a memory cell is accompanied by an automatic check for the validity of the parity bit, and a hardware restart is generated if the parity bit is determined to be inconsistent with the word. Because of this parity check, the existence of an odd number of errors (1, 3, 5, etc.) in the information read from memory can be detected, including cases where all 16 bits are zero or one.

IIB Address Allocation

The character of the address allocation problem in the computer can be described by first considering the hardware which would be necessary to be able to use any instruction with any address. Excluding the special-purpose channel instructions, there are 27 instructions in the computer machine language repertoire, would would require five bits for representation ($2^5 = 32$). There are also 38,912 addressable cells in the computer (36,864 fixed memory, 2040 erasable memory, and 8 non-channel hardware registers), requiring 16 bits for representation ($2^{16} = 65,536$). This would give a total requirement for 21 bits in the instruction word length, or six bits more than the actual hardware instruction word length of 15 bits (plus the 16th odd-parity bit).

In order to obtain the necessary hardware (and software) effect of the "missing" bits at a minimum penalty, the following special design features are found in the computer logic:

1. Instructions which are used comparatively infrequently (such as multiply and divide) require two lines of coding, with the first line setting an "extended-order" flip-flop (which is reset after the order is performed).
2. Several instructions can be used with only one type of memory: most transfer orders, for example, can refer only to addresses in fixed memory, and instructions which load a memory cell can refer only to addresses in erasable memory. In addition, the computer digital (as contrasted with analog-type pulse input/output) input/output information is handled through "channels", which can be addressed only by a special group of instructions intended solely for that purpose.

3. The erasable memory cells are divided into eight "banks" of 256 cells each, with banks 0-2 ("unswitched erasable") addressable directly (bank 0 includes the 8 non-channel hardware registers), and the remaining banks, 3-7, selected with the aid of a three-bit "EBANK" register (cell 0003_8). These non-uniquely addressed banks are referred to as "switched erasable": the cell within the bank, of course, is determined by the address portion of the instruction word.

4. The fixed memory cells are also divided into "banks", but these have a capacity of 1024 cells each. Two of the banks, 02 and 03, are addressable directly, and hence are known as "fixed-fixed" memory. Of the remaining 34 banks, 22 are selected with the aid of a five-bit "FBANK" register (cell 0004_8), and comprise banks 00, 01, and 04-27 (by convention, banks are considered to be octal quantities), and are "variable-fixed" memory. The other 12 banks in fixed memory, banks 30-43, are considered to be "superbanks", since their selection also depends on the setting of channel 7 (SUPERBNK). The three bits of this channel can be considered an "extension" to the FBANK capacity (hence the channel is sometimes referred to as "F EXT"), with a setting of 0-3 selecting superbank S3 (banks 30-37) and a setting of 4 selecting superbank S4 (banks 40-43). The other 4 banks for superbank 4, plus those for superbanks 5-7, are not presently included in the computer hardware. The cell within the superbank, of course, is determined by the FBANK setting and by the address portion of the instruction word.

The logical design of the computer includes a twelve-bit memory address register ("S-register") which, together with suitable EBANK or FBANK and SUPERBNK information if necessary, is used to specify memory cell locations within the computer. The S-register is not necessarily loaded with bits 12-1 of the instruction word, however, since some instructions use these bits to help determine the operation code.

Computer Memory Address Allocation

"True" Address	S-register	EBANK	FBANK	SUPERBNK	Function
00000-00007	0000-0007	---	---	--	Non-channel hardware cells.
	or				
	1400-1407	0			
00010-00060	0010-0060	---	---	--	Special Erasable cells.
	or				
	1410-1460	0			
00061-00377	0061-0377	---	---	--	Bank 0 of Unswitched Erasable.
	or				
	1461-1777	0			
00400-00777	0400-0777	---	---	--	Bank 1 of Unswitched Erasable.
	or				
	1400-1777	1			
01000-01377	1000-1377	---	---	--	Bank 2 of Unswitched Erasable.
	or				
	1400-1777	2			
01400-01777	1400-1777	3	---	--	Bank 3 of Switched Erasable.
02000-02377	1400-1777	4	---	--	Bank 4 of Switched Erasable.
02400-03777	1400-1777	5-7	---	--	Banks 5-7 of Switched Erasable
04000-05777	4000-5777	---	---	--	Bank 02 of Fixed-fixed Memory.
	or				
	2000-3777	---	02	--	
06000-07777	6000-7777	---	---	--	Bank 03 of Fixed-fixed Memory.
	or				
	2000-3777	---	03	--	
10000-13777	2000-3777	---	00-01	--	Banks 00-01 of Variable-fixed. (Cell conversion 00000-03777)
20000-67777	2000-3777	---	04-27	--	Banks 04-27 of Variable-fixed. (Cell conversion 10000-57777)
70000-107777	2000-3777	---	30-37	≤ 3	Superbank S3, banks 30-37. (Cell conversion 60000-77777)
110000-117777	2000-3777	---	30-33	4	Superbank S4, banks 40-43. (Cell conversion 100000-107777)

The addresses in the computer are allocated as shown on the previous page (all numerical quantities are given in octal). As can be seen from the table, the following general rules apply for selection of cells within the computer:

1. If bits 12-11 of the S-register are both zero (S-register in the range 0000-1777), then the erasable memory (or non-channel hardware cells) is read. If, however, one or both of these bits are one (S-register in range 2000-7777), then the fixed memory is read.
2. The contents of EBANK influence the address which is selected if bits 12-11 of the S-register are both zero and bits 10-9 are both one (S-register in range 1400-1777, giving erasable memory bank selection capability of 256 cells).
3. The contents of FBANK influence the address which is selected if bit 12 of the S-register is zero and bit 11 is one (S-register in range 2000-3777, giving fixed memory bank selection capability of 1024 cells).
4. The contents of SUPERBNK influence the address which is selected if the most significant two bits of FBANK are both one (FBANK in range 30-37) and if S-register is in range 2000-3777. Note that values of SUPERBNK between 0 and 3 will <u>all</u> select banks 30-37, contrary to analogous options for EBANK and FBANK (which have non-redundant cell selection).

The quantity listed on the previous page as the "true" address is used for assembler purposes (to specify the absolute starting address for a set of computations). In order to convert an erasable memory "true" address to hardware cell contents, the following process can be used:

S-register = 1400_8 + bits 8-1 of "true" address

EBANK = bits 11-9 of "true" address

The "true" address is the one specified when external inputs that require specification of absolute cell location are required (such as for certain uplink sequences and for address-to-be-specified inputs to the display system). For programming convenience, the three bits of EBANK are connected to bits 11-9 of the computer hardware accumulator.

In order to convert a fixed memory "true" address to hardware cell contents, first subtract 10000_8 if the "true" address is above that value: the result of such a subtraction, identified as "cell conversion", is shown in the Function column of the table. Starting with this "fixed" address, the following process can then be used to determine address selection parameters:

S-register = 2000_8 + bits 10-1 of "fixed" address

SUPERBNK = bits 16-14 of "fixed" address (values of 0-3 the same for hardware)

FBANK = bits 15-11 of "fixed" address for SUPERBNK ≤ 3

= 30_8 + bits 13-11 of "fixed" address for SUPERBNK > 3

For programming convenience, the five bits of FBANK are connected to bits 15-11 of the computer hardware accumulator. In addition, the cell BBANK (address 0006) has both FBANK (bits 15-11) and EBANK (in bits 3-1) connected to it, and hence can be used to sample or load both bank registers, provided the loading information is in the proper format. SUPERBNK is connected to bits 7-5 of the computer hardware accumulator, but it must be loaded by a channel order (note, however, that the bits for SUPERBNK are compatible with the assigned bits in BBANK, permitting one 15-bit computer word to have suitably formatted information for all three quantities).

IIC Hardware Registers

There are eight non-channel hardware cells in the guidance computer, with addresses 0000_8 - 0007_8. These cells are described below (see Section IIE for the computer channels).

Address	Symbol	Function
0000_8	A	Accumulator. Most instructions refer to, or modify, the contents of A. See Section IV.
0001_8	L	L register, or Low Order Accumulator. Used to contain the least significant half of double precision words for those operations which use, or generate, such words, and to contain the remainder after division. Cell also forms channel 01, permitting channel operations to be used for bit manipulation purposes, when symbol conventionally is LCHAN. Cell frequently used for temporary storage purposes within a computation.
0002_8	Q	Q register, or Return Address Register. Loaded with the value of the program count (cell 0005_8) of the step following a TC (transfer control, see Section IVB) instruction, thus retaining return-address information. Cell also forms channel 02, permitting channel operations to be used for bit manipulation purposes, when symbol conventionally is QCHAN. Cell frequently used for temporary storage purposes within a computation.
0003_8	EBANK	Erasable Memory Bank Selector, consisting only of bits 11-9 (which are also connected to bits 3-1 of cell 0006_8). Contents used to specify which bank of 256 erasable memory cells (bits 8-1 of S-register) is to be selected for S-register in the range 1400_8 - 1777_8. See Section IIB.

Address	Symbol	Function
0004_8	FBANK	Fixed Memory Bank Selector, consisting only of bits 15-11 (which are also connected to bits 15-11 of cell 0006_8). Contents used to specify which bank of 1024 fixed memory cells (bits 10-1 of S-register) is to be selected for S-register in the range 2000_8 - 3777_8. See Section IIB.
0005_8	Z	Z register, or Program Counter. Contains address of the next step, and for most instructions it is incremented by 1 under hardware control. Can be loaded directly by program (pseudo-operations DTCB and DTCF) in order to accomplish a transfer of program control. Incrementing takes place as part of termination of previous instruction (so that direct loading of register 0002_8 from Z for a TC order achieves the desired effect).
0006_8	BBANK	Both Banks, a cell which may be used if reference for reading or writing to both EBANK and FBANK is desired. The three bits (11-9) of EBANK are connected to bits 3-1 of BBANK, while the five bits (15-11) of FBANK are connected to bits 15-11 of BBANK. The SUPERBNK bits (7-5) are <u>not</u> connected to BBANK, but instead must be processed by a separate channel instruction (referencing channel 7).
0007_8	---	Address which may be used as a source of 00000_8 for clearing instructions (such as the pseudo-operations ZL and ZQ). No physical register corresponds to this address, so that "loading" of the address has no effect, and hence can be used to achieve desired program performance (such as modification of the A register via the TS order, Section IVB). An attempt to read an unwired fixed memory cell to obtain 0 would give a hardware restart due to parity failure.

IID Special Erasable Cells

The first 41 cells of the erasable memory (locations 0010_8 - 0060_8) are nominally allocated to special functions, with the last 29 (starting at location 0024_8) being used for counter purposes and the first 12 for other specialized purposes (although some function as normal erasable memory cells). In addition, cell 0067_8 serves a special hardware function in monitoring for a program loop and initiating a hardware restart if one is detected. These cells and their functions are described below.

Address	Symbol	Function
0010_8	ARUPT	Normal erasable memory cell used by convention to contain the contents of the accumulator (A register) after program interrupts #1 - #10 (see Section IIH) acted upon, and used to restore these contents before resuming the interrupted computation.
0011_8	LRUPT	Normal erasable memory cell used by convention to contain the contents of the L register after program interrupts #1 - #10 acted upon, and used to restore these contents before resuming the interrupted computation.
0012_8	QRUPT	Normal erasable memory cell used by convention to contain the contents of the Q register (if these contents would be modified during the computations associated with the interrupt) after program interrupts #1 - #10 acted upon, and used to restore these contents before resuming the interrupted computation (if, of course, QRUPT loaded).
0013_8 0014_8	SAMPTIME	Normal erasable memory cells used to retain the value of cells 0024_8 - 0025_8 when certain program steps are performed (e.g. steps for program interrupts #5 - #7), for subsequent possible display.

Address	Symbol	Function
0015_8	ZRUPT	Cell used to contain the value of the program counter (Z register) when a program interrupt acted upon. It is usually loaded and restored to Z by hardware means, although it can also be sensed and stored by software.
0016_8	BANKRUPT	Normal erasable memory cell used by convention to contain the contents of BBANK (if these contents would be modified during the computations associated with the interrupt) after program interrupts #1 - #10 acted upon, and used to restore these contents before resuming the interrupted computation (if, of course, BANKRUPT loaded). For those interrupts changing SUPERBNK, BANKRUPT also used to retain the SUPERBNK of the interrupted computation, thus requiring special restoration coding.
0017_8	BRUPT	Cell used to contain the value of the nonaddressable "B-register" (buffer register, used to contain the next instruction) when a program interrupt is acted upon. It is normally loaded and restored to B by hardware means, although it can also be sensed and stored by software. Loading the cell with a certain program (Z register) count and then executing a RESUME will cause program to start at indicated step, since the TC (transfer control) order has operation code = 0.
0020_8	CYR	Cycle right register. When the contents of the cell are written into, either as part of the original loading or as a result of most sensing operations (such as CA, Clear Add), they are shifted right one place in a cyclic fashion: bit 15 becomes bit 14, bit 14 becomes bit 13, ... bit 2 becomes bit 1, and bit 1 becomes bit 15. The unshifted value (except as it is shifted from a previous loading) is the one sensed. Shifting does <u>not</u> take place for the MASK, MP (Multiply), or DV (Divide) orders. See Section IV.

Address	Symbol	Function
0021_8	SR	Shift right register. When the contents of the cell are written into, either as part of the original loading or as a result of most sensing operations, they are shifted right one place in a non-cyclic fashion: bit 15 becomes bit 15 and bit 14, bit 14 becomes bit 13, ... bit 2 becomes bit 1, and bit 1 is lost. The unshifted value (except as it is shifted from a previous loading) is the one sensed. Shifting does not take place for the MASK, MP, or DV orders. See Section IV.
0022_8	CYL	Cycle left register. When the contents of the cell are written into, either as part of the original loading or as a result of most sensing operations, they are shifted left one place in a cyclic fashion: bit 1 becomes bit 2, bit 2 becomes bit 3, ... bit 14 becomes bit 15, and bit 15 becomes bit 1. The unshifted value (except as it is shifted from a previous loading) is the one sensed. Shifting does not take place for the MASK, MP, or DV orders. See Section IV. The effect of a shift left in a non-cyclic fashion (except for bit 15) can be achieved by addition of accumulator to itself the proper number of times.
0023_8	EDOP	Edit operand register. When the contents of the cell are written into, either as part of the original loading or as a result of most sensing operations, bits 14-8 are loaded into bits 7-1 respectively, and bits 15 and 14-8 are set 0. The unshifted value (except as it is shifted from a previous loading) is the one sensed. Shifting does not take place for the MASK, MP, or DV orders. See Section IV. The right shift of 7 places for the selected bits is used for interpreter and verb/noun pattern editing requirements.

Address	Symbol	Function
0024_8	TIME2	Cell used as the most significant half of the computer "clock", preset and sensed under program control. It is incremented by +1 when cell 0025_8 overflows. TIME2 overflows every 745 hours (i.e. 31 days 1 hour), 39 minutes, 14.56 seconds, and is conventionally reset when liftoff is deduced so as to indicate mission elapsed time.
0025_8	TIME1	Cell used as the least significant half of the computer "clock", preset and sensed under program control. It is incremented by +1 each 0.01 second (i.e. each centi-second). When the cell overflows (each 163.84 seconds), TIME2 is incremented by +1. See Section IIE for phasing with respect to Channel 04 time information.
0026_8	TIME3	Cell used to generate (when overflow takes place) program interrupt #3 (conventionally used for "waitlist" tasks, see Section IIH). Preset to appropriate value under program control (i.e. 2^{14} - required delay in centi-seconds), and incremented by +1 each 0.01 second. See Section VIIA for computations associated with determining proper settings.
0027_8	TIME4	Cell used to generate (when overflow takes place) program interrupt #4 (conventionally used for periodic "T4RUPT" input/output functions, see Section IIH). Preset to appropriate value under program control (i.e. 2^{14} - required delay in centi-seconds), and incremented by +1 each 0.01 second. Incrementing phased so as to take place 0.0075 second after the TIME3 increment.

Address	Symbol	Function
0030_8	TIME5	Cell used to generate (when overflow takes place) program interrupt #2 (conventionally used for computations associated with the digital autopilots, see Section IIH). Preset to appropriate value under program control (i.e. 2^{14} - required delay in centi-seconds), and incremented by +1 each 0.01 second.
0031_8	TIME6	Cell used to generate (after has been decremented to -0) program interrupt #1 (conventionally used for timing of RCS jet commands in output channels 05 and 06 from the digital autopilots, see Section IIH). Preset to appropriate value under program control (i.e. required delay in units of 2^{-4} centi-seconds), and decremented by 1 each 0.000625 second (i.e. at a 1600 pps rate) provided bit 15 of channel 13 is 1. When the counter reaches a value of -0, the next DINC pulse causes bit 15 of channel 13 to be set 0 and the program interrupt to be generated.
0032_8 CDUX 0033_8 CDUY 0034_8 CDUZ		Cells accumulating the output pulses from the three CDU's (Coupling Data Units) associated with the IMU (Inertial Measurement Unit), to provide information on the IMU gimbal angles (and hence on spacecraft attitude). Cells preset and sensed under program control, providing angle information in twos complement form, scale factor B-1 revolutions. One pulse, therefore, is 2^{-15} revolutions, equivalent to 39.55078125 arc sec (about 0.01098633°).
0035_8(CM)	CDUT	Cell accumulating the output pulses from the CM optics trunnion CDU, to provide information on optics trunnion angle. Cell preset and sensed under program control, with optics zeroing (bit 1 of channel 12) giving a setting of 61740_8 (about -19.7754° in twos complement) for the cell. Scale factor B-3 revolutions, with data in twos complement. Zeroing points about 32° 31' 23.19" away from Z axis (towards X axis).

Address	Symbol	Function
0035_8(LM)	CDUT	Cell accumulating the output pulses from the LM rendezvous radar trunnion CDU, to provide information on rendezvous radar trunnion angle. Cell preset and sensed under program control, with radar CDU zeroing (bit 1 of channel 12) giving a setting of 0 for the cell (due to software), and also inhibiting cell increments. Scale factor B-1 revolutions, in twos complement.
0036_8(CM)	CDUS	Cell accumulating the output pulses from the CM optics shaft CDU, to provide information on optics shaft angle. Cell preset and sensed under program control, with optics zeroing (bit 1 of channel 12) giving a setting of 0 for the cell. Scale factor B-1 revolutions, with data in twos complement.
0036_8(LM)	CDUS	Cell accumulating the output pulses from the LM rendezvous radar shaft CDU, to provide information on rendezvous radar shaft angle. Cell preset and sensed under program control, with radar CDU zeroing (bit 1 of channel 12) giving a setting of 0 for the cell (due to software), and also inhibiting cell increments. Scale factor B-1 revolutions, in twos complement.
0037_8 0040_8 0041_8	PIPAX PIPAY PIPAZ	Cells accumulating the output pulses from the three PIPA's (Pulsed Integrating Pendulous Accelerometers) associated with the IMU, to provide information on sensed velocity increments in IMU coordinates. Cells preset and sensed under program control, providing information with scale factor B14 counts. Nominal CM scale factor is 5.85 cm/sec per count; nominal LM scale factor 1.00 cm/sec per count. When the software resets the cells, the accelerometer electronics is <u>not</u> affected, so that fractional counts accumulated there would not be disturbed.

Address	Symbol	Function
0042_8(CM) 0043_8(CM) 0044_8(CM)	BMAGX BMAGY BMAGZ	Not used. Originally intended to provide an accumulation of angle increment data from the Gyro Display Coupler of the Spacecraft Control System BMAG's (Body Mounted Attitude Gyros), to serve as a backup source of attitude information in the event of IMU failure. Inputs to cells enabled if bit 8 of channel 13 is set. Cells preset and sensed under program control.
0042_8(LM) 0043_8(LM) 0044_8(LM)	Q-RHCCTR P-RHCCTR R-RHCCTR	Cells accumulating the output pulses from the RHC (Rotational Hand Controller) pitch, yaw, and roll axes respectively, used if the RHC is employed as a rate commanding device. If RHC used as a minimum impulse or landing point designator device, however, bits 6-1 of channel 31 used instead to determine the status of the controller. Inputs to counters enabled if bit 8 of channel 13 is set, and counters must be reset to 0 under program control. Bit 9 of channel 13 is used to cause a readout of the RHC analog-to-digital converters to be started, and then becomes reset. Separate sign and magnitude information is received from the converter, with magnitude provided by width of a dc pulse (which gates a 3200 pps signal to the counter for digital conversion). Full-scale deflection (into soft stops) of RHC provides an input count of 42, scale factor B14 counts, with corresponding value of rate command determined by software. Software does not enable counting unless bit 15 of channel 31 indicates that RHC is out of detent (giving a minimum control capability of about 10% of full scale). Also known in LM as "ACA" (for "Attitude Controller Assembly").

Address	Symbol	Function
0045_8	INLINK	Cell into which serial binary data is shifted from the uplink receiver (after completion of the checks by the receiver for satisfactory message format) one bit at a time under hardware control. The overflow of this cell (implemented by having the first of the 16 bits sent to the computer be a binary one) causes program interrupt #7 (see Section IIH). Cell must be reset to 0 by software to permit next word to be processed properly. Software performs additional checks on the 15-bit word read from the cell: bits 5-1 are checked to be the same as bits 15-11, and the same as the complement of bits 10-6, before processing the input further (using the five-bit codes listed in Section IIJ, the same as for DSKY inputs). If failure of the software check is encountered, all subsequent inputs are rejected by the software until an error reset pattern (22_8) is received via the uplink (<u>not</u> DSKY). No inputs to the cell are made if bit 6 of channel 13 is 1, nor if either of the spacecraft switches (the CM has two) are set to block uplink inputs (cf. bit 10 of channel 33). In addition, an incoming bit is rejected if a 6400 pps signal has not occurred since the previous bit was accepted, and bit 11 of channel 33 (a flip-flop) is set to be sensed as a binary 0 to indicate such a rejection. Checks for too rapid an uplink rate made only if bit 5 of channel 13 is 0 and if switches set to accept uplink. Bit 5 of channel 13 can be set 1 to select output of "crosslink" hardware (cf. cell 0057_8) instead of the uplink receiver for cell 0045_8 input, but this capability is not used. The spacecraft switches cannot be set to block crosslink inputs from cell 0045_8, although the same monitoring for too fast an input rate is made.

Address	Symbol	Function
0046_8	RNRAD	Cell into which VHF range data (for CM) or landing radar data (velocity and altitude) and rendezvous radar data (range and range rate) for LM is shifted under control of bits 4-1 of channel 13. The rendezvous radar angle data is in cells 0035_8 - 0036_8. The source and type of data is selected by bits 3-1 of channel 13, and when bit 4 of channel 13 becomes 1 the readout process is started, being terminated 90-100 ms later by the generation of program interrupt #9 and the resetting of bit 4 of channel 13. All 15 bits loaded are magnitude bits. When the first 100 pps signal after bit 4 of channel 13 becomes 1 occurs, a 3200 pps pulse train is generated on an appropriate computer output line (depending on the selection made by bits 3-1 of channel 13). This pulse train, which lasts for about 80 ms, is used by the radar to gate the selected data into a radar counter. About 5 ms after the termination of this pulse train, 15 sync pulses (on a separate line from the data gating pulses) are sent, again at a 3200 pps rate, to shift data from the counter to cell 0046_8. Improper shifting results if these sync pulses are not of the proper waveform (due to a channel 13 loading command, for example). After the last sync pulse (or 10 ms after the end of the measurement pulse train), program interrupt #9 is generated. For the CM, the least increment on the quantity loaded into cell 0046_8 is about 0.01 nm (18.52 meters). For the LM, the landing radar measurement is made for about 80.001 ms, with a 153.6 kc bias on rates (bias count of 12288.2), and least increments of about -0.6440, 1.212, and 0.8668 fps/bit for x-z velocities. On the low range LR scale (cf. bit 9 of channel 33), least increment is about 1.079 feet/bit (high range 5.000 times bigger). For rendezvous radar, range on low scale (cf. bit 3 of channel 33) about 9.38 feet/bit; high range 8.000 times bigger. Range rate (counts for 80 ms) bias frequency is 212.5 kc (17000 bias count), and scale -0.6278 fps/bit.

Address	Symbol	Function
0047_8	GYROCMD	Cell which is loaded with the magnitude of the required IMU gyro torquing command, scale factor B14, units counts (one count is 2^{-21} revolution or about 0.61798096 arc seconds). Output pulses are generated at a 3200 pps rate, with power supply for them enabled by bit 6 of channel 14 and the sign and axis of the gyro to be torqued specified by bits 9-7 of channel 14. When bit 10 of channel 14 is set, the pulses are started (and GYROCMD decremented appropriately). When GYROCMD reaches zero, the pulses are terminated and bit 10 of channel 14 reset.
0050_8 0051_8 0052_8	CDUXCMD CDUYCMD CDUZCMD	Cells loaded with values to be transmitted to IMU CDU error counters. Information gated out of cells if bits 15-13 (respectively) of channel 14 are set, and error counters loaded if bit 6 of channel 12 is set. These "error counters" should be considered as being in large measure independent of the "CDU" information in cells 0032_8 - 0034_8, and essentially serve the purpose of digital-to-analog converters. The error counters saturate at a count of 600_8 (or 384 counts), and are incremented at a 3200 pps rate for a count determined by their respective CDUiCMD cell. If bit 4 of channel 12 is set, the error counter data is used for coarse align of the IMU (and the count in the error counter decremented in magnitude as the IMU alignment proceeds). The error counters associated with all 3 cells are reset 0 if bit 6 of channel 12 is reset to 0. The scale factor of the cells for IMU coarse align is B1 revolutions (so that one pulse corresponds to 2^{-13} revolutions or about 158.2 arc seconds). See next page for additional CM-only and LM-only uses.

Address	Symbol	Function
0050_8(CM) 0051_8(CM) 0052_8(CM)		See previous page for items common to CM and LM uses. If bit 9 of channel 12 is set, error counter output (converted to dc) is used for roll, pitch, and yaw control of the Saturn. Error counter output also used for roll, pitch, and yaw attitude error displays respectively on FDAI (Flight Director Attitude Indicator). Software loads cells with data scaled B1 revolutions (saturated error counter = 16.875° except for roll during boost and entry, when scale factor is B3 revolutions (saturation = 67.5° for 384 counts into error counter). Actual display scale determined by spacecraft FDAI SCALE switch (which is <u>not</u> sensed by software): for ERR scale at "5", full scale is 5° for B1 rev. scaling; for "50/15", full scale is 50° in roll for B3 scaling (12.5° for B1 scaling), and 15° in pitch/yaw (B1 scaling).
0050_8(LM) 0051_8(LM) 0052_8(LM)		See previous page for items common to CM and LM uses. Error counter output also used for LM P, Q, and R axes (yaw, pitch, and roll respectively) attitude "error" needles on FDAI: note that in LM vehicle rotation about thrust vector is "yaw" (in CSM it is "roll"). Software controls whether cells loaded with attitude error information (scaled B0 in units of 180°) or vehicle rate information (scaled B0 in units of 450°/sec). For attitude error display, needles pin at about 5 1/16°; for rate display, they pin at about 1 17/64°. These figures correspond to 46 least increments in the error counters.
0053_8(CM) 0054_8(CM)	CDUTCMD CDUSCMD	Cells loaded with values to be transmitted to optics CDU error counters. Information gated out of cells if bits 12-11 respectively of channel 14 are set, and error counters loaded if bit 2 of channel 12 is set (counters set 0 if bit 2 = 0). Used during optics position drive operation to drive the optics trunnion (scale B-1 rev.) and shaft (scale B1 rev.) respectively. Drive of optics inhibited if bit 11 of channel 12 is set 1. May also be used for rate drive of optics on subsequent flights (see mission documentation). The cells are also used for control of SPS engine (see next page).

Address	Symbol	Function
0053_8(CM) 0054_8(CM)	TVCYAW TVCPITCH	Cells loaded with values to be transmitted to "optics" error counters for use in controlling the position of the SPS (Service Propulsion System) engine gimbals. Same cells used to drive optics (see previous page), but the automatic optics drive can be disabled by setting bit 11 of channel 12 (although optics could still drift unless mode specified to be optics zeroing). Information is gated out of cells if bits 12-11 respectively of channel 14 are set, and error counters loaded if bit 2 of channel 12 is set (counters set 0 if bit 2 = 0). Output of error counters, converted to dc, is sent to SPS engine yaw and pitch servos if bit 8 of channel 12 is set (which also inhibits the position feedback to the error counters used when commanding optics positioning). The error counter saturates at 600_8 (384 counts), or about 9.1°, and is loaded at a 3200 pps rate. One count for SPS driving corresponds to 85.41 arc seconds (or 0.023725°), giving about 42.14963 pulses/degree or 388.7104° (about 1.07975111 revolutions) per 2^{14} pulses.
0053_8(LM) 0054_8(LM)	CDUTCMD CDUSCMD	Cells loaded with values to be transmitted to rendezvous radar error counters for use in controlling the position of the rendezvous radar antenna when its position is being controlled by software (when antenna sufficiently close to proper direction, the radar system controls its position provided bit 14 of channel 12 is 1). Information gated out of cells if bits 12-11 respectively of channel 14 are set, and error counters loaded if bit 2 of channel 12 is set (error counter reset to 0 if bit 2 of channel 12 is reset to 0). Cells used to control radar trunnion and shaft drives respectively, with a saturated error counter (384 pulses) corresponding to a drive rate of about 10°/second: position error signal corrected by program for desired dynamic response.

Address	Symbol	Function
0053_8(LM) (cont) 0054_8(LM)		If bit 8 of channel 12 is set, the error counter outputs (converted to dc) are used to provide lateral and forward velocity information respectively to an analog display, scaled about 0.5571 fps/bit.
0055_8(CM)		Not used.
0055_8(LM)	THRUST	Cell used to provide engine throttle commands for the descent engine, giving output pulses at a 3200 pps rate when bit 4 of channel 14 is set, of a polarity determined by the polarity of 0055_8. Cell decremented as pulses are sent, and bit 4 of channel 14 is reset 0 when cell contents have been reduced to 0. Actual throttle command to engine is sum of counter contents (counter incremented by outputs of cell 0055_8) and the position of manual throttle. The counter driven by the pulses controlled by cell 0055_8 is reset 0 when the descent engine is disarmed, and has a saturation level greater than the number of pulses required for full throttle setting. One pulse corresponds to roughly 2.8 pounds of thrust (see mission documentation for specific value).
0056_8		Not used. Originally intended to provide entry monitoring system velocity information for CM (tag EMSD) and a LM monitor function (LEMONM). Cell gives output pulses at a 3200 pps rate when bit 5 of channel 14 is set 1, of a polarity determined by the polarity of cell 0056_8. The cell is decremented in magnitude as pulses are sent, and bit 5 of channel 14 is reset 0 when the cell contents have been reduced to 0.

Address	Symbol	Function
0057_8(CM)	LOCALARM	Cell used for storage of alarm source information (using cell as a normal erasable memory cell, rather than employing the counter feature described for LM). See mission documentation for details.
0057_8(LM)	OUTLINK	Not used. Originally intended for use to provide a "crosslink" capability for serial binary data to cell 0045_8 of another computer (if bit 5 of channel 13 of that computer is set). After loading 0057_8 with the proper data, setting of bit 1 of channel 14 to 1 causes the data to be sent at a 3200 pps rate: first a binary 1 is sent, then the 15 bits in cell 0057_8 (most significant bit first). Bit 1 of channel 14 is reset when the first binary 1 (satisfying the format requirement for program interrupt #7) is generated, which is (1/6.4) ms after the first 200 pps signal following the setting of bit 1 of channel 14.
0060_8(CM)	BANKALRM	Cell used for storage of alarm source information (using cell as a normal erasable memory cell, rather than employing the counter feature described for LM). See mission documentation for details.
0060_8(LM)	ALTM	Cell used to provide altitude and altitude rate information to analog display. Data provided in serial binary form, with bit 2 of channel 14 set to 1 if altitude rate information is supplied (scaled at 0.5 fps/bit), and reset to 0 if altitude information is supplied (scaled about 2.345 ft/bit). After loading 0060_8 with the proper data, setting of bit 3 of channel 14 causes the data to be sent at a 3200 pps rate: first a binary 1 is sent, then the 15 bits of cell 0060_8 (most significant bit first). Bit 3 reset with same timing as bit 1 for cell 0057_8.

Address	Symbol	Function
0067_8	NEWJOB	Cell used in control of job sequencing (see Section VIIB). Each time it is sampled, a flip-flop set by a signal with a 1.28-second period is reset. If the flip-flop is set when another 1.28-second period signal (0.64 seconds out of phase with the first) occurs, a "night watchman" fault (see Section IIH), causing a hardware restart, is produced. Hence maximum allowable interval between samples ranges from 0.64 to 1.92 second.

IIE Input/Output Channels

Binary-level inputs and outputs from the computer, including control signals for portions of the computer hardware, are handed through interfacing hardware called "channels". Analog-type pulse input/output, on the other hand, is mechanized through the special purpose erasable memory cells with their associated counter interrupts, as discussed in Section IID and IIH. One of these special purpose cells (0047_8), for example, is used to contain the magnitude of the required gyro torquing pulse output, while appropriate bits in one of the output channels (bits 9-7 of channel 14) specify not only the sign of the required pulses, but also the gyro axis to which they are to be applied.

Of the twenty channels which are defined, three different types may be identified:

1. Ten output channels, numbered 05, 06, 07, 10, 11, 12, 13, 14, 34, and 35. The first 8 can be both loaded and sensed under program control, but channels 34 and 35 can be loaded only (they are used to provide telemetry output from the computer).

2. Eight input channels, numbered 03, 04, 15, 16, 30, 31, 32, and 33. All eight can be sensed under program control. Bits 15-11 of channel 33 are flip-flop inputs, which can be set to a binary 1 (logic 0) under program control by a "loading" type command (instructions WAND, WOR, or WRITE in Section IVC).

3. Two computer registers, numbered 01 and 02, corresponding to the L register and Q register respectively of Section IIC. These registers are included as "channels" to permit use of the bit manipulation capabilities of the seven channel instructions (see Section IVC) in the computer order code.

Channels are conventionally referenced by their octal channel number (the number appearing in the address portion of the appropriate channel instruction). To permit references to each channel to be flagged by the assembly program (see Section III), however, the program coding generally uses a symbolic reference tag for each channel, as shown in the "mnemonic" column on the following pages.

In order to sense and/or load the input/output channels, only the seven extended-order (see Section IVC) channel instructions may be used: RAND, READ, ROR, RXOR, WAND, WOR, and WRITE. These instructions cannot be used with other computer registers (except, of course, L and Q which are also "defined" to be channels).

The bit assignments given on the following pages are those of the quantities associated with the hardware. Several (such as CM/SM separation) may not be actually sensed by the program for computation control (as contrasted with e.g. telemetry) purposes, and therefore reliable equation information should be consulted to determine which bits serve a purpose in a given computer program configuration.

As part of a hardware restart (signal produced by hardware, see page IIH-9), all output channel bits (except those of channel 07) are reset zero. Consequently, the software must restore the appropriate bits (such as IMU control and engine-on information) as necessary. In addition, the channel loading commands (WAND, WOR, and WRITE) zero all bits of the channel briefly (about $\frac{1}{4}$ microsecond), and some spacecraft hardware is sensitive to this brief zeroing, such as the radar systems in the LM, so special software techniques are required to avoid loading the channel (#13) while shift pulses are being generated (otherwise, a single shift pulse could appear as two pulses).

Channel	Mnemonic	Bits	Function
01	LCHAN	15-1	Computer L register (address 0001_8 in Section IIC).
02	QCHAN	15-1	Computer Q register (address 0002_8 in Section IIC).
03	HISCALAR	14-1	Most significant 14 bits from 33-stage binary counter driven by 102.4 kc signal from computer oscillator (see Section IIA). Counter keeps running when computer placed in low-power (standby) mode of operation, and hence data in counter can be used to restore the proper value of the computer clock (cells 0024_8 - 0025_8 in Section IID) after a period of standby operation. Scale factor for channel 03 data is B23 in units of centi-seconds, so least significant bit is 5.12 seconds and channel information overflows every 23 hours, 18 minutes, 6.08 seconds (about 23.3 hours).
04	LOSCALAR	14-1	Next most significant 14 bits from 33-stage binary counter associated with channel 03. Scale factor for channel 04 data is B9 in units of centi-seconds, so least significant bit is (1/3200) second and channel information overflows (and propagates to channel 03) every 5.12 seconds. Time information in channel 04 is 0.005 seconds out of phase (i.e. $\frac{1}{2}$ centi-second) with cell 0025_8 in Section IID, so that the least significant 5 bits of channel 04 are 20_8 during the first (1/3200) second interval after cell 0025_8 (TIME1) has been incremented by +1. TIME1, in turn, is 5 ms out of phase with the 100 pps signal used for control of the radar (see cell 0046_8 in Section IID).

Channel	Mnemonic	Bits	Function
05	CHAN5 PYJETS	8-1	RCS (Reaction Control System) jet controls: see next two pages.
06	CHAN6 ROLLJETS	8-1	RCS (Reaction Control System) jet controls: see next two pages.
07	SUPERBNK	7-5	Superbank (sometimes called F EXT) register, used to select the appropriate fixed memory bank for FBANK values of 30_8 or more. Channel not reset if get a hardware restart. See Section IIB.
10	OUTO	15-1	Register used to transmit latching-relay driving information to the display system (see Section IIJ). Bits 15-12 are set to the row number (01_8 - 14_8) of the relays to be changed, and bits 11-1 contain the required settings for the relays in the selected row. Since the relays are bi-stable devices, the OUTO setting need be left for only 0.02 second. After a period of 0.02 second in which the channel bits are all reset, a setting for another row of relays can be specified (hence to change all 11 rows that control the DSKY digit/sign displays requires 0.44 seconds). Row 17_8 has been used for mission programmer functions on unmanned flights (e.g. LM-1), when the OUTO setting was retained for 0.03 seconds.
11	DSALMOUT		Register whose individual bits are used for engine on/off control and to drive individual indicators of the display system (see Section IIJ).
		15	Not assigned.
		14(CM)	Not assigned.

Channel 5 Channel 6

Service Module RCS Jets

Bit	Jet	Quad	Reaction	Bit	Jet	Quad	Reaction
8	6/B4	B	+X -Yaw	8	14/C2	C	+Y -Roll
7	7/B3	B	-X +Yaw	7	15/C1	C	-Y +Roll
6	8/D4	D	-X -Yaw	6	16/A2	A	-Y -Roll
5	5/D3	D	+X +Yaw	5	13/A1	A	+Y +Roll
4	2/A4	A	+X -Pitch	4	10/D2	D	+Z -Roll
3	3/A3	A	-X +Pitch	3	11/D1	D	-Z +Roll
2	4/C4	C	-X -Pitch	2	12/B2	B	-Z -Roll
1	1/C3	C	+X +Pitch	1	9/B1	B	+Z +Roll

Command Module RCS Jets

Bit	Jet	System	Control	Bit	Jet	System	Control
8	6/26	B	-Yaw				
7	7/25	B	+Yaw				
6	8/16	A	-Yaw				
5	5/15	A	+Yaw				
4	2/14	A	-Pitch	4	10/22	B	-Roll
3	3/23	B	+Pitch	3	11/21	B	+Roll
2	4/24	B	-Pitch	2	12/12	A	-Roll
1	1/13	A	+Pitch	1	9/11	A	+Roll

"Reaction" means direction of spacecraft motion when the jet fires.
"Control" means direction of spacecraft motion used in software for that jet.
+X direction is same direction as SPS engine thrust (roll axis positive about this axis in right-hand rule sense).
Quads in order B, C, D, A starting at the +Y (pitch) axis and going clockwise (looking forward, i.e. along +X). Control axes offset from spacecraft axes by a rotation of -7.25° (measured from spacecraft to control axes about +X axis).
See spacecraft hardware documentation for locations of individual jets.

Lunar Module RCS Jets

Bit	Jet	Cluster	System	Translation	Rotation	Failure Bit (ch. 32)
			Channel 5			
8	14	1 D	B	+X	+Q, +R; +U	5
7	13	1 U	A	-X	-Q, -R; -U	6
6	10	2 D	A	+X	-Q, +R; +V	8
5	9	2 U	B	-X	+Q, -R; -V	7
4	6	3 D	B	+X	-Q, -R; -U	4
3	5	3 U	A	-X	+Q, +R; +U	2
2	2	4 D	A	+X	+Q, -R; -V	1
1	1	4 U	B	-X	-Q, +R; +V	3
			Channel 6			
8	16	1 S	B	+Y	-P	5
7	4	4 S	A	-Y	+P	1
6	8	3 S	A	-Y	-P	2
5	12	2 S	B	+Y	+P	7
4	11	2 F	A	+Z	-P	8
3	15	1 F	A	-Z	+P	6
2	3	4 F	B	-Z	-P	3
1	7	3 F	B	+Z	+P	4

"Translation" and "Rotation" mean direction of spacecraft motion when jet fires.
+X is through the upper docking tunnel (+P rotations about this axis in right-hand rule sense, for "yaw"), i.e. in direction of APS/DPS thrust.
+Z is through the forward tunnel (+R rotations about this axis in "roll").
+Y completes right-hand set (+Q rotations about this axis in "pitch").
In the software, rotation control for channel 5 is oriented about the U, V axis system, where +U is through cluster 4, between +Z and +Y, and +V is through cluster 1, between +Z and -Y. The actual software outputs are about (non-orthogonal) U', V' axes (defined so as to avoid cross-coupling effects, and restricted by software to be reasonably close (e.g. 15°) to U, V axes).
Clusters are numbered clockwise starting at +Z (looking along +X), with jets pointed up (U), down (D), fore-aft (F), or to the side (S).
See spacecraft hardware documentation for locations of individual jets.

Channel	Mnemonic	Bits	Function
11(cont)		14(LM)	Engine Off signal to engine sequencer for descent and ascent engine. If bits 14-13 = 00_2, the engine remains in its previous state (on or off), but if the vehicle stages with the bits equal to 00_2, the ascent engine would <u>not</u> light. If the descent engine sees the bits equal to 11_2, it likewise remains in previous state; the ascent engine, however, turns on. The 00_2 condition, however, should be avoided when the engine is armed.
		13(CM)	SPS (Service Propulsion System) engine turn-on signal (set 0 to turn engine off).
		13(LM)	Engine On signal to engine sequencer for descent and ascent engine. See bit 14 of channel 11.
		12	Not assigned.
		11	Not assigned. Used in LM-1 program for telemetry purposes as a status indicator of program performance.
		10	Caution Reset signal. It resets the flip-flop holding the Restart light (see Section IIJ) of the display system in the energized state.
		9	Test Connector Outbit (Connector A52 pin 813). Can be used as an indicator for hybrid simulator test purposes that Average-G (two-second navigation cycle using accelerometer outputs) is running, if suitably set by software.
		8	Not assigned.

Channel	Mnemonic	Bits	Function
11(cont)		7	Bit that energizes the Operator Error (see Section IIJ) of the display system. It is set to 1 if an improper operator entry to the keyboard or uplink detected by the software, and it causes the Operator Error light to be flashed.
		6	Bit that energizes the Flash (see Section IIJ) of the display system, that causes the verb and noun register indicators to be flashed on and off (not noticeable unless they are non-blank, of course). Used by the software to signify that an operator response or action is needed.
		5	Bit that energizes the Key Release (see Section IIJ) of the display system. It is set to 1 if the software of the internal display system users is inhibited from using the display system because of operator use. The bit causes the Key Release light to be flashed.
		4	Bit that energizes the Temperature Caution light (see Section IIJ) of the display system. This light is also connected to bit 15 of channel 30.
		3	Bit that energizes the Uplink Activity light (see Section IIJ) of the display system, set by software when program interrupt #7 (see Section IIH) is processed, and reset likewise by the software (at termination of uplink sequence, etc.). Can also be used for informing crew of other situations when uplink information not being received (such as, for CM, the need for an attitude maneuver): see equation documentation.

Channel	Mnemonic	Bits	Function
11(cont)		2	Bit that energizes the Computer Activity light (see Section IIJ) of the display system. It is set by the software if the executive system (Section VIIB) has an active job being performed (i.e. something besides the dummy job routine). The bit remains at its previous value when a task (such as the one initiated by program interrupt #8 for telemetry) is done.
		1	Bit that energizes the ISS (inertial subsystem) Warning light, a red light on the caution and warning panel of the spacecraft, if IMU, IMU CDU, or PIPA fail indications are significant in terms of mission phase (as determined by the software). Bit can be used on unmanned flights to generate a PGNCS (primary guidance, navigation, and control system) failure indication.
12	CHAN12		Register whose individual bits are used to drive miscellaneous navigation and spacecraft hardware.
		15	ISS turn-on delay complete. Bit set by software nominally 90 seconds after receipt of ISS power-on signal, bit 14 of channel 30, and reset to zero nominally 10.24 seconds later. Used to delay the closing of the stabilization loops of the IMU gimbals (to permit the gyro wheels to reach operating speed) and also to delay torquing of the accelerometers. Bit energizes a latching relay which energizes the ISS turn-on relay, removing the signal from bit 14 of channel 30. Same effect achieved by IMU Cage button, bit 11 of channel 30.
		14(CM)	S4B Cutoff command. Command provided via a relay in the DSKY to the Saturn Instrumentation Unit. The relay contact closure is not functional unless CMC control of Saturn is enabled (and hence software may close it unconditionally, see equation documentation)

Channel	Mnemonic	Bits	Function
12(cont)		14(LM)	Enable rendezvous radar lock-on. Command provided via a relay in the DSKY to enable automatic angle tracking by the rendezvous radar when software determines that antenna position (from cells 0035_8 - 0036_8) is sufficiently close to the predicted position of the other vehicle.
		13(CM)	S4B Injection Sequence start. Command provided via a relay in the DSKY to Saturn Instrumentation Unit if backup generation of the signal (which starts the Saturn "Time Base 6") is required.
		13(LM)	Landing radar position command. Command provided via a relay in the DSKY to change landing radar antenna position from position #1 (descent, see bit 6 of channel 33) to position #2 (hover, see bit 7 of channel 33). For the LGC command to have an effect, the landing antenna switch must be in "AUTO" (its other positions are "DESC" and "HOVER").
		12(CM)	Not assigned.
		12(LM)	Descent engine negative roll gimbal trim. Nominal trim rate about 0.2°/sec, and magnitude of trim determined by length of time that signal left at a binary 1. DPS engine trim gimbal actuator driven in such a way as to be rotated in a positive right hand sense about the positive roll (+Z) axis, for -R acceleration.
		11(CM)	Disengage optics DAC (digital to analog converter). Can be used to disconnect optics CDU DAC from shaft and trunnion motor drive amplifiers, if zeroing of optics desired by software with optics in computer mode. Can also be set by software (see equation documentation) prior to use of the optics DAC for SPS gimbal drive purposes (see cells 0053_8-0054_8 in Section IID).

Channel	Mnemonic	Bits	Function
12(cont)		11(LM)	Descent engine positive roll gimbal trim. Nominal trim rate about 0.2°/sec, and magnitude of trim determined by length of time that signal left at a binary 1. DPS engine trim gimbal actuator driven in such a way as to be rotated in a negative right hand sense about the positive roll (+Z) axis, for +R acceleration.
		10(CM)	Zero optics. Function performed also by setting spacecraft "Optics Zero" switch to "ZERO".
		10(LM)	Descent engine negative pitch gimbal trim. Nominal trim rate about 0.2°/sec, and magnitude of trim determined by length of time that signal left at a binary 1. DPS engine trim gimbal actuator driven in such a way as to be rotated in a positive right hand sense about the positive pitch (+Y) axis, for -Q acceleration.
		9(CM)	S4B takeover enable. Connects the dc output of the IMU CDU error counters (loaded from cells 0050_8 - 0052_8, see Section IID) to the Saturn Instrumentation Unit. Used to permit attitude control of Saturn through the guidance computer (which does not necessarily mean the engine sequencing and on/off control of bits 14-13 of this channel, of course).
		9(LM)	Descent engine positive pitch gimbal trim. Nominal trim rate about 0.2°/sec, and magnitude of trim determined by length of time that signal left at a binary 1. DPS engine trim gimbal actuator driven in such a way as to be rotated in a negative right hand sense about the positive pitch (+Y) axis, for +Q acceleration.

Channel	Mnemonic	Bits	Function
12(cont)		8(CM)	TVC (thrust vector control) enable. Connects the dc output of the optics CDU error counters (loaded from cells 0053_8 - 0054_8, see Section IID) to the SPS (service propulsion system) gimbal servo amplifiers.
		8(LM)	Display inertial data. Connects the dc output of the rendezvous radar CDU error counters (loaded from cells 0053_8 - 0054_8, see Section IID) to a spacecraft analog display to provide lateral and forward velocity information. Bit set by software (provided proper computations are being performed) when bit 6 of channel 30 indicates that such a display is desired.
		7	Not assigned. Originally intended for use as engine on command (done now by bits 14-13 of channel 11).
		6	Enable IMU CDU error counters. The error counters are reset to 0 if this bit is 0, and are loaded from cells 0050_8 - 0052_8 (see Section IID).
		5	Zero IMU CDU's. Can be used to force the CDU hardware to a zero value, whereupon zeroing of cells 0032_8 - 0034_8 and then reset of this bit will permit these cells to reflect the IMU gimbal angle information. This bit alone does not cause movement of the stable member: this is done at IMU turn-on or by an IMU cage command (bit 11 of channel 30), or by coarse aligning.
		4	Enable coarse align of IMU. Connects IMU CDU error counters to cause IMU coarse alignment (angle change information loaded into cells 0050_8 - 0052_8, and bit 6 of this channel must be 1).

Channel	Mnemonic	Bits	Function
12(cont)		3	Not used. In CM, assigned to enable star tracker (no longer in vehicle), and in LM assigned to indicate low scale for horizontal velocity output.
		2(CM)	Enable optics CDU error counters. The error counters are reset to 0 if this bit is 0, and are loaded from cells 0053_8 - 0054_8 (used also for yaw and pitch SPS control).
		2(LM)	Enable rendezvous radar CDU error counters. The error counters are reset to 0 if this bit is 0, and are loaded from cells 0053_8 - 0054_8.
		1(CM)	Zero optics CDU's. Can be used to force the optics CDU hardware to a zero value, whereupon setting of cells 0035_8 - 0036_8 and then reset of this bit will permit these cells to reflect the optics angle information.
		1(LM)	Zero rendezvous radar CDU's. Similar function to bit 1(CM), but for rendezvous radar rather than optics. To avoid an excessive number of counter interrupts which can occur if RR mode not set to LGC, software sets this bit 1 if the mode not LGC.
13	CHAN13		Register whose bits are used to control miscellaneous navigation system functions (some bits sensitive to write commands, see page IIE-2).
		15	Bit set to 1 to permit cell 0031_8 (TIME6) to be decremented by 1 each 0.000625 second (i.e. 1600 times a second). When cell has been reduced to -0, the next DINC pulse causes bit to be reset to 0 and program interrupt #1 to be produced (see Section IIH).

Channel	Mnemonic	Bits	Function
13(cont)		14	Bit set to 1 to cause trap 32 to be reset. This input trap circuit is set when program interrupt #10 is generated in response to a signal fed to bits 10-1 of channel 32 (see Section IIH).
		13	Bit set to 1 to cause trap 31B to be reset. This input trap circuit is set when program interrupt #10 is generated in response to a signal fed to bits 12-7 of channel 31 (see Section IIH).
		12	Bit set to 1 to cause trap 31A to be reset. This input trap circuit is set when program interrupt #10 is generated in response to a signal fed to bits 6-1 of channel 31 (see Section IIH).
		11	Bit set to 1 to permit relay in computer power supply to put computer in Standby (low-power) operation when the PRO key (formerly the "Standby" key) on the DSKY is pressed. The bit is set by the software when preparations for standby operation completed, including retention of the computer clock, and it is reset by the software after clock restored.
		10	Bit set to 1 to test the DSKY lights and relays not otherwise accessible to the software. It energizes the Restart light, the Standby light, and the Computer Warning light (via a "warning filter").
		9(CM)	Not assigned.
		9(LM)	Bit set to 1 to initiate readout of analog-to-digital converters associated with the displacement of the rotational hand controller when used as a rate commanding device. See cells 0042_8 - 0044_8 in Section IID.

Channel	Mnemonic	Bits	Function
13(cont)		8	Bit set to 1 to enable inputs to cells 0042_8 - 0044_8 (see Section IID) for LM rotational hand controller rate-command input and for (unused) CM BMAG input.
		7	Bit used as the word order code bit (first bit in the 40-bit downlink sequence sent from computer for digital data) for telemetry, in order to flag certain words in the list.
		6	Bit set to 1 to block all inputs to INLINK (cell 0045_8, see Section IID).
		5	Bit set to 1 to connect (unused) crosslink input instead of uplink receiver to cell 0045_8 (see Section IID).
		4	Bit set to 1 to initiate transmission of radar information to computer. Bit is reset to 0 when program interrupt #9 is generated (see Section IID for timing sequence associated with loading of cell 0046_8).
		3-1(CM)	Bits set to 001_2 in order to specify that range information from VHF range system is to be provided to computer in cell 0046_8 (see Section IID) after bit 4 of channel 13 is set 1.
		3-1(LM)	Bits assigned control functions for sampling of rendezvous radar (RR) if bit 3 is 0 and of landing radar (LR) if bit 3 is 1, to establish information fed to cell 0046_8 when bit 4 of channel 13 is set. For RR, bits 2-1 are set to 01_2 for range data and 10_2 for range rate data. For LR, bits 2-1 are set to 00_2, 01_2, and 10_2 for x-z velocities respectively, and to 11_2 for range (altitude) information.

Channel	Mnemonic	Bits	Function
14	CHAN14		Register whose bits are used to control the computer counter cells (CDU, gyro, and spacecraft functions) described in Section IID.
		15	Bit set to 1 to cause output pulses (at a 3200 pps rate) to be generated from CDUXCMD, cell 0050_8. When cell counted down to 0, bit is reset (at the next DINC, see Section IIH), thereby stopping the pulses. Error counter is loaded if bit 6 of channel 12 is 1.
		14	Bit set to 1 to cause output pulses to be generated from CDUYCMD, cell 0051_8: see bit 15 of channel 14.
		13	Bit set to 1 to cause output pulses to be generated from CDUZCMD, cell 0052_8: see bit 15 of channel 14.
		12	Bit set to 1 to cause output pulses (at a 3200 pps rate) to be generated from cell 0053_8 (CDUTCMD or TVCYAW). When the cell has been counted down to 0, bit is reset (at the next DINC, see Section IIH), thereby stopping the pulses. Error counter is loaded if bit 2 of channel 12 is 1.
		11	Bit set to 1 to cause output pulses to be generated from cell 0054_8 (CDUSCMD or TVCPITCH): see bit 12 of channel 14.
		10	Bit set to 1 to specify "gyro activity": it causes the pulse train whose magnitude is in cell 0047_8 (GYROCMD) to be sent with polarity and destination specified by bits 9-7 of this channel, if bit 6 of this channel is 1. Bit reset 0 after proper pulses sent (cell reduced to 0 and the next DINC).

Channel	Mnemonic	Bits	Function
14(cont)		9	Bit set to 1 to specify a negative-polarity gyro torquing output from GYROCMD (cell 0047_8). Other pulse-type outputs from the computer have the polarity indicated by the polarity of the information in the counter cell itself.
		8-7	Bits used to specify the axis for gyro compensation information from GYROCMD. Conventional output sequence is inner (Y), middle (Z), and outer (X) for torquing, with the following bit configurations:

Bits 8-7	Torque Output
00_2	None
01_2	X-axis Gyro
10_2	Y-axis Gyro
11_2	Z-axis Gyro

6	Bit set to 1 to enable the power supply that produces the torquing pulses used to torque gyros (in a manner determined by bits 7-10 of this channel and cell 0047_8). Software generally leaves bit at 1 after the first gyro torquing is performed (reset to 0 when certain initialization functions performed).
5	Not used. Bit set to 1 to initiate commands from data in cell 0056_8 (see Section IID).
4(CM)	Not used (initiates commands from cell 0055_8).
4(LM)	Bit set to 1 to cause output pulses to be generated from cell 0055_8 (THRUST) for use in controlling the position of the descent engine throttle (see Section IID). When cell has been reduced to -0, the next DINC pulse causes this bit to be reset to 0.

Channel	Mnemonic	Bits	Function
14(cont)		3(CM)	Not used (initiates commands from cell 0060_8).
		3(LM)	Bit set to 1 to initiate shifting of data from cell 0060_8 (ALTM) to spacecraft indicator for altitude (bit 2 of this channel = 0) or altitude rate (bit 2 of this channel = 1) information. Bit reset to 0 just after start of data shift (see Section IID).
		2(CM)	Not used.
		2(LM)	Bit set to 1 to indicate that altitude rate information is being shifted from cell 0060_8; if is 0, altitude information is being shifted.
		1	Not used. Bit set to 1 to initiate shifting of data from cell 0057_8 (see Section IID).
15	MNKEYIN	5-1	Key code input from keyboard of DSKY (see Section IIJ), sensed by the program when program interrupt #5 (see Section IIH) is acted upon. For the CM (which has two DSKY's), channel 15 is associated with the DSKY located on the main display console.
16	NAVKEYIN	7-1	Optics mark information and lower equipment bay (or "navigation panel") DSKY inputs for CM; optics mark information and rate-of-descent control for LM. Sensed by the program when program interrupt #6 (see Section IIH) is acted upon.
		7(CM)	Optics mark reject signal if 1.
		7(LM)	Bit set to 1 if an increase in the rate of descent is desired by crew (i.e. a lower thrust). Generated by moving rate-of-descent switch in the -X direction (i.e. towards engine). Effect of switch and scaling (e.g. 1 fps per "click") controlled by software: see equation documentation.

Channel	Mnemonic	Bits	Function
16(cont)		6(CM)	Optics mark signal if 1.
		6(LM)	Bit set to 1 if a decrease in the rate of descent is desired by crew (i.e. a higher thrust). Generated by moving rate-of-descent switch in +X direction (i.e. away from engine). Processed by software similarly to bit 7(LM).
		5-1(CM)	Lower equipment bay (or "navigation panel") DSKY key code input (see Section IIJ).
		5(LM)	Optics mark reject signal if 1.
		4(LM)	Optics Y-axis mark signal for AOT (alignment optical telescope) if 1.
		3(LM)	Optics X-axis mark signal for AOT if 1.
		2-1(LM)	Not assigned.
17-27			Channels not assigned. Some tentatively allocated for control of additional memory capacity that has been considered for CM (an Auxiliary Core Memory addressed with SUPERBNK settings of 5 and 6).
30	CHAN30		Register whose bits are used to supply miscellaneous input information for the program. All bits are inverted as sensed by the program, so that a value of binary 0 means that the indicated signal is present.
		15	Bit sensed as 0 if the stable member temperature is within its design limits. Software sets bit 4 of channel 11 to 1 if this bit becomes 1. The light controlled by bit 4 of channel 11 is also connected directly to this bit 15 of channel 30.

Channel	Mnemonic	Bits	Function
30(cont)		14	Bit sensed as 0 if the inertial subsystem has been turned on or commanded to be turned on. Bit 15 of channel 12 is set to 1 by the software about 90 seconds after this bit sensed as 0 (if checks passed), resetting this bit to 1.
		13	Bit sensed as 0 if an IMU fail indication has been generated within the IMU hardware (due e.g. to excessive servo errors or degradation of 3200 cps or 800 cps supply). Software controls setting of bit 1 (ISS Warning) of channel 11 based on this input bit and the IMU mode.
		12	Bit sensed as 0 if an IMU CDU fail indication has been generated within the IMU CDU hardware (due e.g. to excessive errors or low voltage). Software controls setting of bit 1 (ISS Warning) of channel 11 based on this input bit and the IMU mode.
		11	Bit sensed as 0 if the "IMU Cage" switch is set by crew to drive all the IMU gimbal angles to zero. The command is also sent directly to the IMU control hardware, and can be used as an emergency technique for recovering a tumbling IMU. The preferred method, however, is to remove power.
		10(CM)	Bit sensed as 0 if the "Launch Vehicle Guidance" switch is set by crew to the "CMC" (as opposed to "IU") position, indicating that control of the Saturn vehicle has been given to the computer.
		10(LM)	Bit sensed as 0 if the "Guidance Control" switch is set by crew to the "PGNS" (as opposed to "AGS", for Abort Guidance Section) position, indicating that control of the vehicle has been given to the computer.

Channel	Mnemonic	Bits	Function
30(cont)		9	Bit sensed as 0 if the IMU is turned on and operating with no malfunctions.
		8	Not assigned.
		7(CM)	Bit sensed as 0 if an optics CDU fail indication produced (due e.g. to excessive errors or low voltage). Software controls setting of bit 8 of row 14_8 (TRACKER light, see Section IIJ) based on this input bit and optics mode.
		7(LM)	Bit sensed as 0 if a rendezvous radar CDU fail indication produced (due e.g. to excessive errors or low voltage). Software controls setting of bit 8 of row 14_8 (TRACKER light, see Section IIJ) based on this input bit and radar selection.
		6(CM)	Bit sensed as 0 if GRR (guidance reference release) signal generated by S4B Instrumentation Unit, indicating that this action has occurred or has been commanded to occur. Software uses bit 5 rather than this bit to halt pre-launch computations (with backup of a DSKY verb).
		6(LM)	Bit sensed as 0 if a display of inertial data from the computer is desired by the crew, by setting the "Mode Select" switch to the "PGNS" position (as opposed to "LDG RADAR" or "AGS"). When the appropriate information has been loaded by the software, bit 8 of channel 12 is set to 1.
		5(CM)	Bit sensed as 0 if liftoff signal generated by S4B Instrumentation Unit, indicating that lift-off has taken place. Software uses this bit to halt pre-launch computations (with backup of a DSKY verb).

Channel	Mnemonic	Bits	Function
30(cont)		5(LM)	Bit sensed as 0 if computer given control of descent engine throttle by the crew, by setting the "Throttle Control" switch to "AUTO" (as opposed to "MAN") position. In the "AUTO" position, computer throttle commands from cell 0055_8 (THRUST, see Section IID) are summed with the manual throttle commands; in the "MAN" position, with bit = 1, the computer commands no longer control the throttle setting.
		4(CM)	Bit sensed as 0 if the S4B separation (or abort) signal is received. Software does not use the bit.
		4(LM)	Bit sensed as 0 if the crew has produced an "Abort Stage" command (with a spacecraft pushbutton switch), indicating that an abort using the ascent engine is required (spacecraft hardware causes descent engine to be staged).
		3	Bit sensed as 0 when preparations for use of the appropriate engine have been completed. Software ~~does not use the bit~~. For CM, it indicates that a "ΔV Thrust" switch has been set to "NORMAL"; for LM, that the "Engine Armed" switch has been set to "ASC" or "DSC".
		2(CM)	Bit sensed as 0 when CM/SM separation has taken place. The bit is generated by the CM/SM reaction jet control transfer unit, but is not used by the software.
		2(LM)	Bit sensed as 0 to indicate that the descent stage is attached ("Stage Verify"): a value of 0 means descent stage and a value of 1 means ascent stage only. Software does not use the bit.

Channel	Mnemonic	Bits	Function
30(cont)		1(CM)	Bit sensed as 0 if "Ullage Thrust Present" signal received from S4B Instrumentation Unit. Bit not sensed by software.
		1(LM)	Bit sensed as 0 if the crew has produced an "Abort" command (with a spacecraft pushbutton switch), indicating than an abort using the descent engine is required. When the descent engine propellants are nearly expended, the crew could then initiate the "Abort Stage" command (bit 4 of this channel).
31	CHAN31		Register whose bits are associated with the attitude controller, translational controller, and spacecraft attitude control. All bits are inverted as sensed by the program, so that a value of binary 0 means that the indicated signal is present.
		15(CM)	Bit sensed as 0 if the computer is in control of the spacecraft. The bit becomes a binary 1 if the IMU is turned off, if the THC (translation hand controller) is twisted in the clockwise direction, or if the "Spacecraft Control" switch is placed in the "SCS" (spacecraft control system) as opposed to the "CMC" position.
		15(LM)	Bit sensed as 0 if ACA (attitude controller assembly) is out of detent. Control also referred to as RHC (rotational hand controller), see LM cells 0042_8-0044_8 in Section IID.
		14(CM)	Bit sensed as 0 if the three-position "CMC Mode" switch is set by crew to "FREE". Software fires RCS jets only in response to controller inputs (as with other manual inputs, bit ignored by software unless RCS DAP is running).

Channel	Mnemonic	Bits	Function
31(cont)		14(LM)	Bit sensed as 0 if the "PGNS Mode Control" switch is set to "AUTO", indicating that the software has complete authority for control of spacecraft (if bit 10 of channel 30 = 0).
		13(CM)	Bit sensed as 0 if the three-position "CMC Mode" switch is set by crew to "HOLD", indicating that attitude hold is desired. If bits 14-13 = 11_2, this means that the third position of the switch, "AUTO", is selected (software has complete authority for control of spacecraft if bits 15-13 = 011_2).
		13(LM)	Bit sensed as 0 if the "PGNS Mode Control" switch is set to "ATT HOLD", indicating to the software that attitude hold is desired. If the switch is set to "OFF", then bits 14-13 = 11_2.
		12	Bit sensed as 0 if translation in the -Z direction commanded by THC (translation hand controller). In LM is "TTCA" (thrust/translation controller assembly).
		11	Bit sensed as 0 if translation in +Z direction commanded by THC.
		10	Bit sensed as 0 if translation in -Y direction commanded by THC.
		9	Bit sensed as 0 if translation in +Y direction commanded by THC.
		8	Bit sensed as 0 if translation in -X direction commanded by THC.
		7	Bit sensed as 0 if translation in +X direction commanded by THC.

Channel	Mnemonic	Bits	Function
31(cont)		6(CM)	Bit sensed as 0 if rotation commanded in negative roll direction by RHC (rotational hand controller).
		6(LM)	Bit sensed as 0 if ACA (attitude controller assembly) is deflected in negative roll direction. If software set to use controller for minimum impulse purposes, then a rotation in the desired direction produced. Otherwise, controller used as a rate command device, with input to cells 0042_8-0044_8. During portion of lunar descent, software senses bit for use as a landing point designation change, giving a "negative azimuth" offset (new site is to left as viewed by crew).
		5(CM)	Bit sensed as 0 if rotation commanded in positive roll direction by RHC.
		5(LM)	Bit sensed as 0 if ACA deflected in positive roll direction (see bit 6(LM) discussion). For landing point designation, input gives a "positive azimuth" offset (new site is to right as viewed by crew).
		4(CM)	Bit sensed as 0 if rotation commanded in negative yaw direction by RHC.
		4(LM)	Bit sensed as 0 if ACA deflected in negative yaw direction (see bit 6(LM) discussion).
		3(CM)	Bit sensed as 0 if rotation commanded in positive yaw direction by RHC.
		3(LM)	Bit sensed as 0 if ACA deflected in positive yaw direction (see bit 6(LM) discussion).
		2(CM)	Bit sensed as 0 if rotation commanded in negative pitch direction by RHC.
		2(LM)	Bit sensed as 0 if ACA deflected in negative pitch direction (see bit 6(LM) discussion). For landing point designation, input gives a "negative elevation" offset (new site beyond the present site).

Channel	Mnemonic	Bits	Function
31(cont)		1(CM)	Bit sensed as 0 if rotation commanded in positive pitch direction by RHC.
		1(LM)	Bit sensed as 0 if ACA deflected in positive pitch direction (see bit 6(LM) discussion). For landing point designation, input gives a "positive elevation" offset (new site short of the present site).
32	CHAN32		Register whose bits are used for miscellaneous inputs from the spacecraft. All bits are inverted as sensed by the program, so that a value of binary 0 means that the indicated signal is present.
		15	Not assigned.
		14	Bit sensed as 0 if the PRO (proceed) key on the DSKY is depressed (see Section IIJ). This key was formerly labeled "STBY" (and also serves that function if bit 11 of channel 13 is 1). Software can cause a logical "Proceed" function to be performed when a binary 1 to binary 0 transition of the bit is sensed by a check done every 0.12 sec.
		13	Not assigned.
		12	Not assigned.
		11(CM)	Bit sensed as 0 if the "ΔV CG" switch set by crew to the "LM/CSM" (as opposed to "CSM") position. The software uses a DSKY input for vehicle status.
		11(LM)	Not assigned.
		10(CM)	Not assigned.
		10(LM)	Bit sensed as 0 if the descent engine gimbal failure monitor detects an apparent gimbal fail in the pitch or roll gimbal trim system. The software does not use the bit (but takes action based on bit 9 of this channel instead).

Channel	Mnemonic	Bits	Function
32(cont)		9(CM)	Not assigned.
		9(LM)	Bit sensed as 0 if the "Engine Gimbal" switch set by crew to "OFF" (as opposed to "ENABLE") position, indicating that the descent engine gimbal drive system has been disabled. If bit 0, software does not attempt to use bits 12-9 of channel 12 to control the position of the descent engine gimbal.
		8(CM)	Not assigned.
		8(LM)	Bit sensed as 0 if System A Quad 2 RCS jets shut off (RCS jets 10 and 11).
		7(CM)	Not assigned.
		7(LM)	Bit sensed as 0 if System B Quad 2 RCS jets shut off (RCS jets 9 and 12).
		6(CM)	Bit sensed as 0 if negative roll commanded by minimum impulse controller.
		6(LM)	Bit sensed as 0 if System A Quad 1 RCS jets shut off (RCS jets 13 and 15).
		5(CM)	Bit sensed as 0 if positive roll commanded by minimum impulse controller.
		5(LM)	Bit sensed as 0 if System B Quad 1 RCS jets shut off (RCS jets 14 and 16).
		4(CM)	Bit sensed as 0 if negative yaw commanded by minimum impulse controller.
		4(LM)	Bit sensed as 0 if System B Quad 3 RCS jets shut off (RCS jets 6 and 7)

Channel	Mnemonic	Bits	Function
32(cont)		3(CM)	Bit sensed as 0 if positive yaw commanded by minimum impulse controller.
		3(LM)	Bit sensed as 0 if System B Quad 4 RCS jets shut off (RCS jets 1 and 3).
		2(CM)	Bit sensed as 0 if negative pitch commanded by minimum impulse controller.
		2(LM)	Bit sensed as 0 if System A Quad 3 RCS jets shut off (RCS jets 5 and 8).
		1(CM)	Bit sensed as 0 if positive pitch commanded by minimum impulse controller.
		1(LM)	Bit sensed as 0 if System A Quad 4 RCS jets shut off (RCS jets 2 and 4).
33	CHAN33		Register whose bits are used for various hardware status data. All bits are inverted as sensed by the program, so that a value of binary 0 means that the indicated signal is present. Bits 15-11 of this channel are flip-flop inputs, which retain a "set" state (binary 0 as sensed) until reset by a "loading" type command (orders WAND, WOR, or WRITE in Section IVC) or hardware restart.
		15	Flip-flop input sensed as 0 if the computer oscillator has stopped. Can be reset by a channel loading command.
		14	Flip-flop input sensed as 0 if a "computer warning" indication produced (e.g. restart, counter fail, voltage fail in standby, scaler double or fail, prime power fail, or alarm test by bit 10 of channel 13). Can be reset by a channel loading command.

Channel	Mnemonic	Bits	Function
33(cont)		13	Flip-flop input sensed as 0 if a PIPA fail indication generated by PIPA (accelerometer) electronics due to improper pulses from a PIPA. Software controls setting of bit 1 (ISS Warning) of channel 11 based on this input bit and the use being made of PIPA outputs. Can be reset by a channel loading command.
		12	Flip-flop input sensed as 0 if a telemetry end pulse occurs too soon after the previous pulse: these pulses cause program interrupt #8 to be generated, see Section IIH. The pulses are considered to be "too fast" if a 100 pps pulse has not occurred since the previous end pulse was received. Can be reset by a channel load.
		11	Flip-flop input sensed as 0 if an input bit to cell 0045_8 (INLINK, see Section IID) is rejected due to an excessive bit rate. Rejection takes place if a 6400 pps pulse has not occurred since the previous input bit was received. Can be reset by a channel loading command.
		10(CM)	Bit sensed as 0 unless both the "CM Up Telemetry" switch (on the main display console) and the "Up Telemetry" switch (in the lower equipment bay) are each set to "ACCEPT" (as opposed to "BLOCK"). Bit not used by software, but it must be a binary 1 for inputs from the uplink receiver to be gated into cell 0045_8 (INLINK, see Section IID).
		10(LM)	Not assigned (would be expected to read a binary 1). Similar blocking function to that for CM could be obtained by setting spacecraft switch to use rf link for voice backup (or by manually setting bit 6 of channel 13 to 1).

Channel	Mnemonic	Bits	Function
33(cont)		9(CM)	Not assigned.
		9(LM)	Bit sensed as 0 if landing radar range ("altitude") on low scale (controlled by landing radar system and changed at an altitude of about 2500 feet). Least increment of range information decreased by a factor of 5.000 when switch to low scale (see cell 0046_8, RNRAD, in Section IID).
		8(CM)	Not assigned.
		8(LM)	Bit sensed as 0 if all three landing radar velocity trackers have locked on, a necessary criterion for landing radar velocities to be valid.
		7(CM)	Not assigned. Formerly used with star tracker to indicate star present.
		7(LM)	Bit sensed as 0 if power applied to landing radar and the antenna is in "position 2" (used for hovering). Antenna can be commanded from position 1 to position 2 by bit 13 of channel 12.
		6(CM)	Not assigned. Formerly used with star tracker to indicate star tracker on.
		6(LM)	Bit sensed as 0 if power applied to landing radar and the antenna is in "position 1" (used for descent prior to hovering, see bit 7(LM) of this channel).
		5(CM)	Bit sensed as 0 if the "Optics Mode" switch in the lower equipment bay is set to "CMC" and the "Optics Zero" switch there is set to "OFF". A binary 0 indicates that the optics can be driven via cells 0053_8-0054_8 (see Section IID) unless inhibited e.g. by setting bit 11 of channel 12 to 1.

Channel	Mnemonic	Bits	Function
33(cont)		5(LM)	Bit sensed as 0 if the landing radar range tracker and two rear velocity-beam trackers have locked on, a necessary criterion for landing radar range ("altitude") data to be valid.
		4(CM)	Bit sensed as 0 if the "Optics Zero" switch in the lower equipment bay is set to "ZERO" (as opposed to "OFF"), regardless of the position of the "Optics Mode" switch there. If bits 5-4 = 11_2, this indicates that the "Optics Zero" switch is set "OFF" and the "Optics Mode" switch is set to "MAN" (for manual positioning of optics).
		4(LM)	Bit sensed as 0 if the rendezvous radar range tracker and frequency tracker are locked on, a necessary criterion for rendezvous radar range and range rate data to be valid.
		3(CM)	Not assigned.
		3(LM)	Bit sensed as 0 if rendezvous radar is on the low range scale. Internal range counter in radar is 18 bits in length, and if most significant 3 bits are 0 then bits 15-1 are sent to cell 0046_8, and this bit is set 0; otherwise, bits 18-4 are sent. Hence least increment decreased by a factor of 8.000 when switch to low scale, and switch occurs at 9.38 (2^{15} - 1) feet, or at about 50.584 nm.
		2(CM)	Bit sensed as 0 if VHF range data considered OK.
		2(LM)	Bit sensed as 0 if rendezvous radar power is on and the rendezvous radar mode switch is in the "LGC" (as opposed to "SLEW" or "AUTO TRACK") position, meaning that CDUs driven from an LGC power supply and control of the antenna position can be accomplished via cells 0053_8 -0054_8. If bit is 1, software can set bit 1 of channel 12 = 1 (see equation documentati

Channel	Mnemonic	Bits	Function
33(cont)		1	Not assigned.
34	DNTM1	15-1	Register used to contain the first word of a pair telemetered periodically. Loading of a new pair is performed by software when program interrupt #8 (see Section IIH) is processed. Channel contents cannot be sensed by a channel sensing instruction (will give zero). See Section IIH for format of output.
35	DNTM2	15-1	Register used to contain the second word of a pair telemetered periodically. Loaded by software when program interrupt #8 (see Section IIH) is processed: channel 34 is loaded also at this time. See Section IIH for format of output. Channel contents cannot be sensed by a channel sensing instruction (will give zero).

IIF Fixed Memory Mechanization

The fixed memory is implemented by a collection of 3072 magnetic cores, each of which is suitably threaded or bypassed by 208 wires (192 sense lines, 14 inhibit lines, 1 set line, and 1 reset line). A given core is used to determine the information from two addresses in each of six consecutive banks, or a total of 12 addresses (12 x 3072 = 36,864). Readout of the contents of a given address is accomplished by appropriate hardware address decoding logic causing a particular core to be set (magnetized in a certain direction) and then reset. The changing magnetic field during the reset induces a voltage in the sense lines that are threaded through the core (like an ordinary transformer), but not in the sense lines that bypass the core. Additional hardware address decoding logic selects the output of a set of 16 sense lines (called a strand: there are 12 strands associated with each core, one for each address associated with the core, giving the 16 x 12 = 192 sense lines with a core). These 16 sense lines have the contents of the indicated address represented by the presence (binary 1) or absence (binary 0) of an induced voltage: as discussed in Section IIA, the 16 bits associated with a given address reflect 15 "information" bits and an odd parity bit to make the total number of binary ones in the 16-bit word an odd number.

Although a detailed knowledge of the logical design of the memory is not required to review the program, some knowledge of its mechanization is desirable for proper evaluation of the impact of program changes upon the hardware. As discussed in Section IIB, the fixed memory is divided into a collection of 36 banks, each of which contains 1024 cells (giving the fixed-memory capacity of 36 x 1024 = 36,864 cells). Banks 02 and 03 can be addressed independently of the FBANK register, and banks 00 - 27

are addressed independently of the contents of SUPERBNK (channel 07). Banks 30 - 37 are addressed for SUPERBNK contents of 3 or less (using FBANK in range 30 - 37), and the remaining banks 40 - 43 are addressed for SUPERBNK contents of 4 (using FBANK in range 30 - 33). It is conventional to use the fixed memory bank number (in octal, of course) to identify individual banks, and this convention is followed below, without further reference to the method whereby the bank number is determined from the contents of the S-register, FBANK, and SUPERBNK registers.

The allocation of the contents of individual banks to computer hardware is reasonably straightforward, but can best be explained after a digression to review the mechanical design of the fixed memory:

1. The fixed memory consists of three "rope assemblies", called "R", "S", and "T". Each rope assembly in turn contains two "modules": B1 and B2 are in rope assembly R; B3 and B4 are in rope assembly S; and B5 and B6 are in rope assembly T.

2. Each module has two "sides", each of which is divided into two "areas" (giving 4 areas per module). Each side has a common "set" line.

3. An area contains 128 cores (hence a module has 4 x 128 = 512 cores, and the 6 modules total 6 x 512 = 3072 cores). Each of the cores in an area is threaded by the same "reset" line.

4. Each core is associated with a set of "inhibit" lines and with 12 strands (which, as mentioned previously, consist of 16 wires each for the 15 "information" bits and the odd parity bit). There is a total of 14 inhibit lines associated with each core.

The selection of a particular word stored in fixed memory is accomplished as described on the following pages. Computer hardware documentation should be consulted for details of timing etc. not covered here.

1) The rope assembly and module in that assembly are selected by the value of the bank number:

Banks 00 - 05 select rope R, module B1.
Banks 06 - 13 select rope R, module B2.
Banks 14 - 21 select rope S, module B3.
Banks 22 - 27 select rope S, module B4.
Banks 30 - 35 select rope T, module B5.
Banks 36 - 43 select rope T, module B6.

Hence each module has 6 banks, with the 6 modules giving the computer capacity of 36 banks.

2) One core (out of 128) in each of the four areas of the selected module is chosen by means of bits 7-1 of the S-register, whose one and zero outputs are connected to a total of 14 inhibit lines so threaded that all except one core in the area will receive an inhibit current (note that $2^7 = 128$).

3) A current pulse through the set line associated with one side of the selected module is produced. Because of the inhibit action of item 2, only two cores (one in each of the two areas on the selected side of the selected module) will become set. The side which is pulsed is selected by bit 9 of the S-register, which drives "side A" if zero and "side B" if one.

4) The strand (one out of 12) within the selected module is chosen by a suitable combination of bank number and S-register information. The strand (in range 1-12, a decimal number) is given by:

2 x (bank number modulo 6) + (bit 10 of S-register) + 1

The "modulo 6" operation is performed upon the decimal equivalent of the bank number: it yields a result of 0 for the first bank in each module and a result of 5 for the last bank (see #1 above). Address 07,3143 (FBANK = 07, S-register = 3143_8) would be strand 4 (of module B2), since the modulo operation yields 1 and bit 10 of S-register = 1.

5) A current pulse through the reset line associated with one area of the selected module is produced. The area (one of 4 in the module) is selected by bits 9-8 of the S-register, thus resetting one of the two cores set in #3 above, and inducing a voltage into the sense lines that are threaded through the core.

6) The output of the strand selected in #4 above is sensed to obtain the required contents of the specified memory cell.

Another term associated with the fixed memory is "paragraph". The paragraph is an octal number giving a kind of "serial number" for the information in fixed memory. Each paragraph consists of 256 words, and the paragraph number is computed as follows:

$$4 \times (\text{bank number}) + (\text{bits 9-8 of S-register}) + F$$

$F = 0$ for fixed-fixed memory
$= 20_8$ for variable-fixed memory.

Hence address 07,3143 would be in paragraph $4 \times 7 + 10_2 + 20_8 = 56_8$.

In addition to the check of the readout of each cell which is provided by the parity bit, individual banks in the fixed memory can be checked by means of a memory-cell summing routine which is included in the computer self-check portion of the fixed memory. This routine sums the contents of the addresses in each bank, halting when the last cell is reached that has been wired, and either checks that the magnitude of the sum is equal to the bank number or provides a display of the sum for manual review, depending upon the original manual inputs that initiated the check. The routine starts with the first cell in the bank and sums successive cells until two consecutive cells with contents equal to their addresses are found (or the last cell in the bank is read). If the cell contents equal its address, this is a one-step loop: two such cells in sequence

would not serve a functional program purpose (one such cell, of course, might be preceded by an index order so that transfer to a different cell would actually be performed). The summing routine halts after including in the sum the cell following the two consecutive cells with contents equal to their addresses (or after the final cell in the bank is reached).

The usual method for ending the wired cells in a bank (for the summing routines to work, no gaps in wired cells within a given bank can be permitted) is with two transfer-control (TC) orders to the present step (giving address contents equal to address, since the octal operation code for TC is 0), followed by a "checksum" (or "bugger word" as it is called by the G&N contractor). This checksum is computed by the assembly program, and it is formed so as to make the sum of the complete bank (including that cell) equal in magnitude to the bank number. The assembler operation BNKSUM is used to generate the required (up to 2) transfer-control orders at the end of the bank (the octal bank number is in the address field of the BNKSUM order), followed by the checksum word. This operation can be placed at any point in the assembly, and has the capability of omitting the generation of the transfer-control orders (indicated by "NO NEED" in the cell contents field) if the bank is full of functional orders. In addition, the number of words left (computed as 1023 minus the number of functional orders) in the bank is printed to the left of the cell contents field: cell 1024, of course, must contain the check sum. If no words are left, the statement "NO WORDS LEFT" is printed. A separate fixed memory constant is used to specify within the summing routine what the last bank entering the sum is to be, in the form of "BBCON*" (an operation which sets the octal cell contents to the proper value in BBCON format), with blank address field.

The algorithm used to compute the sum of each bank consists of the following machine language instructions, whose individual performances are described in Section IVB. The symbol CELL represents the contents of successive fixed-memory cells, read in sequence of increasing S-register contents, and SUM is the value of the sum (set zero at the start of each bank):

```
CA      CELL    (clear add)
AD      SUM     (add)
TS      SUM     (store, skip next order if overflow and set
                 A = 1 sgn SUM)
CA      Zero
AD      SUM
TS      SUM
```

Considering the quantities to have scale factor B14, the algorithm may be described as:

SUM = SUM + CELL

If |SUM| $\geq$ 16384:

SUM = SUM - 16383 sgn SUM

The check sum word is formed by the assembler in such a way as to give it the smaller of its two possible magnitudes: if the sum of the cells prior to the word is positive, for example, the word is formed so as to yield the positive bank number. Bank 00, of course, would have a sum of -0.

IIG Arithmetic and Overflow

Although most of the mechanization details of the arithmetic unit are not of interest from a programming viewpoint, some of its features are instructive for analysis of program performance. The adding-type arithmetic unit (ignoring some special-purpose provisions) makes use of ones complement arithmetic when operating with most computer instructions. Because of this, the quantity "zero" has two possible representations: 00000_8 and 77777_8, which are designated as +0 and -0 respectively. Except for some special cases involving two zero-magnitude operands (including (+0) + (+0) and (+0) - (-0)), the "zero" that results from addition or subtraction will be a negative zero.

Although most of the machine language orders (described in detail in Section IV) make use of the computer hardware arithmetic registers (A, L, or Q) for arithmetic manipulations, three instructions (AUG, DIM, and INCR) are included for changing the contents of an erasable memory cell (by $\pm$ 1) without affecting the information in the arithmetic registers. This feature is included in the computer logical design because of the necessity for processing the counter interrupts described in Section IIH without the execution time penalty that would be required to save and then restore the arithmetic registers. To achieve this capability, the adder in the arithmetic unit is not functionally composed of addressable arithmetic registers: instead, a set of input gates is used to provide binary levels corresponding to the operands, and output levels corresponding to the answer may be gated to the appropriate destination as desired. Most data transfers in the computer hardware

take place by gating various registers to "write amplifier" inputs, and the amplifier outputs are gated to the necessary destinations. Because of this design, it is unnecessary, for example, to go through the adder to load the A register (accumulator, see Section IIC).

In addition to the ones complement arithmetic operations included in the order code, there is also a special instruction (MSU) which may be used to form the ones complement difference of two twos complement numbers: such numbers generally would be obtained from CDU angle data, so that 2^{15}, rather than $(2^{15} - 1)$, different numbers can be represented, a hardware convenience for representing points on a circle. The MSU order is performed in the arithmetic unit by forcing an end-around carry and by setting $S_2 = S_1$ (see below) at the completion of the operation. If the second operand is 00000_8, this process converts the first operand from twos complement to ones complement; if the two operands are equal, the result is +0 (another exception to the rule that -0 results from most computer arithmetic operations that yield a "zero" answer).

When a word is read out of the memory into one of the arithmetic registers (A, L, or Q), bits 14-1 (the magnitude information) are placed in their corresponding bit positions of the register. Bit 15 of the memory word (the sign bit) is placed in both sign positions (identified for the A register as S_2 and S_1, where carries from bit 14 propagate to S_1 and from S_1 to S_2) of the register. The S_2 bit is considered as "the" sign bit (for program control instructions sensing the sign of A), and in general is the bit stored in memory for sign information. The adder of the arithmetic unit, however, is connected to S_1 and S_2 separately, so that arithmetic operations can effectively make use of a 16-bit word. The full 16 bits can be used for transfers

of data between the A and Q registers, but the S_1 bit is lost in transfers between the L and A registers (such as XCH L with overflow in A).

Under normal conditions, the S_1 and S_2 bits will be equal. After an addition or subtraction operation in which overflow took place, however, the bits will be unequal. It should be evident that bit S_1 has the overflow information that would have been propagated to the next most significant magnitude bit (if the word length of the computer were bigger), and advantage of this fact is taken in the TS (Transmit to Storage, see Section IVB) order code instruction. To avoid improper answers, S_1 should equal S_2 before division or multiplication operations are performed; for addition and subtraction, however, the S_1 bit is effectively another magnitude bit and can be used as such: $\frac{1}{4} + \frac{1}{2} + \frac{1}{2} - \frac{1}{2}$ (computed in that order) will give an answer of (3/4), provided of course that the sum of the first three terms is not stored by a TS order.

Storage of the accumulator contents into memory causes the overflow bit (in the quantity stored) to be lost, since the 16-bit memory word has an odd parity bit instead of the overflow bit. If the TS instruction is used, presence of an overflow (established by the fact that $S_1 \neq S_2$) will cause the next instruction to be skipped and the least significant bit of the accumulator to be set to $\pm$ 1, as described in Section IVB. A similar setting is employed for the DAS instruction. Since S_1, as described previously, has the features of an additional magnitude bit, it is used in place of S_2 for the storage of certain counter-incrementing orders that require twos complement arithmetic (and the MSU order), as well as those counter orders requiring the assembly of a serial stream of input bits.

The computer order code includes four instructions that make use

directly of double precision operands: DAS, DCA, DCS, and DXCH. The interpretive language described in Section VI permits portions of the computer program to be written almost as if the whole computer had nothing but double precision operations, however. The double precision machine language orders operate on the least significant half of the double precision word first, using the computer L register. The address associated with the order is then decremented by one and the most significant half of the word processed. Hence DXCH L, for example, starts by putting L in Q and Q in L, and then puts A in L and the (new) L in A, giving the net effect of putting A in L, L in Q, and Q in A.

There is no hardware requirement that sign agreement exist between the two halves of the double precision words: they are treated essentially as independent single precision quantities unless there is need to propagate a carry (or borrow) from the least significant half. The DV (divide) order employs a double precision dividend (in A,L) and forces sign agreement by hardware means before initiating the division sequence.

The assembly program increments the address of the symbol provided for a double precision order so as to read the least significant half first (as described above). Consequently, the symbol provided with the double precision order (either an absolute or a symbolic address) must be that of the most significant half of the word, and naturally the last cell in a switched memory bank cannot be considered the "most significant half" for such orders.

A detailed description of the hardware algorithms employed for multiplication and division can be found in the appropriate hardware documentation, and therefore is not included here. To minimize execution time, these algorithms are fairly elaborate. See Appendix A for more details on addition and overflow.

There are two distinct types of interrupts incorporated in the computer logical design: counter interrupts and program interrupts. Since they are quite different, separate sub-sections are devoted to each below.

Since the counter interrupts represent one hardware approach (others could have been selected, although probably with the need for additional hardware) to the mechanization of computer inputs driven by external signals, their existence for most programming purposes can be ignored. Program interrupts, on the other hand, perform an integral portion of the program control logic: consequently, it is conventional that the term "interrupt", unless otherwise specified, refers to these program interrupts.

Counter Interrupts

The 29 counter interrupts in the computer are associated with the 29 erasable memory cells (0024_8 - 0060_8, see Section IID) that may contain counter-type information. Seven "involuntary" (i.e. not under computer program control) counter instructions are associated with these counters, and can be performed when an appropriate counter interrupt is received. In some cases, a counter interrupt can select different involuntary instructions to be performed, depending on the nature of the external signal (such as positive or negative changes to the value of a counter) or the value of the quantity in the counter (positive or negative output pulses). The seven involuntary instructions, and the cells to which they apply, are given on the following pages.

1. DINC, applying to cell 0031_8(TIME6) and cells 0047_8 - 0056_8 (GYROCMD, CDU error counter drives, THRUST, and not used). If the contents of the cell are positive non-zero, they are decremented by 1 and positive output pulses are provided; if the contents are negative non-zero, the contents are incremented by 1 (i.e. magnitude decreased by 1) and negative output pulses are provided. Output pulses must be enabled from 0047_8 - 0056_8 by bit of channel 14 (10, 15-11, 4, and 5 respectively), which is reset to 0 when the counter contents are equal to -0 and another DINC pulse is generated (-0 is the result of a DINC to a cell equal to +1 or -1). Consequently, zeroing of cells by program means must load -0, not +0. Use of DINC with cell 0031_8 is enabled by bit 15 of channel 13, although the cell's output pulses are not used: instead, its decrement to -0 causes program interrupt #1 to be generated at the next DINC (and the enabling bit to be reset).

2. MCDU, applying to cells 0032_8 - 0036_8 (input CDU angles from IMU and optics/rendezvous radar). This instruction subtracts 1 (in twos complement) from the contents of the cell.

3. MINC, applying to cells 0037_8 - 0044_8 (accelerometer inputs and RHC/unused BMAG analog inputs). This instruction subtracts 1 (in ones complement) from the contents of the cell.

4. PCDU, applying to cells 0032_8 - 0036_8 (input CDU angles from IMU and optics/rendezvous radar). This instruction adds 1 (in twos complement) to the contents of the cell.

5. PINC, applying to cells 0024_8 - 0030_8 (TIMEi, i = 1-5) and cells 0037_8 - 0044_8 (accelerometer inputs and RHC/unused BMAG analog inputs). This instruction adds 1 (in ones complement) to the contents of the cell.

6. SHANC, applying to cells 0045_8 - 0046_8 (INLINK and RNRAD). This instruction shifts the contents of the cell left by one place, and then adds 1 (it is used for a binary one of a serial bit stream).

7. SHINC, applying to cells 0045_8 - 0046_8 (INLINK and RNRAD) and to cells 0057_8 - 0060_8 (unused OUTLINK and ALTM). This instruction shifts the contents of the cell left by one place (it is used for a binary zero of a serial bit stream or to generate a serial output bit stream from the cell overflow bit).

A counter interrupt request can be generated (in general) at any time. All requests are retained by the hardware until the end of the current computer instruction. At that time, provided that the next instruction is not a special-purpose TC order (EXTEND, INHINT, or RELINT), the request is honored. This means, for example, that a double precision computer instruction (such as DCA) can be used to sample the values of cells 0024_8 - 0025_8 (the computer clock) without concern that a counter interrupt will cause the two halves to be inconsistent due to an overflow of cell 0025_8 (see Section IID).

Satisfaction of a counter interrupt takes one MCT (memory cycle time of about 11.7 microseconds) per request (due to the need to read the counter from memory, modify it, and store it back). Priority for satisfaction of the requests is based upon the value of the counter's address (0024_8 has the highest priority and 0060_8 has the lowest), but all requests are satisfied before the next program instruction is started. See Section VII for the sequence with which the computer hardware performs its various functions.

Counter interrupts are not under computer program control (once the appropriate control bits, in some cases, have been set), cannot be inhibited by the program, and in fact can only be determined by the software to have occurred by sampling the cell in question. It is sometimes necessary (such as when the accelerometer cells are sampled) to sample and reset a counter without losing any counts: the machine-

language order XCH (Exchange) can be used for this purpose, since this order exchanges the contents of the A register and the cell specified by the address field of the order. In other instances, it is necessary to change the value of an output-generating counter cell (such as the cell used to generate gyro torquing pulses) while it may be controlling output pulse generation. In this case, the machine-language order ADS (Add and Store) can be used.

Program Interrupts

Eleven program interrupts are incorporated into the computer design. Most interrupts (provided certain conditions are satisfied) cause the performance of the program to be suspended, the contents of certain registers to be saved (some by hardware means, some by software), and the next instruction to be executed be the one at a special address (different for each interrupt) in order to start the "task".

The interrupt is mechanized through the involuntary instruction RUPT, which takes 3 MCT to perform. If necessary, it can also be programmed as EDRUPT, an extended order, using the following sequence:

```
        CA    start address desired, in ADRES form
        TC    bank 3 address  (i.e. in fixed-fixed, form 7xxx8)
        ------------
BNK3    EXTEND          (BNK3 is address to which TC is done, form 7xxx8)
        EDRUPT  BNK3
```

This sequence causes the hardware to initiate computations at the ADRES address contained in the accumulator, with various hardware flip-flops set as they would be for a "normal" hardware-induced program interrupt (FBANK setting is that from which the BNK3 step was entered). In either case, resumption of the program is triggered by the special-purpose

instruction RESUME (triggered by an INDEX order for cell 0017_8, see Section VA), taking 2 MCT to perform. The mnemonic stems from the phrase "Ed Smally's interrupt instruction".

The individual interrupts, with their titles, starting addresses, causes, and functions are:

1. T6RUPT, starting address 4004_8, generated by the next DINC after TIME6 (cell 0031_8, see Section IID) has been reduced to -0. Conventionally used to control the timing of RCS jet commands in output channels 05 and 06 (by suitable software).

2. T5RUPT, starting address 4010_8, generated by overflow of TIME5 (cell 0030_8, see Section IID). Conventionally used to control cycling of computations associated with the digital autopilots (jet timing conventionally controlled by program interrupt #1).

3. T3RUPT, starting address 4014_8, generated by overflow of TIME3 (cell 0026_8, see Section IID). Conventionally used to control performance of "waitlist" tasks (see Section VIIA).

4. T4RUPT, starting address 4020_8, generated by overflow of TIME4 (cell 0027_8, see Section IID). Conventionally used to control cycling of periodic input/output functions (such as driving of DSKY digits, see Section IIJ).

5. KEYRUPT1, starting address 4024_8, generated by depression of a key on the DSKY keyboard (main panel DSKY for CM). Input trap circuit reset when key is released. Used by software to initiate processing of keyboard input from channel 15.

6. KEYRUPT2, starting address 4030_8, generated for CM by depression of a key on lower equipment bay (or "navigation panel") DSKY or depression of optics mark or mark reject button. For LM, it is generated by depression of a mark or mark reject button or by rate-of-descent switch offset. Input trap circuit reset when key or button released, or rate-of-descent switch returned to middle (neutral) position. Used by software to start channel 16 processing.

7. UPRUPT, starting address 4034_8, generated by overflow of cell 0045_8 (INLINK, see Section IID) due to shifting of the first binary 1 (in the 16-bit word sent to the computer) out of the cell. Used by software to start processing of information in INLINK (including its reset). If the checks are passed, the same computational job is established as that for program interrupts #5 and #6 if a DSKY input is involved.

8. DOWNRUPT, starting address 4040_8, generated by an end pulse from the telemetry system. The basic telemetry format consists of eight-bit data words transmitted at a rate depending on the setting of spacecraft switches. At the "high bit rate" (51.2 kbps), 5 of the 128 words in each frame are allocated to computer digital data (giving 40 bits), thus permitting 50 of the 40-bit computer words to be sent per second. Computer words are loaded for downlink transmission in channels 34 and 35 (plus bit 7 of channel 13 for "word order code" information). The 40 bits are transmitted in the following sequence:

   a) Bit #1 is the word order code bit.
   b) Bits #2 - #16 are bits 15-1 (sign first) of channel 34.
   c) Bit #17 is an odd parity bit for channel 34 data.
   d) Bits #18 - #32 are bits 15-1 (sign first) of channel 35.
   e) Bit #33 is an odd parity bit for channel 35 data.
   f) Bits #34 - #40 are the same as bits #2 - #8 (i.e. bits 15-9 of channel 34).

After the final bit, the end pulse from the telemetry system is received, generating the interrupt (request). At the high bit rate, the program has about 19.2 ms in which to respond to the interrupt and load new data into channels 34-35 before the transmission is started again. Garbled downlink data, of course, would result if loading not accomplished (ground resynchronization could be accomplished when the word order code bit flagged data). The "low bit rate" in the CM is 1.6 kbps (200 eight-bit words per second), in which 50 of the 200 words are the digital data (giving an end-pulse rate of one every 0.1 second rather than the rate of one every 0.02 second at the high bit rate). In the LM, no LGC data is transmitted at low bit rate (hence AGS initialization, for example, must be accomplished at high bit rate). If bit 12 of channel 33 is a binary 0, this indicates that a telemetry end pulse was rejected.

9. RADAR RUPT, starting address 4044_8, generated by completion of the shifting of radar data into cell 0046_8 (RNRAD). The time delay between the setting of bit 4 of channel 13 and the generation of the interrupt is 90-100 ms (see Section IID). Used by software to start processing of information in RNRAD.

10. HAND CONTROL RUPT, starting address 4050_8, generated by the setting of interrupt traps 31A, 31B, or 32. These traps are reset by bits 12-14 of channel 13 respectively, and are required because of the duration of the input signals (which otherwise could produce multiple program interrupts). Trap 31A is associated with bits 6-1 of channel 31 (rotational hand controller deflections); trap 31B is associated with bits 12-7 of channel 31 (translation hand controller inputs); and trap 32 is associated with bits 10-1 of channel 32 (CM minimum impulse controller and LM thruster fail and descent engine gimbal fail inputs). A signal fed into the indicated bit positions causes the indicated trap to be set. In the CM software, this program interrupt is not used, since sampling of the input signals involved is done sufficiently often as a consequence of the normal digital autopilot cycling. In the LM software, a similar argument applies (the digital autopilot cycling and logic performs functions equivalent to those originally intended by the hardware design), so that only trap 31A is employed in order to monitor for hand controller deflections associated with the landing point designation (see bits 6, 5, 2, and 1 of channel 31, Section IIE).

11. GOPROG, starting address 4000_8, caused by an internally generated hardware signal in response to various hardware difficulties. A "hardware restart" is produced, as described in more detail below.

Program interrupts #1 - #10 have the following common features:

a) Their first few steps store A in ARUPT, L in LRUPT, and transfer control to a routine that performs the necessary computations (after saving Q and/or BBANK and/or SUPERBNK if necessary).

b) They initiate the performance of a task, at the conclusion of which (after restoration of A, L, and any other cells necessary) the operation RESUME (see Section VA) causes the program to start again from where it was interrupted, provided of course that another program interrupt is not waiting to be processed.

c) Their priority for initiation is the order in which they were listed above (#1 is the highest and #10 is the lowest). Once a program interrupt has had its processing started, however, it will continue on to completion: the "priority" is significant only in determining which interrupt should be processed first.

d) They will not be acted upon (processed), but instead will be retained for future action, if any of the following criteria are satisfied:

1. The current machine language order is not yet complete.
2. An "extended" machine language order is about to be performed (see Section IV), since information retained when interrupt processing is started does not include the "extended order code" bit.
3. An accumulator overflow (see Section IIG) condition exists, since information retained when interrupt processing is started does not include the overflow bit. Other overflows (e.g. Q register) are not protected.
4. The INHINT/RELINT flip-flop (see Section VA) is set to inhibit program interrupts, meaning that interrupts not desired by programmer (permitting flagword bits to be changed, downlink state vectors to be consistent, etc.).
5. A program interrupt (even one of lower priority) is already being processed.
6. A special-purpose TC order (EXTEND, INHINT, or RELINT) is the next instruction to be executed.

For a summary of the sequence in which the computer hardware (and software) performs its various functions, see Section VII.

Program interrupt #11 (sometimes referred to as interrupt #0) differs in a number of respects from the others. It does not result in "normal" resumption of the program (instead, a "restart" is performed, see Section VIIC); it takes absolute priority over other program interrupts; it cannot be inhibited; and it can even "interrupt an interrupt". As part of its generation, a special involuntary interrupt instruction is produced, causing a master clear signal ("GOJAM") to be generated by the hardware. Program interrupt #11, which is also termed a "hardware restart" (to distinguish it from similar functions that can be done solely by software), can be triggered by the following:

1. Indication of power failure on the prime 28-volt supply (below about 22.6 volts), the 14-volt supply (below 12.5 volts or above 16 volts) or the 4-volt supply (below 3.65 volts or above 4.4 volts).

2. Detection of a computer oscillator failure.

3. Detection of a large program loop ("night watchman"), revealed by failure to address erasable memory cell 0067_8 (NEWJOB, see Section IID) in a period ranging from 0.64 to 1.92 seconds.

4. Detection of a transfer control failure ("TC trap"), revealed by having a TC or TCF (see Section IV) order in effect for a period of from 5 to 15 ms (or a counter interrupt), or if no TC or TCF order is executed in this same time interval. The software can cause a hardware restart by this means, through a TC order to the present step.

5. Detection of a parity failure on a word read from fixed or erasable memory (applies to all addresses of value 0010_8 or above). A failure would be revealed if an even number of binary ones were in the word (see Section IIA).

6. Detection of a program interrupt failure ("RUPT lock"), revealed if a program interrupt is continuously in effect for a period of from 140 ms to 300 ms, or if no program interrupt takes place in this same interval.

7. Recovery from "standby" operation. This is analogous to item #2 above, since standby operation removes power from the monitor circuit, causing the hardware to consider standby an "oscillator fail" condition, even though the oscillator keeps on running (so that time information can be obtained from channels 03 and 04, see Section IIE).

Peripheral Equipment Orders

There are five interrupt-type instructions which may be originated from computer peripheral equipment, i.e. the CTS (Computer Test Set) or PAC (Program Analyzer Console). Although these instructions are not used when the computer is in a flight environment, there are listed here in the interests of completeness:

1. FETCH. 2 MCT. Display contents of specified address.
2. INOTLD. 1 MCT. Load specified channel.
3. INOTRD. 1 MCT. Read specified channel and display it.
4. STORE. 2 MCT. Store data in specified cell. Should not be confused with interpretive language order having the same mnemonic.
5. TCSAJ. 2 MCT. Transfer control to specified address.

IIJ Display System

Most of the outputs from the computer for display purposes are transmitted through channel 10, which is assigned the mnemonic "OUTO". Bits 15-12 of this register define a particular row of relays (which are of the latching type) to be driven, while the remaining eleven bits specify the new settings for these relays. Since the relays are bistable devices, retaining either a binary 0 or a binary 1 state until changed, register OUTO need retain the specification of the contents of a row for only 0.02 seconds (which also helps minimize power consumption and heat buildup), following which the channel is zeroed for 0.02 seconds before another row is specified. Under certain conditions (see equation documentation for details), the software allows a new row setting to be specified every 0.04 seconds, permitting a complete change of the eleven rows controlling the DSKY numerical and sign displays in less than $\frac{1}{2}$ second.

Relay rows 1-11 (selected by having bits 15-12 of OUTO equal to 01_8-13_8 respectively) are used to drive the digit and sign displays on the DSKY (display and keyboard assembly), while relay row 12 (bits 15-12 of OUTO equal to 14_8) is used to drive some of the indicators on the DSKY (most of the other indicators are driven from channel 11 bits, see Section IIE). The displays and indicators which are energized consist of three banks (R1, R2, and R3 registers) of five digits and a sign; three banks (noun, verb, and program or mode) of two digits each; 9(CM) or 11(LM) indicators on the DSKY (two others are driven by separate hardware); and a request for an operator action (FLASH, which causes the verb and noun displays to blink on for 0.64 seconds, then off for 0.64 seconds).

Numbers are specified for display on the DSKY by a total of five bits, while sign and indicator information require one bit each. The individual bits of the first eleven rows have the following meanings:

Row	Bit 11	Bits 10-6	Bits 5-1
01	-R3S	R3D4	R3D5
02	+R3S	R3D2	R3D3
03		R2D5	R3D1
04	-R2S	R2D3	R2D4
05	+R2S	R2D1	R2D2
06	-R1S	R1D4	R1D5
07	+R1S	R1D2	R1D3
10			R1D1
11		ND1	ND2
12		VD1	VD2
13		MD1	MD2

In this table, the row numbers are cited in octal (as loaded into bit positions 15-12 of OUTO), and "D" means digit, with D1 the most significant and D5 (or D2) the least significant. R1 - R3 refer to the three display registers, each with sign ("S"), while "N", "V", and "M" refer to the noun, verb, and mode (or program) two-digit registers respectively.

The pattern of the digit displayed on the DSKY panel is specified by the five bits assigned to the character, according to the table on the following page. In this table, the five bits of the pattern are identified as B5 through B1 respectively. The prime in the formula for a particular display segment designates a complement. A numerical entry in the table means that the display segment indicated at the left is energized for the display of that digit.

Display Segment	Formula	Display: Octal Pattern:	Blank 00	0 25	1 03	2 31	3 33	4 17	5 36	6 34	7 23	8 35	9
Top	B5			0		2	3		5	6	7	8	
Middle	B4					2	3	4	5	6		8	
Left Upper	B3			0				4	5	6		8	
Right Upper	B1			0	1	2	3	4			7	8	
Left Lower	B2'(B1 + B5)			0		2				6		8	
Right Lower	B2 + B2' B3			0	1		3	4	5	6	7	8	
Bottom	B5 (B3 + B4)			0		2	3		5	6		8	

Most keys on the DSKY are used to generate a five-bit code which appears in bits 5-1 of channel 15 (and, for CM, channel 16 in the case of the DSKY in the lower equipment bay). The same codes are used for the corresponding characters when transmitted via uplink means to cell 0045_8 (INLINK, see Section IID). The individual key codes are:

Key	Code	Function
0	20_8	Digit zero.
1-9	01_8-11_8	Digits one through nine (code corresponds to the decimal value).
Verb	21_8	Indicates that up to the next two digits specify verb code, indicating "action desired".
Noun	37_8	Indicates that up to the next two digits specify noun code, indicating "action recipient".
+	32_8	Indicates that up to the next five digits specify a positive data number (loaded into Ri register).
-	33_8	Indicates that up to the next five digits specify a negative data number (loaded into Ri register).
Clear (CLR)	36_8	Causes Ri register to be cleared (if software checks passed satisfactorily).
Key Release (KEY REL)	31_8	Indicates to software that operator is releasing display system for internal control of display.
Enter (ENTR)	34_8	Indicates to software that either execution of the verb/noun direction should be performed or that data that has been keyed into an Ri register is complete.
Error Reset (RSET)	22_8	Indicates to software that various error indicators (and internal alarm-code cells) are to be reset. Turns off Restart light by hardware means. Uplink (but not DSKY input) also resets software bit that set after failure of check of INLINK input (see cell 0045_8 in Section IID).

In addition to these key codes, the PRO button is also on the DSKY, and causes bit 14 of channel 32 to be 0 when it is depressed (see Section IIE). If bit 11 of channel 13 is set, this button also used to put computer into standby (lower-power consumption) mode if pressed for 0.64 - 1.92 seconds, and to return it to normal operation when pressed again for the same interval.

The various bits of relay row 14_8 have the following significance:

Bit	Light	Function
11		Not assigned.
10		Not assigned.
9	PROG	Set by program to indicate that a program check has failed.
8	TRACKER	For CM, set by software to indicate an optics CDU fail; for LM, set by software to indicate a radar malfunction. See mission documentation.
7		Not assigned (has been used for test purposes).
6	GIMBAL LOCK	Set by software to indicate approach of middle gimbal angle to a "lock" condition (such as an angle in excess of 70°).
5	ALT(LM-only)	Set by software (or flashed) to indicate data difficulty with landing radar altitude. Not connected in CM.
4	NO ATT	Set by software to indicate that inertial subsystem not suitable as an attitude reference (because it is off, caged, or in coarse align).
3	VEL(LM-only)	Set by software (or flashed) to indicate data difficulty with landing radar velocity. Not connected in CM.
2		Not assigned.
1		Not assigned.

The STBY light is energized by the computer hardware if the computer is in the standby mode of operation, while the RESTART light is energized if a computer restart (program interrupt #11, see Section IIH) is encountered. For testing purposes, both lights can be energized by bit 10 of channel 13. The RESTART light can be turned off by the Error Reset key or by bit 10 of channel 11.

The various lights on the DSKY panel, and the source from which they are driven, are summarized in schematic form on the next page.

UPLINK ACTY ch. 11 bit 3 white	TEMP ch. 11 bit 4 yellow	COMP ACTY ch. 11 bit 2 green	prog MD1 MD2
NO ATT row 14_8 bit 4 white	GIMBAL LOCK row 14_8 bit 6 yellow		
STBY hardware white	PROG row 14_8 bit 9 yellow	verb VD1 VD2	noun ND1 ND2
KEY REL ch. 11 bit 5 white (flashes)	RESTART hardware yellow	R1S	R1D1 R1D2 R1D3 R1D4 R1D5
OPR ERR ch. 11 bit 7 white (flashes)	TRACKER row 14_8 bit 8 yellow	R2S	R2D1 R2D2 R2D3 R2D4 R2D5
no DAP row 14 bit 2	ALT(LM) row 14_8 bit 5 yellow		
Prio row 14 bit 1	VEL(LM) row 14_8 bit 3 yellow	R3S	R3D1 R3D2 R3D3 R3D4 R3D5

TEMP also connected to channel 30 bit 15 (so light comes on if computer is in standby).

Verb and Noun flash: ch. 11 bit 6.

STBY and RESTART also energized by ch. 13 bit 10.

All digits on DSKY display (and sign) driven from rows 01_8 - 13_8, as shown on page IIJ-2.

The COMP ACTY and the digit (and sign) display are electroluminescent displays, while the remaining indicator lights are incandescent.

The keyboard layout is as follows:

VERB	+	7	8	9	CLR	ENTR
	-	4	5	6	PRO	
NOUN	0	1	2	3	KEY REL	RSET

III FORMAT OF GUIDANCE PROGRAM SYMBOLIC LISTING

This section describes the format of a program symbolic listing as reflected by typical programs, based on the currently used assembly program. New features and capabilities can be expected to be added to this program, however, so it should be realized that items may be encountered in a listing which are not described below. In addition, in the past the printer character sets used by MSC and the G&N contractor have had some differences (the G&N contractor's "?", for example, has been printed as "π" or "&" in MSC listings, and some G&N contractor symbols are not printed at all in MSC listings, such as an apostrophe in line printer outputs). Symbols may also appear differently (a colon as an apostrophe, for example).

The assembler (referred to also as the assembly program) is quite flexible in its capabilities, and unusually tolerant in the variety of formats, such as spaces between digits of a number, that it will accept. A "symbol" consists of from one to eight characters (with certain restrictions), and is equated to a unique octal cell address by one of the following means:

a) Specification as the tag associated with a quantity stored in that octal cell location.

b) Specification as a tag equated (by the "EQUALS" or "=" pseudo-operations) to some other quantity, which may be another tag or an absolute address.

A symbol must be eight characters or less in length, and cannot consist of an integer, an integer preceded by a plus or minus sign, or an integer followed by the letter D. Aside from these restrictions, however, the symbols which may be selected are quite varied, as suggested by the following symbols selected at random from a sample program.

0.00167	16/32400	A	BITS6&15	DV-+,+
11DEC	1SEC+1	A+B	BUF+	.166...
11DEC.	+-ZERO	ACOS=0	D--SC	-TAN22.5
13-11,1	(1-K),QR	A(X)	DLOAD*	NXTT6=P

For proper performance of the assembler, of course, symbol definitions must be unique, so that only one octal cell location corresponds to the given symbol. The assembler distinguishes between the letter O and the number 0.

Quantities not satisfying the format restrictions for a "symbol" are used for other purposes by the assembler. An unsigned integer in the address field (frequently found in interpretive language coding) without an operand indication (OCT, DEC, etc.) is considered as an octal absolute address (equivalent to an octal integer for values below 10000_8, as discussed in Section IIB). If the integer is followed by the letter "D", it is treated as a decimal number. Hence both "36" and "30D" would be loaded in the memory as 00036_8. The quantity "20000" (with no operand indication) would be loaded in the memory as 10000_8 in view of its definition as an address: for such numbers, it is conventional (and desirable) to specify explicitly the operand information. The "D" is optional with "8" or "9".

A blank in the address field is considered to mean the address of the step itself, and a signed integer (such as "+2" or "-3") would be translated relative to the step's own address (two steps beyond or three steps earlier respectively). As mentioned previously, signed integers are not allowable symbols, so they sometimes are used in association with the relative addresses as a form of program "remark": they have no effect on the performance of the assembly program.

A symbol followed by a space and then a signed integer is treated by the assembler as if the value of the integer modified the octal instruction (operation code and address). If the integer is of sufficient size, it will cause modification of the operation code, thus giving compatibility with the hardware INDEX order described in Section IV. As with other integers in the address field, the signed integer is considered to be octal unless followed by the letter "D" (for decimal). It should be noted that a space must be left between the symbol and the signed integer, or else the net combination would be considered as another symbol (as indicated by the symbol "1SEC+1" given on the previous page). If it is desired to have the assembled address information be negative (for use with INDEX, for example), this can be accomplished by the artifice of using "0 - n" there, since merely "- n" would be considered as a relative address.

The address-field form (symbol $\pm$ integer) is the only type of address-field modification conventionally allowed by the assembler. The effect of adding two symbols (or subtracting them), however, can be achieved by appropriate use of the address operations "=PLUS" and "=MINUS", which are described in Section VC. Repeated use of these operations, of course, can achieve the effect of multiplication of a symbol's octal equivalent by an integer.

Page Layout

Each page of the program listing has 120 columns of available space for the printing of program information, and a maximum of 56 output lines per page (of which the first four, including two blanks, are for header data).

The location and information which is printed is established by the nature of the original input to the assembler, as explained in more detail below.

The top line on each page contains a program identification which is specified when the run is made. The assembler identification (e.g. "GAP:") appears in print columns 1-4, followed by the assembler action (such as "assemble") and the revision number, name, and "author" of the program being assembled. On the right-hand side of the page, the time when the run was made appears (hours:minutes) for identification, followed by the date on which the run was made. Print columns 112-115 contain "PAGE", and columns 117-120 contain the master page number (starts at 1, right justified with leading zeros suppressed) used throughout the assembly to identify locations in the listing. An additional piece of information included in the top line (before the PAGE print), if applicable, is the Subroutine name and revision (see Information at Start of Listing below).

The second line is blank, and the third is used to supply the "log" identification information. In order to permit different people to work on different areas of the program while minimizing their need for close synchronization during the development of these areas, the assembler information is divided into a number of separate segments ("log sections"), each of which may be modified individually through specification of the appropriate sequence number (line identification) within that particular segment. Each segment is assigned a title, which appears on the left-hand side of this second line of printing. The printing of the title of the log section generally starts in print

column 9, and in addition an "L" (for "log card") appears in print column 1. On the right-hand side of this line appears the "USER's PAGE NO. xxx", where xxx is restarted at 1 at the beginning of each log section (leading zeros again suppressed). This is followed in print columns 111-112 by Ei, giving the most recent erasable memory bank specification. The Ei identification is nulled (printing E0) at the start of each log section, and can be used by the assembler to check for possibly illegal memory references (see EBANK= below). The last piece of information in this line is Si, which appears in print columns 114-115. This gives the most recent SUPERBNK setting specification (see SBANK= below), and can be used to generate BBCON values.

Next comes another blank line, completing the four lines allocated for header data. The remaining lines (up to 54) on the page contain the program information. For each line of this information, print columns 2-7 contain a "sequence number", which is restarted at the beginning of each log section and which increases monotonically (when left justified) throughout that segment. This number is normally incremented by +1 in print column 5, and is used to specify the location of changes when making modifications to a log section. For example, the numbers:

0009
0010
00101
00103
001031
001032
00104
0011

could appear in sequence on successive lines of coding. An assembler

capability exists to cause the sequence numbers within a log section to be redefined so as to count up uniformly in the counting position (column 5), but this option is not necessarily employed when a new listing is made.

Print column 1 is blank for most lines of coding. If it is blank, card columns 9-80 are printed as print columns 49-120, and the remaining print columns are filled with address, address content, and symbol reference information as described in more detail below. Print column 1 can also contain certain letters, which result in assembler operation as follows:

"A" signifies an "aligned remark card", which does not produce any binary memory information. Card columns 9-80 are printed in the same print positions as for a normal card (i.e. 49-120), and are generally used to provide additional comments that could not be fitted onto the same card as the original program step being described.

"L" signifies a "log card", used to specify the segment of the program (and appearing on the second printed line of the page, as described above).

"P" signifies "page", and causes a printer page-eject signal before it is printed (making the "P" line the first line of program information, or fifth line on the page). Otherwise, the "P" is treated the same way as an "R" card.

"R" signifies "remark", and does not produce any binary memory information. Card columns 9-80 are printed in print columns 9-80. Print columns 81-120 may be filled by the information on another card, if that card has a "9" punched in column 8 (and is in the proper sequence-number order).

Print column 8 contains a flag (such as "*") if the card on that print line was changed in the most recent modification of the Subroutine.

Card Layout

Although the assignment of functions to the individual columns of the cards that are input to the assembler is not of direct concern unless cards must be punched, the card format serves as a methodical explanation of some of the features of the assembler, and also can be useful in reviewing lists of program changes that might be provided in the form of a listing of input cards to the assembler.

Column 1 is used for specification of the type of input: blank for a normal input, and otherwise A, L, P, or R as reviewed above. If a change to a log section is provided, "=LOG" appears in columns 1-4 to identify the subsequent information on the card as a log identification.

Columns 2-7 contain the left-justified sequence number.

Column 8 is used to contain printer control information, with values of 0-7 providing the same number of line spacings after the current line is printed (a blank is treated the same as a 1); a value of 8 causing a page eject after the current line is printed; and a value of 9 causing (with R in column 1) the card information in columns 9-48 to be printed in print columns 81-120 with no space since the previous card. If a "9" appears, of course, the sequence number for the card would not appear in the final printout (although it would appear in a list of card changes, naturally).

Columns 9-16 contain the tag of the cell, if any. The information in the tag must observe the constraints on allowable "symbols", since the purpose of the tag is to permit reference to the cell by symbolic means. As pointed out above, a "tag" such as "+2" is essentially a comment, and is ignored by the assembler if it appears in this card field, which is the "location field".

Column 17 may contain a minus sign, in which case the memory information resulting from the remainder of the card will be complemented before being stored.

Columns 18-23 contain the operation code, making use of the appropriate mnemonics assigned to machine-language or interpretive-language orders (Sections IV and VI respectively), or the appropriate assembler pseudo-operations (Section V). In addition to these, however, the following assembler control operations (which do not generate binary memory information) may also be used:

BANK: Set location counter (assembler counter used to determine the assignment of binary memory information to absolute machine addresses) equal to the first unassigned cell in the variable-fixed memory bank specified by the two-digit octal fixed bank number in the address field. If the address field is blank, perform a similar function using the bank of the present location counter setting (generally follows a SETLOC instruction). Cells are assigned in ascending sequence starting from the beginning of each bank, but location counter changes to a different bank must be by an explicit assembler control operation. BANK orders referencing a cell in S3 or S4 cause the Si printout (see SBANK= below) to be changed.

BLOCK: Same function as BANK, but conventionally used for fixed-fixed memory banks (02 and 03). "Blocks" 00 and 01 are erasable memory, 04 is FBANK 00, etc. (cf. Section IIB). A blank BANK card can be used successfully with fixed-fixed memory banks, however.

COUNT: Initiate a count of the number of fixed memory cells, terminating when the next COUNT card is reached, for printout in a table at the end of the listing. The number of cells counted is associated with the tag in the address field (and the previous count, if any, and current total is provided on the printout). The operation COUNT*, if the tag is suitably flagged (e.g. "$$/xxx"), will replace the "$$" with the current fixed memory bank number (as if "$$" had originally been punched in that fashion). See Information at End of Listing.

EBANK=: Set the erasable memory bank portion of the following address pseudo-operation (BBCON, 2CADR, etc., see Section VC) to the erasable memory bank number of the tag in the address field (or to the number in the address field). If the EBANK= is <u>not</u> followed immediately by such an address pseudo-operation, an assembler cell is set to the same value, for use in monitoring machine language references to the erasable memory. This monitoring is reset at the beginning of each log section. The bank being monitored (if any) as of the last line on the previous page of the listing appears in print columns 111-112 (e.g. "E3") of the third line (header log data).

EQUALS (or =): Translate the quantity in the tag field of the card in the same manner as the quantity in the address field of the card (which need not have preceded the EQUALS and which may be an absolute address as well as a symbol). If the address field is blank, the address corresponding to the present value of the location counter (e.g. one greater than the last filled address) is assigned to the tag in the tag field. A distinction sometimes observed in the software is to use "EQUALS" to indicate either a relationship to a previous address ("chaining" of address assignments, useful for erasable memory) or a time-sharing of cells (between thrusting programs and entry guidance, for example); "=", on the other hand, indicates different tags for the same quantity.

ERASE: Allocate erasable memory cells in accordance with the material in the address field. If the address field is blank, one cell is allocated (and location counter advanced); if it is a signed integer (e.g. "+5"), then an additional number of cells (in this example, a total of six, sufficient for a double precision vector) are allocated as specified by this integer. If an unsigned (octal) integer is in the address field, on the other hand, then that absolute erasable memory cell (in ECADR format, see Section VC) is assigned to the tag in the tag field. Allocation of a set of cells can be accomplished in this fashion by ERASE xxx - yyy, where xxx is starting address and yyy the final address.

MEMORY: Allocate memory of the type indicated by location field (functions similarly to ERASE).

SBANK=: Set an assembler control cell to indicate the use of the superbank (i.e. setting of SUPERBNK, channel 07) given by the address field. This setting (in a manner similar to EBANK=, except it is <u>not</u> reset at the start of each log section) appears in print columns 114-115 (e.g. "S3") of the third line (header log data). Address constants such as BBCON and 2CADR (see Section VC), if reference to a cell in S3 or S4 is made (cf. Section IIB), will place the proper SUPERBNK bit setting in bits 7-5; if reference to bank numbers of 27 or less is made, however, these bits will be set to either the most recent SBANK= statement or the last BANK pseudo-operation (whichever was the last to occur). The software is generally arranged so that reference to S3 is made wherever possible. The Si information on the third line, of course, is also changed by the BANK pseudo-operation.

SETLOC: Set location counter to value specified by address field of card, which may be a True Address (see Section IIB) or a symbol. Frequently followed by a BANK card with a blank address field, to facilitate changes to memory bank allocations of the coding (see Information

at Start of Listing). LOC means the same as SETLOC.

SUBRO: Include in the assembly the Subroutine identified by the symbol in the address field: see Information at Start of Listing below.

Column 24 is blank.

Columns 25-40 comprise the normal address field. For machine-language instructions, it may consist of a symbol or a symbol ± an integer (with a space before the sign). A blank means the address of the present step, so that an address of "+2" would mean an address two steps beyond the current step. For interpretive-language orders, the address field contains information as described in Section VI. Values of constants and addresses, of course, appear in the address field too. The information in the address field should end at or prior to card column 40. If the required information is too lengthy to complete in 16 card columns, the number of card columns allocated to the address field may be increased by punching an asterisk following the last character (changing "2DEC" to "2DEC*", for example) of the operation field and another asterisk after the last character of the address information. An asterisk may also be used to obtain special assembler program performance (as mentioned above with COUNT and as also mentioned in Section IIF), or to indicate indexing in the interpretive language (see Section VI).

Columns 41-80 (unless used with the address-field extension technique described for columns 25-40) are used for comments information: the contents of these columns, of course, would not affect the binary information generated by the assembler for the computer memory.

Symbol Reference Information

In the analysis of the performance of the software, it is frequently valuable to be able to identify quickly and reliably all

references to a given tag. Information permitting this to be done is included in print columns 9-26 for those lines of coding with a tag in the address field (for the operations such as 2CADR that generate two lines of coding, the reference information is provided with the first line). The symbol reference information, which is generated for the various assembler control operations as well as for cards that generate binary memory information, has the following print format:

Columns 9-11 contain "REF" (for the serial number of the reference to the tag).

Columns 13-15 contain the serial number of the reference to the tag (starting from the beginning of the listing), with the least significant digit in column 15 and with leading zeros suppressed.

Columns 18-21 contain "LAST" (for the previous time in the listing that the symbol was referenced), provided that columns 13-15 do not contain 1 (if they do, meaning that this is the first reference encountered, the printing of "LAST" is suppressed).

Columns 23-26 contain the master page number (i.e. the one on the top line of the page) where the previous reference (if any) to the symbol was made.

It should be understood that the symbol reference information applies to the symbol in the address field, <u>not</u> to the symbol in the tag field. In order to identify references to the symbol in the tag field, the information printed in the Symbol Table Listing at the end of the program printout may be used (see Information at End of Listing below).

Information at Start of Listing

The first log section of the listing is conventionally titled "Assembly and Operation Information". This log section generally consists solely of remarks information, and hence no binary memory loading information is generated from this segment. Therefore, although the log section is intended to be a convenient source of rapid reference information on the program, it should be clearly understood that this information has no direct effect on the binary memory information. Consequently, unless conscientious management control procedures are enforced the material in this log section can deviate from the actual performance of the software (an observation that applies to all "comments" in the listing, of course). The information generally included in this first log section includes:

A table of Log Sections, giving the various Subroutines in the software and the log sections that comprise them.

A Verb List, giving the various verbs (see Section IIJ) in the software and their numerical codes.

A Noun List, giving the various nouns (see Section IIJ) in the software, their numerical codes, their scaling for the display, and information on their scaling constants.

An Alarm Code list, giving the patterns in the software and their significance.

Checklist and Option Codes, giving the patterns generated by the software to request certain operator actions or decisions, and the significance of each pattern.

It is emphasized once again that this log section is made up solely of remarks cards, and need not be consistent with the actual binary memory information.

The second log section of the listing is conventionally titled "Tags for Relative SETLOC and Blank BANK Cards". This log section is used to assign various portions of the software to different fixed memory banks. This is accomplished by having the software coding itself written so as to specify the assembler location counter value by means of a SETLOC card referencing a tag in this log section, followed by a blank BANK card (see Card Layout above), causing the subsequent binary memory information to be placed in the fixed memory bank dictated by this second log section. The function of this log section, therefore, is to associate a set of tags with appropriate fixed memory banks; it allows absolute memory assignments in the software to be changed (for suitably fine-grained SETLOC and blank BANK cards) without changing the log section in which the software itself appears. This technique also allows some Subroutine information to be identical in different programs, with necessary memory allocation differences handled in this second log section rather than within the individual log sections of the Subroutine. The only binary information generated by this second log section of the listing is conventionally the memory check-sum information, since the BNKSUM operands (see Section IIF) for the different banks are conventionally placed here. In addition to fixed memory bank assignments, some fixed memory tag equivalences can appear in this second log section, as well as some erasable memory bank assignments and tag equivalences reflecting vehicle-peculiar computations.

Following this second log section may be additional log sections for special purposes (such as bank-peculiar constants). The final log section in the front of the program, however, is conventionally titled "Subroutine Calls". During coding, it is convenient to have

the various elements of the software grouped into functions at a higher level than the individual log section. This grouping is accomplished by segmenting the software into groups called "Subroutines" (with a capital "S": if the word appears with a lower-case "s", it has the standard Webster's 1965 definition of "specific instruction(s) whereby a digital computer is guided to perform a precisely defined mathematical or logical operation"). Subroutines are assigned individual names (which are <u>not</u> tags within the Subroutine itself), and software modifications are made on a Subroutine basis (by, of course, specifying log sections to be changed within the Subroutine). The listing flags the last modification(s) made to the Subroutine as described earlier, and an accounting is kept of the serial number of the Subroutine revision (printed with the Subroutine name on line 1 of each page of the Subroutine in the listing).

Subroutines are included in the assembly listing by means of SUBRO cards (see Card Layout above), whose address field is the name of the Subroutine. Each Subroutine, of course, must be compatible with the others as far as memory usage, tag conflict, etc. are concerned (there is no constraint on tag references between Subroutines, nor is there any requirement for special assembler inputs to define such tags). During the course of program development, the SUBRO log section is the final one associated with the complete program, and hence at the end of this log section there is a printout to this effect (such as "*** END OF MAIN PROGRAM ***"). The place for the Subroutine name on line 1 of early pages of the assembly is

filled with "(MAIN)", indicating that no Subroutine is being printed on this part of the listing.

After the program reaches a certain stage in its development, however, it can be desirable to restrict modifications to those which are generated with reference to the complete program, rather than merely to an individual Subroutine. This can be accomplished by suitable assembler control cards, which cause the insertion of an "R" (for Remark) in column 1 of each SUBRO card, thus retaining them in the listing for reference. After this is accomplished, the place for the Subroutine name in line 1 on all pages of the listing is filled with "(MAIN)". This process is known as a "freeze" of Subroutines.

Erasable Memory Information

The next log section in the listing (which can also comprise a Subroutine) is conventionally titled "Erasable Assignments", and gives most of the erasable memory and special register tag assignments to absolute addresses (for convenience, the channel tag assignments also are included). Many tags are assigned octal equivalent addresses by the EQUALS or "=" assembler control operation (see Card Layout above), and in these cases the corresponding S-register contents appear in print columns 33-36. If EBANK is 3 or more for the address (see Section IIB), the quantity "Ei,"(where i is the EBANK number) appears in print columns 30-32.

Other tags are assigned octal equivalent addresses by the ERASE assembler control operation (see Card Layout above), and these have the first address of the "block" (even if only one cell) in print

columns 30-36 and the second in print columns 39-45. As for the assignment of tags by the EQUALS or "=" operation, the "Ei," is suppressed for EBANK values of 0, 1, or 2 (so the address appears in print columns 33-36 and 42-45 only).

The convention is sometimes followed that comments concerning the erasable memory cell use are made in the comments field of the card, such as "B(2)" if two cells are required for the quantity and it is referenced in "basic" (i.e. machine language) coding so that its use is EBANK sensitive; "I(6)" if six cells are required and it is referenced only in "interpretive" coding (so use not EBANK sensitive); "PL(1)" if the quantity is part of a "pad load", needing only 1 cell; etc. As is true of all comments in the listing, however, there is no guarantee that this information necessarily reflects the current status of the software. Other aspects of the listing of erasable memory information (formats, allowable symbols, symbol reference information, etc.) have already been covered.

Fixed Memory Information

Specification of the contents of the fixed memory is the major purpose of the remaining log sections of the listing. The format of the octal information (most of the other portions of this listing have already been described) is as follows:

1. An odd parity bit (to make the sum of the binary ones in the 16-bit memory word, including this bit, an odd value) is given in print position 46 for all words to be loaded into the memory. The only allowable values, of course, that can appear in this print column are 0 and 1.

2. For words loaded into the memory, print positions 33-36 give the contents of the S-register. If the cell address is in variable-fixed memory, the memory bank is in positions 30-31, and a comma appears in print position 32. Words in fixed-fixed memory (banks 02 and 03) have print positions 30-32 blank (and S-register contents in range 4000_8 - 7777_8, cf. Section IIB).

3. For machine language instructions whose operation code is specified completely by bits 15-13 (e.g. those operations which can have addresses in both erasable and fixed memory, see Section IV), the single octal digit of the operation code is in print position 39 and the four octal digits of the operation address are in positions 41-44.

4. For machine language instructions requiring portions of the most significant digit of the (nominal) S-register portion for their specification, and which reference the erasable memory (or a channel), print positions 39-40 contain the two octal digits of the operation and positions 42-44 contain the three octal digits that remain for the address. If the most significant bit of the allowed ten-bit address is a binary 1, the operation code is an odd number (except for the channel operations of Section IVC, only the most significant two bits of the nominal S-register information are used for operation information), and in addition an apostrophe (which may appear as some other character for different print chains) appears in position 41 to emphasize the presence of a binary 1 from the address information in the operation-code octal digits.

5. For address information, constants, and interpretive instructions the five octal digits to be loaded into the memory are printed in print positions 40-44.

6. Addresses generated by assembler functions (BANK, BLOCK, EBANK=, EQUALS, "=", SBANK=, or SETLOC) appear in print positions 30-36 (the full address equivalent of the symbol is given, even though only a portion is functional). Since no binary memory information is generated, no parity bit is printed.

Information at End of Listing

After the final log section of the program, there are several valuable reference tables which give useful information on the program. The first of these is a "Symbol Table Listing", which gives all symbols defined in the program as arranged in the order "sorted" by the assembler (i.e. in order of increasing EBCDIC representation):

. (+ & $ *) - / , ? = A-Z 0-9

Note that, as mentioned previously, some of these characters may appear differently on different print chains (or may not be printed at all).

Given after each symbol is the address (bank register, then S-register: erasable banks of 3 or more are designated by Ei, while those less than 3 can be identified by their S-register contents, less than 1400_8). To the right of the address is given the "health" of the definition, which is blank unless it is defined by EQUALS (or "="), in which case an "=" appears, or if some other difficulty was encountered such as poorly or multiply defined symbols (suitably indicated per table at bottom of each page). To the right of the "health" is given the page number on which the symbol was defined, which of course is the "master" page number appearing on the first line of each page. If the symbol is referenced on several different pages of the program, the next three columns on the page give the total number of references to the symbol, the page number of the first reference, and the page number of the final

reference. If the symbol is only referenced on one page, the page number of the "final" reference is blank, while if it is not referenced at all these three columns are blank. Three symbol columns appear on each page.

If there were undefined symbols detected during the assembly, the table following the "Symbol Table Listing" is the "Undefined Symbol Table Listing", which gives the undefined symbols in the listing, with their "health" (e.g. "UN" for undefined) and the same type of reference information as for the previous table. No address or page number of definition, of course, appears in this table. If there were no undefined symbols, then the printing of this table is suppressed.

Next comes the "Unreferenced Symbol Listing", which lists only those symbols from the "Symbol Table Listing" that are <u>not</u> referenced in the program: this table repeats the information from the first 4 columns (symbol, address, health, and page of definition) of the "Symbol Table Listing". Four symbol columns appear on each page.

Next comes an "Erasable & Equals Cross-Reference Table", which lists all erasable memory tags in the order of increasing erasable memory address: the octal equivalent address is actually used, so that flagword bits and channel mnemonic assignments also appear. Tags assigned to the same octal equivalent address are listed in the order in which they were defined within the assembly (i.e. in order of increasing page number), except those symbols which are equated to the same octal value on the same page of the listing are provided instead in alphabetical order for that page. At the end of the erasable memory tags, those fixed memory tags which are defined by "=" (or EQUALS) assembler operations are shown. Five columns of addresses are given on a page, with each address followed by the page number and associated symbol.

The next table at the end of the listing provides a summary of the addresses assigned ("Reserved") and spare ("Available"), in the form of a "Memory Type & Availability Display", arranged with erasable memory first, followed by fixed-fixed memory and then variable-fixed memory. Figures deduced from this table would differ slightly from numbers obtained from BNKSUM (see Section IIF), since this table includes the two TC orders as "reserved". In addition, this table recognizes erasable memory as "reserved" only by the ERASE assembler operation, so memory cell assignments by "chained" EQUALS cards are not reflected.

Following this table, there is a table which provides information on the number of fixed memory cells that are expended for various functions within the program. Information to make up this table is provided by the COUNT and COUNT* (see Card Layout) cards within the listing. The table lists in order the address-field information associated with the COUNT and COUNT* cards (except that the COUNT* "bank to be specified" information has instead the actual octal fixed memory number inserted): these frequently take the form nn/XXXX, where nn is the fixed memory bank number and XXXX is some convenient mnemonic (normal printing occurs, however, if a COUNT card specified an nn different from that in which the steps involved actually are located). With the address-field information is given the number of references (including both COUNT and COUNT* that results in same "tag"), the first and last pages of the final assembly accumulation of cell counts for that "tag", with the number counted then ("LAST xxx TO yyy: zz"), the total counted for that tag (the same as the final accumulation of cell counts if REF = 1) and finally the cumulative count of cells used since the beginning of the table: the final entry in this last column

in the table, therefore, would give the number of fixed memory cells assigned in the complete program, since provision is made for a "blank" count tag. If more than one reference to a given "tag" occurs, the page number given for the first page of the final accumulation can be checked: on that page will be found (to the left of the COUNT or COUNT* print) the serial number of the reference to the "tag", the previous "span" of counting for that "tag", the number found then, and the number total to that point for the "tag". The count information which is supplied, of course, is only as valid as the original placement of the COUNT and COUNT* cards within the listing, and should be used with caution as an indicator of how many steps would be "saved", for example, if a function with a familiar mnemonic were to be deleted.

After this table comes a list of the "Paragraphs Generated for this Assembly; Address Limits and the Manufacturing Location Code are Shown for Each." The hardware-oriented information presented in this table is given in Section IIF.

Next comes an octal listing of the contents of each Paragraph (256 cells) in the program. Constants and interpretive operations are flagged by "C:" and "I:" respectively before the cell contents (which shows the odd parity bit to the right of the rest of the memory word, separated by a space). The check sum word (the final cell that is wired in the bank) is flagged by "CKSM" before the cell. The checksum is computed by the assembler prior to printing each memory bank, using the same algorithm as described in Section IIF (including stopping when two TC orders to the present step are found). Unwired cells in the memory are flagged by the character "@" (which may appear differently due to other printing hardware), while those cells whose contents were not uniquely defined are suitably flagged.

After the octal listing comes a table which provides for each assigned fixed memory cell the page number in the listing on which the contents of that cell are specified (except for the check sum word itself, see Section IIF, in each bank). This is followed by a list of the Subroutines (if any) that are included in the program, along with their revision numbers.

Finally, there is an indication of whether or not the assembly was satisfactory. If it was, meaning that the assembler program detected no deficiencies, an indication (e.g. "The assembly was good and manufacturable. No lines were cussed.") is provided. If deficiencies ("cussed lines") were detected, the number of these is provided, together with the page number of the first page and last page where faults were noted. Within the listing, each fault is accompanied by information on the reason for flagging as a fault, its serial number, and the page number of the previous fault (unless the previous one is on the same page). Pages III-25 to III-28 give a list of the fault messages appearing in one version of the assembly program, arranged in order of increasing hexadecimal (base 16) serial number of the message (printed to the far right of the line on which the fault message appears in the listing). Also indicated is whether the fault is considered "fatal" (if so, the assembly is considered "unmanufacturable").

Program Changes

Program changes are specified by providing the modification information segregated by individual log sections which are to be modified. The locations of the modifications are specified by the sequence number punched on the card, as described earlier. Deletions

can be accomplished by the pseudo-operation DELETE, with the option of adding "THRU yyyy" to delete the cards (i.e. lines) with sequence numbers ranging from that of the DELETE through yyyy inclusive.

For an extensive insertion, the requirement for punching the sequence number on every card may be avoided by the pseudo-operation INSERT (with BEGIN in the tag field, assigned a suitable sequence number); the end of the insertion is again indicated by INSERT, this time with END in the tag field. Alternatively, "WITH nnnn", rather than a blank, can be provided in the address field of the BEGIN INSERT, in which case the first sequence number of the inserted coding will be that specified. In either case, all subsequent sequence numbers (up through the end of the log section) will be modified so as to count up in the standard "counting position" of these numbers (which allows four digits, i.e. print column 5). If it is merely desired to modify the sequence numbers, this can be accomplished by the CARDNS pseudo-operation.

A capability exists to print revision information by individual Subroutine, reflecting the card inputs which were made to generate the various versions. For such a listing, the first word in the top line is "PRINT*" (as opposed to "ASSEMBLE" for the normal program listing), and instead of the overall program name, the name of the Subroutine is included in the top line (if the overall program name is given, the changes made to the material at the front of the listing, identified by "(MAIN)" on the first line of the program, are supplied). The second printed line for such listings gives the "author" and "date" (preceded by the control characters "./"). A similar listing is generated when the original modification is inserted, and has the first word in the top line as "MODIFY" (or "CREATE", if a new Subroutine being generated). These have "GOOD UPDATE" printed at the bottom if update successful.

Fault Messages Generated by Assembler

Serial	Fatal	Message
		Card Format
01		Queer information in column 17
02		Queer information in column 24
		Erasable Problems
03	x	EBANK/SBANK illegal except with BBCON & 2CADR
04		EBANK conflict with one-shot declaration
		Polish Opcode Problems
05		Erased region should not cross E-banks
06	x	Polish words require blanks in columns 1, 17, & 24
07	x	Previous Polish equation not concluded properly
08	x	Polish push-up requires negative word here
09		Polish address expected here
0A	x	Asterisk illegal on this opcode
0B	x	Interpretive instruction not expected
0C	x	Rt-opcode's mode-in disagrees with mode-out setting
0D	x	Lft-opcode's mode-in disagrees with mode-out setting
0E		Address has no associated Polish opcode
0F	x	Polish address(es) missing prior to this op pair
10	x	Location symbol improper on STADR'ed store word
11	x	Store opcode must be next after "STADR"
12	x	Push-up illegal before store opcode without "STADR"
13		Address words cross over bank or VAC area boundary
14	x	Interpretive address word out of sequence
15	x	Address field should contain a Polish operator
16	x	First Polish operator illegally indexed
17	x	Interpreter opcode requires indexed address here
18	x	Interpreter opcode did not call for indexing
19	x	Second Polish operator illegally indexed
1A	x	Can not handle neg addresses with indexing here

Serial	Fatal	Message
		Numeric Constant Problems
1B		More than 14 octal digits in octal constant
1C		More than 10 digits in decimal constant
1D		Fractional part lost by truncation
1E	x	Range error in constant field
1F		Inexact decimal-to-binary conversion
20		Double precision constant should not cross banks
21		No "D" in decimal number
		Merge Control Problems
22	x	Subroutine name not recognized
23		Multiple calls in one program or subroutine
24	x	Card ignored because it makes memory table too long
25	x	Card ignored because it's too late in the deck
26	x	Conflict with earlier head specification
27		Card number out of sequence
28	x	No match found for second card number
29	x	First card number not less than second
2A	x	No match found for card number or acceptor text
		General Address Field Problems
2B		Blank address field expected
2C	x	" " is undefined
2D	x	" " was undefined in pass1
2E		" " should be symbolic
2F	x	" " was nearly defined by equals
30	x	" " was nearly defined by equals in pass1
31	x	" " given multiple definitions
32	x	" " multiply defined including by equals
33	x	" " multiply defined including nearly by ='s
34	x	" " given oversize definition
35	x	" " associated with conflict
36	x	" " associated with multiple errors
37	x	" " associated with wrong memory type
38	x	" " is in miscellaneous trouble

Serial	Fatal	Message
39		Address is inappropriate for opcode
3A	x	Address is in bank 00 (filled in with bank number)
3B	x	Address depends on unknown location
3C		Irregular but acceptable address
3D	x	Address field is meaningless
3E	x	Addr. must be basic single-precision constant or inst
3F	x	Range error in value of address
40	x	Indexing is illegal here
		Opcode Field Problems
41	x	Illegal or mis-spelled operation field
42		This instruction should be indexed
43	x	This instruction should be extended
44	x	This instruction should not be extended
		Predefinition Problems
45	x	" " shouldn't have been predefined
46	x	Attempt to predefine location symbol failed
		Location Field Problems
47		Illegal location field format
48		Location field should be blank
49	x	Location is in wrong type of memory
4A	x	Numeric location field is illegal here
4B	x	Oversized or ill-defined location
4C	x	Conflict in use of this location
4D	x	" " won't fit in symbol table
4E	x	No such bank or block number in this machine
4F	x	This bank or block is full
		Leftover Problems
50	x	" " is indefinably leftover
51	x	Leftover won't fit in memory
52	x	Improper leftover location field format

Serial	Fatal	Message
		More Cusses
53		Queer information in column 1
54		Address field arithmetic not allowed here
55		Address constant not expected here
56		Address constant expected here
57		Count table full. Address field ignored.
58	x	BBANK type constants require preceding EBANK=
59		One shot SBANK= above was not needed
5A		Address OO,0000 (filled in with address)
5B		"STADR" unnecessary
5C		Assembler finds error but has no specific cuss for it
5D	x	Address is in super bank 0 (filled in with bank)

IV MACHINE LANGUAGE INSTRUCTIONS

IVA General Principles

There are 34 machine-language operation codes which may be performed under program control (the operation EDRUPT is conventionally excluded from the list of operation codes, and is discussed in Section IIH rather than here). Use of special addresses with certain of these instructions permits an additional four special functions (EXTEND, INHINT, RELINT, and RESUME, see Section VA) to be performed, and use of addresses 0020_8 - 0023_8 permits shifting operations to be performed, as described in Section IID. Of the 34 instructions, 15 may be classified as "regular" orders and the remaining 19 as "extended" (or "extra code") orders. The extended orders must be written as two lines of coding (occupying two fixed memory cells, cf. Section IIB), with the first line setting a special bit in the instruction register (by the EXTEND operation: the bit is normally reset after the instruction is performed) and the second line giving the order itself. Without the EXTEND, the second line would be interpreted as a regular order.

Only a few of the instructions can be used with operands in both erasable and fixed memory, since most of them achieve an effective extension of the operation code bits by using the most significant two bits of the 12 bits nominally assigned to the S-register. The seven channel instructions (see Section IIE), however, use the most significant three bits of the nominal S-register information to determine the operation to be performed.

The following two sections list the operation codes in alphabetical sequence, with Section IVB giving the regular orders and Section IVC

giving the extended orders. The symbols A, L, and Q refer to the arithmetic registers defined in Section IIC, while quotation marks around a symbol signify that the value of the address is of interest, rather than the information stored in that address. The following special symbols are also used:

E means an address in the range (S-register) 0000_8 - 1777_8, i.e. a hardware register or an erasable memory cell.

F means an address in the range (S-register) 2000_8 - 7777_8, i.e. a cell in fixed memory.

H means a channel (see Section IIE).

K means an address in the range (S-register) 0000_8 - 7777_8, i.e. a hardware register, an erasable memory cell, or a cell in fixed memory.

N means the address of the step now being performed (i.e. the one containing the operation code being described).

The term "erasable memory" is used in Sections IVB and IVC to signify either an erasable memory cell or a hardware register.

The value shown in the "Operation" column is the operation code that appears in the program listing: if two values appear, the choice between them depends on the value of the address, as discussed in Section III. A parenthetical 1 is used with the orders in Section IVC to emphasize the need for having the extended-order flip-flop set by the EXTEND operation.

The table on the following page summarizes the machine language orders by operation value, separated into the "regular orders" and the "extended orders".

Machine Language Orders

Regular Orders

0	TC
	Address 3 = RELINT
	Address 4 = INHINT
	Address 6 = EXTEND
10-11	CCS
12-17	TCF
20-21	DAS
22-23	LXCH
24-25	INCR
26-27	ADS
3	CA
4	CS
50-51	INDEX
	Address 17 = RESUME
52-53	DXCH
54-55	TS
56-57	XCH
6	AD
7	MASK

Extended Orders

00	READ
01	WRITE
02	RAND
03	WAND
04	ROR
05	WOR
06	RXOR
07	EDRUPT
10-11	DV
12-17	BZF
20-21	MSU
22-23	QXCH
24-25	AUG
26-27	DIM
3	DCA
4	DCS
5	INDEX
60-61	SU
62-67	BZMF
7	MP

IVB Regular Orders

Mnemonic	Operation	Performance
AD "K"	6	Add. Two MCT (23.4 microsec). Address can be in erasable or fixed memory. A = A + K If "K" = "A", this doubles the contents of the accumulator. If "K" = 20_8 - 23_8, K modified as described in Section IID.
ADS "E"	26 27	Add and Store. Two MCT (23.4 microsec). Address can be in erasable memory only. A = A + E E = A If "E" = 20_8 - 23_8, the value stored in "E" modified as described in Section IID.
CA "K"	3	Clear and Add. Two MCT (23.4 microsec). Address can be in erasable or fixed memory. A = K If "K" = 20_8 - 23_8, K modified as described in Section IID.
CCS "E"	10 11	Count, Compare, and Skip. Two MCT (23.4 microsec). Address can be in erasable memory only. Load A with |E| - 1, limited $\geq$ +0, and skip 0 (E > 0), 1 (E = +0), 2 (E < -0), or 3 (E = -0) steps. Overflow bit can be sensed. If E is positive non-zero (in range 00001_8 - 37777_8): A = E - 1 (if E = 1, A = +0) Proceed to "N" + 1 (the next step) If E is +0 (00000_8): A = +0 Proceed to "N" + 2 (skipping one step)

Mnemonic	Operation	Performance
CCS "E" (cont)		If E is negative non-zero (in range 77776_8 - 40000_8): A = -1 - E (if E = -1, A = +0) Proceed to "N" + 3 (skipping two steps) If E is -0 (77777_8): A = +0 Proceed to "N" + 4 (skipping three steps) If "E" = 20_8 - 23_8, E modified as described in Section IID.
CS "K"	4	Clear and Subtract. Two MCT (23.4 microsec). Address can be in erasable or fixed memory. A = - K If "K" = "A", this complements the accumulator. If "K" = 20_8 - 23_8, K modified as described in Section IID.
DAS "E"	20 21	Double Precision Add and Store. Three MCT (35.2 microsec). Address can be in erasable memory only. $E_{dp} = E_{dp} + (A, L)$ If "E" $\neq$ "A" or "L": L = +0 If "E" $\neq$ "A": A = +0 if no overflow A = 1 sgn A if overflow If "E" = "A", this doubles the double precision number in (A, L). If "E" = 20_8 - 23_8, E modified as described in Section IID: also true if "E" +1 does.
DXCH "E"	52 53	Double Precision Exchange. Three MCT (35.2 microsec). Address can be in erasable memory only. Set E_{dp} = (A, L) and (A, L) = E_{dp}. Overflow lost in L. If "E" = "Z", see DTCB in Section VA. If "E" = "L", A ends up in L, L ends up in Q, and Q ends up in A (see Section IIG). If "E" = 20_8 - 23_8, or if "E" +1 does, modification takes place as described in Section IID.

Mnemonic	Operation	Performance
INCR "E"	24 25	Increment. Two MCT (23.4 microsec). Address can be in erasable memory only. E = E + 1 (if E was -1, result is -0) If "E" = 20_8 - 23_8, E modified as described in Section IID. Unless "E" = "A", A not affected. If "E" = 0025_8 - 0030_8 and overflow occurs, action takes place as described in Section IID when such overflow occurs.
INDEX "E"	50 51	Index using Erasable. Two MCT (23.4 microsec). Address can be in erasable memory only. If "E" = 17_8, see Section VA (RESUME). Otherwise, add E to the contents of "N" +1 and use the resulting instruction as the next one to be performed. The order code can be changed (including bit 15, the "sign"), but overflow will not cause an extended order, nor will overflow change bit 15 of the original operand. If "E" = 20_8 - 23_8, E modified as described in Section IID.
LXCH "E"	22 23	Exchange L Register. Two MCT (23.4 microsec). Address can be in erasable memory only. Set E = L and L = E. Overflow lost in loading L. If "E" = 20_8 - 23_8, the value stored in "E" modified as described in Section IID.
MASK "K"	7	Mask. Two MCT (23.4 microsec). Address can be in erasable or fixed memory. Replace A with the logical "and" of A and K: where the corresponding bits of both A and K are 1, a 1 is placed in that bit position of A; where the corresponding bits of A and K have at least one binary 0, a 0 is placed in that bit position of A. K is not disturbed.

Mnemonic	Operation	Performance
TC "K"	0	Transfer Control. One MCT (11.7 microsec). Address can be in erasable or fixed memory. If "K" = 3, 4, or 6, see Section VA (RELINT, INHINT, and EXTEND respectively). Otherwise, set Q = Z and take the step at "K" as the next instruction (proceeding from that point). If "K" is outside the range 1400_8 - 3777_8, the next instruction is unique; otherwise, it is determined by EBANK, FBANK, and/or SUPERBNK (see Section IIB). The information in BBANK is <u>not</u> affected by this instruction, and Q is loaded with "N" + 1 (S-register portion).
TCF "F"	12-17	Transfer Control to Fixed Memory. One MCT (11.7 microsec). Address can be in fixed memory only. Take the step at "F" as the next instruction (proceeding from that point). See TC for discussion of effect of FBANK and/or SUPERBNK. Q is not disturbed.
TS "E"	54 55	Transmit to Storage. Two MCT (23.4 microsec). Address can be in erasable memory only. E = A If "E" = "A", skip next instruction if overflow. If "E" $\neq$ "A": A = 1 sgn A if overflow, and skip next instruction If no overflow, A left alone and next instruction is performed. If "E" = 20_8 - 23_8, the value stored in "E" modified as described in Section IID.
XCH "E"	56 57	Exchange. Two MCT (23.4 microsec). Address can be in erasable memory only. Set E = A and A = E If "E" = 20_8 - 23_8, the value stored in "E" modified as described in Section IID.

The execution times given below for the extended orders includes the one MCT (11.7 microsec) for the EXTEND operation.

Mnemonic	Operation	Performance
EXTEND AUG "E"	(1)24 (1)25	Augment Magnitude. Three MCT (35.2 microsec). Address can be in erasable memory only. E = E + 1 sgn E (+0 is plus, -0 is minus). If "E" = 20_8 - 23_8, E modified as described in Section IID. Unless "E" = "A", A not affected. If "E" = 0025_8 - 0030_8 and overflow occurs, action takes place as described in Section IID when such overflow occurs.
EXTEND BZF "F"	(1)12- (1)17	Branch on Zero to Fixed. Two MCT (23.4 microsec) if branch, and three MCT (35.2 microsec) if do not branch. Address can be in fixed memory only. If A = $\pm$ 0 (including overflow information): Take step at "F" as the next instruction, proceeding from that point. Otherwise, perform instruction at "N" +1. See TC for discussion of effect of FBANK and/or SUPERBNK.
EXTEND BZMF "F"	(1)62- (1)67	Branch on Zero or Minus to Fixed. Two MCT (23.4 microsec) if branch, and three MCT (35.2 microsec) if do not branch. Address can be in fixed memory only. If A = $\pm$ 0 or negative (including overflow information): Take step at "F" as the next instruction, proceeding from that point. Otherwise, perform instruction at "N" +1. See TC for discussion of effect of FBANK and/or SUPERBNK.

Mnemonic	Operation	Performance
EXTEND DCA "K"	(1)3	Double Precision Clear and Add. Four MCT (46.9 microsec). Address can be in erasable or fixed memory. (A, L) = K_{dp}. Overflow lost in L. If "K" = 20_8 - 23_8, or if "K" +1 does, modification takes place as described in Section IID. If "K" = "L", Q is loaded into L and A.
EXTEND DCS "K"	(1)4	Double Precision Clear and Subtract. Four MCT (46.9 microsec). Address can be in erasable or fixed memory. (A, L) = - K_{dp}. Overflow lost in L. If "K" = "A", this complements (A, L). If "K" = 20_8 - 23_8, or if "K" +1 does, modification takes place as described in Section IID.
EXTEND DIM "E"	(1)26 (1)27	Diminish Magnitude. Three MCT (35.2 microsec). Address can be in erasable memory only. If E = $\pm$ 0, E not modified (unless "E" = 20_8 - 23_8). Otherwise, E = E - 1 sgn E (If |E| = 1, -0 → E) If "E" = 20_8 - 23_8, E modified as described in Section IID. Unless "E" = "A", A not affected.
EXTEND DV "E"	(1)10 (1)11	Divide. Seven MCT (82.0 microsec). Address can be in erasable memory only. Divide (A, L) by E, and leave quotient in A and remainder in L. Improper results obtained if "E" = "L" or if any operand has overflow bit set (including L). Sign agreement of (A, L) need not exist before the division is performed. If E = $\pm$ 0, A = $\pm$ MAX (37777_8 or 40000_8) E is not disturbed (unless "E" = "A", in which case sign is reversed if A is positive).

Mnemonic	Operation	Performance
EXTEND EDRUPT "F"	(1)07	See Section IIH.
EXTEND INDEX "K"	(1)5	Index and Extend. Three MCT (35.2 microsec). Address can be in erasable or fixed memory. Add K to the contents of "N" +1 and use the resulting extended-order instruction as the next one to be performed: note that the EXTEND propagates through this INDEX order to affect the following order also. Otherwise, performance like INDEX (order code can be changed, etc.), except that address $0017_8 \neq$ RESUME. If "K" = 20_8 - 23_8, K modified as described in Section IID.
EXTEND MP "K"	(1)7	Multiply. Four MCT (46.9 microsec). Address can be in erasable or fixed memory. Multiply A by K, leaving the most significant half of the product in A and the least significant half of the product in L. The signs of A and L agree. Improper results obtained if either operand has overflow bit set. A zero-magnitude product will be +0 unless the original contents of A were $\pm$ 0 and K was non-zero and of the opposite sign to A. K is not disturbed (unless "K" = "A" or "L").
EXTEND MSU "E"	(1)20 (1)21	Modular Subtract. Three MCT (35.2 microsec). Address can be in erasable memory only. Replace A by the signed ones complement difference (A - E), where both A and E operands are treated as twos complement numbers. The twos complement difference is formed, and the result decremented by 1 if the sign is minus (indicating that the difference angle, for B-1 revolutions scaling, is at least 180^o). If E = +0, effect is to convert A to ones complement; if "E" = "A", +0 left in A, as is also true if E = A. The S_2 bit is set to S_1 (see Section IIG), so no overflow after the operation would be observed.

Mnemonic	Operation	Performance
EXTEND MSU "E" (cont)		If "E" = 20_8 - 23_8, E modified as described in Section IID.
EXTEND QXCH "E"	(1)22 (1)23	Exchange Q Register. Three MCT (35.2 microsec). Address can be in erasable memory only. Set E = Q and Q = E. If "E" = 20_8 - 23_8, the value stored in "E" modified as described in Section IID.
EXTEND RAND "H"	(1)02	Read Masked Channel. Three MCT (35.2 microsec). Address can be a channel only, except that 34 and 35 give a 0 result. Replace A with the logical "and" of A and H: see discussion with MASK instruction.
EXTEND READ "H"	(1)00	Read Channel. Three MCT (35.2 microsec). Address can be a channel only, except that 34 and 35 give a 0 result. Set A = H.
EXTEND ROR "H"	(1)04	Read ORed Channel. Three MCT (35.2 microsec). Address can be a channel only, except that 34 and 35 give a 0 result when sensed, leaving A undisturbed. Replace A with the logical "or" of A and H: where the corresponding bits of both A and H are 0, a 0 is placed in that bit position of A; where the corresponding bits of A and H have at least one binary 1, a 1 is placed in that bit position of A.
EXTEND RXOR "H"	(1)06	Read Exclusive-ORed Channel. Three MCT (35.2 microsec). Address can be a channel only, except that 34 and 35 give a 0 result when sensed, leaving A undisturbed. Replace A with the logical "exclusive or" of A and H: where the corresponding bits of A and H are different, a 1 is placed in that bit position of A; where the corresponding bits of A and H are the same (binary 1 or 0), a 0 is placed in that bit position of A. Gives effect of a bit-by-bit "add without carries".

Mnemonic	Operation	Performance
EXTEND SU "E"	(1)60 (1)61	Subtract. Three MCT (35.2 microsec). Address can be in erasable memory only. A = A - E If "E" = 20_8 - 23_8, E modified as described in Section IID.
EXTEND WAND "H"	(1)03	Write Masked Channel. Three MCT (35.2 microsec). Address can be a channel only. If "H" = 34_8 or 35_8, zero is loaded into these channels and A. If "H" = 33_8, order performs as described for loading A, and bits 15-11 (flip-flops) of this channel are reset (to binary 1 as sensed). For other read-type channels, loading has no effect. In loading channel, all bits set zero briefly. See Section IIE. Replace A with the logical "and" of A and H (see discussion with MASK instruction), and then write A into H.
EXTEND WOR "H"	(1)05	Write ORed Channel. Three MCT (35.2 microsec). Address can be a channel only. See discussion with WAND for channels 33-35 and read/write channels: if "H" = 34_8 or 35_8, A left alone. Replace A with the logical "or" of A and H (see discussion with ROR instruction), and then write A into H.
EXTEND WRITE "H"	(1)01	Write Channel. Three MCT (35.2 microsec). Address can be a channel only. See discussion with WAND for channel 33 and read/write channels. Write A into H.

IVD Machine Language Examples

The following examples of machine language coding have been fabricated to illustrate some of the basic principles of the different machine language instructions, rather than necessarily to illustrate efficient coding techniques or to perform meaningful calculations. The notation VAR is used for "variable" (quantity in erasable memory) and CON for "constant" (quantity in fixed memory). Transfer orders go to STEP, the tag for a program step (likewise assumed to be in fixed memory, either in the same bank as the coding which transfers to it or in fixed-fixed memory). Unless otherwise stated, it is assumed that the numbers entering the computations are scaled so that there is no risk of overflow.

1. VAR = VAR + CON

```
CA      VAR
AD      CON
TS      VAR
```

or

```
CA      CON
ADS     VAR
```

2. VAR = VAR - CON

```
CS      CON
ADS     VAR
```

or

```
CA      CON
EXTEND
SU      VAR         SU order is for erasable memory only.
CS      A           Complements accumulator (same as COM).
XCH     VAR         Does not skip next order if overflow.
```

3. VAR = CON - 1 (CON positive non-zero integer)

```
    CA      CON
    CCS     A           CCS order is for erasable memory only.
    TS      VAR
        or
    CS      CON
    CCS     A
    ---                 Transfer to here is impossible since CON was
    ---                 positive non-zero. These cells could be used
                        to store constants, or transfer to error routine.
    TS      VAR
        or
    CS      CON
    INCR    A
    CS      A
    TS      VAR
        or
    CA      CON
    EXTEND
    DIM     A           If desired to set VAR = VAR -1, could do
    TS      VAR         EXTEND  DIM VAR directly.
```

4. VAR = |VAR| - 1, limited $\geq$ +0

```
    CCS     VAR
    TS      VAR         If positive non-zero originally, to here.
    TS      VAR         If +0 originally, to here.
    TS      VAR         If negative non-zero originally, to here.
    TS      VAR         If -0 originally, to here.
        or
    CCS     VAR
    TCF     STEP
    TCF     STEP
    TCF     STEP        This same as NOOP.
STEP TS     VAR
```

5. VAR = VAR + 1 sgn VAR If VAR is zero magnitude, consider positive.

```
        CA      VAR         Note that EXTEND AUG VAR cannot be used since
                            AUG gives -1 if E = -0.
        EXTEND
        BZF     STEP1       Branch on +0 or -0.
        EXTEND              Must be repeated.
        BZMF    STEP2
STEP1   AD      ONE         To here also if was positive non-zero.  ONE is
                            a constant, 00001(8).
        TCF     STEP3
STEP2   CS      A
        AD      ONE
        CS      A           Net effect same as A - 1.
STEP3   TS      VAR
            or
        CCS     VAR
        AD      ONE         Cancel subtraction of 1 by CCS.
        TCF     STEP
        AD      THREE       Constant, value 00003(8), added if original
                            number negative.
        CS      A           A now is VAR - 2 if original VAR negative non-
                            zero or -0 if original VAR was -0.
 STEP   AD      ONE
        TS      VAR
```

6. Set cells VAR through VAR+11 to zero

```
        CA      ELEVEN      Constant, value 11.
 STEP   TS      TEMP        Temporary storage cell, for indexing/counting.
        CA      7           Absolute address 00007(8) (see Section IIC).
        INDEX   TEMP        First time makes next order TS VAR+11, then
                            TS VAR+10, etc.
        TS      VAR
        CCS     TEMP        Will perform total of 12 iterations.
        TCF     STEP        Decremented value of TEMP in A, ready for storing.
```

or

```
     CA     ELEVEN
STEP LXCH   7          Same as ZL.
     INDEX  A
     LXCH   VAR
     CCS    A
     TCF    STEP
```

or

```
     CA     TEN        Constant, value 10.
STEP TS     TEMP
     EXTEND
     DCA    DPZERO     Double precision zero.
     INDEX  TEMP       First time through makes next order DXCH VAR+10
                       (causing VAR+10 and VAR+11 to be zeroed), next
                       time DXCH VAR+8, etc.
     DXCH   VAR
     EXTEND
     DIM    TEMP       IF TEMP = 0, has no effect.
     CCS    TEMP
     TC     STEP       TEMP counted down by two's (10,8,6,4,2,0).
```

7. VAR = $\frac{1}{4}$ VAR (shifted right two places)

```
     CA     VAR
     XCH    SR         Shift right register, cell 0021_8 (Section IID).
     AD     A          Same as DOUBLE.
     LXCH   A
     CS     SR         Sensing operation causes another shift.
     LXCH   SR         Puts SR register original contents back,
                       and puts VAR shifted right 2 places in L.
     LXCH   VAR
```

or (if desire to round quantity)

```
     CA     VAR
     EXTEND
     MP     BIT13      Constant, value 10000_8.
     XCH    L          Could also do LXCH A.
     AD     A          Doubles number obtained from L (which is the
                       least significant half of product). If result
                       overflows, rounding is required.
```

```
TS      7           Need to check for overflow and change contents
                    of A (see Section IIC).
TCF     +2          This step not performed if overflow obtained;
                    it causes the next step to be skipped.
ADS     L           A has ± 1 (from TS order), i.e. "overflow" data.
LXCH    VAR
```

The CYR register (address 0020_8) could have been used only if it was known that bits 1 and 2 of VAR were the same as the sign bit: CYR is primarily intended for logical manipulations rather than arithmetic computations.

8. VAR = 4 VAR (shifted left two places)

```
CA      VAR
AD      VAR
AD      A           Same as DOUBLE.
TS      VAR
```

or

```
CA      VAR
TS      CYL         Cycle left register, cell 0022₈.
CA      CYL         Dummy sensing operation to force another shift.
CA      CYL
TS      VAR
```

or

```
CA      VAR
LXCH    7           Same as ZL (needed since division uses A and L).
EXTEND
DV      BIT13E      Constant, 10000₈, previously set into erasable.
TS      VAR
```

or

```
CA      VAR
EXTEND
MP      BIT3        Constant, 00004₈, causing movement of VAR right
                    by 12 places.
LXCH    VAR
```

9. $VAR_{dp} = VAR_{dp} + CON_{dp}$

```
EXTEND
DCA     CON
DAS     VAR
```

or

```
EXTEND
DCS     MCON        If constant stored as complement of true value.
DAS     VAR
```

or

```
EXTEND
DCS     CON
EXTEND              Must be repeated.
DCS     A           Same as DCOM.
DAS     VAR
```

or

```
CA      VAR +1      Least significant half.
AD      CON +1
TS      VAR +1      If overflow, next order skipped and overflow
                    data left in A.
CA      7           Zero.
AD      VAR
AD      CON
XCH     VAR         Avoids skip if overflow.
```

10. Transfer to STEP if bit 9 or 10 of VAR = 1, otherwise proceed

```
CA      9OR1O       Constant, 01400_8.
MASK    VAR
CCS     A
TCF     STEP
```

or

```
CA      VAR         If do QXCH VAR, value of VAR destroyed.
EXTEND
QXCH    A           Could also have done TS Q.
CA      9OR1O
EXTEND
RAND    QCHAN       Same as Q.
```

```
CCS     A           CCS  Q would not work here, since RAND leaves
                    answer in A.
TCF     STEP
    or
CA      VAR
EXTEND
WRITE   LCHAN       Same as L.
CA      9OR1O
EXTEND
WAND    LCHAN
EXTEND              Must be repeated.
READ    LCHAN       (Redundant since L already in A from WAND).
CCS     A
TCF     STEP
```

11. Transfer to STEP if bits 7-1 of VAR are all 1, otherwise proceed

```
CS      VAR         Gives complement of VAR in A.
MASK    LOW7        Constant, 00177_8.
EXTEND
BZF     STEP
    or
CA      VAR
MASK    LOW7
EXTEND
AUG     A           Leaves bit 8 = 1 if low 7 bits all 1.
TS      EDOP        Edit operand register, cell 0023_8.
CCS     EDOP        Will be 00001_8 if low 7 bits were all 1.
TC      STEP
    or
LXCH    7           Same as ZL.
CA      VAR
EXTEND
WOR     LCHAN       Since was zero at start, same as WRITE.
CA      LOW7
EXTEND
ROR     LCHAN       Bits 7-1 of A remain at 1.
```

```
        EXTEND
        RXOR    LCHAN       Any bits set 1 by ROR set to zero again in A,
                            as are those bits of 7-1 in A that are 1 in L:
                            ROR could have left some of bits 15-8 at 1.
        EXTEND
        BZF     STEP
```

12. VAR = CON - VAR, with ones complement difference of single precision twos complement vectors being formed.

```
        CA      TWO         Constant, value 00002(8).
STEP    TS      TEMP        For counting and indexing.
        INDEX   A
        CA      CON
        EXTEND
        INDEX   TEMP        The EXTEND "carries through" to the MSU: note
                            that instructions must appear in this sequence
                            or the INDEX would change the EXTEND to a
                            transfer order (see Section VA).
        MSU     VAR
        INDEX   TEMP
        TS      VAR
        CCS     TEMP
        TCF     STEP
```

13. Save routine's return address (in Q register), then perform a subroutine starting at STEP to add quantity at calling address +1 to VAR, and then return to calling address +2.

```
        EXTEND
        QXCH    TEMP
        TC      STEP        TCF would not work, since it does not load Q.
        DEC     xxxx        Constant to be added.
        EXTEND
        QXCH    TEMP        Restore Q register.
        ---                 (proceed)
STEP    INDEX   Q
        CA      0           Obtain value of constant.
        ADS     VAR
        INDEX   Q
        TC      1           Return to (address in Q) +1, or calling address +2.
```

14. If bit 5 of FLAGWRD3 = 1, set VAR = CON1; if it is 0, set VAR = CON2.

```
CA      FLAGWRD3
MASK    BIT5        Constant, value 00020(8).
CCS     A
EXTEND              Skipped if bit 5 = 0.
DCA     CON1        If EXTEND not done, becomes CA  CON2 (CON1 and CON2
                    stored in consecutive memory cells), since assembler
                    increments address so that least significant half
                    of double precision operand taken first.
TS      VAR
```

15. VAR = VAR - CON, limited $\geq$ 0 (CON > 0)

```
CS      CON
AD      VAR
AD      POSMAX      Constant, value 37777(8).
TS      VAR
CS      VAR         This line skipped if  VAR - CON ≥ 00001(8), so
                    that addition of 37777(8) forced overflow (and
                    left A = 00001(8)).
ADS     VAR         If overflowed above, A = 00001(8) which, for the
                    word length, cancels out the POSMAX addition.
                    If did not overflow, VAR = VAR - VAR, which = 0.
```

V SPECIAL ASSEMBLER OPERATIONS

VA Equivalent Machine Language Instructions

The assembler has the capability of recognizing several operation codes in addition to the "standard" ones listed in Section IV. Most of these additional codes are alternate mnemonics which happened to be considered useful by the G&N contractor, rather than a reflection of supplemental hardware capability. Four of the codes, however, do represent special hardware performance (EXTEND, INHINT, RELINT, and RESUME).

Some of the additional mnemonics which may be encountered in the listing are given below. The octal information generated by the assembler for these additional mnemonics is identical to that which would be produced for the coding in the "equivalent" column. Use of these additional codes, of course, is not a requirement when generating coding. Functions performed for mnemonics other than those listed can be determined from the octal codes given in Section IV. In the tabulation below, the symbol "--" means "not provided" (true of addresses for many of the operation codes), and "E", "F", and "K" are defined in Section IVA.

Additional Operation Mnemonic	Equivalent Octal Information	Function
CAE "E"	CA "E"	Clear and Add from Erasable. In the listing, CAE is shown as 30 xxx or 31'xxx (CA is 3 xxxx).
CAF "F"	CA "F"	Clear and Add from Fixed.
COM --	CS "A"	Complement accumulator contents.
EXTEND	EXTEND	Double Precision Complement (i.e. (A, L)).
DCOM --	DCS "A"	

Additional Operation Mnemonic	Equivalent Octal Information	Function
DDOUBL --	DAS "A"	Double Precision Double (i.e. double (A, L)).
DOUBLE --	AD "A"	Double accumulator contents.
DTCB --	DXCH "Z"	Exchange BBANK with L register and Z register with accumulator. Serves as a "double precision transfer order", with the additional feature of retaining in (A, L) return address information. See 2CADR in Section VC.
DTCF --	DXCH "FBANK"	Exchange Z register with L register and FBANK with accumulator. Serves as another type of "double precision transfer order", differing from DTCB in that EBANK data is not changed or retained. See 2FCADR in Section VC.
EXTEND --	TC 0006_8	Cause an extended order (see Section IVC) to be performed as the next step, by setting the extended order flip-flop in the instruction register (flip-flop would be reset after the instruction performed, unless the next instruction is INDEX, see Section IVC). Note that this is not the usual significance of the TC order, but no capability is lost since 0006_8 is BBANK address.
INHINT --	TC 0004_8	Cause a flip-flop to be set that results in having program interrupts inhibited (no effect if generated during a program interrupt, see Section IIH). Note that this is not the usual significance of the TC order, but no capability is lost since 0004_8 is FBANK address.
MSK "K"	MASK "K"	Alternate notation.
NDX "K" or "E"	INDEX "K" or "E"	Alternate notation.
NOOP --	TCF +1 (fixed) CA "A" (erasable)	No operation: program takes next order in sequence without disturbing registers.
OVSK --	TS "A"	Overflow skip: skip next instruction if accumulator overflow is present, but leaves accumulator contents undisturbed. If it is desired also to set accumulator, this can be done by TS 7 (see Section IIC).

Additional Operation Mnemonic	Equivalent Octal Information	Function
RELINT --	TC 0003_8	Cause a flip-flop to be reset that results in having program interrupts released (i.e. enabled) again: it resets the flip-flop that is set by the INHINT command. Note that this is not the usual significance of the TC order, but no capability is lost since 0003_8 is EBANK address.
RESUME --	INDEX "BRUPT"	Trigger the performance of the instruction hardware sequence that causes the program performance to be resumed after the computations required to satisfy a program interrupt have been completed (see Section IIH). The Z register is loaded with ZRUPT (cell 0015_8), and BRUPT (cell 0017_8) is taken as the next order. Note that this is not the normal significance of the INDEX order, but no capability is lost since BRUPT is intended, in general, for hardware rather than software loading (see Section IID).
RETURN --	TC "Q"	Return from a subroutine entered by a TC order (provided FBANK and SUPERBNK are proper and no additional TC commands, or other loadings of Q, took place).
EXTEND SQUARE --	EXTEND MP "A"	Square the contents of the accumulator.
TCAA --	TS "Z"	Transfer Control to Address in A.
TCR "K"	TC "K"	Alternate notation (Transfer Control setting up Return).
XLQ --	TC "L"	Execute instructions in L and Q. The Q register, of course, would be loaded with return address information by the TC, so effect is merely to execute the order in L and then return.
XXALQ --	TC "A"	Execute instructions in A, L, and Q. See XLQ, except here would execute orders in A and L: if A = 00006_8, then contents of L would be an extended order.
ZL --	LXCH 0007_8	Zero L register (see Section IIC for cell 0007_8 information).
EXTEND ZQ --	EXTEND QXCH 0007_8	Zero Q register.
0-7	Considered to be TC, CCS, DAS, CA, CS, INDEX, AD, MASK respectively.	

VB Representation of Numbers

The value of numbers (generally considered to be "constants") processed by the assembler may be specified in several different ways. This section is concerned with representations of decimal and octal numbers, as well as special combinations (such as the display system verb and noun information), while Section VC is devoted to a discussion of various address representation methods.

Decimal Numbers

Single precision decimal numbers (those to be stored in one cell of 14 magnitude bits and a sign bit) are specified by the operation DEC. Double precision decimal numbers are stored in two consecutive memory cells, each with 14 magnitude bits and the same sign bit (negative numbers are stored in ones complement form in the software, unless otherwise specified), and are specified by the operation 2DEC. The value of the constant may be specified in several different ways:

a) As a simple decimal quantity less than 1 (e.g. ".2"). In this case, it is converted to a binary number with scale factor B0 (see Appendix A), so that the most significant magnitude bit corresponds to 2^{-1}, the next bit 2^{-2}, etc.

b) As a simple decimal integer (e.g. "200" or "200."). In this case, it is converted to a binary number so scaled that the least significant bit of the constant (whether single or double precision) corresponds to the value of "1". This means that for single precision the binary number has a scale factor B14 and for double precision the binary number has a scale factor B28.

c) As a decimal quantity (integer or fraction, with or without a fractional part) together with an indicated binary scale factor (e.g. "200 B-9" or ".1 B2"). In this case, it is converted to a binary number with scale factor given by the complement of the quantity following the "B": this would provide a scale factor of B9 and B-2 for the two examples cited. It should be evident that "200 B-14" (single precision) or "200 B-28" (double precision) give the same result as the numbers without specification of the scale factor explicitly. If the quantity processed as described would overflow, however, then the conversion process of "d" instead is used.

d) As a decimal quantity with an indicated scale factor that would cause overflow if processed in accordance with "c" above. In this case, the number is converted to a binary number with scale factor B (14 - S') for single precision and B (28 - S') for double precision. Hence single precision "200 B5" would result in the same number as "200 B-9". The B5 representation is convenient in some applications in which the "basic" scaling or computations are done in integer arithmetic, so that a scaling up by 5 places is conveniently input to the assembler as a "B5".

Decimal exponents are specified by an E followed by the exponent value: .32 E2 (or .32 E 2 or .32 E+2) causes the same result as a simple 32. Plus signs may optionally be omitted, but minus signs must be specified. As mentioned previously, a minus sign before the value of the number causes it to be converted in ones complement form. Both decimal and binary exponents, of course, may be associated with the same number. If no decimal indication of the value of the number is supplied, then it is assumed that the value is 1 (hence "B-5" and ".1E1 B-5", for example, produce the same binary information).

Several special-purpose assembler operations are included in order to facilitate the generation of binary cell contents. The operation VN (which can also be written as NV, although VN is preferred because verb digits must precede noun digits) is available to specify a verb and noun combination for the display system (see Section IIJ) which presently uses decimal verbs and nouns, each in range 00 - 99 (although not necessarily all of the possible patterns are assigned). The VN operation causes the last two digits of the number in the address field to be converted to binary and assigned to bits 7-1 of the memory word (the noun); the preceding two digits are also converted to binary, and assigned to bits 14-8 of the memory word (the verb). The number specified, of course, in the address field with this operation should not exceed 9999.

Another special operation is MM, which achieves the same effect as DEC in converting the address field information to binary information. This operation is sometimes used when program numbers (also referred to as "major modes", see Section IIJ) are specified, and hence the number in the address field should not exceed 99.

Octal Numbers

Single precision octal numbers are specified by the operation OCT or OCTAL, which perform the same function. The quantity specified may have leading zeros suppressed: 23, 023, and 00023 will all be stored as the same memory cell contents. A sign may be included if desired, in which case the usual ones complement for a negative number is formed: OCT -37 becomes 77740_8 in the memory. Fractional octal numbers

(such as 12.3B3) may also be provided, with the scale factor indicating a left shift required: the example, therefore, would be stored in memory as 00123_8.

Double precision octal numbers can be written either as two consecutive single precision cells (via OCT), or by the operation 2OCT, in which case all the digits in both halves of the word must be included in the input. The operation 2OCTAL is treated the same as 2OCT.

Particularly in the interpretive language, it may appear unnecessary to specify explicitly the OCT for octal numbers. As has been mentioned previously, however, such quantities are considered as addresses by the assembler (in "true address" form, see Section IIB). For values less than 10000_8, the true and memory information would be the same; above that value, however, 10000_8 is subtracted from the input (so that 20000, for example, would be loaded into the memory as 10000). It is good practice to specify explicitly the assembler operation which is desired.

VC Representation of Addresses

The value of addresses formed as separate quantities in the listing can use the following operations (BNKSUM is discussed in Section IIF).

iDNADR (i = 1-6): Special-format word used for construction of telemetry downlist, where i is the number of consecutive pairs of cells to be sent (hence to send a double precision vector, i = 3). Bits 14-12 of the memory word are set to (i - 1), while bits 11-1 give the ECADR form of the address (see below) for the first word to be sent (its most significant half).

2BCADR: Same as 2CADR.

2CADR: Double precision CADR (complete address), occupying two consecutive cells in the memory. The first cell is the S-register portion of the address (bits 15-13 would be 0, see Section IIB), while the second is the same information as described below for BBCON. Both cells, of course, apply to the symbol in the address field, and because of the BBCON format, an EBANK= card (see Card Layout in Section III) must precede the 2CADR. Address information in this form can be used with DTCB (see Section VA).

2FCADR: Double precision CADR (complete address) for fixed-memory portion only, occupying two consecutive cells in the memory. The first cell is the same information as described below for BBCON (except bits 3-1 are 0 and no EBANK= is required), and the second in the S-register portion of the address (bits 15-13 would be 0, see Section IIB). Address information in this form can be used with DTCF (see Section VA).

=MINUS: Special-purpose assembly operation used to define the address (octal equivalent) of the tag in the tag field to be equal to the address of the tag in the address field minus the current value of the location counter (see Section III): this may be used to achieve the effect of subtraction of two addresses

in the following sequence. Assume NUM is to be set to the value of "BBB" minus "AAA" (the ending and starting addresses of a table, for example). This can be done by:

```
    SETLOC   AAA
TEM =MINUS   BBB      Note that this gives "BBB" - "AAA".
    SETLOC   (tag in bank where NUM is to be stored)
    BANK     (Blank BANK card, see Section III)
NUM ADRES    TEM
```

The last line is the only one that generates binary memory information.

=PLUS: Special-purpose assembly operation used to define the address (octal equivalent) of the tag in the tag field to be equal to the address of the tag in the address field plus the current value of the location counter. See =MINUS: successive applications of =PLUS can achieve the effect of multiplication of an address by an integer, for such applications as generating flagword-bit identifications.

ADRES: Address of information in address field, generally in 12-bit S-register format (will be full 15 bits, however, if address in address field artificially generated by e.g. =PLUS and =MINUS operations). Assembler can check that the tag in the address field is in the same fixed memory bank as the operand (or in the same erasable bank as that being monitored by the assembler).

BBCON: Address of information in address field in BBANK format, with bits 15-11 giving the FBANK portion and bits 3-1 giving the EBANK portion (as specified by a required EBANK= card immediately preceding the BBCON). In addition, bits 7-5 contain the required setting for SUPERBNK (for addresses in S3 and S4), or reflects the last S3 or S4 BANK card or SBANK= card, whichever was most recent: see SBANK= of Card Layout in Section III. The operand "BBCON*", with a blank address field, is used for a special purpose (see page IIF-5).

CADR: "Complete Address" information for a fixed memory cell, a single precision quantity (that excludes, therefore, SUPERBNK information). Bits 15-11 of the cell correspond to the FBANK setting and bits 10-1 to the least significant 10 bits of the S-register. The CADR operation is used with the appropriate program service routines to give an effective 15-bit addressing capability. Conversion between the five octal digits of the CADR information and the address information as printed in the listing may be accomplished as follows:

a) Divide the most significant 2 octal digits by 2 (shift the binary information right one place), and discard the remainder. This gives the FBANK setting.

b) If the second octal digit of the CADR information is even, the S-register contents are 2xxx, where xxx are the last 3 digits of the CADR information; if the second octal digit is odd, the S-register contents are 3xxx.

DNCHAN: Special-format word used for construction of telemetry downlist. Bits 14-12 are all binary 1, while bits 5-1 give the channel number (the first of the pair that is sent).

DNPTR: Special-format word used for construction of telemetry downlist. Bits 14-12 are set to 110_2, and bits 11-1 are set to the address (in ADRES format, but this is sufficient since the list is in variable-fixed memory) of the next "sub-list" to be sent.

ECADR: "Erasable Complete Address" information for an erasable memory cell, a single precision quantity. Bits 11-9 correspond the EBANK number and bits 8-1 to the appropriate S-register contents for the cell (provided that bits 10 and 9 of the S-register are both one, see Section IIB). Hence the EBANK can be found from bits 11-9, and the S-register setting is 1400_8 + bits 8-1.

FCADR: "Fixed Complete Address" information. The quantity is the same as that resulting from CADR.

GENADR: Address of information in address field. The quantity is the same as that resulting from ADRES. The assembler, however, makes no check on the consistency between the EBANK/FBANK of the quantity in the address field and the GENADR operand itself.

REMADR: Address of information in address field. The quantity is the same as that resulting from ADRES. The assembler, however, checks that the EBANK/FBANK of the quantity whose tag is in the address field is in a different bank ("remote") from that which is checked for ADRES.

VI INTERPRETIVE LANGUAGE

VIA General Principles

In addition to the machine language instructions given in Section IV, the software may also be provided in an "interpretive language", which permits mission-peculiar memory requirements to be reduced, and in many cases sharply reduces the coding labor which is necessary to implement the required software functions. Although the properties of the interpretive language could, in principle, be implemented in hardware, this would have created a considerably more complex computer design with its attendant power, volume, weight, and reliability penalties. The penalty paid for the interpretive language use is execution time: the double precision add (DAS) machine language order of Section IVB takes about 35 microseconds to be executed, for example, while the analogous interpretive language order (DAD) has been estimated to take about 660 microseconds (but it is more powerful in that the bank restrictions are not as strict when it is used). For most of the "guidance oriented" computations (such as orbital integration), this increase in execution time is not disadvantageous when weighed against the reduced memory requirements which result, but in other cases (such as most of the digital autopilot calculations) the execution time penalty cannot be tolerated, and therefore these computations are performed using machine language coding in spite of the increased memory requirements which may result.

Transition from machine language to interpretive language is accomplished by performing the instruction TC "INTPRET", which causes the computer hardware to start performing computations at "INTPRET"

(conventionally a cell in fixed-fixed memory bank 03). This step is the beginning of a program in fixed memory referred to as an "interpreter", which processes the information starting at the cell following the one transferring to "INTPRET". This processing continues until an interpretive language instruction is sensed (BOVB, EXIT, or RTB) which causes return to machine language execution at the appropriate point in the software: it should be realized, of course, that the hardware itself is incapable of performing anything <u>but</u> machine language operations. While the software interpreter is in operation, it processes the information in the software (which must, of course, be provided in the proper format for compatibility with the interpreter) as if this information was a list of parameter-word constants comprised of instruction-word and address-word items: hence the name "list-processing interpreter". Although the net software effect can generally be realistically approximated by considering that the interpretive language operations are being "performed" by the hardware, the actual technique whereby the software interpreter processes the list of parameter-word constants should be understood.

Computations in the interpretive language are written with one line of operation code information, followed by the necessary address parameters for the operations (if any are required) and then the next line of operation code information. For example, consider the following equation (the notation B,1 means that address involved is modified by an index register quantity):

$$B,1 = \left| \frac{\sin^{-1} (H + B,1)^2}{C} \right|$$

This could be written in the interpretive language as follows:

```
DLOAD    DAD*
         H
         B,1
DSQ      ASIN
DDV      ABS
         C
STORE    B,1
```

This achieves the same effect as the following coding that is written in a more conventional general-purpose computer form:

DLOAD	H	Double precision accumulator load
DAD*	B,1	Add from address B modified by index register #1
DSQ	--	Square accumulator contents
ASIN	--	Take arc sine of accumulator contents
DDV	C	Double precision divide
ABS	--	Form absolute value
STORE	B,1	Store in the same address as the one from which operand originally obtained.

This "conventional" mechanization, although perhaps somewhat easier to follow, would have the disadvantage of incompatibility with the computer hardware word length of 15 bits, thus restricting unduly the addresses which could be obtained (see Section IIB).

Most operations in the interpretive language are seven bits in length, meaning that two of them can be stored in a single 15-bit fixed memory word (plus, of course, the odd parity bit). For these operations, the first operation code is stored in bits 7-1 and the second (if any) in bits 14-8. For convenience in processing by the software interpreter, the information actually stored is formed as follows:

1. Determine the first operation code, increment it by +1, and place it in bits 7-1 of the word.

2. Determine the second operation code (which is zero if no operation is required), increment it by +1 if operation required, and place it in bits 14-8 of the word.

3. Complement the result from items 1 and 2 (thus making the word negative) to form the binary information stored in memory.

Several operations (including logical bit checks and some shifts) require supplemental information beyond that contained in the seven bits in order to determine what operation is to be performed. In such cases, the assembler automatically includes the necessary information in the address parameter. Some operations (such as absolute value of a scalar or length of a vector) are distinguished within the interpreter software by the value of a special cell within the Job Register Set (see Section VIIB): this cell, MODE, is set to +1 if triple precision (TP) operations are performed, to +0 if double precision (DP) operations are performed, and to -1 if vector (VC) operations are performed. The value of MODE is also used to establish the number of words to be stored by a storage order (of 15 bits each) and the type of operand required by operations using both vectors and scalars (VXSC and V/SC).

The four storage instructions (STCALL, STODL, STORE, and STOVL) form an exception to the storage format described above. Unless preceded by the STADR operation, these storage instructions are loaded in the memory as positive numbers, with bits 14-12 containing the type of command required (including index information), and bits 11-1 containing the erasable memory address where the information is to be stored. The STADR operation causes the storage instruction to be loaded by the assembler in complement form (for use with push-down information as described later).

Provision for two index registers is included in the interpretive language. These single precision registers are stored in the job VAC area (see Section VID) and have notations X1 and X2. They may be used to modify most operand addresses (except transfer orders) if desired, and several interpretive language instructions are available to load and modify these registers (which, of course, are erasable memory cells rather than hardware flip-flops). In common with some general-purpose computers (such as IBM 7090 series machines), the index register contents are subtracted from the base address to find the net address to be used.

The seven-bit operation code for most operations in the interpretive language is divided into a two-bit "prefix" specifying the operation category and a five-bit operation selection code. The prefix information is stored in bits 2-1 or 9-8 of the quantity stored in memory, and hence could also be labeled "suffix information": since the two bits are the first to be processed in decoding the operation, however, the "prefix" terminology is employed. The prefixes are assigned the following significance:

00_2 signifies a Unary Operation
01_2 signifies an Indexable Operation (index not used)
10_2 signifies a Miscellaneous Operation
11_2 signifies an Indexable Operation (index used)

Because of certain similarities between operations of the same prefix, it is convenient to summarize the interpretive language software capabilities using the same divisions, and this is done on the following pages.

Indexable Operations are those which, with one exception (SETPD), may specify an index register to modify the address-word parameter. Specification of an index register is indicated by an asterisk after the operation code. Three operations in this category (CCALL, CGOTO, and SSP) require two address-word parameters and the others require one. Most operations (all except CCALL, CGOTO, MXV, NORM, SETPD, SLOAD, SSP, VXM, and general shifts) will take information properly from the push-down list (see Section VII) if necessary. The operations included in this prefix category include scalar and vector addition (both double and triple precision scalars); scalar and vector subtraction and backwards subtraction (address contents from accumulator and accumulator from address contents respectively); single, double, triple, and vector accumulator loading; rounded and unrounded multiplication; vector dot and cross products; scalar times vector and divided into vector; general shifts (vector or scalar, right or left, rounded or unrounded); vector projection; vector times matrix and matrix times vector; the "sign" operation (effect of multiplying by $x/|x|$); computation of a cell containing a transfer address as the sum of two other quantities (with and without return address information retained); storage of information in push-down list combined with vector or double precision accumulator load; normalization (shift left to make magnitude of number at least $\frac{1}{2}$ and store number of shifts); storage of a single precision constant; setting of push-down list pointer to a specific value; and scalar division and backwards division (accumulator by address contents and address contents by accumulator respectively).

Miscellaneous Operations are those which do not affect the accumulator. With the exception of the logical bit operations that can cause a transfer, all operations in this category require one address-word parameter (and will not obtain information from the push-down list). No asterisk is used for those operations in this category that affect an index register: instead, these orders have the form AXT,1 or AXT,2 for operations affecting index register #1 or #2 respectively. The operations included in this prefix category include those for performing a subroutine in machine language and allowing return to interpreter software (thus effectively expanding the interpretive language operations); transferring

if the accumulator is positive, zero, negative, if most significant part is zero, if overflow (to either interpretive or machine language), or unconditionally; retaining return address information for an unconditional transfer; storing return address information; performing a number of manipulations with logical bits (setting them individually to zero, one, complement, or leave alone), while with the same order causing either no transfer to take place or transfer if the previous value of the bit was a binary zero, a binary one, or either; and performing a variety of operations with either of the two index registers, including setting them equal to true or complemented addresses or address contents, incrementing them by an address, adding or subtracting address contents from them, storing them in an address with or without loading the previous contents of that address in the index register, and transferring with index register decremented by corresponding "step" register if resulting index register contents still positive.

Unary Operations are those which require no address-word parameter, and most of them operate on information already in the accumulator. The operations included in this prefix category include those for taking the sine, cosine, $\sin^{-1}$, and $\cos^{-1}$ of the accumulator contents; square of vector and scalar; square root; complement of vector and scalar; absolute values of vector (i.e. length) and scalar; rounding to double precision; formation of a unit vector; vector definition from components; storage of accumulator in push-down list; transfer making use of return address information; returning to machine language; and causing operands to be taken from push-down list before a storage order. In addition, a variety of "short shift" orders (scalars from 1-4 places, left or right, rounded or unrounded; vectors from 1-8 places, left or right, rounded on right shifts) are also included with operations in this prefix category.

The "accumulator" used in the interpreter software is actually a set of seven cells with identification MPAC (for "multi-purpose accumulator"), located in the Job Register Set (see Section VIIB). Double precision words are stored in the first two cells (MPAC+0 and

MPAC+1), while triple precision words occupy these cells as well as MPAC+2. Vectors have the x component in MPAC+0 and MPAC+1; the y component in MPAC+3 and MPAC+4; and the z component in MPAC+5 and MPAC+6 (MPAC+2 is irrelevant in this case). Hence the first two cells in MPAC can be either a complete double precision word, the most significant two-thirds of a triple precision word, or the x component of a vector, depending on the particular computation in progress at the time. Use is made of the previously discussed MODE cell in those cases (such as storage commands) where it is necessary to identify the type of information present in MPAC.

Interpretation of address-word parameters depends on the operation involved to establish whether this quantity should be considered as an operand address, an address to which transfer is made, an integer (number) to be used directly, or a parameter giving supplemental information on the operation to be performed (such as the number of shifts and their type). The first address used by Indexable Operations is incremented by +1 before being loaded into the memory, but other address-word parameters are stored directly. This first address for Indexable Operations is restricted to 14 bits, since the 15th bit is used to indicate index register #2 (the whole parameter is stored in complemented form) if 1. Because of this, Indexable Operations are restricted to referencing operands with the same value for bit 15 of FBANK: a program step in bank 23 cannot reference a constant in bank 14, for example (if the value of the constant is needed by both "high" and "low" banks, it must be stored twice within the software). In addition, the interpreter software does not modify SUPERBNK, so that

coding within S3 (see Section IIB), for example, can make no references at all (either Indexable Operations or Miscellaneous Operations) to information within S4. Addresses below 4000_8 (addresses are stored in ECADR or FCADR form, see Section VC) are considered to be in erasable memory, and those below 0055_8 are assumed to be relative addresses (see Section VID). Addresses between 0055_8 and 0077_8 should be avoided, since these too are sometimes interpreted as relative addresses, even though the VAC area size is insufficient to have this interpretation be proper.

The following list of generalizations concerning the performance of the interpreter software has been assembled to give an over-all view of some aspects of interpreter software, and to indicate some of the features of this software which may not otherwise be apparent. -

1. The quantity -0 is considered a positive number ("Branch if Positive", for example, will take the branch if the accumulator contents are -0), as well as being of zero magnitude.

2. Sign agreement of various portions of multiple-precision words is generally not forced except if overflow is suspected.

3. Direct reference to input counter and other special erasable memory cells (see Section IID) cannot be made by interpretive language instructions: instead, a return to machine language must be done if it is necessary to sample these cells, since addresses in this range are considered to be in the VAC area (Section VID).

4. Although direct reference to interpretive registers generally will yield the proper results, execution time can be saved by using special orders if available (e.g. DSQ rather than DMP).

5. Although operations are provided with different mnemonics for scalars and vectors, they frequently are the same octal operation code, with the proper manipulation established by the current value of MODE. Hence an "absolute value of scalar" order, if MODE indicates a vector, will produce the length of the vector rather than the magnitude of the x component: the assembler, however, could indicate an error.

6. Since the same MPAC cells are used for vector and scalar computations, they must be specifically loaded and saved: the vector will <u>not</u> remain undisturbed if scalar computations are performed (although the y and z components may remain untouched).

7. A number of the normal computer hardware registers (including the shifting registers) are used by the interpreter, and hence must not be expected to retain their values if the interpretive language is entered: by the same token, any task that may interrupt a job must ensure that the contents of these registers are not lost.

8. A special cell (OVFIND) is "set" (to a value of $\pm$ 1, although the sign is not significant) if overflow is encountered in addition, subtraction, division, shifting, vector operations (cross and dot products, projection, squaring, multiplication by a matrix, unit vector, division by a scalar), and rounding. The OVFIND cell (part of the Job Register Set information when a job not active, see Section VIIB) is set 0 at the start of a job, and reset by a branch on overflow (BOV and BOVB) order or by specific setting e.g. by an SSP order. In many cases, if an overflow after manipulating the most significant part of the answer is encountered, sign agreement of the answer is forced to ensure that the overflow is "genuine", and OVFIND is not set unless this sign-agreement forcing demonstrated that a true overflow condition indeed exists.

9. For scalar division, the accumulator is left at $\pm$ MAX if overflow is encountered, and this feature can be used to obtain an automatic limiting of quotients in applications such as the computation of $\sin^{-1}$ or $\cos^{-1}$ arguments. In most other cases, however, the resulting numerical answer usually will be a poor representation of the answer, since the overflow bit is generally lost.

10. Variables in different banks of erasable memory can be used, since the interpreter software automatically switches these banks. When return from the interpretive language to machine language, EBANK will be left at its value when the interpreter was originally entered.

11. Constants in either the low (below bank 20) or high (above bank 21) part of fixed memory can be referenced by Indexable Operations, but only by programs stored in the same half of the memory. Program transfers, however, can be made freely.

12. The interpreter software performs no modifications of SUPERBNK, and therefore coding in banks 30-37 cannot reference information in banks 40-43 (and SUPERBNK must be set properly if e.g. coding in bank 25 references S3 or S4 information).

13. Banks 00, 01, 20, and 21 of fixed memory cannot be referenced by Indexable Operations, since the address would be interpreted as erasable memory, nor can banks 00 and 01 be entered by transfer orders (same reason, so address would be considered to be indirect).

14. Some instructions will not interface properly with the push-down list, and these are noted in Section VIB with the command in question.

15. Several instructions (noted in Section VIB) require operands in erasable memory, and in general will malfunction if an attempt is made to reference an operand in fixed memory.

16. The number of shifts specified in the general shift orders must not be excessive (i.e. beyond those necessary to remove information from MPAC), or improper results could be obtained. The same restriction applies to the resulting shift amount if an index-register modification to the shift count is employed.

17. The STADR instruction, which complements the following store command to permit operands to be obtained from the push-down list, must be the final order before the store command in question, since STADR performance involves decoding the store command itself.

18. Transfers which retain return address information (CALL and CCALL, since STCALL satisfies the format constraint automatically) should be the final instruction in a sequence, so that return to the cell following the transfer-address information will produce a proper operation code. The EXIT command should be similarly located.

19. If it is desired to withdraw quantities from the push-down list and then transfer control, this will be done properly only if the transfer address is in the high portion of fixed memory (so that it will be negative) or, if RVQ is used, if the next binary memory information is negative.

20. If it is desired to store a triple precision result, the value of the MODE cell must be proper. Although the multiply order (DMP) leaves a triple precision product, and the TAD order adds a triple precision operand, neither order sets MODE for triple precision (instead, a TLOAD must be done or a special-purpose setting of MODE accomplished).

VIB Interpretive Language Operations

For convenience in presentation, the 124 mnemonics (excluding alternate mnemonics for the same operation) available for use with the interpreter software have been divided into seven groups. These groups, with the mnemonics in each (excluding the alternates) are:

Scalar Computation Operations: ABS, ACOS, ASIN, BDDV, BDSU, COS, DAD, DCOMP, DDV, DMP, DMPR, DSQ, DSU, ROUND, SIGN, SIN, SQRT, TAD.

Vector Computation Operations: ABVAL, BVSU, DOT, MXV, UNIT, VAD, VCOMP, VDEF, VPROJ, VSQ, VSU, VXM, VXSC, VXV, V/SC.

Shifting Operations: NORM, SL, SL1-SL4, SLR, SL1R-SL4R, SR, SR1-SR4, SRR, SR1R-SR4R, VSL, VSL1-VSL8, VSR, VSR1-VSR8.

Transmission Operations: DLOAD, ITA, PDDL, PDVL, PUSH, SETPD, SLOAD, SSP, STADR, STCALL, STODL, STORE, STOVL, TLOAD, VLOAD.

Control Operations: BHIZ, BMN, BOV, BOVB, BPL, BZE, CALL, CCALL, CGOTO, EXIT, GOTO, RTB, RVQ.

Index Register Oriented Operations: AXC, AXT, INCR, LXA, LXC, SXA, TIX, XAD, XCHX, XSU.

Logical Bit Operations: BOFCLR, BOFF, BOFINV, BOFSET, BON, BONCLR, BONINV, BONSET, CLEAR, CLRGO, INVERT, INVGO, SET, SETGO.

For each operation, the standard mnemonic and the mnemonic for specification of an index register (if applicable) are given, together with the corresponding seven-bit octal order (one octal digit, in range 0-3, for prefix and two octal digits, in range 00 - 37, for operation).

The first line of the description of each order contains an abbreviated description of the order's function (frequently the full expression corresponding to the abbreviation of the mnemonic). Additional lines contain an expanded description of the performance of the order, and then the detailed formulation of the order's mechanization in the interpreter software. In some cases, "logically equivalent" formulations are presented in the interests of clarity: reference should be made to the symbolic listing for information on the actual coding formulation.

Symbol	Order	Performance

ABS 0 26 Absolute Value of Scalar.

No Address Replace $MPAC_{tp}$ with absolute value of $MPAC_{tp}$.

If MODE = VC, proceed to ABVAL instruction

If $MPAC_{tp} < -0$:

$MPAC_{tp} = -MPAC_{tp}$

Proceed to "DANZIG"

ACOS 0 12 Arc Cosine. ARCCOS may also be used as symbol.

No Address Replace $MPAC_{dp}$ with arc cosine of $MPAC_{dp}$. Input scaling is assumed B1, and output is in units of revolutions with scale factor B0, in range 0 to $+\frac{1}{2}$ (i.e. 0° to 180°).

$X = MPAC_{tp}$

Perform "ARCCOM"

$MPAC_{dp} = TS$ (MPAC+2 not necessarily meaningful)

Proceed to "DANZIG"

ARCCOM (Routine used by ACOS and ASIN)

If $|X| = 0$:

TS = 0.25

Return

$TS_1 = X$

$X = |X|$

If $X \geq (1 + 2^{-12} = 1.000244)$, or if $X > (1 + 2^{-13} = 1.000122)$ with least significant half of X negative:

Perform "ALARM" (pattern 1301_8) CM only

Perform "ALARM1" (pattern 1301_8, TS = (LOC, BANKSET)) LM only

If $X \geq 1$:

Set TS = 0 (TS_1 positive) or TS = $\frac{1}{2}$ (TS_1 negative)

Return

Symbol	Order	Performance
ACOS (cont)	$TS = (1 - X)^{\frac{1}{2}}$	Square root performed via "SQRTSUB" (see SQRT order), with subsequent right shift rounded. Scale factor B1 when enter "SQRTSUB".

$$TS = TS\,(K_{as0} + K_{as1}\,X + K_{as2}\,X^2 + K_{as3}\,X^3 + K_{as4}\,X^4 + K_{as5}\,X^5 + K_{as6}\,X^6 + K_{as7}\,X^7)$$

If $TS_1 < 0$:

$TS = \frac{1}{2} - TS$

Return

Constant	True Value	$\sqrt{2\pi}$ x Value
K_{as0}	0.35355 3385	1.57079 6302
K_{as1}	-0.04830 17006	-0.21459 8801
K_{as2}	0.02002 73085	0.08897 8987
K_{as3}	-0.01129 31863	-0.05017 4305
K_{as4}	0.00695 311612	0.03089 1881
K_{as5}	-0.00384 617957	-0.01708 8126
K_{as6}	0.00150 1297736	0.00667 0090
K_{as7}	-0.00028 4160334	-0.00126 2491

The numbers in the last column agree closely with the Hastings values quoted on page 81 of "Handbook of Mathematical Functions," National Bureau of Standards Applied Mathematical Series #55.

All constants are stored with scale factor B-I in the program (e.g. K_{as5} has scale factor B-5). The $\sqrt{2}$ factor is required because (1-X) was scaled at B1 when its square root taken.

ARCCOS — Same as ACOS (alternate mnemonic).

ARCSIN — Same as ASIN (alternate mnemonic).

Symbol	Order	Performance
ASIN	0 10	Arc Sine. ARCSIN may also be used as symbol.
No Address		Replace $MPAC_{dp}$ with arc sine of $MPAC_{dp}$. Input scaling is assumed B1, and output is in units of revolutions with scale factor B0, in range $-\frac{1}{4}$ to $+\frac{1}{4}$ (i.e. -90^{o} to $+90^{o}$).

$X = MPAC_{tp}$
Perform "ARCCOM" (see ACOS)
$MPAC_{dp} = \frac{1}{4} - TS$ (MPAC+2 not necessarily meaningful)
Proceed to "DANZIG"

BDDV	1 22	Backwards Double Precision Divide.
BDDV*	3 22	Replace $MPAC_{dp}$ with quotient of quantity at specified address divided by $MPAC_{dp}$. Set OVFIND if overflow, and leave $MPAC_{dp}$ with special patterns in that case.

$Num = E_{ADDRWD_{dp}}$
$Den = MPAC_{dp}$
Proceed to "DIVCOM"

DIVCOM (Routine used by BDDV and DDV)

If $|Den+0| = 00001_{g}$:
    Force sign agreement of Den
If Den+0 = 0:
    Force sign agreement of Num
    If Num+0 $\neq$ 0:
        $MPAC_{dp}$ = +MAX sgn (Num+0/Den+1) (0 is positive)
        Set OVFIND
        Proceed to "DANZIG"
    Shift Num and Den left 14 places (Num+1 into Num+0 etc.)
    If Den+0 = 0:
        $MPAC_{dp}$ = +MAX sgn (Num+0) (0 is positive)
        Set OVFIND
        Proceed to "DANZIG"

Symbol	Order	Performance

BDDV (cont)

If Num = 0:

$MPAC_{dp}$ = Num

Proceed to "DANZIG"

Determine proper sign of quotient and store in DVSIGN (set to -0 if quotient negative, otherwise $\pm$ 1).

Num = |Num|

Den = |Den|

If Den+0 - Num+0 - 1 $\leqslant$ 0:

Force sign agreement of Den and Num

If Den+0 - Num+0 $<$ 0:

$MPAC_{dp}$ = +MAX sgn (quotient, from DVSIGN data)

Set OVFIND

Proceed to "DANZIG"

If Den+0 = Num+0:

If Den+1 - Num+1 $\leqslant$ 0:

$MPAC_{dp}$ = +MAX sgn (quotient, from DVSIGN data)

Set OVFIND

Proceed to "DANZIG"

Perform the division of Num by Den, using the algorithm steps below, and store the result in $MPAC_{dp}$.

MPAC+2 = 0

Proceed to "DANZIG"

The algorithm employed to perform the division makes use of the following sequence of activity (DVSIGN set with data on quotient sign before enter algorithm).

1. Normalize Den by shifting it left one place at a time until overflow is sensed: the overflowing shift is not employed. Then shift Num left the same number of places (because of previous overflow checks, Num will not overflow here, nor will the number of shifts required exceed 13).

2. For notational convenience, let:

   $Num = A + 2^{-14} B$

   $Den = C + 2^{-14} D$

   Where A, B, C, and D each 15 bit numbers (including sign), and both A and C are positive. Algorithm involves essentially multiplication of Num and Den by $(C - 2^{-14} D)$ and neglecting high-order terms.

Symbol Order	Performance

BDDV (cont)

3. If A = C:

   MPAC+0 = +MAX

   TS = B - D + C

   Proceed to step #8

4. Divide (using hardware divide order, DV, of Section IVC) Num_{dp} by C, storing quotient in MPAC+0 and remainder in MPAC+1.

5. TS = MPAC+1 - (MPAC+0) D

6. If $|TS| \geq 1.0$: (i.e. overflows: "1" as used below is one least increment).

   TS = TS - C

   MPAC+0 = MPAC+0 + 1

   Proceed to step #7

   If TS = 0:

   MPAC+1 = TS

   Proceed to step #9

   If TS < 0:

   MPAC+0 = MPAC+0 - 1

   TS = TS + C

   Proceed to step #8

7. If $TS - C \geq 0$:

   TS = TS - C

   MPAC+0 = MPAC+0 + 1

8. Divide (using hardware divide order, DV, with L-register set to 0) TS by C, storing quotient in MPAC+1.

9. If quotient should be negative (from DVSIGN data), complement the contents of $MPAC_{dp}$.

Using the notation of #2 above, the manipulation may be summarized as follows (deleting overflow checks):

$$\frac{Num}{Den} = \frac{Num}{C} + 2^{-14}\left[\frac{Remainder - (Num/C)\,D}{C}\right]$$

As mentioned in step #2, this effectively reflects a multiplication of Num and Den by $(C - 2^{-14} D)$ and a neglect of most 2^{-28} terms.

Symbol	Order	Performance

BDSU 1 33 Backwards Double Precision Subtract.

BDSU* 3 33 Replace $MPAC_{dp}$ by (quantity at specified address) - $MPAC_{dp}$. Set OVFIND if overflow.

$MPAC_{dp} = E_{ADDRWD_{dp}} - MPAC_{dp}$

If most significant half overflows:

$MPAC_{dp} = MPAC_{dp} + 2^{28}$ sgn MPAC+0 (forces sign agreement and corrects answer if not an overflow case).

If above computation overflows:

Set OVFIND

Proceed to "DANZIG"

COS 0 06 Cosine. COSINE may also be used as symbol.

No Address Replace $MPAC_{dp}$ with cosine of $MPAC_{dp}$. Input value scaling is assumed BO in units of revolutions, and output is with scale factor B1.

$X = \frac{1}{4} - |MPAC_{dp}|$

Perform "SICOM"

$MPAC_{dp}$ = TS (MPAC+2 also loaded, not necessarily with significant bits).

Proceed to "DANZIG"

SICOM (Routine used by COS and SIN)

If $|X| \geq \frac{1}{2}$, $X = \frac{1}{2}$ sgn X - X

If $|X| \geq \frac{1}{4}$, $X = \frac{1}{2}$ sgn X - X

x = X, rescaled to scale factor B-1 revolutions (B1 in $\pi/2$ units)

$TS = K_{sn1}\, x + K_{sn3}\, x^3 + K_{sn5}\, x^5 + K_{sn7}\, x^7 + K_{sn9}\, x^9$

Return

Constant	Scaling	Stored Value	True Value x $(2/\pi)^i$
K_{sn1}	B2	0.39269 90796	0.99999 9995
K_{sn3}	BO	-0.64596 37111	-0.16666 6567
K_{sn5}	B-2	0.31875 8717	0.00833 3025
K_{sn7}	B-4	-0.07478 0249	-0.00019 8074
K_{sn9}	B-6	0.00969 4988	0.00000 2603

Per program comments, constants from a Hastings series. "Scaling" column quoted for x in $(\pi/2)$ units.

Symbol	Order	Performance
COSINE		Same as COS (alternate mnemonic).
DAD	1 34	Double Precision Add.
DAD*	3 34	Replace $MPAC_{dp}$ by (quantity at specified address) + $MPAC_{dp}$. Set OVFIND if overflow.

$MPAC_{dp} = E_{ADDRWD_{dp}} + MPAC_{dp}$

If most significant half overflows:

$MPAC_{dp} = MPAC_{dp} + 2^{28}$ sgn MPAC+0 (see note with BDSU)

If above computation overflows:

Set OVFIND

Proceed to "DANZIG"

DCOMP	0 20	Double Precision Complement.
No Address		Replace $MPAC_{tp}$ with complement of $MPAC_{tp}$.

If MODE = VC, proceed to VCOMP instruction

$MPAC_{tp} = - MPAC_{tp}$

Proceed to "DANZIG"

DDV	1 21	Double Precision Divide.
DDV*	3 21	Replace $MPAC_{dp}$ with quotient of $MPAC_{dp}$ divided by quantity at specified address. Set OVFIND if overflow, and leave $MPAC_{dp}$ with special patterns in that case.

Num = $MPAC_{dp}$

Den = $E_{ADDRWD_{dp}}$

Proceed to "DIVCOM" (see BDDV)

DMP	1 36	Double Precision Multiply.
DMP*	3 36	Replace $MPAC_{tp}$ with product of $MPAC_{dp}$ and quantity at specified address.

Symbol	Order	Performance
DMP (cont)		AMAL = $MPAC_{dp}$ BMBL = $E_{ADDRWD_{dp}}$ Perform "MULCOM" Proceed to "DANZIG"
		MULCOM (Routine used by DMP, DMPR, etc.) Note 1. AMAL = AM + 2^{-14} AL Note 2. BMBL = BM + 2^{-14} BL Note 3. The hardware multiply order (MP, Section IVC) gives a double precision answer, using single precision operands. TS = most significant half of (AL x BL) $MPAC_{tp}$ = (AM x BM) + 2^{-14} (AM x BL + AL x BM) + 2^{-28} TS (carries propagate, no overflows) Return
DMPR	1 20	Double Precision Multiply and Round.
DMPR*	3 20	Replace $MPAC_{dp}$ with product of $MPAC_{dp}$ and quantity at specified address, rounded to double precision. Set OVFIND if overflow. AMAL = $MPAC_{dp}$ BMBL = $E_{ADDRWD_{dp}}$ Perform "MULCOM" (see DMP) MODE = DP $MPAC_{tp}$ = $MPAC_{tp}$ + MPAC+2 (carries propagate) If most significant part overflows: Set OVFIND MPAC+2 = 0 Proceed to "DANZIG"
DSQ	0 14	Double Precision Square Operation.
No Address		Replace $MPAC_{tp}$ by the square of $MPAC_{dp}$.

Symbol	Order	Performance
DSQ (cont)		TS = most significant half of (MPAC+1 x MPAC+1) $MPAC_{tp}$ = (MPAC+0 x MPAC+0) + 2^{-14} (2 (MPAC+0 x MPAC+1)) + 2^{-28} TS (see notes with "MULCOM", DMP) Proceed to "DANZIG"
DSU	1 32	Double Precision Subtract.
DSU*	3 32	Replace $MPAC_{dp}$ by $MPAC_{dp}$ - (quantity at specified address). Set OVFIND if overflow. $MPAC_{dp}$ = $MPAC_{dp}$ - $E_{ADDRWD_{dp}}$ If most significant half overflows: $MPAC_{dp}$ = $MPAC_{dp}$ + 2^{28} sgn MPAC+0 (see note with BDSU) If above computation overflows: Set OVFIND Proceed to "DANZIG"
ROUND	0 16	Round to Double Precision.
No Address		Replace $MPAC_{dp}$ with rounded version of $MPAC_{tp}$, setting OVFIND if overflow. MODE = DP $MPAC_{tp}$ = $MPAC_{tp}$ + MPAC+2 (carries propagate) If most significant part overflows: Set OVFIND MPAC+2 = 0 Proceed to "DANZIG"
SIGN	1 02	Sign Function.
SIGN*	3 02	If quantity at specified address (which may be in fixed or erasable memory) is negative non-zero, complement contents of MPAC (scalar or vector).

Symbol	Order	Performance
SIGN (cont)		If $E_{ADDRWD_{dp}} \geq -0$, proceed to "DANZIG" If MODE = VC: $MPAC_{vc} = -MPAC_{vc}$ Proceed to "DANZIG" $MPAC_{tp} = -MPAC_{tp}$ Proceed to "DANZIG"
SIN	0 04	Sine. SINE may also be used as symbol.
No Address		Replace $MPAC_{dp}$ with sine of $MPAC_{dp}$. Input value scaling is assumed B0 in units of revolutions, and output is with scale factor B1. $X = MPAC_{dp}$ Perform "SICOM" (see COS) $MPAC_{dp}$ = TS (MPAC+2 also loaded, not necessarily with significant bits). Proceed to "DANZIG"
SQRT	0 02	Square Root Function.
No Address		Replace $MPAC_{tp}$ with square root of $MPAC_{tp}$ (most significant non-zero bits only), with scale factor one-half the scale factor of the original number. Perform "SQRTSUB" If MPTEMP = 0, proceed to "DANZIG" MPAC+2 = 0 Shift $MPAC_{tp}$ right MPTEMP places Proceed to "DANZIG"
		SQRTSUB (Routine used by SQRT, UNIT, etc.) MPTEMP = 0 (gives number of shifts for output) If $\lvert MPAC_{tp} \rvert = 0$: $MPAC_{tp} = +0$ MPTEMP = 14 Return

Symbol	Order	Performance
SQRT (cont)		If $MPAC_{tp} < -2^{-14}$:
		Proceed to "POODOO" (pattern 21302_8) CM only
		Proceed to "POODOO1" (pattern 21302_8, TS = (LOC, BANKSET)) LM only
		If $MPAC_{tp} < 0$:
		$MPAC_{dp} = 0$
		If MPAC+0 was non-zero, MPTEMP = 0; Return
		If MPAC+1 was non-zero, MPTEMP = 7; MPAC+2 = 0; Return.
		MPAC+2 = 0
		MPTEMP = 14
		Return
		If MPAC+0 = 0:
		Shift $MPAC_{tp}$ left 14 places
		MPAC+2 = 0
		MPTEMP = 7
		If MPAC+0 = 0: (i.e. original MPAC+1)
		Shift $MPAC_{tp}$ left 14 places
		MPAC+2 = 0
		MPTEMP = 14
		If MPAC+0 $< \frac{1}{4}$:
		Shift $MPAC_{tp}$ left 2 places
		MPTEMP = MPTEMP + 1
		Repeat check against $\frac{1}{4}$
		If MPAC+0 $\geq \frac{1}{2}$:
		$TS_1 = 0.5884$ (MPAC+0) + 0.4192 (single precision)
		If MPAC+0 $< \frac{1}{2}$: TS_1 scaled B1
		$TS_1 = 0.8324$ (MPAC+0) + 0.2974 (single precision)
		$TS_1 = (\frac{1}{2}$ (MPAC+0)$) / TS_1 + \frac{1}{2} TS_1$ (single precision)
		$TS = \frac{1}{2} MPAC_{dp}$
		$MPAC_{dp} = \frac{1}{2} TS_1$ (right shift 1, result double precision)
		TS_2 = single-precision quotient from TS_{dp} / TS_1 (using hardware divide order, DV)
		TS_3 = remainder from division to compute TS_2, single prec.
		$TS_4 = TS_3 / TS_1$ (single precision, L-register zeroed)
		$MPAC_{dp} = MPAC_{dp} + TS_2 + 2^{-14} TS_4$ (carries propagate)
		If above addition overflows, set $MPAC_{dp}$ = +MAX
		Return

Symbol	Order	Performance
TAD	1 01	Triple Precision Add.
TAD*	3 01	Replace $MPAC_{tp}$ by (quantity at specified address, triple precision) + $MPAC_{tp}$. Set OVFIND if overflow.

$MPAC_{tp} = E_{ADDRWD_{tp}} + MPAC_{tp}$

If most significant part overflows:

$MPAC_{dp} = MPAC_{dp} + 2^{28}$ sgn MPAC+0

Note that MPAC+2 left alone; see note with BDSU.

If above computation overflows:

Set OVFIND

Proceed to "DANZIG"

Symbol	Order	Performance
ABVAL	0 26	Absolute Value (length) of Vector.
No Address		Load into $MPAC_{tp}$ the length of $MPAC_{vc}$ (same scaling as $MPAC_{vc}$). Leave in LVSQUARE the square of this length. Set OVFIND if overflow.

If MODE $\neq$ VC, proceed to ABS instruction

$TS_{tp} = MPAC_x^2 + MPAC_y^2$ (DSQ routine forms squares)

If most significant part overflows:

Set OVFIND

$MPAC_{tp} = TS_{tp} + MPAC_z^2$ (DSQ routine forms square)

If most significant part overflows:

$MPAC_{dp} = MPAC_{dp} + 2^{28}$ sgn MPAC+0 (see note with BDSU)

If above computation overflows:

Set OVFIND

LVSQUARE = $MPAC_{dp}$ (Relative Addresses 34D-35D, see Section VID)

MODE = DP

Proceed to SQRT instruction to form square root of $MPAC_{tp}$ and then exit to "DANZIG". Length of vector, left in $MPAC_{dp}$, has same scale factor as original $MPAC_{vc}$.

BVSU	1 26	Backwards Vector Subtract.
BVSU*	3 26	Replace $MPAC_{vc}$ by (vector at specified address) - $MPAC_{vc}$. Set OVFIND if overflow.

Perform the following for i = y, z, x:

$MPAC_i = E_{ADDRWD_i} - MPAC_i$

If most significant half overflows:

$MPAC_i = MPAC_i + 2^{28}$ sgn $MPAC+0_i$ (see note with BDSU)

If above computation overflows:

Set OVFIND

Proceed to "DANZIG"

Symbol	Order	Performance
DOT	1 27	Dot Product of Vectors.
DOT*	3 27	Load into $MPAC_{tp}$ the dot product of $MPAC_{vc}$ and (vector at specified address). Set OVFIND if overflow.

$AVEC = MPAC_{vc}$

$BVEC = E_{ADDRWD_{vc}}$

Perform "DOTCOM"

MODE = DP

Proceed to "DANZIG"

DOTCOM (Routine used by DOT, MXV, etc.)

$AMAL = AVEC_x$

$BMBL = BVEC_x$

Perform "MULCOM" (see DMP)

$BUF_{tp} = MPAC_{tp}$

$AMAL = AVEC_y$

$BMBL = BVEC_y$

Perform "MULCOM" (see DMP)

$BUF_{tp} = BUF_{tp} + MPAC_{tp}$

If most significant part overflows:

    Set OVFIND

$AMAL = AVEC_z$

$BMBL = BVEC_z$

Perform "MULCOM" (see DMP)

$MPAC_{tp} = MPAC_{tp} + BUF_{tp}$

If most significant part overflows:

    $MPAC_{dp} = MPAC_{dp} + 2^{28}$ sgn MPAC+0 (see note with BDSU)

    If above computation overflows:

        Set OVFIND

Return

Symbol	Order	Performance
MXV	1 13	Matrix Times Vector.
MXV*	3 13	Use the matrix stored with double precision components (3 x 3) at the specified address to premultiply the vector in $MPAC_{vc}$, and leave the result in $MPAC_{vc}$. Set OVFIND if overflow. Matrix is stored with first element at $E_{ADDRWD_{dp}}$, and successive elements of first row, then second row, then third row in consecutive cells after the first element. Computation forms x component of result as dot product of first row and the vector; y component of result as dot product of second row and the vector; and z component of result as dot product of third row and the vector. Instruction will not push up properly.

$VBUF = MPAC_{vc}$
$AVEC = VBUF_{vc}$
$BVEC = E_{ADDRWD_{vc}}$
Perform "DOTCOM" (see DOT)
$BUF_x = MPAC_{dp}$
$AVEC = VBUF_{vc}$
$BVEC = E_{ADDRWD+6_{vc}}$
Perform "DOTCOM" (see DOT)
$BUF_y = MPAC_{dp}$
$AVEC = VBUF_{vc}$
$BVEC = E_{ADDRWD+12_{vc}}$
Perform "DOTCOM" (see DOT)
$BUF_z = MPAC_{dp}$
$MPAC_{vc} = BUF_{vc}$
Proceed to "DANZIG"

Symbol	Order	Performance
UNIT	O 24	Form a Unit Vector.
No Address		Replace $MPAC_{vc}$ with a unit vector corresponding to $MPAC_{vc}$, and with scale factor B1. Leave in LVSQUARE the square of the original vector length, and in LV the vector length (same scale factor as original $MPAC_{vc}$). Set OVFIND if overflow.

Force sign agreement of each component of $MPAC_{vc}$

$VBUF = MPAC_{vc}$

$TS_{tp} = MPAC_x^2 + MPAC_y^2$ (DSQ routine forms squares)

If most significant part overflows:

    Set flag indicating overflow

$MPAC_{tp} = TS_{tp} + MPAC_z^2$ (DSQ routine forms square)

If most significant part overflows:

    $MPAC_{dp} = MPAC_{dp} + 2^{28}$ sgn MPAC+O (see note with BDSU)

    If above computation overflows:

        Set flag indicating overflow

If flag set indicating overflow above:

    Set OVFIND

    $MPAC_x = \pm$ MAX (sign not significant)

    Proceed to "DANZIG"

$LVSQUARE = MPAC_{dp}$ (Relative Addresses 34D-35D, see Section VID)

Perform "SQRTSUB" (see SQRT)

If MPAC+O = O: (i.e. input to "SQRTSUB" was O)

    LV = O (Relative Addresses 36D-37D)

    Set OVFIND

    $MPAC_x = \pm$ MAX (sign not significant)

    Proceed to "DANZIG"

$LV = 2^{-MPTEMP}\ MPAC_{dp}$ (Relative Addresses 36D-37D. MPTEMP computed in "SQRTSUB")

If MPTEMP = O:

    Shift each component of VBUF right 1 place

Symbol	Order	Performance
UNIT (cont)		If MPTEMP $\neq$ 0: Shift each component of VBUF left (MPTEMP -1) places (unit vector scaling B1) TS = $MPAC_{dp}$ (still normalized from "SQRTSUB") Divide each component of VBUF by TS, setting DVSIGN to sign of VBUF component and using algorithm of "DIVCOM" (see BDDV), starting at step #4 (after making Num positive). $MPAC_{vc}$ = vector results from previous line Proceed to "DANZIG"
VAD	1 24	Vector Add.
VAD*	3 24	Replace $MPAC_{vc}$ by (vector at specified address) + $MPAC_{vc}$. Set OVFIND if overflow. Perform the following for i = y, z, x: $MPAC_i = E_{ADDRWD_i} + MPAC_i$ If most significant half overflows: $MPAC_i = MPAC_i + 2^{28}$ sgn $MPAC+O_i$ (see note with BDSU) If above computation overflows: Set OVFIND Proceed to "DANZIG"
VCOMP	0 20	Vector Complement.
No Address		Replace $MPAC_{vc}$ with the complement of $MPAC_{vc}$. If MODE $\neq$ VC, proceed to DCOMP instruction $MPAC_{vc} = - MPAC_{vc}$ Proceed to "DANZIG"
VDEF	0 22	Vector Define.
No Address		Load $MPAC_{vc}$ with information in $MPAC_{dp}$ (x component), top two cells in push-down list (y component), and second from top pair of cells in push-down list (z component).

Symbol	Order	Performance
VDEF (cont)		$MPAC_x = MPAC_{dp}$ (these are the same pair of cells) $MPAC_y = E_{PUSHLOC-2_{dp}}$ (see Section VID) $MPAC_z = E_{PUSHLOC-4_{dp}}$ PUSHLOC = PUSHLOC - 4 MODE = VC Proceed to "DANZIG"
VPROJ	1 31	Vector Projection on New Vector.
VPROJ*	3 31	Replace $MPAC_{vc}$ with the dot product of $MPAC_{vc}$ and (vector at specified address) times (vector at specified address). Set OVFIND if overflow. Scale factors add. $AVEC = MPAC_{vc}$ $BVEC = E_{ADDRWD_{vc}}$ Perform "DOTCOM" (see DOT) MODE = DP Proceed to VXSC instruction to complete manipulation and then exit to "DANZIG".
VSQ	0 30	Vector Square Operation.
No Address		Load into $MPAC_{tp}$ the square of the length of the vector in $MPAC_{vc}$, with scale factor twice that of the vector. Set OVFIND if overflow. $TS_{tp} = MPAC_x^2 + MPAC_y^2$ (DSQ routine forms squares) If most significant part overflows: Set OVFIND $MPAC_{tp} = TS_{tp} + MPAC_z^2$ (DSQ routine forms square) If most significant part overflows: $MPAC_{dp} = MPAC_{dp} + 2^{28}$ sgn·MPAC+0 (see note with BDSU) If above computation overflows: Set OVFIND MODE = DP Proceed to "DANZIG"

Symbol	Order	Performance
VSU	1 25	Vector Subtract.
VSU*	3 25	Replace $MPAC_{vc}$ by $MPAC_{vc}$ - (vector at specified address). Set OVFIND if overflow.

Perform the following for i = y, z, x:

$MPAC_i = MPAC_i - E_{ADDRWD_i}$

If most significant half overflows:

$MPAC_i = MPAC_i + 2^{28}$ sgn $MPAC+0_i$ (see note with BDSU)

If above computation overflows:

Set OVFIND

Proceed to "DANZIG"

VXM	1 16	Vector Times Matrix.
VXM*	3 16	Use the matrix stored with double precision components (3 x 3) at the specified address to postmultiply the vector in $MPAC_{vc}$, and leave the result in $MPAC_{vc}$. Set OVFIND if overflow. Matrix is stored with first element at $E_{ADDRWD_{dp}}$, and successive elements of first row, then second row, then third row in consecutive cells after the first element. Computation forms x component of result as dot product of first column and the vector; y component of result as dot product of second column and the vector; and z component of result as dot product of third column and the vector. Effect is equivalent to premultiplication of the vector by the transpose of the given matrix (cf. MXV). Instruction will not push up properly.

$VBUF = MPAC_{vc}$

$AVEC = VBUF_{vc}$

$BVEC = (E_{ADDRWD_{dp}}, E_{ADDRWD+6_{dp}}, E_{ADDRWD+12_{dp}})$

Perform "DOTCOM" (see DOT)

$BUF_x = MPAC_{dp}$

Symbol	Order	Performance
VXM (cont)		$AVEC = VBUF_{vc}$ $BVEC = (E_{ADDRWD+2_{dp}}, E_{ADDRWD+8_{dp}}, E_{ADDRWD+14_{dp}})$ Perform "DOTCOM" (see DOT) $BUF_y = MPAC_{dp}$ $AVEC = VBUF_{vc}$ $BVEC = (E_{ADDRWD+4_{dp}}, E_{ADDRWD+10_{dp}}, E_{ADDRWD+16_{dp}})$ Perform "DOTCOM" (see DOT) $BUF_z = MPAC_{dp}$ $MPAC_{vc} = BUF_{vc}$ Proceed to "DANZIG"
VXSC	1 03	Vector Times Scalar.
VXSC*	3 03	Multiply a vector times a scalar and store the result in $MPAC_{vc}$. If have been computing with vectors, the quantity at the specified address is considered a scalar; if have been computing with scalars, the quantity at the specified address is considered a vector. Rounding is performed for each component, and OVFIND set if overflow. If MODE = VC: $VBUF = MPAC_{vc}$ $BMBL = E_{ADDRWD_{dp}}$ If MODE $\neq$ VC: $VBUF = E_{ADDRWD_{vc}}$ $BMBL = MPAC_{dp}$ $AMAL = VBUF_x$ Perform "MULCOM" (see DMP) $TS_x = MPAC_{tp}$ + MPAC+2 (carries propagate) If most significant part overflows: Set OVFIND

Symbol	Order	Performance
VXSC (cont)		AMAL = $VBUF_y$ Perform "MULCOM" (see DMP) $TS_y = MPAC_{tp} + MPAC+2$ (carries propagate) If most significant part overflows: Set OVFIND AMAL = $VBUF_z$ Perform "MULCOM" (see DMP) $TS_z = MPAC_{tp} + MPAC+2$ (carries propagate) If most significant part overflows: Set OVFIND $MPAC_{vc} = TS_{vc}$ MODE = VC Proceed to "DANZIG"
VXV	1 30	Vector Cross Product.
VXV*	3 30	Replace $MPAC_{vc}$ with ($MPAC_{vc}$) * (vector at specified address). Set OVFIND if overflow. $(M_x, M_y, M_z) = MPAC_{vc}$ $(A_x, A_y, A_z) = E_{ADDRWD_{vc}}$ All multiplications done with "MULCOM" (see DMP) $TS_z = M_x A_y - M_y A_x$ If most significant part overflows: $TS_z = TS_z + 2^{28}$ sgn $TS+O_z$ (see note with BDSU) If above computation overflows: Set OVFIND $TS_y = M_z A_x - M_x A_z$ If most significant part overflows: $TS_y = TS_y + 2^{28}$ sgn $TS+O_y$ (see note with BDSU) If above computation overflows: Set OVFIND

Symbol	Order	Performance
VXV (cont)		$TS_x = M_y A_z - M_z A_y$ If most significant part overflows: $TS_x = TS_x + 2^{28}$ sgn $TS+0_x$ (see note with BDSU) If above computation overflows: Set OVFIND $MPAC_{vc} = TS_{vc}$ Proceed to "DANZIG"
V/SC	1 07	Vector Divided by Scalar.
V/SC*	3 07	Divide a vector by a scalar and store the result in $MPAC_{vc}$. If have been computing with vectors, the quantity at the specified address is considered a scalar; if have been computing with scalars, the quantity at the specified address is considered a vector. Set OVFIND if overflow. If MODE = VC: $BUF_{dp} = E_{ADDRWD_{dp}}$ If MODE $\neq$ VC: $BUF_{dp} = MPAC_{dp}$ $MPAC_{vc} = E_{ADDRWD_{vc}}$ MODE = VC Force sign agreement of each component of $MPAC_{vc}$ Force sign agreement of BUF_{dp} If BUF+0 = 0: Shift $MPAC_{vc}$ left 14 places If any component of $MPAC_{vc}$ overflows: $MPAC_x = \pm$ MAX (sign not significant) Set OVFIND Proceed to "DANZIG"

Symbol	Order	Performance
V/SC	(cont)	If $BUF_{dp} = 0$: $MPAC_x = \pm$ MAX (sign not significant) Set OVFIND Proceed to "DANZIG" If $\lvert BUF_{dp} \rvert \leq$ magnitude of any component of $MPAC_{vc}$: $MPAC_x$ = +MAX sgn (overflowing division) Set OVFIND Proceed to "DANZIG" Divide each component of $MPAC_{vc}$ by BUF_{dp}, setting DVSIGN to proper value and using algorithm of "DIVCOM" (see BDDV), starting at step #1 (with Den = BUF_{dp}, of course). $MPAC_{vc}$ = vector results from previous line Proceed to "DANZIG"

Shifting Operations

All general shift instructions (SL, SLR, SR, SRR, VSL, and VSR) use the same octal order (1 23), with the distinction between them made by bits 10-9 of the address-word parameter, and by the value of MODE. The format of the address-word parameter is as follows:

Bit 15: 0

Bit 14: 1

Bits 13-11: 0

Bit 10: 1 for rounded shifts, 0 for unrounded ones (always 0 for vector shifts)

Bit 9: 1 for right shift, 0 for left shift

Bit 8: 1 (as a "pseudo-sign bit" for indexed shifts)

Bits 7-1: Amount of shift

If index register 2 is specified, the complete word is complemented; as stored in memory, of course, one is added to the quantity (as discussed in Section VIA for all Indexable Operations).

Symbol	Order	Performance
NORM	1 17	Normalize. SLC may also be used as symbol.
NORM*	3 17	Force sign agreement of $MPAC_{tp}$, and then shift left until most significant magnitude bit is different from the sign bit. Store the negative of the number of shifts performed in the address specified (single precision). Order will not push up properly.

TS = 0

Force sign agreement of $MPAC_{tp}$ (unless magnitude 0)

If $MPAC_{tp} = 0$:

$E_{ADDRWD_{sp}} = -0$

Proceed to "DANZIG"

If MPAC+0 + MPAC+0 overflows:

$E_{ADDRWD_{sp}} = -TS$

Proceed to "DANZIG"

$MPAC_{tp} = MPAC_{tp} + MPAC_{tp}$ (i.e. left shift of 1 place)

TS = TS + 1

Proceed to fourth line of NORM (recheck for overflow)

Symbol	Order	Performance
SL	1 23	Scalar Shift Left.
SL*	3 23	Shift $MPAC_{tp}$ left by appropriate number of places specified by address-word parameter. If shift amount is negative, shift right by the corresponding number of places. Set OVFIND if overflow.
Address bit 10-9= 00_2		

If bits 7-1 of ADDRWD = 0, proceed to "DANZIG"

MPTEMP = (bits 7-1 of ADDRWD) - 1

If bit 8 of ADDRWD = 0:

MPTEMP = 126 - MPTEMP

Proceed to fourth line of SR instruction

(Number of shifts effectively stored as 128 + count in bits 8-1; if SL* causes number to be negative, then MPTEMP set to (number of right shifts -1), and SR performed)

If MODE = VC, proceed to VSL instruction

Shift $MPAC_{tp}$ left by (MPTEMP +1) places, by performing:

$MPAC_{tp} = MPAC_{tp} + MPAC_{tp}$ repeatedly

If most significant part overflows: (for any shift)

Set OVFIND

Continue shifting

Proceed to "DANZIG"

Symbol	Order	Performance
SL1	0 05	Short Scalar Shift Left.
SL2	0 15	
SL3	0 25	Shift $MPAC_{tp}$ left by the number of places specified by the operation code. Set OVFIND if overflow.
SL4	0 35	
No Address		

If MODE = VC, proceed to VSLi instruction

Shift $MPAC_{tp}$ left by the appropriate number of places, by performing:

$MPAC_{tp} = MPAC_{tp} + MPAC_{tp}$ repeatedly

If most significant part overflows: (for any shift)

Set OVFIND

Continue shifting

Proceed to "DANZIG"

Symbol	Order	Performance
SLC		Same as NORM (alternate mnemonic).
SLR	1 23	Scalar Shift Left and Round.
SLR*	3 23	Shift $MPAC_{tp}$ left by appropriate number of places specified by address-word parameter. If shift amount is negative, shift right by the corresponding number of places. Round $MPAC_{tp}$ to double precision after shifting. Set OVFIND if overflow.

Address bit 10-9= 10_2

If bits 7-1 of ADDRWD = 0:

MODE = DP

$MPAC_{tp} = MPAC_{tp} + MPAC+2$ (carries propagate)

If most significant part overflows:

Set OVFIND

MPAC+2 = 0

Proceed to "DANZIG"

MPTEMP = (bits 7-1 of ADDRWD) - 1

If bit 8 of ADDRWD = 0:

MPTEMP = 126 - MPTEMP

Proceed to fourth line of SRR instruction

(Cf. SL discussion)

If MODE = VC, proceed to VSL instruction

Shift $MPAC_{tp}$ left by (MPTEMP +1) places, by performing:

$MPAC_{tp} = MPAC_{tp} + MPAC_{tp}$ repeatedly

If most significant part overflows: (for any shift)

Set OVFIND

Continue shifting

MODE = DP

$MPAC_{tp} = MPAC_{tp} + MPAC+2$ (carries propagate)

If most significant part overflows:

Set OVFIND

MPAC+2 = 0

Proceed to "DANZIG"

Symbol	Order	Performance
SL1R SL2R SL3R SL4R	0 01 0 11 0 21 0 31	Short Scalar Shift Left and Round. Shift $MPAC_{tp}$ left by the number of places specified by the operation code. Round $MPAC_{tp}$ to double precision after shifting. Set OVFIND if overflow.
No Address		If MODE = VC, proceed to VSLi instruction Shift $MPAC_{tp}$ left by the appropriate number of places, by performing: $MPAC_{tp} = MPAC_{tp} + MPAC_{tp}$ repeatedly If most significant part overflows: (for any shift) Set OVFIND Continue shifting MODE = DP $MPAC_{tp} = MPAC_{tp} + MPAC+2$ (carries propagate) If most significant part overflows: Set OVFIND MPAC+2 = 0 Proceed to "DANZIG"
SR	1 23	Scalar Shift Right.
SR*	3 23	Shift $MPAC_{tp}$ right by appropriate number of places specified by address-word parameter. If shift amount is negative, shift left by the corresponding number of places. Set OVFIND if overflow (only on left shift).
Address bit 10-9= 01_2		If bits 7-1 of ADDRWD = 0, proceed to "DANZIG" MPTEMP = (bits 7-1 of ADDRWD) - 1 If bit 8 of ADDRWD = 0: MPTEMP = - (MPTEMP - 126) (0 result is +, for coding) Proceed to fourth line of SL instruction (Cf. SL discussion) If MODE = VC, proceed to VSR instruction

Symbol	Order	Performance
SR (cont)		Shift $MPAC_{tp}$ right by (MPTEMP +1) places, by multiplying by appropriate negative power of 2 (shifts in excess of 13 are accomplished by movement of the components of $MPAC_{tp}$ as complete words, until the remaining number of shifts is less than 14) Proceed to "DANZIG"
SR1 SR2 SR3 SR4 No Address	0 07 0 17 0 27 0 37	Short Scalar Shift Right. Shift $MPAC_{tp}$ right by the number of places specified by the operation code. If MODE = VC, proceed to VSRi instruction Shift $MPAC_{tp}$ right by the appropriate number of places, by multiplying by appropriate negative power of 2 Proceed to "DANZIG"
SRR	1 23	Scalar Shift Right and Round.
SRR*	3 23 Address bit 10-9= 11_2	Shift $MPAC_{tp}$ right by appropriate number of places specified by address-word parameter. If shift amount is negative, shift left by the corresponding number of places. Round $MPAC_{tp}$ to double precision after shifting. Set OVFIND if overflow. If bits 7-1 of ADDRWD = 0: MODE = DP $MPAC_{tp}$ = $MPAC_{tp}$ + MPAC+2 (carries propagate) If most significant part overflows: Set OVFIND MPAC+2 = 0 Proceed to "DANZIG" MPTEMP = (bits 7-1 of ADDRWD) - 1

Symbol	Order	Performance
SRR (cont)		If bit 8 of ADDRWD = 0: MPTEMP = - (MPTEMP - 126) (0 result is +, for coding) Proceed to fourth line of SLR instruction (Cf. SL discussion) If MODE = VC, proceed to VSR instruction Shift $MPAC_{tp}$ right by (MPTEMP +1) places, by multiplying by appropriate negative power of 2 (shifts in excess of 13 are accomplished by movement of the components of $MPAC_{tp}$ as complete words, until the remaining number of shifts is less than 14) $MPAC_{tp} = MPAC_{tp} + MPAC+2$ (carries propagate, no overflow) MODE = DP MPAC+2 = 0 Proceed to "DANZIG"
SR1R SR2R SR3R SR4R No Address	0 03 0 13 0 23 0 33	Short Scalar Shift Right and Round. Shift $MPAC_{tp}$ right by the number of places specified by the operation code. Round $MPAC_{tp}$ to double precision after shifting. If MODE = VC, proceed to VSRi instruction Shift $MPAC_{tp}$ right by the appropriate number of places, by multiplying by the appropriate negative power of 2 $MPAC_{tp} = MPAC_{tp} + MPAC+2$ (carries propagate, no overflow) MODE = DP MPAC+2 = 0 Proceed to "DANZIG"
VSL	1 23	Vector Shift Left.
VSL*	3 23	Shift each component of $MPAC_{vc}$ left by appropriate number of places specified by address-word parameter. If shift amount is negative, shift right by the corresponding number of places. Set OVFIND if overflow.
Address bit 10-9= 00_2		

Symbol	Order	Performance
VSL (cont)		If bits 7-1 of ADDRWD = 0, proceed to "DANZIG" MPTEMP = (bits 7-1 of ADDRWD) - 1 If bit 8 of ADDRWD = 0: MPTEMP = 126 - MPTEMP Proceed to fourth line of VSR instruction (Cf. SL discussion) If MODE $\neq$ VC, proceed to SL instruction Shift $MPAC_{vc}$ left by (MPTEMP +1) places, by performing: $MPAC_{vc} = MPAC_{vc} + MPAC_{vc}$ repeatedly If most significant part overflows: (any component, any shift) $MPAC_i = MPAC_i + 2^{28}$ sgn $MPAC+O_i$ (see note with BDSU) If above computation overflows: Set OVFIND Continue shifting Proceed to "DANZIG"
VSL1 VSL2 VSL3 ⋮ VSL8	0 01 0 05 0 11 ⋮ 0 35	Short Vector Shift Left. Shift each component of $MPAC_{vc}$ left by the number of places specified by the operation code. Set OVFIND if overflow.
No Address		If MODE $\neq$ VC: If shifts odd, proceed to SLiR instruction If shifts even, proceed to SLi instruction Shift $MPAC_{vc}$ left by the appropriate number of places, by performing: $MPAC_{vc} = MPAC_{vc} + MPAC_{vc}$ repeatedly If most significant part overflows: (any component, any shift) $MPAC_i = MPAC_i + 2^{28}$ sgn $MPAC+O_i$ (see note with BDSU) If above computation overflows: Set OVFIND Continue shifting Proceed to "DANZIG"

Symbol	Order	Performance
VSR	1 23	Vector Shift Right.
VSR*	3 23	Shift each component of $MPAC_{vc}$ right by appropriate number of places specified by address-word parameter, and then round. If shift amount is negative, shift left by the corresponding number of places. Set OVFIND if overflow (only if left shift).
Address bit 10-9= 01_2		

If bits 7-1 of ADDRWD = 0, proceed to "DANZIG"

MPTEMP = (bits 7-1 of ADDRWD) - 1

If bit 8 of ADDRWD = 0:

MPTEMP = - (MPTEMP - 126) (0 result is +, for coding)

Proceed to fourth line of VSL instruction (Cf. SL discussion)

If MODE $\neq$ VC, proceed to SR instruction

If MPTEMP $\geq$ 13:

MPTEMP = MPTEMP - 13

Shift each component of $MPAC_{vc}$ right 14 places and round to double precision

If MPTEMP = 0, proceed to "DANZIG"

If MPTEMP $\geq$ 14:

MPTEMP = MPTEMP - 14 and proceed to 3rd previous line (shift $MPAC_{vc}$ again)

MPTEMP = MPTEMP - 1

Shift $MPAC_{vc}$ right by (MPTEMP +1) places, by multiplying by appropriate power of 2, and round each component to double precision (each component treated as a triple-precision number with least significant third initially 0). For a right shift of 15 or 29 places, rounding could give a one-bit error in the answer.

Proceed to "DANZIG"

VSR1	0 03	Short Vector Shift Right.
VSR2	0 07	
VSR3	0 13	Shift each component of $MPAC_{vc}$ right by the number of places specified by the operation code, then round.
⋮	⋮	
VSR8	0 37	
No Address		

Symbol	Order	Performance
VSRi (cont)		If MODE $\neq$ VC: If shifts odd, proceed to SRiR instruction If shifts even, proceed to SRi instruction Shift $MPAC_{vc}$ right by the appropriate number of places, by multiplying by appropriate power of 2, and round each component to double precision (cf. VSR) Proceed to "DANZIG"

Transmission Operations

Symbol	Order	Performance
DLOAD	1 06	Double Precision Load.
DLOAD*	3 06	Load $MPAC_{dp}$ with double precision quantity at specified address.

$MPAC_{dp} = E_{ADDRWD_{dp}}$
MPAC+2 = 0
MODE = DP
Proceed to "DANZIG"

ITA	2 33	Interpretive Transfer Address to Storage. STQ may also be used as symbol.
No Indexing		Load specified address with QPRET (which is loaded with return address information by e.g. a CALL order). Instruction does not work properly if address in fixed memory.

Perform "15ADRERS" (see Section VIC)
$E_{POLISH_{sp}} = QPRET$ (Relative Address 42D, see Section VID)
Proceed to "DANZIG"

PDDL	1 12	Store in Push-down List and Double Precision Load.
PDDL*	3 12	Store MPAC (double precision, triple precision, or vector) in push-down list (see Section VID) and load $MPAC_{dp}$ with quantity at specified address. If no address is given, $MPAC_{dp}$ loaded from push-down list, and MPAC stored in same push-down list locations.

Set $E_{PUSHLOC_{dp}} = MPAC_{dp}$ and $MPAC_{dp} = E_{ADDRWD_{dp}}$
If MODE = DP:
    MPAC+2 = 0
    PUSHLOC = PUSHLOC +2
    Proceed to "DANZIG"

Symbol	Order	Performance

PDDL (cont)

If MODE = TP:

MODE = DP

$E_{PUSHLOC+2}$ = MPAC+2

MPAC+2 = 0

PUSHLOC = PUSHLOC +3

Proceed to "DANZIG"

MODE = DP

$E_{PUSHLOC+2_{dp}} = MPAC_y$ (note that $MPAC_{dp}$ same as $MPAC_x$)

$E_{PUSHLOC+4_{dp}} = MPAC_z$

MPAC+2 = 0

PUSHLOC = PUSHLOC +6

Proceed to "DANZIG"

PDVL 1 14 Store in Push-down List and Vector Load.

PDVL* 3 14 Store MPAC (double precision, triple precision, or vector) in push-down list (see Section VID) and load $MPAC_{vc}$ with quantity at specified address. If no address is given, $MPAC_{vc}$ loaded from push-down list, and MPAC stored in same push-down list locations.

Set $E_{PUSHLOC_{dp}} = MPAC_{dp}$ and $MPAC_x = E_{ADDRWD_{dp}}$ (note that $MPAC_{dp}$ same cells as $MPAC_x$).

If MODE = DP:

PUSHLOC = PUSHLOC +2

$MPAC_y = E_{ADDRWD+2_{dp}}$

$MPAC_z = E_{ADDRWD+4_{dp}}$

MODE = VC

Proceed to "DANZIG"

Symbol	Order	Performance
PDVL	(cont)	If MODE = TP: $MPAC_y = E_{ADDRWD+2_{dp}}$ $MPAC_z = E_{ADDRWD+4_{dp}}$ $E_{PUSHLOC+2} = MPAC+2$ PUSHLOC = PUSHLOC +3 MODE = VC Proceed to "DANZIG" Set $E_{PUSHLOC+2} = MPAC_y$ and $MPAC_y = E_{ADDRWD+2_{dp}}$ Set $E_{PUSHLOC+4} = MPAC_z$ and $MPAC_z = E_{ADDRWD+4_{dp}}$ PUSHLOC = PUSHLOC +6 Proceed to "DANZIG"
PUSH	0 36	Store in Push-down List.
No Address		Store MPAC (double precision, triple precision, or vector) in push-down list (see Section VID) and leave MPAC loaded.
		$E_{PUSHLOC_{dp}} = MPAC_{dp}$ If MODE = DP: PUSHLOC = PUSHLOC +2 Proceed to "DANZIG" If MODE = TP: $E_{PUSHLOC+2} = MPAC+2$ PUSHLOC = PUSHLOC +3 Proceed to "DANZIG" $E_{PUSHLOC+2} = MPAC_y$ (note that $MPAC_{dp}$ same cells as $MPAC_x$) $E_{PUSHLOC+4} = MPAC_z$ PUSHLOC = PUSHLOC +6 Proceed to "DANZIG"

Symbol	Order	Performance
SETPD	1 37	Set Push-down List Address.
No Indexing		Set PUSHLOC (see Section VID) to value of address-word parameter information.
		PUSHLOC = ADDRWD (FIXLOC already added as part of generation of ADDRWD value)
		Proceed to second line of "DANZIG"
SLOAD	1 10	Single Precision Load.
SLOAD*	3 10	Load $MPAC_{sp}$ with single precision quantity at specified address (setting 0 the other two components of $MPAC_{tp}$). Instruction will not push up properly.
		$MPAC+0 = E_{ADDRWD_{sp}}$
		MPAC+1 = 0
		MPAC+2 = 0
		MODE = DP
		Proceed to "DANZIG"
SSP	1 11	Store Single Precision Constant.
SSP*	3 11	Set the single precision cell whose address is given by the first address-word parameter to the value (number) of the second address-word parameter. The second address-word parameter is a constant used directly. Instruction will not push up properly.
Two Address Words		
		LOC = LOC +1
		$E_{ADDRWD_{sp}} = E_{LOC_{sp}}$ (loads quantity at address LOC into E)
		Proceed to "DANZIG"
STADR	0 32	Cause Push-up on Store Address.
No Address		Complement the next line of coding and treat it as a storage command (the assembler performs a compensating complement). This causes the normally positive storage command with which STADR employed to be negative, forcing operands to be taken from push-down list (see Section VID).

Symbol	Order	Performance
STADR	(cont)	FBANK = bits 15-11 of BANKSET LOC = LOC +1 TS = - ($E_{LOC_{sp}}$) (positive value of store command) Proceed to "DOSTORE" (to process storage command), Section VIC.
STCALL	---	Store and do a CALL Instruction.
No Indexing		Store MPAC (double precision, triple precision, or vector) in specified address and do a CALL instruction. Neither address can be indexed. See "DOSTORE" (Section VIC) for mechanization.
STODL	---	Store and do a DLOAD Instruction.
STODL*	---	Store MPAC (double precision, triple precision, or vector) in specified address (which cannot be indexed) and do a DLOAD instruction (which can be indexed via STODL*). See "DOSTORE" (Section VIC) for mechanization.
STORE	---	Store in Address. Store MPAC (double precision, triple precision, or vector) in specified address (which can be indexed), and leave MPAC loaded. See "DOSTORE" (Section VIC) for mechanization.
STOVL	---	Store and do a VLOAD Instruction.
STOVL*	---	Store MPAC (double precision, triple precision, or vector) in specified address (which cannot be indexed), and do a VLOAD instruction (which can be indexed via STOVL*). See "DOSTORE" (Section VIC) for mechanization.

Symbol	Order	Performance
STQ		Same as ITA (alternate mnemonic, Store QPRET).
TLOAD	1 05	Triple Precision Load.
TLOAD*	3 05	Load $MPAC_{tp}$ with triple precision quantity at specified address.

$MPAC_{tp} = E_{ADDRWD_{tp}}$

MODE = TP

Proceed to "DANZIG"

VLOAD	1 00	Vector Load.
VLOAD*	3 00	Load $MPAC_{vc}$ with vector quantity at specified address.

$MPAC_{vc} = E_{ADDRWD_{vc}}$

MODE = VC

Proceed to "DANZIG"

Control Operations

Symbol	Order	Performance
BHIZ	2 31	Branch if High-order Zero.
No Indexing		If magnitude of MPAC+0 = 0, transfer to specified address.
		If MPAC+0 = $\pm$ 0: Proceed to GOTO instruction Proceed to "DANZIG"
BMN	2 27	Branch if Minus.
No Indexing		If $MPAC_{tp}$ negative non-zero, transfer to specified address.
		If $MPAC_{tp} <$ -0: Proceed to GOTO instruction Proceed to "DANZIG"
BOV	2 37	Branch on Overflow.
No Indexing		If OVFIND set (i.e. non-zero), reset it (to 0) and transfer to specified address.
		If OVFIND $\neq$ 0: OVFIND = 0 Proceed to GOTO instruction Proceed to "DANZIG"
BOVB	2 36	Branch on Overflow to Basic.
No Indexing		If OVFIND set (i.e. non-zero), reset it (to 0) and transfer to specified address expecting machine-language orders (rather than the interpretive-language orders of BOV).
		If OVFIND $\neq$ 0: OVFIND = 0 Proceed to RTB instruction Proceed to "DANZIG"

Symbol	Order	Performance
BPL	2 26	Branch if Positive.
No Indexing		If $MPAC_{tp}$ is zero or positive, transfer to specified address.
		If $MPAC_{tp} \geq -0$:
		Proceed to GOTO instruction
		Proceed to "DANZIG"
BZE	2 24	Branch if Zero.
No Indexing		If magnitude of $MPAC_{tp} = 0$, transfer to specified address.
		If $MPAC_{tp} = \pm 0$:
		Proceed to GOTO instruction
		Proceed to "DANZIG"
CALL	2 32	Transfer with Return Address. CALRB is used as symbol if return is accomplished in machine-language (to suppress assembler alarm).
No Indexing		Transfer to specified address, leaving return address in QPRET (Relative Address 42D, see Section VID). QPRET format satisfactory for use by BANKCALL etc. (i.e. in FCADR form, see Section VC).
		QPRET = (bits 15-11 of BANKSET) + (LOC + 1 - 2000_8)
		Proceed to GOTO instruction
CALRB		Same as CALL (alternate mnemonic, Call with Return in Basic, to suppress assembler alarm).

Symbol	Order	Performance
CCALL	1 15	Computed CALL. CCLRB used if return in machine language.
CCALL* Two Address Words	3 15	Transfer to a step whose address is selected from an address table in fixed memory, leaving return address in QPRET (Relative Address 42D, see Section VID). The address table entry is selected from the contents of the cell specified by the first address-word parameter and the value (number) of the second address-word parameter. The second address-word parameter would be expected to contain the base address of the address table, with the cell specified by the first address-word parameter giving the increment to this base address. Note that this "composed" address is used to find the cell where the required transfer address is stored: it is <u>not</u> the address to which control is transferred. Hence the order is an "indexed indirect transfer", rather than an "indexed transfer", order: use of X1 or X2 is limited to determining memory location of first address. Instruction will not push up properly.

LOC = LOC +1

TS = $E_{ADDRWD_{sp}}$ + $E_{LOC_{sp}}$ (second term is quantity at address LOC)

FBANK = bits 15-11 of TS

TS = (bits 10-1 of TS) + 2000_8

POLISH = E_{TS}

Proceed to CALL instruction

CCLRB		Same as CCALL (alternate mnemonic, Computed CALL with Return in Basic, to suppress assembler alarm).
CGOTO	1 04	Computed GOTO.
CGOTO* Two Address Words	3 04	Transfer to a step whose address is selected from an address table in fixed memory. Except for the fact that QPRET not set, performance same as CCALL (see above).

Symbol	Order	Performance
CGOTO (cont)		TS = $E_{ADDRWD_{sp}}$ + $E_{LOC+1_{sp}}$ FBANK = bits 15-11 of TS TS = (bits 10-1 of TS) + 2000_8 POLISH = E_{TS} Proceed to GOTO instruction
EXIT	0 00	Exit from Interpreter.
No Address		Return to machine-language coding, starting at address following the last one used in interpretive language. BBANK = BANKSET Proceed to address specified by FBANK and (LOC +1)
GOTO	2 25	Transfer Control.
No Indexing		Transfer to specified address. Except for the fact that QPRET is not set, performance same as CALL (the coding for GOTO serves as a common control-transfer sequence). If specified address is in erasable, the contents of that address are used as the transfer address. If these contents are also an erasable address, it is treated the same way, etc. (until an address in fixed memory is obtained). If bits 15-12 of POLISH $\neq$ 0: BBANK = BANKSET FBANK = bits 15-11 of POLISH LOC = (bits 10-1 of POLISH) + 2000_8 Proceed to third line of "INTPRET" If POLISH $< 55_8$: POLISH = $E_{FIXLOC + POLISH}$ Proceed to GOTO instruction (repeat checks again) EBANK = bits 11-9 of POLISH TS = (bits 8-1 of POLISH) + 1400_8 POLISH = E_{TS} Proceed to GOTO instruction (repeat checks again)

Symbol	Order	Performance
ITCQ		Same as RVQ (alternate mnemonic, Interpretive Transfer Control via QPRET).
RTB	2 30	Return to Basic.
No Indexing		Return to machine-language (i.e. "basic") orders starting at specified address. Differs from EXIT in that a transfer address is specified. Can return via Q register or via "DANZIG". Proceed to address specified in CADR format by POLISH (using "SWCALL" routine, with return address set to cause transfer to "DANZIG")
RVQ	0 34	Return Via QPRET. ITCQ may also be used as symbol.
No Address		Transfer to address specified by contents of QPRET (Relative Address 42D, see Section VID). Can be used to return from a subroutine entered via a CALL, CCALL, or STCALL instruction, provided that other CALL-type instructions not given after subroutine entered. QPRET can refer to fixed memory only. BANK = BANKSET FBANK = bits 15-11 of QPRET LOC = (bits 10-1 of QPRET) + 2000_8 Proceed to third line of "INTPRET"

Index Register Oriented Operations

Symbol	Order	Performance
AXC,1	2 03	Address to Index Complemented.
AXC,2	2 02	Load single precision index register X1 or X2 with the complement of the value (number) of the address-word parameter.

If AXC,1:

X1 = - POLISH

If AXC,2:

X2 = - POLISH

Proceed to "DANZIG"

AXT,1	2 01	Address to Index True.
AXT,2	2 00	Load single precision index register X1 or X2 with the value (number) of the address-word parameter.

If AXT,1:

X1 = POLISH

If AXT,2:

X2 = POLISH

Proceed to "DANZIG"

INCR,1	2 15	Increment Index by Address.
INCR,2	2 14	Increment single precision index register X1 or X2 with the value (number) of the address-word parameter, ignoring overflow. A machine language order has same mnemonic.

If INCR,1:

X1 = X1 + POLISH (modulo 2^{14})

If INCR,2:

X2 = X2 + POLISH (modulo 2^{14})

Proceed to "DANZIG"

Symbol	Order	Performance
LXA,1	2 05	Load Index from Address.
LXA,2	2 04	Load single precision index register X1 or X2 with the contents of the erasable memory cell whose address is given by address-word parameter. Instruction does not work properly if address in fixed memory.

Perform "15ADRERS" (see Section VIC)
If LXA,1:

$$X1 = E_{POLISH}$$

If LXA,2:

$$X2 = E_{POLISH}$$

Proceed to "DANZIG"

LXC,1	2 07	Load Index from Complement of Address.
LXC,2	2 06	Load single precision index register X1 or X2 with the complement of the contents of the erasable memory cell whose address is given by address-word parameter. Instruction does not work properly if address in fixed memory.

Perform "15ADRERS" (see Section VIC)
If LXC,1:

$$X1 = -E_{POLISH}$$

If LXC,2:

$$X2 = -E_{POLISH}$$

Proceed to "DANZIG"

SXA,1	2 11	Store Index in Address.
SXA,2	2 10	Store single precision index register X1 or X2 in the erasable memory cell whose address is given by address-word parameter. Instruction does not work properly if address in fixed memory (an erasable memory cell would be modified anyhow).

Symbol	Order	Performance
SXA,i	(cont)	Perform "15ADRERS" (see Section VIC) If SXA,1: E_{POLISH} = X1 If SXA,2: E_{POLISH} = X2 Proceed to "DANZIG"
TIX,1	2 17	Transfer on Index.
TIX,2	2 16	If the specified index register minus the corresponding step register is positive, set index register to the difference and transfer to the specified address. The step registers are Relative Addresses 40D and 41D (see Section VID), and can be set by an SSP instruction. Overflow is ignored when index register updated. If TIX,1: TS = X1 - S1 If TS > 0: X1 = TS (modulo 2^{14}) Proceed to GOTO instruction If TIX,2: TS = X2 - S2 If TS > 0: X2 = TS (modulo 2^{14}) Proceed to GOTO instruction Proceed to "DANZIG"
XAD,1	2 21	Add to Index from Address.
XAD,2	2 20	Add to the single precision index register X1 or X2 the contents of the erasable memory cell whose address is given by address-word parameter. Instruction does not work properly if address in fixed memory. Overflow is ignored.

Symbol	Order	Performance
XAD,i	(cont)	Perform "15ADRERS" (see Section VIC) If XAD,1: $X1 = X1 + E_{POLISH}$ (modulo 2^{14}) If XAD,2: $X2 = X2 + E_{POLISH}$ (modulo 2^{14}) Proceed to "DANZIG"
XCHX,1	2 13	Exchange Contents of Index and Address.
XCHX,2	2 12	Exchange the contents of the single precision index register X1 or X2 with the contents of the erasable memory cell whose address is given by address-word parameter. Instruction does not work properly if address in fixed memory (erasable memory cell set anyhow). Perform "15ADRERS" (see Section VIC) If XCHX,1: Set $X1 = E_{POLISH}$ and $E_{POLISH} = X1$ If XCHX,2: Set $X2 = E_{POLISH}$ and $E_{POLISH} = X2$ Proceed to "DANZIG"
XSU,1	2 23	Subtract Address from Index.
XSU,2	2 22	Subtract from the single precision index register X1 or X2 the contents of the erasable memory cell whose address is given by address-word parameter. Instruction does not work properly if address in fixed memory. Overflow is ignored. Perform "15ADRERS" (see Section VIC) If XSU,1: $X1 = X1 - E_{POLISH}$ (modulo 2^{14}) If XSU,2: $X2 = X2 - E_{POLISH}$ (modulo 2^{14}) Proceed to "DANZIG"

Logical Bit Operations

All logical bit operations use the same octal order (2 34) with the distinction between them being made by bits 8-5 of the first address-word parameter (given below as the "Code"): these bits are set suitably by the assembler. The following parameters are common to all the logical bit operations:

SWBIT: bit specified by bits 4-1 of POLISH. Bit 15 to bit 1 respectively are selected for bits 4-1 equal to 00 - 16_8, and all bits (bit 15 through bit 1) are selected for bits 4-1 of POLISH equal to 17_8 (not used). Setting of bits accompanied with suitable inhibiting of interrupts as necessary.

SWWORD: word selected by bits 14-9 of POLISH. The word selected has address "STATE" + 2^{-8} x these bits, so that if bit 10 were a binary 1 and bits 14-11 and 9 were 0, the word with address "STATE" +2 would be selected. "STATE" is the address of the first of a series of flag-bit words stored in erasable memory, and conventionally is assigned address 0074_8.

Symbol	Code	Performance
BOF		Same as BOFF (alternate mnemonic).
BOFCLR	12	Clear Bit and Transfer if Bit Was Off.
Two Address Words		Set SWBIT in SWWORD to zero. If this bit was already zero, transfer to specified address.

```
TS = SWWORD
Set SWBIT of SWWORD = 0
If SWBIT of TS = 0:
     FBANK = BANKSET
     POLISH = E LOC+1
     Proceed to GOTO instruction
LOC = LOC +1
Proceed to "DANZIG"
```

Symbol	Code	Performance
BOFF	16	Transfer if Bit Is Off. BOF may also be used as symbol.
Two Address Words		If SWBIT in SWWORD is zero, transfer to specified address.

If SWBIT of SWWORD = 0:

FBANK = BANKSET

POLISH = E_{LOC+1}

Proceed to GOTO instruction

LOC = LOC +1

Proceed to "DANZIG"

BOFINV	06	Invert Bit and Transfer if Bit Was Off.
Two Address Words		Complement SWBIT in SWWORD. If this bit was previously zero, transfer to specified address.

TS = SWWORD

Complement SWBIT in SWWORD

If SWBIT of TS = 0:

FBANK = BANKSET

POLISH = E_{LOC+1}

Proceed to GOTO instruction

LOC = LOC +1

Proceed to "DANZIG"

BOFSET	02	Set Bit and Transfer if Bit Was Off.
Two Address Words		Set SWBIT in SWWORD to one. If this bit was previously zero, transfer to specified address.

TS = SWWORD

Set SWBIT of SWWORD = 1

If SWBIT of TS = 0:

FBANK = BANKSET

POLISH = E_{LOC+1}

Proceed to GOTO instruction

Symbol	Code	Performance
BOFSET (cont)		LOC = LOC +1 Proceed to "DANZIG"
BON	14	Transfer if Bit Is On.
Two Address Words		If SWBIT in SWWORD is one, transfer to specified address.
		If SWBIT of SWWORD = 1: FBANK = BANKSET POLISH = E_{LOC+1} Proceed to GOTO instruction LOC = LOC +1 Proceed to "DANZIG"
BONCLR	10	Clear Bit and Transfer if Bit Was On.
Two Address Words		Set SWBIT in SWWORD to zero. If this bit was previously one, transfer to specified address.
		TS = SWWORD Set SWBIT of SWWORD = 0 If SWBIT of TS = 1: FBANK = BANKSET POLISH = E_{LOC+1} Proceed to GOTO instruction LOC = LOC +1 Proceed to "DANZIG"
BONINV	04	Invert Bit and Transfer if Bit Was On.
Two Address Words		Complement SWBIT in SWWORD. If this bit was previously one, transfer to specified address.
		TS = SWWORD Complement SWBIT in SWWORD

Symbol	Code	Performance
BONINV	(cont)	If SWBIT of TS = 1: FBANK = BANKSET POLISH = E_{LOC+1} Proceed to GOTO instruction LOC = LOC +1 Proceed to "DANZIG"
BONSET	00	Set Bit and Transfer if Bit Was On.
Two Address Words		TS = SWWORD Set SWBIT of SWWORD = 1 If SWBIT of TS = 1: FBANK = BANKSET POLISH = E_{LOC+1} Proceed to GOTO instruction LOC = LOC +1 Proceed to "DANZIG"
CLEAR	13	Clear Bit. CLR may also be used as symbol. Set SWBIT in SWWORD to zero. Set SWBIT of SWWORD = 0 Proceed to "DANZIG"
CLR		Same as CLEAR (alternate mnemonic).
CLRGO	11	Clear Bit and Transfer.
Two Address Words		Set SWBIT in SWWORD to zero and transfer to specified address. Set SWBIT in SWWORD = 0 FBANK = BANKSET POLISH = E_{LOC+1} Proceed to GOTO instruction
INV		Same as INVERT (alternate mnemonic).

Symbol	Code	Performance
INVERT	07	Invert Bit. INV may also be used as symbol. Complement SWBIT in SWWORD. Complement SWBIT of SWWORD Proceed to "DANZIG"
INVGO	05	Invert Bit and Transfer.
Two Address Words		Complement SWBIT in SWWORD and transfer to specified address. Complement SWBIT of SWWORD FBANK = BANKSET POLISH = E_{LOC+1} Proceed to GOTO instruction
SET	03	Set Bit. Set SWBIT in SWWORD to zero. Set SWBIT of SWWORD = 1 Proceed to "DANZIG"
SETGO	01	Set Bit and Transfer.
Two Address Words		Set SWBIT in SWWORD to one and transfer to specified address. Set SWBIT of SWWORD = 1 FBANK = BANKSET POLISH = E_{LOC+1} Proceed to GOTO instruction

Interpretation of Code

Bit Value	Bits 4-3	Bits 2-1
00_2	Set bit.	Transfer if bit was on.
01_2	Invert bit.	Transfer unconditionally.
10_2	Clear bit.	Transfer if bit was off.
11_2	Leave bit alone.	Do not transfer.

VIC Addresses and Interpreter Control

Although the bulk of the interpreter performance capabilities is embodied in the various instructions in its repertoire, an understanding of the methods by which addresses are determined and the interpreter control logic is implemented can also be useful. Many of the special features of the interpreter are due to these two areas, and if their mechanization is well understood, then the need for exhaustive (and sometimes puzzling) lists of special cases can be eliminated.

For presentation convenience, this section has been divided into the following categories:

Overall Interpreter Control
Interpreter Address Determination
Interpreter Storage Orders
Interpreter Transfer to Operation

In a few cases, it may be noticed that there seems to be redundancy between the logic shown in this section and that shown for the individual instructions. This is usually due to the fact that the individual instructions in Section VIB are presented as self-contained computations, even though the actual program mechanization is not necessarily designed in this manner. For example, separate mnemonics are assigned to vector and scalar shifts, although both use the same octal codes. Consequently, it was necessary in Section VIB to show checks for vector vs. scalar computations within each shift description, although the actual interpreter software makes these checks in order to determine what operation was specified originally.

Overall Interpreter Control

INTPRET — To enter interpretive language coding operation, the program performs the instruction TC "INTPRET" (an address in fixed-fixed memory bank 03).

Release interrupts

LOC = Q (FBANK and LOC now have address after the TC "INTPRET")

BANKSET = BBANK (includes EBANK retention)

INTBT15 = bit 15 of BBANK (distinguish between low and high banks)

EDOP = 0

Proceed to "NEWOPS"

DANZIG — Most instructions (including RTB orders) end by giving control to the start of "DANZIG".

BBANK = BANKSET

If EDOP > 0:

CYR = (EDOP - 1), cycled right 1 place

Shift EDOP right 7 places (making it zero)

Proceed to "OPJUMP"

If NEWJOB > 0:

Proceed to "CHANG2" (see Section VIIB)

LOC = LOC +1

Proceed to "NEWOPS"

NEWOPS

TS = E_{LOC} (address also specified by FBANK contents)

If TS > 0:

Proceed to "DOSTORE"

If TS = 0, error

TS = |TS| - 1, limited $\geq +0$

EDOP = bits 14-8 of TS, shifted right 7 places

CYR = bits 7-1 of TS, cycled right 1 place

Proceed to "OPJUMP"

OPJUMP

If CYR = 0, proceed to EXIT instruction

If bit 15 of CYR = 0:

Cycle CYR right 1 place and proceed to "OPJUMP2" (prefix was 0 or 2 for transfer)

Cycle CYR right 1 place

If bit 15 of CYR = 1:

Proceed to "INDEX" (prefix was 3)

Proceed to "DIRADRES" (prefix was 1)

OPJUMP2

If bit 15 of CYR = 0:

Cycle CYR right 1 place and proceed to "OPJUMP3" (prefix was 0 if transfer)

Cycle CYR right 1 place

LOC = LOC +1

POLISH = E_{LOC} (address also specified by FBANK contents)

FBANK = 1 (i.e. bit 11 set)

Proceed to "MISCJUMP"

OPJUMP3

FBANK = 0

If bit 15 of CYR = 0:

Proceed to "UNAJUMP" (operation code was even if transfer)

If MODE = VC, proceed to appropriate short vector shift (VSLi or VSRi)

Proceed to appropriate short scalar shift (SLi, SLiR, SRi, or SRiR)

Interpreter Address Determination

DIRADRES — Entered from "OPJUMP" if prefix is 1, and from "DOSTORE" if a non-indexed loading operation is specified.

TS = - (E_{LOC+1}) (address also specified by FBANK contents)
If TS > 0:
  Proceed to "PUSHUP" (next word is negative, meaning operation)
ADDRWD = |TS| - 1 (error if TS = 0)
LOC = LOC +1
If ADDRWD < 55_8:
  ADDRWD = ADDRWD + FIXLOC (Cf. Section VID)
  Proceed to "INDJUMP"
If ADDRWD ≤ 3777_8:
  Proceed to "GEADDR"
TS = ADDRWD + INTBT15 (INTBT15 saved in "INTPRET")
FBANK = bits 15-11 of TS
ADDRWD = bits 10-1 of (ADDRWD - 3777_8) + 1777_8 (can't use last cell in bank)
Proceed to "INDJUMP"

INDEX — Entered from "OPJUMP" if prefix is 3, and from "DOSTORE" if an indexed loading operation is specified.

LOC = LOC +1
TS = - (E_{LOC}) (address also specified by FBANK contents)
ADDRWD = |TS| - 1
If bits 15-12 of ADDRWD ≠ 0:
  ADDRWD = ADDRWD + INTBT15
If TS < 0:
  ADDRWD = ADDRWD - X1 modulo 2^{14} (E_{LOC} was positive)
If TS > 0:
  ADDRWD = ADDRWD - X2 modulo 2^{14} (E_{LOC} was negative)
If ADDRWD ≤ 77_8:
  ADDRWD = ADDRWD + FIXLOC
  Proceed to "INDJUMP"
If ADDRWD ≤ 3777_8:
  Proceed to "GEADDR" (different step, but same function)

FBANK = bits 15-11 of ADDRWD

ADDRWD = (bits 10-1 of ADDRWD) + 2000_8

Proceed to "INDJUMP"

PUSHUP Entered from "DIRADRES"

If operation code (see "INDJUMP") $\geq 20_8$:

PUSHLOC = PUSHLOC - i (i = 2,3,6 for MODE = DP, TP, VC)

If operation code is VXSC, V/SC, MXV, or NORM:

PUSHLOC = PUSHLOC - j (j= 6,6,2 for MODE = DP, TP, VC)
(should not be used for MXV,NORM)

If operation code is TAD, TLOAD, SSP, or CCALL:

PUSHLOC = PUSHLOC - 3 (should not be used for SSP or CCALL)

If operation code is SIGN, DLOAD, PDDL, or VXM:

PUSHLOC = PUSHLOC - 2 (should not be used for VXM)

If operation code is VLOAD, CGOTO, SLOAD, or PDVL:

PUSHLOC = PUSHLOC - 6 (should not be used for CGOTO,SLOAD)

ADDRWD = PUSHLOC

Proceed to "INDJUMP"

GEADDR Entered from "DIRADRES" and "INDEX" if non-VAC E-memory ref.

EBANK = bits 11-9 of ADDRWD

ADDRWD = (bits 8-1 of ADDRWD) + 1400_8

Proceed to "INDJUMP"

15ADRERS Entered from ITA and index-register operations requiring an E-memory address.

If POLISH $< 55_8$:

POLISH = POLISH + FIXLOC

Return

EBANK = bits 11-9 of POLISH

POLISH = (bits 8-1 of POLISH) + 1400_8 (note that this not usual POLISH information)

Return

Interpreter Storage Orders

DOSTORE Entered from "NEWOPS" if operation word positive, and at end of STADR order.

ADDRWD = TS - 1

TS = (bits 14-12 of ADDRWD), shifted right 11 places

ADDRWD = bits 11-1 of ADDRWD

Proceed as indicated in table below (X is address for store order)

TS	Instruction	Performance
0	STORE X	Perform "STORE"; Proceed to "DANZIG"
1	STORE X,1	Perform "STORE,1"; Proceed to "DANZIG"
2	STORE X,2	Perform "STORE,2"; Proceed to "DANZIG"
3	STODL X	Perform "STORE"; Proceed to "DODLOAD"
4	STODL* X	Perform "STORE"; Proceed to "DODLOAD*"
5	STOVL X	Perform "STORE"; Proceed to "DOVLOAD"
6	STOVL* X	Perform "STORE"; Proceed to "DOVLOAD*"
7	STCALL X	Perform "STORE" CYR = 00015_8 (CALL order) Proceed to third line of "OPJUMP2"

STORE,1

ADDRWD = ADDRWD - X1 (modulo 2^{14})

Proceed to "STORE"

STORE,2

ADDRWD = ADDRWD - X2 (modulo 2^{14})

Proceed to "STORE"

STORE

If ADDRWD $\geq 55_8$:

EBANK = bits 11-9 of ADDRWD

ADDRWD = (bits 8-1 of ADDRWD) + 1400_8

If ADDRWD $< 55_8$:

ADDRWD = ADDRWD + FIXLOC

If MODE = DP:

$E_{ADDRWD_{dp}} = MPAC_{dp}$

Return

If MODE = TP:

$E_{ADDRWD_{tp}} = MPAC_{tp}$

Return

If MODE = VC:

$E_{ADDRWD_{vc}} = MPAC_{vc}$

Return

DODLOAD

CYR = 20006_8 (DLOAD order)

Proceed to "DIRADRES"

DODLOAD*

CYR = 60006_8 (DLOAD* order)

Proceed to "INDEX"

DOVLOAD

CYR = 20000_8 (VLOAD order)

Proceed to "DIRADRES"

DOVLOAD*

CYR = 60000_8 (VLOAD* order)

Proceed to "INDEX"

Interpreter Transfer to Operation

INDJUMP Entered from "DIRADRES", "INDEX", "GEADDR", or "PUSHUP" for Indexable Operations (those with prefix of 1 or 3).

Proceed to routine specified by bits 5-1 of CYR (i.e. the order bits alone, since prefix bits already removed):

Octal Digit	Orders
00-03	VLOAD; TAD; SIGN; VXSC
04-07	CGOTO; TLOAD; DLOAD; V/SC
10-13	SLOAD; SSP; PDDL; MXV
14-17	PDVL; CCALL; VXM; NORM
20-23	DMPR; DDV; BDDV; (SL, SLR, SR, SRR, VSL, VSR)
24-27	VAD; VSU; BVSU; DOT
30-33	VXV; VPROJ; DSU; BDSU
34-37	DAD; (spare); DMP; SETPD

MISCJUMP Entered from "OPJUMP2" for Miscellaneous Operations (those with prefix of 2), with FBANK = 1

Proceed to routine specified by bits 4-1 of CYR (i.e. the most significant 4 bits of order code, since CYR contains original instruction cycled right 3 places):

Octal Digit	Shifted Digits	Orders
00-03	00-07	AXT; AXC; LXA; LXC
04-07	10-17	SXA; XCHX; INCR; TIX
10-13	20-27	XAD; XSU; BZE/GOTO; BPL/BMN
14-17	30-37	RTB/BHIZ; CALL/ITA; (all Logical Bit Operations)/ (spare); BOVB/BOV

The shifted value is odd if index register #1 is specified or if the operation following the slash mark is selected, and otherwise is even. The shifted value is the one given in Section VIB.

UNAJUMP Entered from "OPJUMP3" if the prefix is 0 (Unary Operation) and the five-bit operation code is even, with FBANK = 0.

Proceed to routine specified by bits 4-1 of CYR (i.e. the most significant 4 bits of order code, since CYR contains original instruction cycled right 3 places):

Octal Digit	Shifted Digits	Orders
00	00	EXIT (detected in "OPJUMP")
01-03	02-06	SQRT; SIN; COS
04-07	10-16	ASIN; ACOS; DSQ; ROUND
10-13	20-26	(DCOMP, VCOMP); VDEF; UNIT; (ABS, ABVAL)
14-17	30-36	VSQ; STADR; RVQ; PUSH

The shifted value is the one given in the order code description, and is equal to twice the octal digit (i.e. an even number). Odd values are detected in "OPJUMP3" and used for short shifts.

USPRCADR Entered from machine-language coding to access interpretive language computations (such as divide or trig functions) that must be accessed by changing FBANK. Calling address +1 contains FCADR of desired coding. Incoming values of (A, L) retained.

EDOP = 200_8 shifted right 7 places (leaves 00001_8, the EXIT code, becoming 0 in "DANZIG")

BANKSET = BBANK

FBANK = (bits 15-11 of E_Q)

TS = (bits 10-1 of E_Q) + 2000_8

Proceed to address specified by TS (and FBANK)

VID Relative Addresses, Push-down List, and VAC Areas

In addition to the instructions for performing multiple-precision and vector computations, the interpreter software can also make special use of a set of 43 erasable memory cells which can be assigned to a job when it is established (see Section VIIB). For historical reasons (in the Block 1 software design, the vector accumulator was located there), this set of cells is conventionally called a "VAC area": there are five such areas allocated in the erasable memory, each with a control cell (VAC1USE, VAC2USE, ...VAC5USE) preceding the first cell of the corresponding area. When the associated VAC area is assigned, the VACiUSE cell is set 0 (assignment is when the job is established, in general); otherwise, the control cell contains its own address (hence VAC3USE +1 provides the starting address of the third VAC area, for example). As shown in Section VIIB, when a VAC area is assigned to a job the starting address of the VAC area is placed in bits 9-1 of the job's PRIORITY cell; when a job is first established, the job's PUSHLOC cell is also set to this same VAC area starting address. As part of the standard set of initial conditions (e.g. at computer turn-on), each VACiUSE cell is set to its own address.

When a job is active, the starting address of the selected VAC area is contained in FIXLOC. If the convention were adopted (as it is) that all addresses below some value in the interpretive language programs were to be with respect to the address contained in FIXLOC, then a relative addressing capability would be achieved (i.e. addresses would not indicate absolute memory cells, but instead a cell which bears a certain relationship to the address in FIXLOC, which for five VAC areas would mean one of five absolute erasable memory cells). As discussed in Section IID, addresses of 60_8 and below in the memory are

assigned either to hardware registers or special-purpose applications, and therefore by convention can be excluded from direct reference in the interpretive language (instead, a return to machine language, perhaps via an RTB order, must be made). Since each VAC area is only 43 cells in capacity, only relative addresses in the range 0 - 52_8 (42D) inclusive are meaningful, however. For programming convenience, the absolute address below which FIXLOC is added is variously considered to be 55_8 and 100_8: the additional excluded cells are assigned functions of such a nature that interpretive language reference to them would not be appropriate.

By convention, certain of the cells in the upper portion of each VAC area are assigned to particular quantities. It is customary in the listing to see references to these cells either by a special mnemonic or by the number of decimal locations they are located above the address in FIXLOC (i.e. the relative decimal address):

Relative Address	True Address	Mnemonic	Function
34D-35D	FIXLOC +34D-35D	LVSQUARE	Square of length of vector after ABVAL and UNIT orders, scale factor twice that of vector.
36D-37D	FIXLOC +36D-37D	LV	Length of vector after UNIT, same scale factor as that of (original) vector.
38D	FIXLOC +38D	X1	Index register #1 (Section VIA).
39D	FIXLOC +39D	X2	Index register #2 (Section VIA).
40D	FIXLOC +40D	S1	Step register #1 (see TIX order).
41D	FIXLOC +41D	S2	Step register #2 (see TIX order).
42D	FIXLOC +42D	QPRET	Return address information after CALL, CCALL, and STCALL orders; sensed by RVQ and ITA orders. A NOVAC job sets program cells such that QPRET storage location is in MPAC+6 (see Section VIIB).

If these functions are not required, of course, their cells may be used for utility storage purposes, as can any of the other cells in the VAC area. As mentioned before, the cells can be used either by direct specification of their relative address or by assignment of a tag to have the appropriate octal equivalence.

In addition to the explicit specification of relative addresses in the VAC area, a provision is also included in the interpreter software to allow addresses in the VAC area to be "implied" by the coding, thus eliminating the need for address-word parameters for these quantities and reducing the amount of fixed memory which is required. The major control parameter for these implied-address uses of the VAC areas is the PUSHLOC cell (part of the Job Register Set, see Section VIIB): this cell is initialized to the same value as FIXLOC when the job is established, and it may be set to some other value (which should be less than FIXLOC +43D, of course) by using the SETPD order. PUSHLOC is also incremented positively or negatively as determined by the nature of the computation.

Three orders are used to store information in the VAC area in cells determined by PUSHLOC: PDDL (store in push-down list and perform a double precision load), PDVL (store in push-down list and perform a vector load), and PUSH (store in push-down list). Each of these operations stores the appropriate number of 15-bit words determined by the current computational value of MODE. Each time a quantity is stored into the VAC area by one of these operations, the contents of PUSHLOC are incremented by the appropriate amount (2, 3, or 6 for double precision, triple precision, and vector storages respectively), so that PUSHLOC contains the address of the cell which should be loaded next.

Withdrawal of information from the VAC area cells determined by PUSHLOC takes place properly only for certain Indexable Operations (see Section VIA), as controlled by the "PUSHUP" routine described on page VIC-5. These withdrawals of information are accomplished by decrementing PUSHLOC by the appropriate amount (2, 3, or 6 for double precision, triple precision, and vector removals respectively) and storing the resulting value of PUSHLOC in ADDRWD for use as the operand address. If desired, the contents of the MPAC set and the push-down list can be exchanged (by the PDDL or PDVL orders) merely by omitting the address following the orders. The STADR instruction is used if it is desired to obtain operands from the push-down list before a store order: the STADR instruction causes the assembler to place the store order in the memory as a negative quantity, thus causing it to appear to the program logic as a normal operation (rather than as an address), so as to cause operands to be taken from the VAC area.

If desired, it may be useful to think of PUSHLOC as a "pointer" which specifies the cell in the VAC area that is to be loaded next by an operation that inserts information into the push-down list. Storage of information by such an operation causes the pointer to be advanced so as to reflect the address of a cell further down within the VAC area (a cell with a higher address can be regarded as being "lower" for a ranking of cells in ascending sequence), and hence "pushes down" the pointer. Conversely, when information is withdrawn, the pointer moves to a cell with a lower address (or one "higher" on the ranking), and hence "pushes up" the pointer. It should be evident that there is no physical movement of cell contents "up" and "down", but merely a change in the contents of PUSHLOC. This process gives the push-down list a "last in, first out" characteristic.

The VAC area and the "push-down list", of course, are the same 43-cell erasable memory segment, and references to individual cells within this segment can be either by the various operations making use of the push-down list properties or by explicit use of relative addresses (values in range 0 - 42D). There is no magnitude check made in the interpreter software before changes to PUSHLOC are caused, so the program must be written to keep PUSHLOC in the proper 0-42D range of relative addresses, remembering that cells 34D-42D are also assigned to particular quantities. Movement of PUSHLOC does not cause the contents of VAC area cells to be destroyed, so these cells could still be sensed by using their relative addresses after PUSHLOC has caused them to be read out. The conventional programming practice, however, is to use the relative address for all except the final reference to the quantity (and use the push-down list capability for this final reference).

VIE Interpretive Language Examples

The following examples of interpretive language coding have been fabricated to illustrate some of the principles of the different orders available with the interpretive language, rather than necessarily to illustrate efficient coding techniques or perform meaningful calculations. Except where noted, scaling considerations have been ignored.

1. $VAR = D / (HB + \frac{1}{2}\sqrt{VAR})$ H's address modified by index register #2

```
DLOAD*   DMPR
         H,2
         B
PDDL     SQRT        Store first term in push-down list
         VAR
SR1      DAD         DAD takes operand from push-down list
BDDV
         D
STORE    VAR
```

2. $VAR = D + FC - H^2 B$; $VAR1 = \cos^{-1}\left(|VAR| / (2^5 \sin VAR)\right)$ Magnitude of arc cosine argument limited to 1; leave VAR1 in push-down list.

```
DLOAD    DSQ
         H
DMP      PDDL
         B
         C
DMP      DAD
         F
         D
DSU      STADR       STADR forces DSU to get operand from push-
                     down list
STORE    VAR
ABS      SRR
         5           Divide numerator by 2^5
```

```
PDDL      SIN
          VAR
BDDV      SR1       Right shift 1 to scale cos^-1 argument B1;
                    BDDV operation incorporates limiting as
                    part of algorithm (see Section VIB)
ACOS      PUSH      Leave VAR1 in push-down list
```

3. $VAR = 2^{10} (H - B + C) / \left(\cos (D - \sin^{-1} E)\right)$ If overflow, do not load VAR, and transfer to STEP (machine language coding). H, B, C triple precision, all positive.

```
TLOAD     DCOMP     DCOMP works for triple precision too
          B
TAD       BOV
          H
          +1        Just resets OVFIND
TAD       SLR
          C
          10D       Without the "D", would be 10_8 (decimal 8)
PDDL      ASIN
          E
BDSU      COS
          D
BDDV
BOVB                Cannot be on same line as BDDV (push-down list)
          STEP
STORE     VAR
```

4. VAR = Z component of: D unit (HB, 0, - C) sgn E

```
DLOAD     DCOMP
          C
PDDL      PDDL      Z component into push-down list first, then Y
          ZERODP    Constant, double precision 0
          H
DMP       VDEF      X component in MPAC_dp for VDEF operation
          B
```

```
        UNIT    VXSC        MPAC now a vector, hence operand scalar
                D
        SIGN    DLOAD
                E
                MPAC +5     Z component
        STORE   VAR
```

5. Transfer double precision quantities at X, Y, Z (6 consecutive cells) to PD+18, PD+20, PD+22 (push-down cells).

```
        SETPD   VLOAD
                18D
                X
        PUSH
            or
        SSP     AXT,1
                S1          Location of store address before what is put
                2           into it
                6
STEP    DLOAD*
                X +6, 1     Index registers use subtractive algorithm
        STORE   18D +6,1
        TIX,1
                STEP
```

6. Compute single precision the value of H = 17 + B - C - 5 - H without permanently changing any erasable memory cell (except H and interpreter ones) or using MPAC.

```
        XCHX,2  LXC,2       Save previous X2
                H
                X2          Complement contents
        INCR,2  INCR,2
                17D
        DEC     - 5         Without DEC would be address of first line
        XAD,2   XSU,2
```

```
        B
        C
XCHX,2
        H
```

7. H = 2 B/C. If H $\geq$ 0, set bit 23 (of flagword assignments) to 1 and transfer to STEP1; otherwise, set bit to 0 and transfer to STEP2.

```
DLOAD   DDV
        B
        C
SL1R                    The round is not functional since DDV leaves
                        MPAC+2 = 0
STORE   H
SET     BPL
        23D
        STEP1
CLEAR   GOTO
        23D
        STEP2
```

or

```
DLOAD   SR1R
        C
BDDV
        B
STORE   H
BMN     SETGO
        +3
        23D
        STEP1
CLRGO
        23D
        STEP2
```

or

```
        DLOAD   DDV
                B
                C
        SL1
        STORE   H
        BMN     BON
                STEP'
                23D
                STEP1       Bit already set to 1
        BOFINV
                23D
                STEP1       Will transfer since know bit was 0
STEP'   BOFF    BONINV
                23D
                STEP2
                23D
                STEP2       Will transfer since know bit was 1
```

or

```
        DLOAD   DDV
                B
                C
        SL1
        STORE   H
        BMN     BOFSET
                STEP'
                23D
                STEP1
        BONSET
                23D
                STEP1       Will transfer since bit was 1
STEP'   BONCLR  BOFCLR
                23D
                STEP2
                23D
                STEP2
```

8. $\underline{VECT} = 2^{-10}\left(\left[REFSMMAT\right]\ \underline{VECT} + \underline{VECT1} - \frac{1}{2}\ \underline{VECT2} - 2^{11}\ \underline{VECT3}\ \left[MTBY\right]\right)$

```
      VLOAD    MXV
               VECT
               REFSMMAT
      VAD      VSL1        The ½ factor handled by shifts
               VECT1
      VSU      VSR1
               VECT2
      PDVL     VXM
               VECT3
               MTBY
      VSL      BVSU        BVSU operand from push-down list
               11D
      VSR
               10D
      STORE    VECT
```

9. If $|(\underline{H} \cdot \underline{B})\ \underline{B}| - (\underline{C})^2 > 0$, go to STEP1 (in basic); if zero, go to STEP2; otherwise, set $\underline{VECT} = \text{unit}(\underline{H} * \underline{B}) \times (\underline{C} \cdot - \underline{B})/|\underline{H} * \underline{B}|$ and leave interpreter.

```
      VLOAD    VPROJ
               H
               B.
      ABVAL    PDVL
               C
      VSQ      BDSU
      BZE      BMN
               STEP2
               STEP'
      RTB                  Can be used to return to basic (machine
                           language) at an address without return to
                           interpreter.
               STEP1
STEP' VLOAD    VXV
               H
               B
```

```
UNIT    PDVL
        B
VCOMP   DOT
        C
VXSC                MPAC has scalar, hence operand from push-down
                    list is vector
V/SC
        36D         Left with magnitude by UNIT operation
STORE   VECT
EXIT
```

10. Divide H^4 by B^4, scaling each to minimize errors.

```
DLOAD   NORM
        H
        X1
SR1R    INCR,1
        1
PDDL    NORM
        B
        X2
BDDV    DSQ
ROUND   DSQ
TLOAD               Sets MODE to TP, for the store
        MPAC
SETPD   PUSH
        0
SXA,2   SSP
        3           Addresses in VAC area
        4           Set to 0
        0
SLOAD   DSU
        X1
        3
SL2     LXA,1
        MPAC
```

```
        TLOAD   SL*
                0
                4,1             Shift either right or left to give (assumed
                                desired) new scaling
        STORE   ANSWR
```

11. $X2 = 10000_8 - X1$. If X2 = 0, proceed to STEP; otherwise, set bits 7-1 of FLAGWRD1 = 1 and set X2 = -4970.

```
        AXT,2   XSU,2
                20000           Could also have used OCT  10000 (see Section VB)
                X1
        DLOAD   BHIZ
                X2
                STEP
        RTB     AXC,2
                SETFLG
        DEC     4970
        .       .
        .       .
        .       .

SETFLG  INHINT
        CS      FLAGWRD1
        MASK    LOW7            (constant, octal 00177)
        ADS     FLAGWRD1        (bits that 0 in FLAGWRD1 now set 1; bits
                                 that were 1 are left alone)
        RELINT
        TC      Q               RTB leaves Q set so that return to "DANZIG"
```

12. $H = B/C$; $J = BC$; $K = 2^{-12}$ VECT1 · VECT2; transfer to STEP0, STEP1, STEP2, or STEP3 depending on whether $|K| \times 2^{14}$ (K single precision) is 0,1,2, or 3.

```
        DLOAD   DDV
                B
                C
        STODL   H
                B
        DMP
                C
```

```
        STOVL   J
                VECT1
        DOT     SR          K considered only a control quantity, hence
                VECT2       not stored
                12D
        ABS     CGOTO
                MPAC
                CONST
CONST   CADR    STEP0       Computations at these steps not shown
        CADR    STEP1
        CADR    STEP2
        CADR    STEP3
```

13. Within a subroutine (entrance to which is not shown), set H = B and perform subroutine starting at STEP. This subroutine computes VAR = H + f (B^2), where f (B^2) is computed by subroutines starting at STEP1, STEP2, or STEP3 depending on values of cell CONTL (possible values 1, 2, and 3 respectively). If CONTL odd, complement bit 17 of the flagword assignments.

```
        STQ     DLOAD
                S1          Used for temporary storage
                B
        STCALL  H
                STEP
        GOTO                Return to original calling routine
                S1
STEP    ITA     DLOAD       Loading repeated since MPAC might not be
                            proper for other users of subroutine
                S2
                B
        DSQ     CCALL
                CONTL
                CONST - 1
        DAD     CALL
                H
                S2          Return to calling routine
```

```
CONST CADR    STEP1
      CADR    STEP2
      CADR    STEP3

STEP1 DSU     INVERT
              R
              17D
      RVQ

STEP2 DSU     RVQ
              S

STEP3 DSU     INVGO
              T
              17D
              QPRET     Since QPRET is in erasable, INVGO logic will
                        cause transfer to address in QPRET
```

VII PROGRAM PERFORMANCE CONTROL

The guidance computer is basically capable of performing only one computation at a time: the proper performance of the guidance system in a flight environment, therefore, requires a carefully integrated combination of hardware and software design so that the computations may be sequenced to meet the demands of the flight. This means that the computer must be responsive to external stimuli (originated either by the crew or by the spacecraft hardware), and must also provide output signals of the proper characteristics for use by other spacecraft systems.

Once the computer arithmetic unit starts executing a machine language instruction, it will continue to execute that instruction until it is completed. The next function performed by the arithmetic unit depends on the character of the computations being performed, as well as on the nature of the signals which may have occurred while the instruction was being executed. These functions, in order of decreasing priority, are:

1. Process counter interrupts (Section IIH). These affect cells 0024_8 - 0060_8, with the lower cell addresses being processed before the higher ones. All counter interrupts, however, are processed before proceeding to the next function.

2. Initiate computations associated with a restart (program interrupt #11, Section IIH) if such computations are required. The software must periodically update the "phase information" covered in Section VIIC so the restart will be successful.

3. If the previous machine language instruction was part of a task (program interrupts #1 - #10, Section IIH), continue with the next program step in sequence. This is done even if a program interrupt of higher priority has been generated, since the program interrupt priority ranking is for the purposes of starting the appropriate processing only. The fact that another program interrupt has been generated, of course, is retained by the hardware and will cause appropriate program action at a later time.

4. If action on program interrupts is blocked (by setting of the interrupt inhibit flip-flop or other means, see Section IIH), continue with the next program step in sequence.

5. If a program interrupt has been received, suspend the present computations and start those pertinent to the task associated with the program interrupt. If several program interrupts are waiting to be processed, perform the one with the highest priority (Section IIH: priority is order of numbering of the interrupt for program interrupts #1 - #10). Once the processing is started, it is continued to completion, even though a higher priority interrupt may have subsequently been generated. The present value of the program counter is saved by hardware means, while other addressable registers must be saved by program means. The interrupts which are produced include those whose timing is specified by the software in order to perform time-dependent computations via the waitlist system (see Section VIIA).

6. By software means, a task can cause the performance of a job to be suspended for a longer interval than that required to do the task itself: this is distinguished from the function of #8 below in that the "check for higher priority" is (in concept) done as part of the task performance, with the foregone conclusion that the new computation will be initiated. One term for this function is "jask", since to other jobs it has the over-riding "priority" of a task; to other tasks, however, it is a "job",

since it can be suspended in favor of "normal" tasks (such as telemetry interrupts). The software must preserve and restore properly the necessary special-purpose erasable memory cells, and the implementation involves use of the EDRUPT order described in Section IIH. The jask does not use the executive system.

7. Continue with the next program step in sequence.

8. Periodically, the program while executing any job must check for a computation of higher job priority (cf. #6 above). If one is waiting, the computations of the present job are suspended and the one of higher priority started. Certain erasable memory cells (the Job Register Set and, if assigned, VAC area, see Section VIIB and Section VID) from the suspended job are retained. The program must be written so as to avoid undesirable interactions between different jobs due either to sequences of computation or time-sharing of erasable memory cells. There is no limit on the length of time or amount of computation required by a particular job, contrary to the case with tasks.

9. If no jobs or tasks are to be performed, the program continues in an idling loop ("dummy job", Section VIIB) while waiting for program interrupts to require tasks to be performed. The tasks, in turn, could establish functional jobs. There is no "stop" command in the computer order code (a one-step loop is a fault condition).

VIIA Waitlist System for Tasks

Time-dependent tasks in the guidance program can be implemented by means of a "waitlist" system, for which the program specifies the starting address of the task computations and the time delay (from the present time) that should elapse before these computations are entered. The time delay is specified in units of centi-seconds (0.01 seconds, the least increment of cell 0026_8, TIME3 in Section IID). The general calling sequence for the waitlist entries is:

```
CA      yy      (yy is time delay in centi-seconds)
TC      WAITLIST    (in fixed-fixed memory)
2CADR   XXXX    (XXXX is starting address)
```

If the waitlist task to be called is in the same fixed memory bank, the BBCON part of the 2CADR can be omitted, thus saving a fixed memory cell, by using the following calling sequence:

```
CA      yy
TC      TWIDDLE     (in fixed-fixed memory)
ADRES   XXXX
```

In order to delay the performance of a waitlist-initiated task for an additional period of time, transfer can be made to "VARDELAY" (required delay in accumulator) or "FIXDELAY" (required delay in calling address +1, returns to calling address +2 after expiration of the delay). Finally, for one long time interval, entrance to "LONGCALL" can be made with (A, L) containing the double precision time delay required before the task is started, again in units of centi-seconds. Waitlist-initiated tasks are terminated by transfer to "TASKOVER", and can be removed from the waitlist by entrance to "KILLTASK" (LM only).

The following constraints and limitations on the use of the waitlist system exist:

1. The value of the specified time delay (yy above) must be positive non-zero. Negative or zero values generally cause a software restart to occur ("POODOO" or "POODOO1"). The lower limit on the time delay, therefore, is 1 centi-second.

2. By convention, the value of yy is restricted to a maximum of 162.5 seconds, although slightly larger values could be accommodated, for direct entrances to "WAITLIST". Larger values of yy are handled (one task at a time) by the "LONGCALL" routine, which uses the waitlist system logic to count down the specified time in increments of 81.92 seconds. The fifteen-bit wordlength of the computer gives a "hard" limit for "WAITLIST" entrances of 163.83 seconds (TIME3 would not be set to intervals bigger than 81.93 seconds due to the "dummy task" discussed later).

3. With no timing constraints, up to seven tasks can be handled in the waitlist system simultaneously ("handled" means "queued": once an interrupt for a task has been acted upon, its place in the queue is vacated). As can be seen from the equations on the following pages, if a new task cannot be accommodated in the queue a software restart ("BAILOUT" or "BAILOUT1") is caused. Under certain conditions, an eighth (and even a ninth) task can be accommodated in the queue.

4. More than one task can be scheduled for initiation at the same TIME3 overflow point, in which case the tasks would be performed in the order in which they were originally scheduled (i.e. the first task scheduled for the time would be the first one performed, etc.). If too much time is consumed in the conduct of tasks, however, a hardware restart (program interrupt #11, Section IIH) would be caused due to the hardware "RUPT lock" check.

5. Tasks scheduled by a job (such as the countdown to send an engine-off command based on a guidance time-to-go) measure the time delay (yy) from the most recent occurrence of the counter interrupt that triggers the incrementing of TIME3. Consequently, the task would be started roughly 0-10 milliseconds before the value indicated by yy. Tasks scheduled by waitlist tasks, of course, such as the periodic accelerometer sampling, measure the time delay from the TIME3 increment that last took place, and hence such self-perpetuating tasks can be recycled indefinitely.

The equations given below describe the computations performed by the various program routines associated with the waitlist system.

WAITLIST — Entered with desired time delay before start of task in A (accumulator), scale factor B14 in units of centi-seconds, and with the 2CADR (see Section VC) of the task in the two cells following the TC "WAITLIST". Return is to the cell following the 2CADR.

Inhibit interrupts

If $A \leq 0$: CM only

Proceed to "POODOO" (pattern 21204_8)

WAITEXIT = Q

DELT = A (Value of input time, B14 centi-seconds)

$WAITADR_{dp} = E_{WAITEXIT_{dp}}$ (The 2CADR information of task starting address)

Proceed to "WAIT2"

TWIDDLE — Entered with desired time delay before start of task in A, and with the ADRES (see Section VC) of the task in the cell following the TC "TWIDDLE". Return is to the cell following the ADRES.

Inhibit interrupts

Q = Q - 1

TS_2 = BBANK + SUPERBNK (or'ed into bits 7-5)

Inhibit interrupts (redundant, to use common coding)

If $A \leq 0$: CM only

Proceed to "POODOO" (pattern 21204_8)

WAITEXIT = Q

DELT = A

$WAITADR_{dp} = (E_{WAITEXIT+1}, TS_2)$ (the +1 compensates for the Q -1)

Proceed to "WAIT2"

FIXDELAY — Entered with required amount of delay (B14 centiseconds) in calling address +1; returns to calling address +2 after completion of the delay.

$A = E_Q$ (Required amount of delay)

$Q = Q + 1$

Proceed to "VARDELAY"

VARDELAY — Entered with required amount of delay (B14 centiseconds) in A; returns to calling address +1 after completion of the delay.

$WAITADR_{dp}$ = (Q, BBANK + SUPERBNK (or'ed into bits 7-5))

WAITEXIT = "TASKOVER" - 2

DELT = A (Required amount of delay)

Proceed to "WAIT2"

WAIT2

If DELT ≤ 0: LM only

Proceed to "POODOO1" (pattern 21204_8, TS = (WAITEXIT, caller's BBANK))

If TIME3 > 128: (interrupt not yet taken place for time presently being delayed)

T1 = 16384 - TIME3

If TIME3 = 128:

T1 = 1 (improper performance)

If TIME3 < 128:

T1 = - TIME3

TS = DELT - T1 + 1 (T1 is time interval between "now" and when present task #1 was supposed to be performed)

If TS > 0:

TS = TS - 1 + LST1 (Sets to DELT - T2 +1)

Proceed to "WTLST5" (DELT ≥ T1)

Set TS = TIME3 and TIME3 = (8192 - DELT) + 8192 modulo 2^{14}

Move all LST1 entries down one place (LST1+i = LST1+i-1, for i from 7 down to 1)

LST1+0 = DELT - (16384 - TS) + 1

Move all $LST2_{dp}$ entries down one place ($LST2{+}i_{dp} = LST2{+}i{-}2_{dp}$, for i from 16 down to 2 in decrements of 2)

$LST2{+}0_{dp} = WAITADR_{dp}$

If previous LST2+16 (before movement) $\neq$ "SVCT3": (Displaced task not dummy task)

Proceed to "BAILOUT" (pattern 31203_8) CM only

Proceed to "BAILOUT1" (pattern 31203_8, TS = (WAITEXIT, BBANK)) caller's LM only

Proceed to address specified by WAITEXIT +2 (and BBANK as used in LM-only alarms)

WTLST5

I = 0 (Index to count cycles and select cells)

I = I + 1

If I = 8:

If TS > 0: (time too big)

Proceed to "BAILOUT" (pattern 31203_8) CM only

Proceed to "BAILOUT1" (pattern 31203_8, TS = (WAITEXIT, BBANK)) caller's LM only

If TS > 0:

TS = TS + LST1+I - 1

Proceed to second line of "WTLST5"

LST1+I-1 = LST1+I-1 - TS + 1 (Notation means e.g. LST1+3 if I = 4)

If I $\leq$ 6:

Move LST1 entries from LST1+I to LST1+7 down by one place

If I $\leq$ 7:

LST1+I = TS

Move $LST2_{dp}$ entries from $LST2+2I_{dp}$ to $LST2+16_{dp}$ down one place

$LST2+2I_{dp}$ = $WAITADR_{dp}$

Proceed to top line of this page

LONGCALL Entered with desired time delay before start of task as a double precision number in (A, L), scale factor B28 in units of centi-seconds, and with the 2CADR of the task in the two cells following the TC "LONGCALL". Return is to the cell following the 2CADR. Only one task at a time can use "LONGCALL".

$LONGTIME_{dp}$ = (A, L)

$LONGCADR_{dp}$ = $E_{Q_{dp}}$ (2CADR of task to be called)

$LONGEXIT_{dp}$ = (Q +2, BBANK) (i.e. return address)

If $LONGTIME_{dp} \leq 0$: LM only

Proceed to "POODOO1" (pattern 21204_8, TS = $LONGEXIT_{dp}$)

Proceed to "LONGCYCL"

LONGCYCL

$LONGTIME_{dp} = LONGTIME_{dp} - 8192$

If $LONGTIME_{dp} > 0$:

Call "LONGCYCL" in 81.92 seconds

If $LONGTIME_{dp} \leq 0$:

$LONGTIME_{dp} = LONGTIME_{dp} + 8192$

Call "GETCADR" in LONGTIME centi-seconds

Set TS = $LONGEXIT_{dp}$ and LONGEXIT = "TASKOVER"

Proceed to address specified by TS

GETCADR

Proceed to address specified by $LONGCADR_{dp}$

SVCT3 — Entered nominally each 81.93 seconds (to "service TIME3", since the hardware incrementing of the clock cannot be terminated by software means), and known as the "dummy task".

Check for flight-program requirements for functions performed with the 81.93-second period (conventionally used for free-flight gyro bias computation). See mission documentation.

Proceed to "TASKOVER"

T3RUPT — Entered when program interrupt #3 (see Section IIH) is acted upon. (A, L) already loaded in (ARUPT, LRUPT) before "T3RUPT" is entered.

BANKRUPT = BBANK + SUPERBNK (or'ed into bits 7-5)

QRUPT = Q

Move LST1 information up by one place, putting -8192 into LST1+7 and setting TS = LST1+0

TS = TIME3 + TS + 16383

RUPTAGN = -0

If TS $\geq$ 16384:

TS = TS - 16384

RUPTAGN = 1

TIME3 = TS (coding done so that no TIME3 increments lost)

Move $LST2_{dp}$ information up by one place, putting "SVCT3" in $LST2+16_{dp}$ and setting $TS_{dp} = LST2+0_{dp}$

SUPERBNK = bits 7-5 of TS+1

Proceed to address specified by TS_{dp} (most significant half is S-register data, least significant half is BBANK information; SUPERBNK already loaded)

TASKOVER — Entered to terminate the performance of a task initiated under waitlist control (the End of task function).

If RUPTAGN = 1:

Proceed to third line of "T3RUPT" (movement of LST1)

SUPERBNK = bits 7-5 of BANKRUPT

Proceed to "RESUME"

RESUME — Entered to terminate the performance of a non-waitlist task that has saved Q in QRUPT, etc. (as well as from "TASKOVER").

Q = QRUPT

Proceed to "NOQRSM"

NOQRSM

BBANK = BANKRUPT (note that SUPERBNK restored only by "TASKOVER")

Proceed to "NOQBRSM"

NOQBRSM

(A, L) = (ARUPT, LRUPT)

Release interrupts

Execute RESUME pseudo-operation (see Section VA)

KILLTASK — Entered with CADR of task to be removed from waitlist in calling address +1; returns to calling address +2. Routine is LM only.

Inhibit interrupts

TS_1 = (bits 10-1 of E_Q) + 2000_8 (i.e. S-register format)

TS_2 = (bits 15-11 of E_Q) (i.e. FBANK information)

I = 0

If $LST2_I$ = TS_1:

If (bits 15-11 of $LST2_{I+1}$) = TS_2:

$LST2_I$ = "TASKOVER" (in fixed-fixed, hence sufficient)

Return

If I = 16:

Return

I = I + 2

Proceed to fifth line of "KILLTASK"

Waitlist System Tables

Most of the notation employed for the waitlist system computations on the previous pages is either that of hardware registers (defined in Section II) or is defined by the equations in which the quantities are employed. The two tables defined by LST1+0 through LST1+7 and $LST2+0_{dp}$ through $LST2+16_{dp}$, however, merit further explanation.

As discussed in Section IID, cell 0026_8 (TIME3) is incremented by +1 each 0.01 second (each centi-second or 10 milliseconds) in response to a signal from the computer hardware oscillator countdown chain. When TIME3 reaches a value of 37777_8 (or 16383 decimal), the next pulse will cause the clock to overflow, leaving a value of 00000_8 in the cell and generating program interrupt #3 (see Section IIH). When this interrupt is acted upon, the software causes "T3RUPT" to be entered. Consequently, a setting of TIME3 to:

16384 - (time interval in units of centi-seconds)

will cause program interrupt #3 to occur in (time interval) centi-seconds. A setting of TIME3 to 37775_8, or 16381 decimal, for example, would cause overflow of TIME3 when the third increment pulse is received, or in three centi-seconds.

In order to achieve conveniently the required performance of the waitlist system (which, except for the TIME3 clock, is completely a software package), the tasks to be performed are stored sequentially in the order in which they must be performed (in the LST2 table in the form of 2CADR addresses of the task starting points). The required time information associated with these tasks is stored in the form of time intervals between the successive tasks (rather than as the absolute

times when these tasks should be initiated). Consequently, when task #1 is initiated, the time before the next task (next required generation of program interrupt #3) is used to set TIME3, and the lists of times (LST1) and task addresses (LST2) are moved to different erasable memory cells (literally "pushed up"), so that the previous "task #2" becomes the new "task #1", etc.

Using T1 to indicate the required time of performance of task #1 (in the same units as T_{now}, the present time, and with the same origin), the values of the various cells associated with the waitlist system are:

TIME3 = 16384 - (T1 - T_{now}) if program interrupt for task #1 not yet generated

= - (T1 - T_{now}) if program interrupt for task #1 already generated.

A breakpoint of 128 is used to decide which value is stored in TIME3 (if TIME3 exceeds 128, it is the first; if less than 128, the second). Since the "SVCT3" dummy task is entered every 81.93 seconds, the value of T1 - T_{now} would not be expected (in general) to exceed this figure.

LST1+0 = - (T2 - T1) +1

LST1+1 = - (T3 - T2) +1

LST1+2 = - (T4 - T3) +1

LST1+3 = - (T5 - T4) +1

LST1+4 = - (T6 - T5) +1

LST1+5 = - (T7 - T6) +1

LST1+6 = - (T8 - T7) +1

LST1+7 = - (T9 - T8) +1

The +1 is for software convenience, and helps in two ways:

a) Since 16383 is the largest number read from memory into the accumulator, it permits TIME3 to be set to (16383 + LST1+0) to get overflow in (T2 - T1) centi-seconds.

b) New tasks are inserted using a CCS order, whose decrement is cancelled by the +1.

Initial conditions for all entries in LST1 (set e.g. by a "fresh start" performed under operator control at computer turn-on) are a setting of -8192, corresponding to an interval of 81.93 seconds.

$LST2+0_{dp}$ = 2CADR Task #1 in LST2+0 and LST2+1

$LST2+2_{dp}$ = 2CADR Task #2 in LST2+2 and LST2+3

$LST2+4_{dp}$ = 2CADR Task #3 in LST2+4 and LST2+5

$LST2+6_{dp}$ = 2CADR Task #4 in LST2+6 and LST2+7

$LST2+8_{dp}$ = 2CADR Task #5 in LST2+8 and LST2+9

$LST2+10_{dp}$ = 2CADR Task #6 in LST2+10 and LST2+11

$LST2+12_{dp}$ = 2CADR Task #7 in LST2+12 and LST2+13

$LST2+14_{dp}$ = 2CADR Task #8 in LST2+14 and LST2+15

$LST2+16_{dp}$ = 2CADR Task #9 in LST2+16 and LST2+17

Initial conditions for all entries in LST2 (set e.g. by a "fresh start" performed under operator control at computer turn-on) are a setting of 2CADR for "SVCT3", the dummy task.

When a new task is inserted into the waitlist system, its value of DELT is compared against $(T_I - T_{now})$, with I from 1 to 9 (sequentially). If no task presently in the waitlist system has a longer wait (i.e. will be performed later) than the new task, a software restart is caused. If there are six "active" tasks in the waitlist system, there will be three entries of the "dummy task" (since at all times there must be an entry into all items of both lists). Since the dummy task entries must be separated by 81.93 seconds (because the dummy task is only inserted in this fashion), the last dummy task must be waiting at least 163.86 seconds, which of course is larger than the maximum possible single precision DELT of 163.83 seconds ($2^{14} - 1$ centi-seconds). Hence a seventh active task can always be accommodated in the waitlist system, and for suitably small time intervals between tasks an eighth and even a ninth task will be accepted.

When the waitlist system software has determined the proper place in the table of LST1 times for the new task, it is inserted there (before the first task that must wait longer than the new task), and the appropriate adjustments made to the two time differences affected by the new task. Later entries in the LST1 table are "pushed down", and a corresponding insertion and push down is made in the LST2 table. If the task that is "bumped" from the bottom of the LST2 table is <u>not</u>

the dummy task, meaning that a functionally required computation is being discarded, then the software restart is caused.

When a task is performed under control of the waitlist system, the entries in both the LST1 and LST2 tables are pushed up to replace the vacancy, and the dummy task (with an associated time delay of 81.93 seconds entered into LST1) is entered at the bottom of both lists. The push up and push down for the LST1 and LST2 tables of the waitlist system is an actual movement of cell contents, as contrasted with merely a change in an address indicator (cf. Section VID).

VIIB Executive System for Jobs

Jobs are performed under the overall jurisdiction of what is called an "executive" system. Jobs are interrupted to perform tasks when a program interrupt is received (see Section IIH), and the job's performance is generally resumed upon completion of the effort required to satisfy the program interrupt. Consequently, if the computer is not in the "interrupt mode" (performing a task) or doing a jask, it can by definition be considered doing a job (or dummy job). Uses of the interpretive language (Section VI) must be by jobs, since the necessary registers for the interpreter are not saved in special buffer cells when a task is started; aside from this restriction, however, it is usually impossible to determine from isolated coding whether the coding is associated with a job or a task, and in fact some subroutines in the program are used with both.

The distinction between a job and a task as far as suspension of a job's performance is concerned should be well understood:

> A job can be interrupted for a <u>task</u> by hardware means (unless such interrupts are explicitly inhibited) after any machine-language program step.
>
> A job can be interrupted for another <u>job</u> only by a specific check made in the program (of cell 0067_8, NEWJOB). If this cell is not checked sufficiently often, program interrupt #11 is generated (see Section IIH).

Because of this distinction, tasks must save all hardware registers (A, L, Q, etc.) which they may affect and restore them before resuming the interrupted computations; jobs do not have this requirement. The "jask" mentioned in item #6 of Section VII must also observe this

restriction, since it is "a task to other jobs; a job to other tasks".

The relative significance (in terms of how soon the computation should be completed) of a job is specified by its "priority", a five-bit quantity ranging between 01_8 (minimum) and 37_8 (maximum) that is specified at the time the job is "established" (entered into the executive system for performance when its priority is sufficient). A job can also cause its own priority to be changed. The "priority" assigned to different computations controls the allocation of computer computational capacity among several different functions that may be required in the same time period: for example, processing of DSKY keyboard inputs is conventionally assigned a priority of 30_8 (since it should be completed before another key is pressed); the computations associated with the two-second powered-flight navigation and guidance cycle conventionally have a priority of 20_8; and the computations to obtained a new velocity-required vector for rendezvous guidance computations (via a Lambert routine) have been assigned a priority of 10_8. Should a special display computation be desired, this has been assigned a priority of 05_8 (so it will not interfere with the "in-line" computations).

When the performance of a job is terminated, the job remaining that has the highest priority will be performed next (if there are no other jobs, then a dummy job routine, discussed later, is entered to await tasks or jobs). Performance of a job can be suspended (to await a DSKY input, for example) by setting its priority negative: this action is termed "putting the job to sleep". When the appropriate action has been accomplished, the job is "awakened". No undesirable conflicts take place if two jobs with the same priority are in the executive

system at the same time, since both will be done before jobs of lower priority, although the sequence in which the two jobs will be performed cannot necessarily be uniquely determined prior to flight. Once jobs of short duration (such as the processing of a keyboard input) are started, they run to completion (subject, of course, to interruption by tasks); jobs of longer duration check cell 0067_8 (NEWJOB) periodically for a job of higher priority (the cell is set positive non-zero if one is waiting). Conventionally, NEWJOB is checked only by the "DANZIG" routine of the interpretive language (see Section VIC), with other job changes made as a consequence of explicit entrance to the executive system routines (to change job priority, terminate a job, put a job to sleep/awaken it, etc.), or by the dummy job routine with its optional computer self-check activities. Some special-purpose test programs also perform checks of the NEWJOB cell, as do other portions of the coding when checks are necessary for proper sequencing (such as restart or display logic).

When a job is established, it is assigned one of the seven(CM) or eight (LM) Job Register Sets discussed later in this section, each of which consists of 12 erasable memory cells. If all of these Job Register Sets have been allocated already (to either active or "dormant", i.e. sleeping, jobs), then a software restart is caused. In addition, a job when established can also be assigned a VAC area (see Section VID), one of five groups of 43 cells used for storage of some interpretive language special-purpose cells and for the "push-down list" of that language. If all the VAC areas have been allocated already, a software restart is caused. A VAC area can also be assigned by another job when it is running for some other storage purpose (one is conventionally used, for example, to retain optics mark information). If no VAC area is specified when the job is established, the software information that

normally would be set to the starting address of the selected VAC area (see Section VII) is set so that the interpretive language QPRET cell (used for return address information retention) would be placed in MPAC+6 of the Job Register Set: this permits a limited use of the interpretive language to be made by jobs without tying up a whole VAC area.

Jobs can be established either by tasks or by other jobs. The two most frequently used sequences are:

```
CA      yy      (Priority, in bits 14-10)
TC      FINDVAC (Assigns a VAC area)
2CADR   XXXX    (Job starting address)
```

and

```
CA      yy
TC      NOVAC   (No VAC area assigned)
2CADR   XXXX
```

A third method, for a VAC area, is to enter "SPVAC" with the priority information stored in NEWPRIO and with the 2CADR of the starting address in (A, L). To put a job to sleep, the job itself must transfer to "JOBSLEEP" with the accumulator containing the CADR of the job's starting address when it is awakened (SUPERBNK must be the same as that of job at present); the job is awakened by transferring to "JOBWAKE" with the accumulator containing the same CADR. Priority changing is accomplished by entering "PRIOCHNG" with bits 14-10 of the accumulator containing the desired new priority for the job. Finally, to terminate the performance of a job transfer is made to "ENDOFJOB".

As mentioned previously, tasks take precedence over jobs, since (in general) a task is capable of interrupting a job after any machine-

language program step. Jobs, in turn, are performed in a sequence as generally determined by their relative priorities. If no tasks and no jobs are to be performed, a routine known as the "dummy job" is performed: this routine starts by turning off the Computer Activity light (bit 2 of channel 11), and then checks to see if performance of the computer self-check routine is required. If it is not required (and periodically if computer self-check functions have been specified), the NEWJOB cell is checked for an indication of a job, and if one is found the Computer Activity light is turned back on. If no job is indicated, the routine rechecks for specification of the computer self-check functions, and the loop is repeated. Since this "dummy job" routine does not have its own Job Register Set, it is not actually a job, but instead a subroutine of the executive system: it is convenient, however, to refer to it as a "job".

The equations given below describe the computations performed by the various program routines associated with the executive system: ignoring the program interrupt #11 initiation by NEWJOB, this system is completely a software package. Priority information for a job is conventionally obtained from cells "PRIO1" through "PRIO37", tags associated with fixed-fixed memory having the necessary bit configuration (in bits 14-10, with remaining bits 0).

FINDVAC — Entered with priority information in A, to assign a VAC area as the job is established.

Inhibit interrupts

NEWPRIO = A

$NEWLOC_{dp} = E_{Q_{dp}}$ (The 2CADR information on job starting address)

Scan VACIUSE cells from I = 1 to I = 5 (in order) for the first one that is positive non-zero. If none found:

Proceed to "BAILOUT" (pattern 31201_8) CM only

Proceed to "BAILOUT1" (pattern 31201_8, TS = (Q, FBANK of caller)) LM only

NEWPRIO = NEWPRIO + VACIUSE + 1

VACIUSE = 0

Proceed to "NOVAC2"

NOVAC — Entered with priority information in A, to establish a job that does not require a VAC area (when established).

Inhibit interrupts

NEWPRIO = A + ("MPAC+6" - 42) (causes QPRET to be MPAC+6 when job is running)

$NEWLOC_{dp} = E_{Q_{dp}}$ (The 2CADR information on job starting address)

Proceed to "NOVAC2"

SPVAC — Entered with NEWPRIO set to priority information and with(A, L) set to 2CADR of starting address. Causes a VAC area to be assigned.

Q = Q -2

$NEWLOC_{dp}$ = (A, L)

Proceed to fourth line of "FINDVAC" (the VACIUSE scan)

NOVAC2

Scan $PRIORITY_I$ cells (in order) from I = 0 to I = 6(CM) or 7(LM) for the first one that is -0. If none found:

Proceed to "BAILOUT" (pattern 31202_8) CM only

Proceed to "BAILOUT1" (pattern 31202_8, TS = (Q, FBANK of caller)) LM only

LOCCTR = 12 I

$PRIORITY_I$ = NEWPRIO

$PUSHLOC_I$ = bits 9-1 of NEWPRIO

If LOCCTR > 0:

Proceed to "SETLOC"

OVFIND = 0

FIXLOC = PUSHLOC

Proceed to "SPECTEST"

SPECTEST

If NEWJOB $\neq$ -0:

Proceed to "SETLOC"

NEWJOB = +0

$(LOC, BANKSET)_0 = NEWLOC_{dp}$

Proceed to address specified by Q +2 (Return to routine that entered executive system)

SETLOC

$(LOC, BANKSET)_I = NEWLOC_{dp}$

TS = NEWJOB / 12

If NEWPRIO - $PRIORITY_{TS} > 0$:

NEWJOB = LOCCTR (Same priority job without VAC area is "lower" than with VAC area)

Proceed to address specified by Q +2

CHANG1 Entered when a non-interpreter job senses that NEWJOB is non-zero.

TS_1 = Q

TS_2 = BBANK

Proceed to "CHANJOB"

CHANG2 Entered when interpreter (in "DANZIG", Section VIC) senses that NEWJOB is non-zero.

TS_1 = - LOC (Negative means interpreter entrance)

TS_2 = BANKSET

Proceed to "CHANJOB"

CHANJOB

Inhibit interrupts

$TS_2 = TS_2$ + SUPERBNK (or'ed into bits 7-5)

I = NEWJOB / 12

Set $(LOC, BANKSET)_0 = (LOC, BANKSET)_I$ and $(LOC, BANKSET)_I = (TS_1, TS_2)$

SUPERBNK = bits 7-5 of $BANKSET_0$

Set $MPAC+i_O = MPAC+i_I$ and $MPAC+i_I = MPAC+i_O$ (i = 0-6)

Set $MODE_O = MODE_I$ and $MODE_I = MODE_O$

If OVFIND $\neq$ 0:

    $PUSHLOC_O = -PUSHLOC_O$

OVFIND = 0

Set $PUSHLOC_O = PUSHLOC_I$ and $PUSHLOC_I = PUSHLOC_O$

Set $PRIORITY_O = PRIORITY_I$ and $PRIORITY_I = PRIORITY_O$

FIXLOC = bits 9-1 of $PRIORITY_O$

If $PUSHLOC_O < 0$:

    $PUSHLOC_O = -PUSHLOC_O$

    OVFIND = 1

NEWJOB = +0

Release interrupts

If $LOC_O > 0$:

    Proceed to address specified by $(LOC, BANKSET)_O$

$LOC_O = 1 - LOC_O$ (From interpreter, now set to original LOC +1)

BBANK = $BANKSET_O$

Proceed to third line of "INTPRET" (Section VIC)

JOBSLEEP — Entered to put a job "to sleep", with A set to CADR of desired starting address when job is awakened (serves also as identification of job when "JOBWAKE" entered).

$LOC_O = A$

Inhibit interrupts

$PRIORITY_O = -PRIORITY_O$

$BANKSET_O$ = bits 7-1 of BBANK + SUPERBNK (or'ed into bits 7-5)

BUF+1 = -0

Proceed to "EJSCAN"

JOBWAKE — Entered to "awaken" job put to sleep by "JOBSLEEP".

Inhibit interrupts

NEWLOC = A (Has CADR of job, used as identification)

Q = Q - 2

Scan $PRIORITY_I$ cells (in order) from I = 0 to I = 6(CM) or 7(LM) for contents < 0 and for LOC_I = NEWLOC: the first one found terminates the scan. If none found:

LOCCTR = -1

Proceed to address specified by Q +2

LOCCTR = 12 I

NEWPRIO = - $PRIORITY_I$

$PRIORITY_I$ = NEWPRIO

TS_1 = (bits 10-1 of NEWLOC) + 2000_8

TS_2 = (bits 15-11 of NEWLOC) + $BANKSET_I$

$NEWLOC_{dp}$ = (TS_1, TS_2)

If LOCCTR = 0:

    Proceed to "SPECTEST"

Proceed to "SETLOC"

PRIOCHNG — Entered to change the priority of a job, with A set to the desired new priority.

Inhibit interrupts

NEWPRIO = A

$BANKSET_0$ = BBANK

LOC_0 = Q

BUF+0 = +0

$PRIORITY_0$ = NEWPRIO + (bits 9-1 of $PRIORITY_0$)

BUF+1 = - $PRIORITY_0$

Proceed to "EJSCAN"

ENDOFJOB — Entered to terminate the performance of a job and release the memory areas allocated.

Inhibit interrupts

BUF+1 = -0

TS = bits 9-1 of $PRIORITY_0$

If TS - ("MPAC+6" - 42) > 0:

    TS = TS - 1

    E_{TS} = TS (Releases VAC area)

$PRIORITY_0$ = -0

Proceed to "EJSCAN"

EJSCAN

I = 0

I = I + 1

If $PRIORITY_I > 0$:

BUF+2 = $PRIORITY_I$ -1 (the 1 is least significant bit of cell, not of bits 14-10 that are the actual job priority)

If BUF+2 + BUF+1 > 0:

BUF+1 = - BUF+2

BUF+0 = 12 I

If I $\leq$ 5(CM) or 6(LM):

Proceed to second line of "EJSCAN"

If BUF+1 = -0:

Proceed to "DUMMYJOB"

If BUF+0 = +0: (Priority change, present job still the highest priority)

NEWJOB = +0

Proceed to address specified by $(LOC, BANKSET)_0$

NEWJOB = BUF+0

$TS_1 = LOC_0$

$TS_2 = BANKSET_0$

Proceed to "CHANJOB"

DUMMYJOB Entered (as a routine in the executive system, not as a true "job") if no active jobs available.

NEWJOB = -0

Release interrupts

Set bit 2 (Computer Activity) of channel 11 = 0

Proceed to "ADVAN"

ADVAN Entered from computer self-check routine (part of "idle loop") to check for availability of a functional job.

If NEWJOB = -0:

Proceed to address specified by SELFRET and BBCON of "SELFCHK" (computer self-check routine; SELFRET initialized as part of standard initial conditions)

If NEWJOB = +0:

    Set bit 2 (Computer Activity) of channel 11 = 1

    SUPERBNK = bits 7-5 of $BANKSET_0$

    BBANK = $BANKSET_0$

    Proceed to address specified by LOC_0 (and BBANK)

Inhibit interrupts

If NEWJOB = +0:

    Release interrupts

    Set bit 2 (Computer Activity) of channel 11 = 1

    SUPERBNK = bits 7-5 of $BANKSET_0$

    BBANK = $BANKSET_0$

    Proceed to address specified by LOC_0 (and BBANK)

Set bit 2 (Computer Activity) of channel 11 = 1

$(TS_1, TS_2) = (LOC, BANKSET)_0$

Proceed to third line of "CHANJOB"

Contents of Job Register Sets

When a job is established, it is assigned one of 7(CM) or 8(LM) available Job Register Sets, each of which is comprised of 12 erasable cells. When a job is active, its Job Register Set occupies the first group of Job Register Set cells, so that the various tags assigned to the 12 cells can be referenced without use of the INDEX order (contrary, for example, to the assignment of VAC areas discussed in Section VID). The individual cells in a Job Register Set are assigned the following functions:

Tag	Contents if Job Active	Contents if Not Active
1. MPAC+0	Most significant part of multi-purpose accumulator (scalar) or vector x component in interpreter; otherwise a storage cell.	Same as Active.
2. MPAC+1	Next most significant part of multi-purpose accumulator (scalar) or least significant half of vector x component in interpreter; otherwise a storage cell.	Same as Active.
3. MPAC+2	Least significant part of multi-purpose accumulator (scalar) in interpreter; otherwise a storage cell.	Same as Active.
4. MPAC+3	Most significant half of vector y component in interpreter; otherwise a storage cell.	Same as Active.
5. MPAC+4	Least significant half of vector y component in interpreter; otherwise a storage cell.	Same as Active.

Tag	Contents if Job Active	Contents if Not Active
6. MPAC+5	Most significant half of vector z component in interpreter; otherwise a storage cell.	Same as Active.
7. MPAC+6	Least significant half of vector z component in interpreter; otherwise a storage cell.	Same as Active.
8. MODE	Identification of type of computating being done in interpreter; otherwise a storage cell. It is +1 for TP (triple precision), +0 for DP (double precision), and -1 for VC (vectors).	Same as Active.
9. LOC	Address information for operand in interpreter; otherwise an unsaved storage cell.	If via "CHANG2", the complement of Active contents; otherwise has S-register information where job should be initiated (CADR if "JOBSLEEP").
10. BANKSET	BBANK associated with LOC in interpreter; otherwise an unsaved storage cell.	BBANK value associated with LOC (and SUPERBNK); EBANK and SUPERBNK if from "JOBSLEEP".
11. PUSHLOC	Address of next cell to be loaded in push-down list if interpreter; otherwise a storage cell.	If OVFIND $\neq$ 0, set to - PUSHLOC; if is 0, same contents as Active.
12. PRIORITY	Bits 14-10 contain priority of job; bits 9-1 contain FIXLOC (starting address of VAC area if assigned, otherwise "MPAC+6" - 42). Cell is complemented if job "put to sleep".	Same as Active. A value of -0 means that the Job Register Set is available (set as part of initial conditions e.g. at computer turn-on).

As is evident from this table, most of the cells in the Job Register Set are oriented towards the interpreter, and can be used for other functions by a job which has no need for the interpreter language. By using LOCCTR as an indexing parameter after establishing a job, it is possible to load the MPAC cells, for example, with information pertinent to the job that has been established (a similar function can be performed after a job is awakened).

The Job Register Set for the currently active job (if there is one, of course) is always stored in the first Job Register Set, as mentioned previously. A job when established is assigned a group of cells comprising such a set, and this group is moved as an ensemble by the "CHANJOB" routine. The 12 cells of the set remain together, but their absolute erasable memory cells can vary (although when the job is active it always has its set in the first location). Job Register Sets are moved by XCH (or DXCH) operations, so no separate set of buffer cells or duplication of the cells of the currently active job is necessary. Since the currently active job is always in the first location, the software can refer to e.g. MPAC+2 directly (this cell is shown in this section as MPAC+2_0 to avoid confusion with the inactive sets).

When a job is established, the only specific initial conditions which are set are words 9-12 above. Since word 11 is set positive, this gives an initial condition of 0 for OVFIND. No special setting of words 1-8 is performed, so they will contain residual information from the previous job employing these cells. As part of the standard set of initial conditions (e.g. at computer turn-on), each PRIORITY cell is set to -0 (and each VACiUSE cell, as mentioned in Section VID, is set to its own address).

VIIC Mechanization of Restart Capability

When program interrupt #11 (see Section IIH) is generated, a hardware restart is initiated. This program interrupt could be triggered by the software itself, as well as by various hardware-detected problems. In addition, a "software restart" (similar computations) can be done if appropriate. Although the details of the computations done when a restart occurs are better left to mission-peculiar information, the general philosophy behind the restart scheme employed is reviewed in this section.

A hardware restart can be generated at literally any location in the program (it cannot be inhibited by program means), and it could be due to hardware difficulties (such as a parity failure on a cell read from memory) as well as to some software-detected difficulty (such as too many tasks in the waitlist system, causing "software restart" activity). The responsibility of the restart logic in the software is to attempt to get the program running again with (hopefully) a minimum of disturbance to the mission. This requires that, fundamentally, the appropriate tasks be called at the end of the correct time intervals (from a suitable base time), and that the appropriate jobs be re-established with the proper priorities. In some cases, the proper starting addresses for the jobs and/or tasks should not be at their beginning, but instead at some intermediate location or even at a special location entered only if a restart is encountered (such as a restart while the accelerometers are being sensed and reset).

In order to accomplish the required restart functions, the various activities performed by the program software in essentially independent computations are divided into "restart groups", of which there is provision

in the restart software for a maximum of six. One group, for example, might be concerned with the periodic powered-flight navigation cycling; another with orbital integration (perhaps required with powered-flight to determine relative CSM/LM information); a third with the timing of events leading to engine ignition; a fourth with generation of a time display on the DSKY; a fifth with computation of required-velocity information for a rendezvous ("Lambert") maneuver; and a sixth with a special computation performed shortly before engine ignition (to estimate the length of the burn). All six of these functions could be part of the complete program's computational load (as jobs or tasks) at one time (and be in various stages of completion), and consequently they could be associated with separate restart groups. Not all computational activity in the program is restart protected in this fashion: if a restart occurs while data are being loaded via the DSKY, for example, the loading sequence would have to be initiated again at its beginning.

A restart "group", therefore, can generally be considered to be associated with a particular functional software activity. Each group, in turn, is conventionally divided into a number of "phases" indicating just where the computations should be re-initiated in the event of a restart. The phase information for a given group is retained in both true and complemented form in the erasable memory (giving a total of 12 cells for the six pairs of cells associated with the six restart groups). When the restart software is entered, a check is made that all six pairs of cells have the proper internal complement relationship. If they do not, it is concluded that insufficient information is available to permit a satisfactory resumption of computations, and the attempt to perform the restart is abandoned (in favor of a "fresh start" that leaves

the guidance system in essentially an "idling" configuration). The complement relationship could be destroyed if the erasable memory was modified by whatever caused the restart action (such as a power transient), or if the restart occurred during certain portions of the programs that change the restart phase information.

If the restart software concludes that adequate phase information is available (as evidenced by agreement of the complement relationship for the six pairs of phase data), the "RESTARTS" routine can be entered for each restart group that is "active" (a group is made "inactive" by setting the phase of that group to 0, indicating that it is not restart protecting any of the computations). This "RESTARTS" routine, depending on the value of the phase associated with that group, can cause jobs to be established (with or without VAC areas) and/or waitlist tasks to be called at appropriate times (via "LONGCALL" or the normal "WAITLIST" routines). The value of the phase information also determines whether one or two such jobs and/or tasks are to be re-initiated, and in addition whether the parameters associated with the re-initiation are to be obtained from fixed or erasable memory.

Phase values are stored in both true and complemented form, and it is conventional to refer to the phase by the "true" value: since negative phase values are not implemented in the logic, this means that all "true" phase values are positive. These phase values are stored in erasable memory cells with the tags PHASE1, PHASE2, ... PHASE6 (the complemented ones are in -PHASE1, -PHASE2, ... -PHASE6). Given on the following pages are the meanings of the various phase values as stored in the memory (values other than those mentioned should not normally exist).

1. A phase value of 00000_8 (+0) means that the restart group is "inactive" (it is not restart protecting any computations).

2. A phase value of 00001_8 means that a special display-interface-routines job is to be established to cause the last software-generated DSKY display to be restored (only one group should have this phase value). The job is given a VAC area (priority 14_8).

3. An odd phase value in the range 00003_8 - 00177_8 causes a fixed-memory single restart: the necessary restart data is obtained from a fixed memory table for either a task, a job, or "LONGCALL".

4. An even phase value in the range 00002_8 - 00176_8 causes a fixed-memory double restart: the necessary restart data is obtained from two successive three-entry fixed memory table quantities, either or both of which can be for a task or a job (one can be for "LONGCALL", since only one "LONGCALL" user can exist at a given time).

5. A phase value of $00X01_8$ (X in range 4-7) causes an erasable-memory single restart of a waitlist task. Time information is given in PHSPRDTi and address information in PHSNAMEi (where i is group).

6. A phase value of $00X02_8$ (X in range 4-7) causes an erasable-memory single restart of a job. Priority information is given in PHSPRDTi and address information in PHSNAMEi (where i is group).

7. A phase value of $00X04_8$ (X in range 4-7) causes an erasable-memory single restart of a "LONGCALL" task. Time information is given in PHSPRDTi and address information in PHSNAMEi (where i is group).

8. A phase value of 01001_8 causes an erasable-memory single restart of a job and also a restart of the special display-interface-routines job to restore the last DSKY display: this is a combination of #2 and #6 above.

9. A phase value in the range 01002_8 - 01777_8 causes an erasable-memory single restart of a job and also a fixed-memory single restart. If the phase value is odd, this is a combination of #3 and #6 above; if it is even, however, only the first of the two entries in fixed memory for the "double restart" of #4 is started (plus the erasable restart of #6 above). Only bits 7-1 are used for the fixed memory restart phase information.

The value of the phase for a particular restart group, interpreted as described above, is used to select an appropriate table entry in fixed and/or erasable memory. The table entries, which are separated by groups (and with even restart phase information for fixed memory stored before odd phase information of phase 3 or above, so memory capacity is not wasted if there are more fixed memory tables of one type than the other), each consist of three cells, which have the following significance:

Cell	Fixed-Memory Table	Erasable-Memory Table	Use
1	PRDTTAB	PHSPRDTi	Priority/time Information
2	CADRTAB	PHSNAMEi	GENADR data
3			BBCON data

The polarity with which information is stored in the tables is used to determine whether the table information pertains to a job, a waitlist task, or a "LONGCALL" task, and in addition to determine which of several available options for defining the re-initiation parameters is to be employed. These characteristics are summarized below:

A job is identified by the fact that cell #2 is positive (bit 15 of the cell is 0). The job is established with the 2CADR information specified by cells #2 and #3, and with priority given by the absolute value of cell #1. If cell #1 is positive, "FINDVAC" is used (VAC area assigned); if it is negative, "NOVAC" is used (no VAC area assigned).

A waitlist task is identified by the fact that cell #2 and #3 are both stored in complement form (bit 15 of cell #2 and bit 10 of cell #3 are both 1). The task is called with the starting address specified by the complement of (cell #2, cell #3), and with a time delay determined as follows:

a) If cell #1 is positive, the time delay is specified by the value of that cell, measured from the time when TBASEi was last set to - TIME1 (cell 0025_8, see Section IID). There is a TBASEi (i = 1 - 6) for each group.

b) If cell #1 is -0 (77777_8), the time delay is the minimum allowed by the waitlist system logic, namely 0.01 second.

c) If cell #1 is negative non-zero, the time delay is specified by the contents of the cell whose GENADR address complement is in cell #1, with these contents interpreted as for (a). The fixed or erasable memory bank information (if necessary) to determine this cell is the same as that specified by the complement of cell #3.

A "LONGCALL" task is identified by the fact that cell #2 is in complement form (bit 15 = 1) and cell #3 is not in complement form (i.e. bit 10 = 0). The starting address of the task is specified by (- cell #2, cell #3). The GENADR of the double precision cell containing the required time delay (measured from the time when $LONGBASE_{dp}$ last set to (TIME2, TIME1)) is specified by cell #1. The fixed or erasable memory bank information (if necessary) to determine this cell is the same as that specified by cell #3. Cell #1 negative for erasable memory table information, positive for fixed memory.

The "base" times of TBASEi (i = 1 - 6) and $LONGBASE_{dp}$ must be set under program control to be consistent with the task restart information time-delay values. If the indicated time delay between when "RESTARTS" is performed and when the task is desired is less than 1 centi-second, including negative values, the "delay" is set to 1 centi-sec.

During the course of the computations, it is necessary to update the phase value associated with the appropriate group. This can be done directly by loading new phase information into the appropriate group's phase cells (-PHASEi for a given group is in an erasable memory cell immediately before PHASEi for that group), or it can be accomplished through use of one of several available phase-changing subroutines. The simplest of these is "NEWPHASE", which has the following calling sequence (CM only):

```
CA    ppp        (or otherwise load A with positive phase)
TC    NEWPHASE
OCT   g          (group number)
```

This will cause what is conventionally referred to as "phase g.ppp" to be set (conventionally used for fixed-memory table data), such as phase 2.11 for g = 2 and ppp = 11_8, since phase table information for fixed memory is conventionally numbered in octal. Erasable memory phase information, to avoid confusion with fixed memory, should <u>not</u> be referred to in this fashion: instead, a notation such as "phase 2. " for such restart settings is sometimes used. If "NEWPHASE" is entered with a negative accumulator, the phase that is set will be the complement of A, and in addition the group's TBASEi will be set to - TIME1 (for use in defining the time when a waitlist task should be initiated).

The most commonly used phase-changing subroutine is "PHASCHNG", which has a variety of options depending on the details of the calling sequence. The general form is:

```
TC      PHASCHNG
OCT     N5 N4 N3 N2 N1
(possibly additional lines)
```

The 15 bits of the word following the TC "PHASCHNG" are used to identify the nature of the restart that is desired: these restarts are conventionally known as "Type A" (fixed-memory table only), "Type B" (fixed and erasable tables), and "Type C" (erasable-memory table only). The format of the word for these three types is:

Type	N_5	N_4	N_3	N_2	N_1
A	T L O	O O P	P P P	P P P	G G G
B	T L 1	D A P	P P P	P P P	G G G
C	T L O	1 A D	x x x	C J W	G G G

x means ignored, 0 means binary 0, 1 means binary 1

A = 0 if address for restart is return address from "PHASCHNG".
  = 1 if address specified by "PHASCHNG" calling sequence.

C = 0 if J or W are 1.
  = 1 if a "LONGCALL" restart task is to be done.

D = 0 if priority/time information already in PHSPRDTi.
  = 1 if priority/time information in "PHASCHNG" calling sequence.

G's give the restart group number.

J = 0 if C or W are 1.
  = 1 if a job restart to be done.

L = 0 if $LONGBASE_{dp}$ is to be left alone.
  = 1 if $LONGBASE_{dp}$ is to be set to (TIME2, TIME1).

P's are phase number (for fixed memory restart table data).

T = 0 if TBASEi is to be left alone.
  = 1 if TBASEi is to be set to - TIME1.

W = 0 if C or J are 1.
  = 1 if a "WAITLIST" restart task is to be done.

The table information on the following pages summarizes the various restart options available by use of the "PHASCHNG" routine. Since in all cases the most significant two bits for the parameter control settings of TBASEi (values of 4-7) and $LONGBASE_{dp}$ (values of 2,3,6,7), the value of N_5 has merely been indicated as "odd" or "even". Similarly, since N_1 always specifies the restart group, it is not included either.

	N_5	N_4	N_3	N_2	Meaning
1.	Even	0,1	0-7	0-7	Set the restart group indicated by N_1 to the phase indicated by N_4 - N_2. Set time base per N_5 if specified.
2.	Even	4	0-7	1	Store a waitlist restart in erasable memory, using information in PHSPRDTi for time and return address from "PHASCHNG" as starting address. Set time base per N_5 if specified. N_3 ignored.
3.	Even	4	0-7	2	Store a job restart in erasable memory, using information in PHSPRDTi for priority and return address from "PHASCHNG" as starting address. Set time base per N_5 if specified. N_3 ignored.
4.	Even	4	0-7	4	Store a "LONGCALL" task restart in erasable memory, using information in PHSPRDTi for time and return address from "PHASCHNG" as starting address. Set time base per N_5 if specified. N_3 ignored.
5.	Even	5	0-7	1	Store a waitlist restart in erasable memory, using information in cell following N_5 - N_1 for time and return address from "PHASCHNG" for starting address. Set time base per N_5 if specified. N_3 ignored.
6.	Even	5	0-7	2	Store a job restart in erasable memory, using information in cell following N_5 - N_1 for priority and return address from "PHASCHNG" for starting address. Set time base per N_5 if specified. N_3 ignored.

	N_5	N_4	N_3	N_2	Meaning
7.	Even	5	0-7	4	Store a "LONGCALL" task restart in erasable memory, using information in cell following $N_5 - N_1$ for time and return address from "PHASCHNG" for starting address. Set time base per N_5 if specified. N_3 ignored.
8.	Even	6	0-7	1	Store a waitlist restart in erasable memory, using information in PHSPRDTi for time and the two cells following $N_5 - N_1$ as the 2CADR of the starting address. Set time base per N_5 if specified. N_3 ignored.
9.	Even	6	0-7	2	Store a job restart in erasable memory, using information in PHSPRDTi for priority and the two cells following $N_5 - N_1$ as the 2CADR of the starting address. Set time base per N_5 if specified. N_3 ignored.
10.	Even	6	0-7	4	Store a "LONGCALL" task restart in erasable memory, using information in PHSPRDTi for time and the two cells following $N_5 - N_1$ as the 2CADR of the starting address. Set time base per N_5 if specified. N_3 ignored.
11.	Even	7	0-7	1	Store a waitlist restart in erasable memory, using information in cell following $N_5 - N_1$ for time and the two subsequent cells as the 2CADR of the starting address. Set time base per N_5 if specified. N_3 ignored.

	N_5	N_4	N_3	N_2	Meaning
12.	Even	7	0-7	2	Store a job restart in erasable memory, using information in cell following N_5 - N_1 for priority and the two subsequent cells as the 2CADR of the starting address. Set time base per N_5 if specified. N_3 ignored.
13.	Even	7	0-7	4	Store a "LONGCALL" task restart in erasable memory, using information in cell following N_5 - N_1 for time and the two subsequent cells as the 2CADR of the starting address. Set time base per N_5 if specified. N_3 ignored.
14.	Odd	0,1	0-7	0-7	Set the group indicated by N_1 to the phase indicated by N_4 - N_2 (using only the least significant bit of N_4), to control a fixed memory table restart. In addition, store a job restart in erasable memory, using information in PHSPRDTi for priority and return address from "PHASCHNG" as starting address. Set time base per N_5 if specified.
15.	Odd	2,3	0-7	0-7	Set the group indicated by N_1 to the phase indicated by N_4 - N_2 (using only the least significant bit of N_4), to control a fixed memory table restart. In addition, store a job restart in erasable memory, using the information in PHSPRDTi for priority and the two cells following N_5 - N_1 as the 2CADR of the starting address. Set time base per N_5 if specified.

	N_5	N_4	N_3	N_2	Meaning
16.	Odd	4,5	0-7	0-7	Set the group indicated by N_1 to the phase indicated by N_4 - N_2 (using only the least significant bit of N_4), to control a fixed memory table restart. In addition, store a job restart in erasable memory, using information in the cell following N_5 - N_1 for priority and return address from "PHASCHNG" as starting address. Set time base per N_5 if specified.
17.	Odd	6,7	0-7	0-7	Set group indicated by N_1 to the phase indicated by N_4 - N_2 (using only the least significant bit of N_4), to control a fixed memory table restart. In addition, store a job restart in erasable memory, using information in the cell following N_5 - N_1 for priority and the two subsequent cells as the 2CADR of the starting address. Set time base per N_5 if specified.

The third phase-changing subroutine is "2PHSCHNG", which has the following general form of calling sequence:

```
TC      2PHSCHNG
OCT     M5 M4 M3 M2 M1
OCT     N5 N4 N3 N2 N1
```

The word represented by M_5 - M_1 follows the format of a "Type A" phase change discussed for "PHASCHNG", except that M_5 is only examined for values of 0 and 4 (to determine if TBASEi should be set). The N_5 - N_1 word is treated the same as the corresponding word discussed for "PHASCHNG", including capability to set $LONGBASE_{dp}$.

The equations given below describe the computations performed by the various program routines associated with the mechanization of restarts.

NEWPHASE — Entered with A set to desired phase, and cell following TC "NEWPHASE" set to group to be changed. Routine CM only.

Inhibit interrupts

$TS_1 = A$

$TS_2 = E_Q$

If $TS_1 \leq -0$:

$TS_1 = |TS_1|$

$TBASEi_{TS_2} = -$ TIME1

Set $(-PHASEi, PHASEi)_{TS_2} = (-TS_1, TS_1)$ and $(A, L) = (-PHASEi, PHASEi)_{TS_2}$ (i = 1-6 for TS_2 = 1-6)

Release interrupts

Proceed to address specified by Q +1

2PHSCHNG

Inhibit interrupts

$TS = E_Q$

Q = Q +1

TEMPG2 = bits 3-1 of TS

TEMPP2 = (bits 13-4 of TS), shifted right 3 places

TEMPSW2 = bit 15 of TS (negative if 1)

Proceed to third line of "PHASCHNG"

PHSCHNGA — Entered with argument in A. Routine LM only.

Inhibit interrupts

TEMPSW2 = 1

TS A

Proceed to fifth line of "PHASCHNG"

PHASCHNG

Inhibit interrupts

TEMPSW2 = 1

TS = E_Q

Q = Q +1

TEMPG = bits 3-1 of TS

TEMPP = (bits 13-4 of TS), shifted right 3 places

If bits 13-12 of TS $\neq 00_2$:

Proceed to "ONEORTWO"

$PHASEi_{TEMPG}$ = TEMPP

Proceed to "BELOW1"

BELOW1

If TEMPSW2 $\leq$ 0:

$(-PHASEi, PHASEi)_{TEMPG2}$ = (- TEMPP2, TEMPP2) (i = 1-6 for TEMPG2 = 1-6)

IF TEMPSW2 < 0:

$TBASEi_{TEMPG2}$ = - TIME1

If bit 15 of TS = 1:

$TBASEi_{TEMPG}$ = - TIME1

If bit 14 of TS = 1:

$LONGBASE_{dp}$ = (TIME2, TIME1)

$-PHASEi_{TEMPG}$ = - TEMPP

Release interrupts

Proceed to address specified by Q

ONEORTWO

If bits 13-12 of TS = 01_2:

If bit 7 of TEMPP = 1: (i.e. bit 10 of original input)

TEMPPR = E_Q

Q = Q +1

If bit 7 of TEMPP = 0:

TEMPPR = $PHSPRDTi_{TEMPG}$

If bits 13-12 of TS = 10_2:

TEMPPR = $PHSPRDTi_{TEMPG}$

If bits 13-12 of TS = 11_2:

TEMPPR = E_Q

Q = Q +1

If bit 8 of TEMPP = 1: (i.e. bit 11 of original input)

TEMPNM = $E_{Q_{dp}}$

Q = Q +2

If bit 8 of TEMPP = 0:

TEMPNM = (Q, BBANK + SUPERBNK (or'ed into bits 7-5))

$PHASEi_{TEMPG}$ = TEMPP

$PHSPRDTi_{TEMPG}$ = TEMPPR

$PHSNAMEi_{TEMPG}$ = $TEMPNM_{dp}$

Proceed to "BELOW1"

RESTARTS — Entered from mission program performed when restart generated after it is concluded that valid phase information is available. TPHS contains the phase information and TGRP the corresponding restart group.

TEMPSWCH = "PHSPART2"

GOLOC+2 = Return address (to routine from which "RESTARTS" entered)

If bits 10-9 of TPHS $\neq 00_2$, proceed to "ITSAVAR"

Proceed to "GETPART2"

GETPART2

If TPHS = 1:

Establish job to re-generate displays (cf. #2, page VIIC-4)

Return to routine from which "RESTARTS" entered

If bit 1 of TPHS = 1: (an odd number)

GOLOC+2 = Return address (to routine from which "RESTARTS" entered)

POINTER = index to locate table entry (from TPHS and TGRP)

Proceed to "CONTBL2"

GOLOC+2 = TEMPSWCH

POINTER = index to locate table entry (from TPHS and TGRP)

Proceed to "CONTBL2"

CONTBL2

(GOLOC, GOLOC+1) = $CADRTAB_{POINTER_{dp}}$ (CADRTAB is start of fixed-memory restart table data for addresses)

If GOLOC $\geq$ 0:

TS = $PRDTTAB_{POINTER}$ (PRDTTAB is start of fixed-memory restart table data for priority/time information)

If TS $>$ 0:

Establish a job with a VAC area and with 2CADR starting address given by (GOLOC, GOLOC+1) and priority of TS

Proceed to address specified by GOLOC+2

If TS $\leq$ 0:

Establish a job without a VAC area, with 2CADR starting address given by (GOLOC, GOLOC+1) and priority of (- TS)

Proceed to address specified by GOLOC+2

GOLOC = - GOLOC

If bit 10 of GOLOC+1 = 1:

GOLOC+1 = - GOLOC+1

TS = $PRDTTAB_{POINTER}$

Proceed to "TIMETEST"

TS = $PRDTTAB_{POINTER}$

Proceed to "ITSLGCL1"

ITSLGCL1

LONGTIME = $E_{TS,\ GOLOC+1_{dp}}$ (BBANK setting from GOLOC+1, S-register from TS)

LONGTIME = LONGTIME - (TIME2, TIME1) + $LONGBASE_{dp}$

If $LONGTIME_{dp} \leq 0$:

Call task with starting address $GOLOC_{dp}$ in 0.01 second (via normal waitlist)

Proceed to address specified by GOLOC+2

Call task with starting address $GOLOC_{dp}$ in $LONGTIME_{dp}$ centi-sec (via "LONGCALL" routine)

Proceed to address specified by GOLOC+2

PHSPART2

POINTER = POINTER +3 (Get next table entry for even restarts)

GOLOC+2 = Return address (to routine from which "RESTARTS" entered)

Proceed to "CONTBL2"

TIMETEST

If TS = -0:

Call task with starting address $GOLOC_{dp}$ in 0.01 second

Proceed to address specified by GOLOC+2

If TS < 0:

TS = - TS

TS = $E_{TS,\ GOLOC+1}$ (BBANK setting from GOLOC+1, S-register from TS)

TS_1 = TIME1 - (- $TBASEi_{TGRP}$)

If TS_1 < 0:

TS_1 = 16384 + TS_1 (now has elapsed time since TBASEi set)

TS = TS - TS_1, limited ≥ 1 centi-second

Call task with starting address $GOLOC_{dp}$ in TS centi-seconds

Proceed to address specified by GOLOC+2

ITSAVAR

If bit 10 of TPHS = 1: (i.e. a "Type B" setting)

TEMPSWCH = Return address (to routine from which "RESTARTS" entered)

GOLOC+2 = "GETPART2"

TPHS = bits 7-1 of TPHS

$GOLOC_{dp}$ = $PHSNAMEi_{TGRP}$

TS = $PHSPRDTi_{TGRP}$

If TS > 0:

Establish a job with a VAC area and with 2CADR starting address given by (GOLOC, GOLOC+1) and priority of TS

Proceed to address specified by GOLOC+2

If TS ≤ 0:

Establish a job without a VAC area, with 2CADR starting address given by (GOLOC, GOLOC+1), and priority of (- TS)

Proceed to address specified by GOLOC+2

$GOLOC_{dp}$ = $PHSNAMEi_{TGRP}$

TS = (bits 3-1 of TPHS) - 2

If TS > 0:

TS = - $PHSPRDTi_{TGRP}$

Proceed to "ITSLGCL1"

If TS = 0:

TS = $PHSPRDTi_{TGRP}$

If TS > 0:

Establish a job with a VAC area and with 2CADR starting address given by (GOLOC, GOLOC+1) and priority of TS

Proceed to address specified by GOLOC+2

If TS $\leq$ 0:

Establish a job without a VAC area, with 2CADR starting address given by (GOLOC, GOLOC+1), and priority of (- TS)

Proceed to address specified by GOLOC+2

TS = $PHSPRDTi_{TGRP}$

Proceed to "TIMETEST"

VIID Standard Program Subroutines

There are a number of standard program subroutines, not discussed elsewhere in this document, which may be encountered during the course of a review of a symbolic listing. Those which perform a standardized function (such as facilitating communication between memory banks) are listed below, together with a brief explanation of their functions. For other subroutines, and for more details on some of the ones listed below, reference should be made to appropriate mission-oriented program information.

Tag	Function
ALARM	Entered if a "minor" (recoverable, in general) problem is encountered by the software, such as improper input data. Causes Program Alarm light (see Section IIJ) to be energized and the storage of alarm pattern (contained generally in cell following TC "ALARM"); in some cases (depending on the mission software design) an automatic display is subsequently initiated.
ALARM1	LM-only routine, entrance to "ALARM" allowing address information on what triggered the alarm (as opposed to why the alarm routine was entered) to be retained.
BAILOUT	Entered for a software restart due to a software problem that is expected to be recoverable, such as overflow of job register sets. Causes an effect similar to a hardware restart (see mission documentation for details).
BAILOUT1	LM-only routine, entrance to "BAILOUT" allowing address information on what triggered the software restart to be retained.
BANKCALL	Entered with FCADR of desired destination in calling address +1, and with (A, L) used as a communication cell (if desired) with routine whose address is the FCADR. Overflow bit of A would be lost, and subroutine does not change SUPERBNK, nor should it be used with a task. The routine which is entered via

Tag	Function
BANKCALL (cont)	"BANKCALL" can do a TC "SWRETURN" or TC "Q" (if no TC orders in routine itself) to get back to originating program. Only one "BANKCALL" or "SWCALL" can be used at a given time if return address information must be saved.
BANKJUMP	Entered with FCADR of destination in A. No return address information is retained, nor is SUPERBNK changed.
CCSHOLE	A special entrance to the "POODOO" routine when no alarm pattern is specified: it transfers to that subroutine with pattern 21103_8. Tag usually used for "illegal" CCS transfer points (see Section IVB) or other steps which should not be entered.
CHECKMM	Entered with a value of MODREG (mode or program register number) in calling address +1. Returns to calling address +2 if present MODREG value not that specified, and to calling address +3 if present value same as that specified.
CURTAINS	Entered if a hardware interface difficulty (such as attempting to pulse torque IMU gyros with IMU in coarse align) encountered. Generates a special alarm code pattern and returns (on unmanned flights such as LM-1, causes computations to be idled instead).
DELAYJOB	Entered by a job with A set to desired delay (B14 centiseconds) before job resumed. Puts job to sleep and sets up a waitlist call to awaken it after suitable interval. Up to four (CM) or three (LM) jobs can be accommodated at one time: if more than the limit are attempted, a BAILOUT software restart (pattern 31104_8) is generated. Due to mechanization, cannot be used for jobs running in erasable memory.
DMPNSUB	Entered with $MPAC_{dp}$ a fraction (scale factor BO) and A an integer (scale factor B14). Returns to calling address +1 with $MPAC_{dp}$ = (A, L) = product of $MPAC_{dp}$ and A, scale factor BO (i.e. with contents of $MPAC_{dp}$ normally increased, such as moved left 2 places if A was 00004_8). Overflow information is lost.

Tag	Function
DOWNFLAG	Routine in fixed-fixed memory that sets the flag bit given in calling address +1 to zero and then returns (after releasing interrupts) to calling address +2. Flag identification given symbolically, with bit 15 of STATE (or FLAGWRD0) equivalent to 0, bit 15 of FLAGWRD1 equivalent to 15, etc., so routine must divide by 15 to determine the word (with remainder giving the bit within word). Note that this differs from the interpretive language scheme for identifying bits (page VIB-50).
DPAGREE	Routine entered to force double precision sign agreement (MPAC+2 is set zero) for MPAC.
GENTRAN	Routine entered to cause a transfer of consecutive cells (which must be in the same fixed or erasable memory bank) from one part of memory to the other. Transfer done with interrupts inhibited (and they are not released before return). Accumulator set when enter with one less than the number of single precision cells to be moved; calling address +1 has ADRES of first cell of "source" data; calling address +2 has ADRES of first cell of "destination" for the data; and return is to calling address +3.
IBNKCALL	Interrupt analog of "BANKCALL", which can be used during an interrupt or with interrupts inhibited. Same format and restrictions (on "IBNKCALL" and "ISWCALL") as indicated for "BANKCALL". Return is via "ISWRETRN" or Q.
ISWCALL	Interrupt analog of "SWCALL", which can be used during an interrupt or with interrupts inhibited. Same format and restrictions (on "IBNKCALL" and "ISWCALL") as indicated for "SWCALL". Return is via "ISWRETRN" or Q.
ISWRETRN	Exit designation from a routine entered via "IBNKCALL" or "ISWCALL", causing return to original program with (A, L) preserved from the routine.
LOADTIME	Routine used by an RTB order (in interpretive language) to load the computer clock (TIME2, TIME1) into $MPAC_{dp}$.

Tag	Function
MAKECADR	Entered to form an address in FCADR format (and leave it in A) of return address information to be used by "SWRETURN": it is formed from S-register information in BUF2 and FBANK information in BUF2+1.
NORMUNIT	Routine used by an RTB order (in interpretive language) to shift all components of a vector left 13 places if the most significant halves of all components (after forcing sign agreement) are zero. This helps avoid numerical problems with the interpretive language UNIT order (which is then entered).
NORMUNX1	Special entrance to NORMUNIT that leaves interpretive index register X1 with the number of shifts performed (zero or +13).
POLY	Entered to evaluate a polynomial of the form $A_n X^n + A_{n-1} X^{n-1} + \ldots + A_0$, and returns with answer in $MPAC_{dp}$. When enter subroutine, $MPAC_{dp} = X$ (argument); calling address +1 = (n - 1); calling address $+2_{dp} = A_0$; calling address $+4_{dp} = A_1$; etc. Return is to calling address +(2 n + 4): n, of course, is the order of the polynomial.
POODOO	Entered for a software restart due to a software problem that is not expected to be recoverable, such as an attempt to take the square root of a negative number. If powered-flight computations not being done, all restart groups are set zero (see mission documentation for details). Cf. "BAILOUT" description.
POODOO1	LM-only routine, entrance to "POODOO" allowing address information on what triggered the software restart to be retained.
POSTJUMP	Entered with FCADR of desired destination in calling address +1, and with (A, L) used as a communication cell with routine whose address is the FCADR (if desired). No return address information is retained (in contrast to "BANKCALL").

Tag	Function
POWRSERS	Entered to evaluate a polynomial of the same form as for "POLY", and returns to calling address +1 with answer in $MPAC_{dp}$. When enter subroutine, $MPAC_{dp}$ = X (argument); A = "A_n" - 3; and L = (n - 1). Coefficients stored in increasing order (A_0 first, then A_1, etc.) as for "POLY", with the address of the coefficient of X^n, decremented by 3 (for coding convenience), in accumulator when enter "POWRSERS".
SGNAGREE	Routine used by an RTB order to force sign agreement of $MPAC_{tp}$. It enters "TPAGREE" and then transfers to "DANZIG".
SHORTMP	Entered with $MPAC_{tp}$ a fraction (scale factor BO) and A a single precision fraction (likewise scale factor BO). Return is to calling address +1 with product of $MPAC_{tp}$ and A in $MPAC_{tp}$ (and with (A, L) = 0). Differs from "DMPNSUB" in that MPAC contents would be moved right here, and are moved left there.
SIGNMPAC	Routine entered (generally by an RTB order) to set $MPAC_{dp}$ equal to +MAX (for original MPAC+0 $\geq$ +0) or -MAX (for the original MPAC+0 $\leq$ -0). MPAC+2 set zero and return is through "DANZIG".
SPCOS	Entered with A set to desired cosine argument, scale factor B-1 in units of revolutions; returns to calling address +1 with A equal to cosine of angle, scale factor BO, in range $\pm$ 1. Incoming argument is incremented by 90° (with suitable limiting) and the "SPSIN" expansion then used.
SPSIN	Entered with A set to desired sine argument, scale factor B-1 in units of revolutions; returns to calling address +1 with A equal to sine of angle, scale factor BO, in range $\pm$ 1. After reducing angle to range $\pm$ 90°, the following expansion is used (coefficients provided for X in units of radians): $ANS = 0.999925X - 0.165951X^3 + 0.007608X^5$

Tag	Function
SUPDACAL	Entered with FCADR of desired address in A, and with required SUPERBNK bits in bits 7-5 of L. Returns to calling address +1 with A set to contents of cell whose address information was in (A, L).
SUPDXCHZ	Entered with (A, L) set to the 2CADR of destination address. Loads SUPERBNK with bits 7-5 of L, then loads BBANK with L and does a TC to address that is in A. Differs from the pseudo-operation DTCB (see Section VA) in that SUPERBNK set.
SUPERSW	Entered with desired setting for SUPERBNK in bits 7-5 of A. Loads SUPERBNK and returns.
SWCALL	Entered with FCADR of desired destination in A, but otherwise the same as "BANKCALL" (including same return to "SWRETURN").
SWRETURN	Exit designation from routine entered via "BANKCALL" or "SWCALL". Causes return to original program with (A, L) preserved from the routine.
TPAGREE	Entered to force sign agreement for $MPAC_{tp}$, with $MPAC_{tp}$ left at +0 if was initially $\pm$ 0. Returns to calling address +1 with A set to +1, +0, or -1 (scale factor B14) for $MPAC_{tp}$ positive, zero, or negative respectively.
TPMODE	Routine entered by an RTB order to set interpretive language MODE cell to indicate triple precision (i.e. to +1) to cause subsequent storage order, for example, to be triple precision.
UPFLAG	Routine in fixed-fixed memory that sets the flag bit given in calling address +1 to one and then returns (after releasing interrupts) to calling address +2. See "DOWNFLAG".

APPENDIX A

Review of Computer Concepts

Familiarity with some of the general principles of digital computer design is necessary in order to permit a meaningful analysis to be conducted of the performance of a program. There is currently available a large variety of textbooks that review these principles in considerable detail, and in addition several "Study Guides" have been published to provide tutorial information directly applicable to the guidance computer. This appendix is not intended to perform the function served by those books, but instead to summarize briefly some of the more salient computer principles of value in reviewing the program listing.

Number Systems

The traditional decimal number system, in which each digit has a weight (going from right to left) of ten times the weight of its neighbor, is usually unsuited from hardware considerations for direct implementation in a digital computer hardware design. Instead, a binary number system is employed, in which each digit has a weight (again going from right to left) of two times that of its neighbor. This has an advantage over a decimal machine in that the value of a binary digit (or "bit") can be represented by only two states of some physical hardware device (such as a relay) or the performance of some piece of hardware (such as the switching of a ferrite core when subjected to a certain sequence of driving currents, but not to another sequence). Rather than create new symbols for the two possible values of a bit, they are designated "zero" and "one".

Although occasionally useful when the condition of distinct quantities

(such as program logic control flags) is of interest, reliable manual handling of quantities expressed in binary, if they involve more than a few bits, is usually quite difficult. Consequently, bits are conventionally grouped in sets of three to form "octal" numbers, which, as the name suggests, have a weight (from right to left) of eight times the weight of their neighbors. The grouping is generally started from the least significant end, although for the guidance computer, since it has an information word length of 15 bits, grouping from the most significant end (with the sign taken as the first bit) would give the same result.

Conversion between decimal and octal numbers can be done most conveniently if the octal number involved is defined to be an integer: any necessary conversion to other than integral values can be done by suitable manipulations upon the decimal number. The octal quantity should also be positive (sign representations will be discussed later). If the octal number is positive and an integer, the conversion of a decimal number DEC to an octal number involving 14 magnitude bits (i.e. the word length of the guidance computer) permits a maximum value of DEC of 16383. Representing the octal number as $(a_4\ a_3\ a_2\ a_1\ a_0)$, the conversion (in either direction) between octal and decimal requires solution of the following equation:

$$\begin{aligned} DEC &= 8^4 a_4 + 8^3 a_3 + 8^2 a_2 + 8^1 a_1 + 8^0 a_0 \\ &= 4096\, a_4 + 512\, a_3 + 64\, a_2 + 8\, a_1 + a_0 \end{aligned}$$

If the octal number is known (a_4 has a maximum value of 3 to satisfy the limit given for DEC), then the conversion formula is obvious; if the decimal number is known, successive division of DEC by 8 will yield

remainders which will be the values of a_0, a_1, etc.

Tables may be used to advantage in the conversion between octal and decimal, and an abbreviated one is presented on the following page. To use this table, add the value found for the weight of a_4 (below) to the weight given by the table for $(a_3\ a_2)$, and in turn add this to the value of $8\ a_1 + a_0$.

a_4	Weight
0	00000
1	04096
2	08192
3	12288
4-7	illegal (gives DEC exceeding 16383)

Also given on the next page is a table of the powers of two: it is convenient to remember that 2^{10} is about 10^3 and 2^{20} is about 10^6. For completeness, the following table of octal-to-binary conversion is supplied:

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

A computer must, of course, be able to represent numbers which are negative. A separate bit (the "sign bit") is used to retain the sign information, and conventionally it is zero for a positive number and one for a negative number. Although the sign bit is used to distinguish between positive and negative numbers, it is convenient from a hardware standpoint to modify the magnitude bits of the number when it is negative, rather than using a "sign-plus-magnitude" representation.

Octal-Decimal Conversion Table

a_3a_2	Weight	a_3a_2	Weight	a_3a_2	Weight	a_3a_2	Weight
00	0000	20	1024	40	2048	60	3072
01	0064	21	1088	41	2112	61	3136
02	0128	22	1152	42	2176	62	3200
03	0192	23	1216	43	2240	63	3264
04	0256	24	1280	44	2304	64	3328
05	0320	25	1344	45	2368	65	3392
06	0384	26	1408	46	2432	66	3456
07	0448	27	1472	47	2496	67	3520
10	0512	30	1536	50	2560	70	3584
11	0576	31	1600	51	2624	71	3648
12	0640	32	1664	52	2688	72	3712
13	0704	33	1728	53	2752	73	3776
14	0768	34	1792	54	2816	74	3840
15	0832	35	1856	55	2880	75	3904
16	0896	36	1920	56	2944	76	3968
17	0960	37	1984	57	3008	77	4032

Powers of Two

Exponent	Value of 2^{Exponent}
0,1,2,3,4,5	1,2,4,8,16,32
6	64
7	128
8	256
9	512
10	1 024
11	2 048
12	4 096
13	8 192
14	16 384
15	32 768
16	65 536
17	131 072
18	262 144
19	524 288
20	1 048 576
21	2 097 152
22	4 194 304
23	8 388 608
24	16 777 216
25	33 554 432
26	67 108 864
27	134 217 728
28	268 435 456

The formulation of negative numbers used in most computations in the guidance computer follows what is called a "ones complement" representation. In this representation, negative numbers of a given magnitude are found by subtracting each octal digit of the positive value from seven (which, of course, is equivalent to complementing each bit). Hence if it is desired to complement a complete number, this may be done by inverting each bit, a function conveniently realized in hardware by reading out the other side of a flip-flop (i.e. the side giving the complement, rather than true, value of the input number).

As suggested by the technique for converting to ones complement (subtracting each digit from seven), the ones complement representation has the feature that two possible representations of zero exist: 00000_8 and 77777_8. The first is conventionally called "plus zero", while the second is "minus zero" (see Section IIG). The interpretive language does not distinguish between them, nor does the "Branch on Zero to Fixed" (BZF) machine language instruction. The "Count, Compare, and Skip" (CCS, Section IVB) order, however, does distinguish between +0 and -0, and therefore it is important to understand the distinction between the two zeros. Most orders giving a zero result from non-zero operands will produce a -0, but there are several exceptions (see Section IIG and Section IV), including the decrementing property of the CCS order itself.

Since ones complement has two representations of zero, this means that the 15-bit computer word length can only express $(2^{15} - 1)$ different numbers. This distinction (32767 instead of 32768 numbers) turns out to be significant when hardware representations of angle data are considered, such as the CDU angles in the guidance system, and therefore another formulation of negative numbers, called "twos complement", is employed

for angle information. To convert a negative ones complement number to twos complement, merely add 1 to the least significant octal digit and allow the carries to propagate (hence the ones complement -0, 77777_8, becomes 00000_8). Positive numbers, of course, are the same in either system. For example, the positive representation of decimal 26 would be 32_8; the ones complement representation (for -26, assuming six bits for sign and magnitude) would be 45_8 ($45_8 + 32_8 = 77_8$); and the twos complement representation of -26 would be 46_8.

Another way of thinking of the twos complement numbers used in the computer is as 15-bit signless integers. With this viewpoint, angles are always positive, and for a least increment of 2^{-15} revolution may be summarized as follows:

0°	00000_8
45°	10000_8
90°	20000_8
180°	40000_8
270°(-90°)	60000_8
315°(-45°)	70000_8
360° = 0°	00000_8

A special computer instruction (MSU, see Section IIG and Section IVC) is available to form the ones complement difference of two twos complement numbers (if the second operand is zero, this is a convenient technique for converting a twos complement number to ones complement).

As mentioned previously, the sign bit in the guidance computer is considered to be to the left of the most significant magnitude bit: consequently, values of a_4 of 4-7 would mean the number is negative, and should be complemented before attempting to use the tables.

Arithmetic and Overflow

The guidance computer performs its arithmetic (multiplication and division, for example) as if it was a fractional machine, with the "binary point" between bit 15 (the sign) and bit 14 (the most significant magnitude bit). If the number were 20000_8 (bit 14 a 1 and the rest zero), the magnitude would be $\frac{1}{2}$ in this fractional machine representation (the "true value" of the number, however, would be determined from its scale factor, as discussed later). Addition is performed in the standard ones complement manner, with an "end-around carry" generated from the sign position (see also Section IIG). For example:

```
 00015    (decimal 13)
+00021    (decimal 17)
 00036    (decimal 30)

 00015    (decimal 13)
+77756    (decimal -17, ones complement)
 77773    (decimal -4)  Note that 6 + 5 = 11(10) = 13(8)

 77762    (decimal -13)
+00021    (decimal 17)
(1)00003
     1    (end-around carry)
 00004

 77762    (decimal -13)
+77756    (decimal -17)
(1)77740
     1    (end-around carry)
 77741    (decimal -30)

 00015    (decimal 13)
+77762    (decimal -13)
 77777    (-0)

 77777    (-0)
+00015    (decimal 13)
(1)00014
     1    (end-around carry)
 00015    (decimal 13)
```

To accomplish twos complement arithmetic, the end-around carry is suppressed, so that $77777_8 + 00001_8 = 00000_8$ (and the other relationships are also proper). Although complementing in the twos complement system is more difficult, arithmetic operations are easier, since no end-around carry is required (a useful feature for serial machines, which operate serially upon the bits of a word, instead of upon all the bits at once as in a parallel machine such as the guidance computer).

Overflow takes place when the result of an operation is too big for the word length. This fact is detected in the guidance computer by having two "signs" when a word is processed in the arithmetic unit (see Section IIG). The two sign bits are normally equal, and the most significant sign position is generally the one that is retained (unless twos complement computations are desired).

With this additional sign bit, computations under non-overflow conditions, of course, proceed as described previously:

```
  1 77762
 +0 00021
(1)0 00003
        1
 0 00004
```

If overflow takes place, however, the effect is different:

```
 0 20000   (08192)
+0 30000   (12288)
 0 50000   ("20480")
```

Here the two sign bits are unequal, indicating overflow. The most significant sign bit plus bits 14-1 give 10000_8 (4096), the correct answer if the overflow information ($2^{14} = 16384$: note that 16384 + 4096 = 20480) can be propagated to a more significant part of the word. Such propagation is performed automatically by the computer hardware (the DAS order,

Section IVB) for double precision computations, and can also be programmed with single precision operands through the special features of the TS order, Section IVB. If overflow is not encountered, however, the two halves of a double precision word are treated independently (hence, obviously, they have independent signs which need not be the same). This setting of overflow information takes place only on addition and subtraction (which is merely the addition of the complement of the word): overflow in division results in an improper answer.

Orders and Addresses

In order to be usable for the solution of problems, a digital computer must have the ability to perform a certain fundamental set of functions, through suitable combinations of the instructions that make up its "order code" (the built-in hardware operations performed in direct response to program steps read from memory). More "efficient" coding (in terms of execution time or memory storage requirements) can be performed if the order code permits a variety of operations to be performed by separate instructions, however. The Block 1 guidance computer had 11 machine language orders while the Block 2 computer has about three times that number, yet both machines, ignoring execution time and memory constraints, are capable of performing the same calculations. The interpretive language (Section VI) provides a good illustration of the "new" functions which may be performed (over 100 orders in the interpretive language are available) using the machine language orders which are provided by the computer hardware.

Most order code operations require the specification not only of the order ("do what"), but also the operand involved ("to what"). The

information concerning the operand is provided by the "address" part of the instruction, which may specify the cell location of a variable or the program step to which a transfer is required. In the guidance computer, the operation "Clear and Add" is assigned the operation code of 3. Hence the instruction

3 1234

would be interpreted by the hardware as "Clear and Add" (the 3) the quantity located in address 1234_8. The instruction

6 1245

(6 is Add) would add the contents of cell 1245_8 to the contents of the accumulator. It should be evident that the quantities 1234_8 and 1245_8 themselves do not enter the accumulator: instead, the quantities located at these addresses are used.

Scaling

As mentioned in the discussion on arithmetic, the guidance computer performs its arithmetic as if it was a fractional machine, meaning that all numbers in the machine are fractions lying in the interval between -1 and +1, so located in a single precision word that $\frac{1}{2}$ is 20000_8. The luxury of built-in floating point is not available in the guidance computer, although a few computations essentially are performed in this fashion in the interpretive language (by suitable programming). Since built-in floating point is not available, the responsibility is upon the program (and therefore programmer) to be aware of the maximum value of a quantity (to ensure that the computer number does not exceed 1) and its least significance.

One technique for avoiding overflow (computer number exceeding 1)

is to assign suitable units to a quantity so that its range is the same as that of the numbers in the machine (i.e. -1 to +1). For example, angles whose trigonometic functions are of interest might be expressed in units of "revolutions" (360°), so that $\frac{1}{2}$ would be 180° and - $\frac{1}{4}$ would be - 90°. This technique is employed for angle information in the interpretive language (see ACOS, ASIN, COS, and SIN in Section VIB).

Although suitable definition of units to provide a quantity in the proper value range (-1 to +1) in principle could be applied to all variables in the program, in practice this would rapidly become unwieldy and subject to errors. Consequently, it is conventional to allow quantities to keep their "normal" units and dynamic ranges when their computations in the program are being analyzed, and to express the conversion between these "normal" units and the computer number (in the range -1 to +1) by associating with each quantity a "scale factor". The scale factor for a quantity is defined as follows:

> The scale factor is the power of two by which the number in the computer must be multiplied to obtain its true value. The scale factor is frequently shown as "Bxx", to signify binary as opposed to decimal exponent information.

Since "shifting" operations (moving bits right or left in a register) can be performed conveniently, the advantages of selecting the scale factor as a power of two are evident: if a number is moved two places to the right in a register, its scale factor is increased by two.

From the definition of "scale factor" given above, it should be evident that angle information (expressed in units of revolutions, as discussed above) would have a scale factor B0, since the computer number would have to be multiplied by 2^0, or 1, to find the "true value". There is frequent need in the program to consider quantities to be integers,

giving the least significant bit of the accumulator a weight of 1 (such as for accelerometer counter outputs, in units of counts). Since the computer number weight for this bit is 2^{-14}, this means that the computer number would have to be multiplied by 2^{14} to convert from the computer value to the true value, and therefore the scale factor for such integers is B14. Note that the table conversion process given earlier would provide directly the decimal value of a number with scale factor B14: in general, however, the integer which results from the table process would have to be divided by 2^{14} - scale factor to find the true value.

Double precision numbers, especially in interpreter (Section VI) calculations, can be handled in a manner identical to that for single precision, since the weights for the most significant half are those for a single precision number, and the weights for the least significant half are 2^{-14} times the corresponding bit positions of a single precision number. A double precision integer, therefore, would have a scale factor of B28, and could represent magnitudes up to $(2^{28} - 1)$, or 268,435,455. The fact that the scale factor is (almost) the maximum possible value of the number follows from its definition (computer numbers are all fractions), and is frequently useful in reviewing the performance of a program.

Conversion of a double precision octal number to decimal can proceed as if the two halves are independent single precision numbers, which indeed is generally the case. After conversion to an integer, the most significant half should be multiplied by 16384 (2^{14}) and added to the least significant half (which may not be of the same sign), and the result then divided by 2^{28} - scale factor to find the true value. It is sometimes useful to use the fact that the least significant bit of a single precision number has a weight of $2^{\text{scale factor} - 14}$; for double precision it is $2^{\text{s.f.} - 28}$.

Computations involving numbers with various scale factors proceed within the computer arithmetic unit without any knowledge of what these scale factors are (since no built-in floating point is available), and therefore it is necessary to write (or analyze) the program in order to determine the manipulations which are being performed. Since scale factors are merely exponents, they follow the normal rules for the handling of exponents by arithmetic operations:

1. Addition and Subtraction leave unchanged.
2. Multiplication adds exponents and scale factors.
3. Division subtracts exponents and scale factors.
4. Squaring doubles exponents and scale factors.
5. Extracting square root halves exponents and scale factors.
6. Shifting left n places is the same as multiplying by 2^{-n} (in the sense that it is the same basic value of the number that must remain) and therefore reduces the scale factor by n.
7. Shifting right n places increments the scale factor by n.

Assume B has a scale factor B14 and K has scale factor B-10:

C = K B leaves C with a scale factor B4

$D = C^2$ leaves D with a scale factor B8

E = B, shifted left 5 places, leaves E with scale factor B9

F = (D + E) cannot immediately be analyzed. Gives effect of F = 2 D + E, scale factor B9, or F = D + $\frac{1}{2}$ E, scale factor B8 (or other permutations, of course). Decision as to which is performed might be resolved by G&N contractor equation documentation, if available.

$G = \sqrt{F}/K$ suggests strongly that F has an even scale factor(e.g. B8), and gives a scale factor for G of 4 - (-10) = B14. If other evidence (such as DSKY display) were available to suggest that G was an integer (single precision), confidence in the validity of the analysis would be enhanced.

To assist in the analysis of the scaling of a program, there is almost nothing which is more valuable than a reasonable valid definition of what

the program "should" be doing. Such information might be provided by G&N contractor equation documentation or flow charts, mission-oriented program documentation, information in the program comments field (not necessarily consistent with the coding), etc. If the program version has been released for fabrication into memory ropes, it is reasonable to assume that the coding performs a valid function. A knowledge of some of the "standard" scale factors, particularly for the interpretive language, can also prove useful. Some of these scale factors, with the associated "standard" units, that may be encountered include the following:

1. Angles in the interpreter generally are in revolutions, scale factor B0. Incoming IMU CDU angles are in revolutions with scale factor B-1 (in twos complement). See Section IID for information on scaling of other input counters.

2. It is usually possible to consider inputs and outputs of the computer to be scaled as integers (B14 in units of "counts", for example).

3. Angular rate information in digital autopilots frequently represents an angle change per computing interval (such as 0.1 second). It will generally be found more convenient to retain the standard angle units of revolutions, with the time unit suitably defined, rather than generating a "new" unit (i.e. consider rates to be scaled B-3 in units of revolutions/deci-second instead of B0 in units of 450°/sec).

4. Position information can have various scale factors depending on dynamic range requirements. Units in most computations are meters (although early programs sometimes used kilometers). In orbital integration, position data scaling generally B29 meters for earth-centered and B27 meters for moon-centered computations. A standard value of B29 meters appears in most other places, except that B24 meters has been used in LM during ascent and descent computations.

5. Velocity information likewise can have various scale factors depending on dynamic range requirements. Units in most computations are meters/centi-second (although early programs in orbital integration used special units). In orbital integration, velocity data scaling generally B7 meters/centi-second for earth-centered and B5 meters/centi-second for moon-centered computations. A standard value of B7 meters/centi-second appears in most other places, except that parts of the LM descent coding use other scaling (such as B10 meters/centi-second).

6. Time information almost always is in units of centi-seconds, since this is the standard least increment of the computer clock (0.01 second). Single precision times, therefore, have scale factor B14 and double precision ones B28 (information to be loaded into TIME6, however, is scaled B10).

7. Trigonometric functions (sines and cosines) as used in the interpretive language have scale factor B1. The single-precision routines in the program, however, derive these functions with scale factor B0.

8. Unit vectors have a standard scale factor of B1 (note that a scale factor of B0 would risk overflow in forming a unit vector if two of its components were 0), although other scalings, particularly for erasable memory constants, may also occur.

Individual portions of the program, however, may have particular units for quantities that are not subject to the above generalizations. The entry guidance program, for example, conventionally scales acceleration B0 in units of 805 fps^2 (25 x 32.2 = 805) and velocity B1 in units of 25766.1973 fps (the circular satellite velocity 300,000' above the one-time standard earth radius of 2.09029E7 feet).

Conventional scaling equivalents that are used in the program include the following:

1 foot	= 0.3048 (exact) meters
1 nautical mile	= 1852 (exact) meters
1 pound	= 0.45359237 (exact) kilograms
1 newton-meter	= 1.355817948 foot-pound
1 kilogram-$meter^2$	= 1.355817948 $slug\text{-}foot^2$
Standard gravity	= 9.80665 (exact) $meters/second^2$ (earth)
Standard pad radius	= 6 373 338 meters
π	= 3.14159265

Software Difficulties

During the course of the development of flight software, several software difficulties have become evident as sources of coding deficiencies. The lists given below are intended to be summaries of such difficulties, rather than be summaries of the software features covered elsewhere in this document. Excluded from the lists are those which are in the category of design troubles (such as multiple loaders of an autopilot deadband), as well as those items which reflect logic troubles with software service routines.

General and Machine Language

1. Constants must be checked carefully. The assembler, rather than a desk calculator, should be use to incorporate binary scale factors. The conversion factors should be applied carefully (see page A-15): the value of π is not (22/7), the number of meters in a foot (12/39.37), nor the number of meters in a nautical mile 1853.25. The proper units on the constant must also be observed (revolutions rather than radians for most angles, for example).

2. Implementation of restart logic is a considerable source of potential difficulty. Considerable checking is required to insure that the logic that is supplied is proper. Checking should not be restricted to the existing phase-change entrances, but should also consider if other areas of the coding need protection that presently are lacking it.

3. Changes should be made carefully, and only after the existing mechanization is thoroughly understood. Restart logic, for example, has been seriously compromised due to ill-advised changes whose only justification may have been the saving of a few program steps.

4. Time sharing of erasable memory is necessary because of the limited size of the memory. A number of difficulties have arisen because of unexpected "sneak paths" through the coding that resulted in attempts to use some cell after it had been written over with another quantity. A complete understanding of how a cell is used within one routine should be obtained before an attempt is made to use it for another purpose (the impact upon the required telemetered information should also be considered).

5. Changes in basic coding design should be implemented only after a thorough review of all the consequences. The addition of a "software restart" capability, for example, led to a number of difficulties with the autopilots that had been avoided when a hardware restart was forced.

6. The use of relative addresses, while convenient during coding, has led to erroneous program transfers, especially if program changes are subsequently made.

7. There are several routines which inhibit and/or release interrupts, and hence caution must be observed if it is desired to protect a batch of coding from interrupts that some subroutine (such as a flag-bit changing routine) has not released interrupts.

8. Proper erasable memory bank settings for 2CADR addresses, such as those in restart tables, have been a source of difficulty. Pending an upgrading of the assembly program, it is necessary to check this information manually.

9. It must always be remembered that interpretive language jobs can be interrupted for a separate job of higher priority, and all jobs can be interrupted for tasks. Consequently, any use of an erasable memory cell between jobs, or between a task and a job, must be done with extreme caution. Difficulties may only arise for a particular phasing of input information (such as crew DSKY manipulations), and hence methodical validation of the coding is a formidable effort.

10. Transitions in operating configuration (such as between one autopilot and another) must be handled carefully, both for the normal case and in the presence of restarts, in order to avoid undesirable interactions.

11. Residual flagword bit settings have been a source of difficulty in program operation, particularly in the presence of a restart or a non-nominal crew exit from a program. The obvious solution, of course, is a methodical reset of the bits in a master initialization routine that is performed prior to entry to a new program.

12. In a program loop that can last for 5 ms or more, it is necessary to ensure the presence of a TC or TCF order in order to avoid a hardware restart, even though the order itself may not be functional (and loop control exercised by a BZMF instruction, for example).

13. Symbolic references to flagword bits can be convenient, but checks must be made that the bit does not correspond to the sign bit of the word (bit 15) before determining the appropriate branching order (such as CCS) that should be used.

14. The possibility of hardware failures, and their effects on the input bits read from channels, should be considered in the coding (as well as being documented in precise detail in the software documentation, of course).

15. Residual SUPERBNK settings when using fixed memory banks 30-43 should be checked (for example, entrance to an alarm routine from banks 40-43 should not be followed by performance of a routine in banks 30-37 without ensuring that SUPERBNK no longer set to 4).

16. Interrupts are inhibited by hardware means only in the case of overflow in the accumulator. If coding is done to cause overflow in other registers (such as Q), then this coding must ensure that interrupts are inhibited appropriately.

17. The executive system assumes that the job reflected in Job Register Set zero is the one currently active. Consequently, after a software/hardware restart (that initializes all Job Register Set cells to indicate that they are available), establishment of a single job followed by an entrance to a display interface routine will cause that job to be lost (since the job that is put to sleep is not the "current" one, but instead the one that was established).

18. During a periodic display generation cycle (such as when the accelerometers are being read for guidance), a DSKY display response which is intended to generate another display may be ineffective (since the periodic display would occur again and destroy the single-shot display).

19. The logic of the display interface routines gives a quite powerful display capability, but its restrictions and features must be well understood. Displays of different display priorities cannot be intermixed, and the fact that a response was a "data enter" cannot be determined from the return address to which the interface routine transfers (due to interruption by a higher priority display or by a restart).

Interpretive Language

1. Since NEWJOB is checked within the interpreter, special techniques (such as a priority change or particular coding) must be used if desire to sample a set of cells which may be modified by a higher priority job (such as DSKY activity or periodic powered flight navigation).

2. The UNIT operation can give numerical difficulties for vectors with several leading zeros in each component (the magnitude of the vector for division is only computed in triple precision), causing the resulting vector magnitude to exceed 1.0000. One solution is to execute the UNIT instruction twice. If there are too many leading zeros, of course, the vector cannot be formed, and the result (scaled B1) is approximately (2, 0, 0).

3. In using the push-down list within a coding loop, care must be taken that all paths through the loop give a consistent setting when the loop is entered again (or else a SETPD order must be used). This includes such special exits as those due to overflow, for example.

4. Triple precision computations can be done (including the TAD order) without setting the MODE cell (which controls the subsequent storage commands) correspondingly.

5. Although it is possible to load MPAC cells in machine language as well as interpretive language, care must be used in combining the two types of coding to ensure that MODE and the low-order parts of MPAC contain the information which is desired.

6. If it is desired to read counter cells (or load them), it is necessary to return to machine language. Attempts to reference these cells in the interpretive language generally will yield a VAC area reference instead.

7. Overflow due to a left shift is a "modulo" operation, and hence checks against a limit should be done before, not after, such a shift.

8. The SIGN operation treats zero as a positive number. Hence if shift a negative quantity that is a power of two such as to cause overflow (losing the bit), the SIGN operation cannot be used on what remains in order to give the proper result.

9. Although some computations have different scaling for earth-centered and moon-centered computations, others require a consistent scaling regardless of the coordinate origin.

10. In generating a program patch, a CALL/RVQ sequence, even though only one entrance to the patch is made, can save a memory cell over a GOTO/GOTO sequence.

APPENDIX B

Changes Made for Revision 2

This appendix gives an abbreviated comparison between this revision and Revision 1 of this document. It is intended that most of the changes made for this revision be included in the list, rather than the additions or deletions that have occurred.

Hardware

The hardware design of the Block 2 computer, of course, was completed several years ago. Nevertheless, the following modifications are reflected in the earlier sections of this revision:

Section IID

1. Delay information for radar inputs (0046_8).
2. Scale ratio between low and high landing radar scale (0046_8).
3. Scaling for throttle output (0055_8).

Section IIE

1. Channel 07 exempt from hardware restart.
2. Channel bits reset during a loading command.
3. Channel 10 loading period increased.
4. Channel 31 bits 14-13 apply to PGNS Mode Control switch, for LM.
5. Channel 33 flip-flops reset by a hardware restart.
6. Channel 33 bit 9 landing radar scale ratio.
7. Channel 33 bits 5-4 (CM) reflect separate spacecraft switches.
8. Channel 33 bit 2 (LM) indicates that RR CDUs have supply voltage phased suitably with LGC voltage.

Section IIH

1. Low bit rate telemetry in LM does not transmit LGC digital data.
2. Delay interval before radar interrupt changed.
3. No interrupts (counter/program) before special-purpose TC orders.

Section IIJ

1. Channel 10 loading period increased.
2. ALT and VEL lights added to DSKY for LM.

Software

The support software has changed in only minor ways (the next section reviews the interpretive language).

Section IID

1. The OPTi tags are no longer used in LM for RR CDUs.
2. FDAI error needles in LM can display vehicle rates.
3. Cells 0057_8-0060_8 used in CM for alarm source information.

Section IIE

1. Software uses different axes in LM for RCS control.
2. Uplink activity light used in CM for attitude maneuver message.
3. Channel 12 bit 15 delay now 10.24 seconds.
4. Channel 12 bit 11 set before TVC in CM.
5. Channel 12 bit 1 set 1 if RR not in LGC mode.
6. RR/LR performance sensitive to channel 13 loads.
7. Software logic added to check validity of PRO key entry.

Section III

1. Line skipping descriptions added and made more explicit.
2. Additional options for listings of changes available.

Section VIIA

1. Software abort (pattern 21204_8) added for negative or zero waitlist calling argument.

2. Waitlist overflow alarm pattern and routine interface modified.

Section VIIB

1. LM has additional Job Register Set.

2. Executive area overflow alarm patterns and routine interface modified.

3. NEWJOB checked in additional locations.

Section VIIC

1. "NEWPHASE" routine restricted to CM only.

2. "PHSCHNGA" routine added to LM only.

Section VIID

1. The "ABORT" and "DATACALL" routines have been deleted from coding.

2. "DELAYJOB" can accommodate 4 jobs in CM, 3 in LM.

3. Several additional routines added.

Interpretive Language

1. Arc cosine/arc sine no longer causes software restart.

2. Scaling units for sine/cosine routine clarified.

3. Square root software restart alarm pattern and routine interface modified.

4. "USEPRET" routine deleted from coding.

APPENDIX C

Summary of Computer Inputs and Outputs

This Appendix summarizes the computer inputs and outputs by the following functional areas:

A. Computer Information
B. Crew Input Quantities
C. Display Quantities
D. External Input Signals
E. IMU Quantities
F. Optics Quantities
G. Radar Quantities
H. Telemetry Quantities
J. Uplink Quantities
K. Vehicle Control Quantities

More details on the particular quantities involved can be found in the sections on Special Erasable Cells (Section IID), Input/Output Channels (Section IIE), and Display System (Section IIJ).

A Computer Information

1. Time Information

a) Computer clock: cells 0024_8 - 0025_8.
b) Clock used to maintain account of time in standby: channels 03 and 04.
c) Clock for control of waitlist tasks: cell 0026_8, program interrupt #3.
d) Clock for control of periodic input/output: cell 0027_8, program interrupt #4.
e) Clock for digital autopilot computations: cell 0030_8, program interrupt #2.
f) Clock for control of RCS jet firings: cell 0031_8, program interrupt #1. Cell countdown enabled when bit 15 of channel 13 = 1.

2. Memory Control

a) Erasable memory bank: cell 0003_8.
b) Fixed memory bank: cell 0004_8.
c) Superbank control: channel 07.

d) Fixed and erasable bank set/sense: cell 0006_8.

e) Program counter: cell 0005_8.

f) Return address information: cell 0002_8.

g) Program loop check: cell 0067_8.

3. Accumulator and L register: cells 0000_8 and 0001_8.

4. Interrupt traps (program interrupt #10) reset: channel 13, bits 14-12.

5. Standby enable: channel 13, bit 11.

6. Alarm test: channel 13, bit 10.

7. Oscillator stop alarm: channel 33, bit 15.

8. Computer warning alarm: channel 33, bit 14.

9. Test connection output: channel 11, bit 9.

B Crew Input Quantities

1. DSKY Keyboard (program interrupts #5 and #6)

a) Main panel DSKY keyboard: channel 15, bits 5-1.

b) Navigation (LEB) panel DSKY keyboard, CM only: channel 16, bits 5-1.

c) Proceed key: channel 32, bit 14.

2. Rotational hand controller binary outputs: channel 31, bits 6-1.

3. Rotational hand controller analog outputs (LM only): cells 0042_8 - 0044_8. Input to cells enabled by channel 13, bit 8; digital-to-analog conversion initiated by channel 13, bit 9.

4. Translation hand controller binary outputs: channel 31, bits 12-7.

5. Minimum impulse controller binary outputs, CM only: channel 32, bits 6-1. LM uses B.2 with suitable software settings.

6. Rate of descent control, LM only: channel 16, bits 7-6. Release of switch to neutral position resets trap for interrupt #6.

7. Engine armed (ready for ignition): channel 30, bit 3.

8. Spacecraft control modes

a) Computer in control, CM only: channel 31, bit 15.

b) Computer in control, LM only: channel 30, bit 10.

c) Attitude controller out of detent, LM only: channel 31, bit 15.

d) Attitude hold mode: channel 31, bit 13.

e) Free mode, CM only: channel 31, bit 14.

f) Automatic mode, LM only: channel 31, bit 14.

9. Abort command, LM only: channel 30, bit 1.
10. Abort Stage command, LM only: channel 30, bit 4.
11. LM attached to CM, CM only: channel 32, bit 11.
12. Landing point designator, LM only: channel 31, bits 6,5,2,1.

C Display Quantities

1. DSKY panel display drive: channel 10, bits 15-1.
   a) Row 14_8 has following assignments:
      PROG light bit 9
      TRACKER light bit 8
      GIMBAL LOCK light bit 6
      ALT light (LM only) bit 5
      NO ATT light bit 4
      VEL light (LM only) bit 3
   b) LM mission programmer (unmanned flights): row 17_8.
2. Caution reset signal (for Restart light): channel 11, bit 10.
3. Operator error light: channel 11, bit 7.
4. Key Release light: channel 11, bit 5.
5. Verb/Noun Flash: channel 11, bit 6.
6. Temperature Caution light: channel 11, bit 4; channel 30, bit 15.
7. Computer Activity light: channel 11, bit 2.
8. ISS Warning light: channel 11, bit 1.
9. FDAI attitude error needles: cells 0050_8 - 0052_8, if channel 14 bits 15-13 and channel 12 bit 6 are set.
10. Special display of inertial data, LM only:
    a) Display request: channel 30, bit 6.
    b) Lateral and forward velocity: cells 0053_8 - 0054_8, if channel 14 bits 12-11 and channel 12 bits 8 and 2 set.
    c) Altitude and altitude rate: cell 0060_8 if channel 14 bit 3 is set. Channel 14 bit 2 is set if altitude rate; otherwise is altitude.

D External Input Signals (CM only)

1. Guidance Reference Release: channel 30, bit 6.
2. Liftoff: channel 30, bit 5.
3. S4B Separation: channel 30, bit 4.
4. Ullage (of S4B): channel 30, bit 1.
5. CM/SM Separation: channel 30, bit 2.

E IMU Quantities

1. CDU angles: cells 0032_8 - 0034_8.
2. Accelerometer outputs: cells 0037_8 - 0041_8.
3. Coarse Align
   a) Mode specified: channel 12, bit 4.
   b) Angle changes: Cells 0050_8 - 0052_8, if channel 14 bits 15-13 and channel 12 bit 6 set.
4. Turn-on request: channel 30, bit 14.
5. Turn-on delay (90 seconds) complete: channel 12, bit 15.
6. Gyro Torquing
   a) Power supply: channel 14, bit 6.
   b) Number of pulses: cell 0047_8.
   c) Sign and axis: channel 14, bits 9-7.
   d) Enable cell 0047_8 output: channel 14, bit 10.
7. Status Information
   a) Turned on and operating: channel 30, bit 9.
   b) Temperature in limits: channel 30, bit 15.
   c) IMU fail signal: channel 30, bit 13.
   d) IMU CDU fail signal: channel 30, bit 12.
   e) PIPA fail signal: channel 33, bit 13.
8. IMU cage command (crew switch): channel 30, bit 11.
9. Enable IMU error counters: channel 12, bit 6 (driven from cells 0050_8 - 0052_8).
10. Zero IMU CDU's: channel 12, bit 5.

F Optics Quantities

1. Optics for CM only
   a) Mark: channel 16, bit 6.
   b) Mark reject: channel 16, bit 7.
   c) Computer control of optics: channel 33, bit 5.
   d) Optics in zero optics mode: channel 33, bit 4.
   e) Command to zero optics: channel 12, bit 10.
   f) Disengage optics DAC: channel 12, bit 11.
   g) Enable optics error counters: channel 12, bit 2 (driven from cells 0053_8 - 0054_8).
   h) Zero optics CDU's: channel 12, bit 1.

i) Optics CDU failure: channel 30, bit 7.

j) Optics angle measurement (CDU's): cells 0035_8 - 0036_8.

k) Optics drive information: cells 0053_8 - 0054_8, if channel 14 bits 12-11 and channel 12 bit 2 set.

2. Optics for LM only

a) X-axis mark: channel 16, bit 3.

b) Y-axis mark: channel 16, bit 4.

c) Mark reject: channel 16, bit 5.

G Radar Quantities

1. Digital input data (except angles): cell 0046_8, with selection of source by channel 13, bits 3-1. Data gating initiated by channel 13, bit 4; elapse of time delay after setting bit 4 causes program interrupt #9.

2. VHF range data good, CM only: channel 33, bit 2.

3. Landing Radar information, LM only

a) Command antenna to position #2: channel 12, bit 13.

b) Antenna in position #1 (descent): channel 33, bit 6.

c) Antenna in position #2 (hover): channel 33, bit 7.

d) Tracker locked on: channel 33, bit 5.

e) Velocity information valid: channel 33, bit 8.

f) Altitude information on low scale: channel 33, bit 9.

4. Rendezvous Radar information, LM only

a) Power on and mode switch to computer: channel 33, bit 2.

b) Locked on: channel 33, bit 4.

c) Angle information (CDU's): cells 0035_8 - 0036_8.

d) Pointing commands for antenna: cells 0053_8 - 0054_8, if channel 14 bits 12-11 and channel 12 bit 2 set.

e) Range information on low scale: channel 33, bit 3.

f) Radar CDU failure: channel 30, bit 7.

g) Enable radar CDU error counters: channel 12, bit 2 (driven from cells 0053_8 - 0054_8).

h) Zero radar CDU's: channel 12, bit 1.

i) Enable radar lockon: channel 12, bit 14.

H Telemetry Quantities

1. Data output: channels 34-35 (load when program interrupt #8).
2. Word order code: channel 13, bit 7.
3. Downlink rate too high: channel 33, bit 12.

J Uplink Quantities

1. Data input: cell 0045_8 (overflow generates program interrupt #7).
2. Bit rate too high: channel 33, bit 11.
3. Spacecraft switches set to inhibit: channel 33, bit 10.
4. Uplink activity light: channel 11, bit 3.
5. Block inputs to cell 0045_8: channel 13, bit 6 (bit 5 for uplink only).

K Vehicle Control Quantities

1. RCS Jets
   a) Jet turn-on: channels 05 and 06 (bits 8-1).
   b) Jet failure, LM only: channel 32, bits 8-1.
2. Saturn control, CM only
   a) Control of Saturn to computer: channel 30, bit 10.
   b) Injection Sequence Start: channel 12, bit 13.
   c) S4B Cutoff: channel 12, bit 14.
   d) Steering commands to Saturn: channel 12, bit 9.
   e) Steering command data: cells 0050_8 - 0052_8, if channel 14 bits 15-13 and channel 12 bit 6 set.
3. SPS control, CM only
   a) Engine on: channel 11, bit 13.
   b) Steering command data: cells 0053_8 - 0054_8, if channel 14 bits 12-11 and channel 12 bit 2 set.
   c) Steering commands to SPS engine: channel 12, bit 8.
4. Stage Verify, LM only: channel 30, bit 2.
5. Engine on/off, LM only: channel 11 bits 13 and 14.
6. Descent engine throttle control, LM only
   a) Computer given control of throttle: channel 30, bit 5.
   b) Throttle command: cell 0055_8, gated if channel 14 bit 4 set.
7. Descent engine gimbal trim control, LM only
   a) Roll and pitch trim: channel 12, bits 12-11 and 10-9.
   b) Gimbal drive amplifiers turned off: channel 32, bit 9.
   c) Gimbal drive indicated malfunction: channel 32, bit 10.

APPENDIX D

ALPHABETICAL LISTINGS

Machine Language and
Other Assembler Codes

Symbol	See Page	Function
0-7	VA-3	Machine operation codes
iDNADR (i = 1-6)	VC-1	Telemetry downlist data
2BCADR	VC-1	Same as 2CADR
2CADR	VC-1	Double precision address, first half ADRES format and second half BBCON format
2DEC	VB-1	Double precision decimal number
2FCADR	VC-1	Double precision address for fixed memory bank loading
2OCT	VB-4	Double precision octal number
2OCTAL	VB-4	Same as 2OCT
=MINUS	VC-1	Address computation function
=PLUS	VC-2	Address computation function
AD	IVB-1	Add
ADRES	VC-2	Address information (usually 12-bit S-register)
ADS	IVB-1	Add and Store
AUG	IVC-1	Augment Magnitude
BANK	III-8	Set location counter in assembly program to first unassigned cell in indicated bank
BBCON	VC-2	BBANK setting corresponding to given symbol and preceding EBANK= specification, plus SUPERBNK information
BLOCK	III-8	Set location counter in assembly program to first unassigned cell in specified fixed-fixed bank
BNKSUM	IIF-5	Generate coding for terminating indicated fixed memory bank
BZF	IVC-1	Branch on Zero to Fixed
BZMF	IVC-1	Branch on Zero or Minus to Fixed
CA	IVB-1	Clear and Add
CADR	VC-3	Complete Address for fixed memory (FBANK and S-register information packed)
CAE	VA-1	Clear and Add from Erasable
CAF	VA-1	Clear and Add from Fixed
CARDNS	III-24	Modify card numbers
CCS	IVB-1	Count, Compare, and Skip
COM	VA-1	Complement

Symbol	See Page	Function
COUNT	III-9	Monitor use of fixed memory cells for tabulation
CS	IVB-2	Clear and Subtract
DAS	IVB-2	Double Precision Add and Store
DCA	IVC-2	Double Precision Clear and Add
DCOM	VA-1	Double Precision Complement
DCS	IVC-2	Double Precision Clear and Subtract
DDOUBL	VA-2	Double Precision Double (A, L)
DEC	VB-1	Single precision decimal number
DELETE	III-24	Delete indicated lines of coding
DIM	IVC-2	Diminish Magnitude
DINC	IIH-2	Decrement magnitude (counter interrupt)
DNCHAN	VC-3	Telemetry downlist channel
DNPTR	VC-3	Telemetry downlist sub-list pointer
DOUBLE	VA-2	Double Accumulator contents
DTCB	VA-2	Transfer to step given in 2CADR form in (A, L)
DTCF	VA-2	Transfer to step given in 2FCADR form in (A, L)
DV	IVC-2	Divide
DXCH	IVB-2	Double Precision Exchange
EBANK=	III-9	Set assembler cell to indicated EBANK value
ECADR	VC-3	Erasable Memory Complete Address
EDRUPT	IIH-4	Programmed RUPT
EQUALS (or =)	III-9	Give quantity in tag field same definition as quantity in address field
ERASE	III-10	Allocate erasable memory storage
EXTEND	VA-2	Permit an extended operation to be performed
FCADR	VC-3	Fixed memory complete address, same as CADR
FETCH	IIH-10	Computer test set instruction
GENADR	VC-4	S-register information, same as ADRES
INCR	IVB-3	Increment
INDEX	IVB-3 IVC-3	Index
INHINT	VA-2	Inhibit interrupts
INOTLD	IIH-10	Computer test set instruction
INOTRD	IIH-10	Computer test set instruction
INSERT	III-24	Insert lines of coding
LOC	III-10	Same as SETLOC

Symbol	See Page	Function
LXCH	IVB-3	Exchange L Register
MASK	IVB-3	Mask
MCDU	IIH-2	Decrement (counter interrupt) in twos complement
MEMORY	III-10	Allocate memory cells
MINC	IIH-2	Decrement (counter interrupt) in ones complement
MM	VB-3	Major Mode (program number)
MP	IVC-3	Multiply
MSK	VA-2	Mask
MSU	IVC-3	Modular Subtract (twos complement subtract)
NDX	VA-2	Index
NOOP	VA-2	No Operation
NV	VB-3	Verb-Noun information (same as VN)
OCT	VB-3	Single precision octal number
OCTAL	VB-3	Same as OCT
OVSK	VA-2	Overflow Skip
PCDU	IIH-2	Increment (counter interrupt) in twos complement
PINC	IIH-2	Increment (counter interrupt) in ones complement
QXCH	IVC-4	Exchange Q Register
RAND	IVC-4	Read Masked Channel
READ	IVC-4	Read Channel
RELINT	VA-3	Release interrupts
REMADR	VC-4	S-register contents, same as ADRES
RESUME	VA-3	Resume computations interrupted by program interrupt
RETURN	VA-3	Exit from subroutine
ROR	IVC-4	Read ORed Channel
RUPT	IIH-4	Implement in hardware the program interrupt
RXOR	IVC-4	Read Exclusive-ORed Channel
SBANK=	III-10	Set assembler cell to specified superbank
SETLOC	III-10	Set location counter in assembly program to indicated value
SHANC	IIH-2	Shift (counter interrupt) and add 1
SHINC	IIH-3	Shift (counter interrupt)

Symbol	See Page	Function
SQUARE	VA-3	Square contents of Accumulator
STORE	IIH-10	Computer test set instruction (same mnemonic also used for interpretive language order)
SU	IVC-5	Subtract
SUBRO	III-11	Include in assembly the Subroutine with the indicated identification
TC	IVB-4	Transfer Control
TCAA	VA-3	Transfer Control to Address in A
TCF	IVB-4	Transfer Control to Fixed Memory
TCR	VA-3	Transfer Control and save Return (same as TC)
TCSAJ	IIH-10	Computer test set instruction
TS	IVB-4	Transmit to Storage
VN	VB-3	Verb-Noun information
WAND	IVC-5	Write Masked Channel
WOR	IVC-5	Write ORed Channel
WRITE	IVC-5	Write Channel
XCH	IVB-4	Exchange
XLQ	VA-3	Execute instructions in L and Q
XXALQ	VA-3	Execute instructions in A, L, and Q
ZL	VA-3	Zero L register
ZQ	VA-3	Zero Q register

Alphabetical Listing of Interpretive Language Instructions

Symbol	Page VIB-	Function
ABS	3	Absolute Value of Scalar
ABVAL	15	Absolute Value (length) of Vector
ACOS	3	Arc Cosine
ARCCOS	3(ACOS)	Arc Cosine
ARCSIN	5(ASIN)	Arc Sine
ASIN	5	Arc Sine
AXC	46	Address to Index Complemented
AXT	46	Address to Index True
BDDV	5	Backwards Double Precision Divide
BDSU	8	Backwards Double Precision Subtract
BHIZ	41	Branch if High-order Zero
BMN	41	Branch if Minus
BOF	51(BOFF)	Transfer if Bit Is Off
BOFCLR	50	Clear Bit and Transfer if Bit Was Off
BOFF	51	Transfer if Bit Is Off
BOFINV	51	Invert Bit and Transfer if Bit Was Off
BOFSET	51	Set Bit and Transfer if Bit Was Off
BON	52	Transfer if Bit Is On
BONCLR	52	Clear Bit and Transfer if Bit Was On
BONINV	52	Invert Bit and Transfer if Bit Was On
BONSET	53	Set Bit and Transfer if Bit Was On
BOV	41	Branch on Overflow
BOVB	41	Branch on Overflow to Basic
BPL	42	Branch if Positive
BVSU	15	Backwards Vector Subtract
BZE	42	Branch if Zero
CALL	42	Transfer with Return Address
CALRB	42(CALL)	Transfer with Return Address, Return in Basic
CCALL	43	Computed CALL
CCLRB	43(CCALL)	Computed CALL, Return in Basic
CGOTO	43	Computed GOTO
CLEAR	53	Clear Bit
CLR	53(CLEAR)	Clear Bit

Symbol	Page VIB-	Function
CLRGO	53	Clear Bit and Transfer
COS	8	Cosine
COSINE	8(COS)	Cosine
DAD	9	Double Precision Add
DCOMP	9	Double Precision Complement
DDV	9	Double Precision Divide
DLOAD	35	Double Precision Load
DMP	9	Double Precision Multiply
DMPR	10	Double Precision Multiply and Round
DOT	16	Dot Product of Vectors
DSQ	10	Double Precision Square Operation
DSU	11	Double Precision Subtract
EXIT	44	Exit from Interpreter
GOTO	44	Transfer Control
INCR	46	Increment Index by Address
INV	54(INVERT)	Invert Bit
INVERT	54	Invert Bit
INVGO	54	Invert Bit and Transfer
ITA	35	Interpretive Transfer Address to Storage
ITCQ	45(RVQ)	Interpretive Transfer Control via QPRET
LXA	47	Load Index from Address
LXC	47	Load Index from Complement of Address
MXV	17	Matrix Times Vector
NORM	26	Normalize
PDDL	35	Store in Push-down List and Double Precision Load
PDVL	36	Store in Push-down List and Vector Load
PUSH	37	Store in Push-down List
ROUND	11	Round to Double Precision
RTB	45	Return to Basic
RVQ	45	Return Via QPRET
SET	54	Set Bit
SETGO	54	Set Bit and Transfer
SETPD	38	Set Push-down List Address
SIGN	11	Sign Function
SIN	12	Sine
SINE	12(SIN)	Sine

Symbol	Page VIB-	Function
SL	27	Scalar Shift Left
SL1-SL4	27	Short Scalar Shift Left
SLC	26(NORM)	Shift Left and Count
SLOAD	38	Single Precision Load
SLR	28	Scalar Shift Left and Round
SL1R-SL4R	29	Short Scalar Shift Left and Round
SQRT	12	Square Root Function
SR	29	Scalar Shift Right
SR1-SR4	30	Short Scalar Shift Right
SRR	30	Scalar Shift Right and Round
SR1R-SR4R	31	Short Scalar Shift Right and Round
SSP	38	Store Single Precision Constant
STADR	38	Cause Push-up on Store Address
STCALL	39	Store and do a CALL Instruction
STODL	39	Store and do a DLOAD Instruction
STORE	39	Store in Address
STOVL	39	Store and do a VLOAD Instruction
STQ	35(ITA)	Store QPRET
SXA	47	Store Index in Address
TAD	14	Triple Precision Add
TIX	48	Transfer on Index
TLOAD	40	Triple Precision Load
UNIT	18	Form a Unit Vector
VAD	19	Vector Add
VCOMP	19	Vector Complement
VDEF	19	Vector Define
VLOAD	40	Vector Load
VPROJ	20	Vector Projection on New Vector
VSL	31	Vector Shift Left
VSL1-VSL8	32	Short Vector Shift Left
VSQ	20	Vector Square Operation
VSR	33	Vector Shift Right
VSR1-VSR8	33	Short Vector Shift Right
VSU	21	Vector Subtract
VXM	21	Vector Times Matrix
VXSC	22	Vector Times Scalar

Symbol	Page VIB-	Function
VXV	23	Vector Cross Product
V/SC	24	Vector Divided by Scalar
XAD	48	Add to Index from Address
XCHX	49	Exchange Contents of Index and Address
XSU	49	Subtract Address from Index

An asterisk after an operation (e.g. DLOAD*) means that index register selection information is to be obtained from the address specification. For an index-affecting operation, the two index registers are indicated with the operation symbol itself, such as: AXC,1 and AXC,2.

Alphabetical Listing of Registers, Program Steps, and Storage References

This list includes only those symbols which serve a significant function in the program control logic or which are assigned to hardware-oriented registers and cells. Most of the program-step tags used solely within a particular routine have been omitted, as have various cell tags whose function is self-explanatory from the method in which they are used (such as TS, BUFF, etc.).

Symbol	See Page	Function
34D,35D	VID-2	Relative addresses with respect to FIXLOC, containing LVSQUARE (square of vector after ABVAL and UNIT operations)
36D,37D	VID-2	Relative addresses with respect to FIXLOC, containing LV (length of vector after UNIT)
2PHSCHNG	VIIC-13	Routine to change 2 phase table entries for restarts
15ADRERS	VIC-5	Load EBANK and POLISH with erasable address
A	IIC-1	Hardware accumulator
ADDRWD	VIC-4	S-register portion of interpreter address
ADVAN	VIIB-10	Looping point in dummy job routine
ALARM	VIID-1	Routine entered if software encounters a non-severe problem
ALARM1	VIID-1	Entrance to "ALARM" to retain triggering data
ALTM	IID-14	Computer output shift register for altitude and altitude rate meter
ARUPT	IID-1	Used to save A after a program interrupt
BAILOUT	VIID-1	Software restart for recoverable problem
BAILOUT1	VIID-1	Entrance to "BAILOUT" to retain triggering data
BANKALRM	IID-14	BBANK of alarm source
BANKCALL	VIID-1	Permit two-way communication in (A, L) between routines in different banks
BANKJUMP	VIID-2	One-way transfer of program control to a step in a different bank
BANKRUPT	IID-2	Used to save BBANK after a program interrupt
BANKSET	VIIB-13	Cell for storage of BBANK information
BBANK	IIC-2	Both banks, i.e. FBANK and EBANK, information
BMAGX,Y,Z	IID-7	Body-mounted attitude gyro input cells
BRUPT	IID-2	Used to save B-register after a program interrupt

Symbol	See Page	Function
CADRTAB	VIIC-5	Entry of address information in restart table
CCSHOLE	VIID-2	Same effect as "POODOO"
CDUS	IID-6	Shaft CDU input
CDUT	IID-5,6	Trunnion CDU input
CDUiCMD i = S,T	IID-11,12	Shaft and trunnion CDU command
CDUX,Y,Z	IID-5	IMU CDU angle information
CDUiCMD i = X,Y,Z	IID-10	Driving information for CDU D to A converter (IMU CDU's)
CHAN5	IIE-4	Channel 05
CHAN6	IIE-4	Channel 06
CHAN12	IIE-9	Channel 12
CHAN13	IIE-13	Channel 13
CHAN14	IIE-16	Channel 14
CHAN30	IIE-19	Channel 30
CHAN31	IIE-23	Channel 31
CHAN32	IIE-26	Channel 32
CHAN33	IIE-28	Channel 33
CHANG1	VIIB-7	Cause a higher priority job to be started
CHANG2	VIIB-7	Same as "CHANG1", entered only from "DANZIG"
CHANJOB	VIIB-7	Complete function of changing job
CHECKMM	VIID-2	Check for equality with MODREG (program register)
CURTAINS	VIID-2	Routine entered if serious hardware difficulty
CYL	IID-3	Cycle left register
CYR	IID-2	Cycle right register
DANZIG	VIC-2	Standard return from most interpretive orders
DELAYJOB	VIID-2	Wait for time interval given by A, then start performing job computations again
DIRADRES	VIC-4	Determine address information for non-indexed Indexable Operation in interpreter
DMPNSUB	VIID-2	Multiply $MPAC_{dp}$ and A, and shift result
DNTM1,2	IIE-32	Channels 34 and 35, used for telemetry
DOSTORE	VIC-6	Process interpreter storage commands
DOWNFLAG	VIID-3	Set flagword bit to zero
DOWNRUPT	IIH-6	Computations when telemetry interrupt received
DP	VIA-4	Setting of MODE if double precision computations being performed (is +0)
DPAGREE	VIID-3	Force sign agreement of $MPAC_{dp}$

Symbol	See Page	Function
DSALMOUT	IIE-4	Channel 11
DUMMYJOB	VIIB-10	Idling routine in executive system
EBANK	IIC-1	Erasable memory bank register
EDOP	IID-3	Edit operand register, shifts bits 14-8 to bits 7-1
EJSCAN	VIIB-10	Scan for highest priority job
EMSD	IID-13	Cell assigned for entry monitoring purposes
ENDOFJOB	VIIB-9	Terminate computations associated with a job and release associated memory cells
F EXT	IIE-4	Channel 07 (same as SUPERBNK)
FBANK	IIC-2	Fixed memory bank register
FINDVAC	VIIB-5	Establish a new job with a VAC area
FIXDELAY	VIIA-4	Delay a waitlist task a certain amount of time, then return to computations
FIXLOC	VID-1	Starting point for relative addresses in VAC area
GEADDR	VIC-5	Determine proper cell settings for erasable memory reference by Indexable Operation
GENTRAN	VIID-3	Move set of cells from one location to another
GOPROG	IIH-7	Perform computations associated with restart
GYROCMD	IID-10	Cell containing number of gyro torquing pulses
HAND CONTROL RUPT	IIH-7	Initiate computations associated with hand-controller interrupt
HISCALAR	IIE-3	Channel 03
IBNKCALL	VIID-3	Interrupt analog of "BANKCALL"
INDEX	VIC-4	Determine effect on address of index register
INDJUMP	VIC-8	Transfer to Indexable Operation
INLINK	IID-8	Input to computer from uplink receiver
INTBT15	VIC-2	Cell used to distinguish between low and high fixed memory banks in interpreter
INTPRET	VIC-2	Starting address for interpreter
ISWCALL	VIID-3	Interrupt analog of "SWCALL"
ISWRETRN	VIID-3	Exit after "IBNKCALL" or "ISWCALL"
ITSAVAR	VIIC-17	Restart computations if information in erasable memory
ITSLGCL1	VIIC-16	Restart computations if "LONGCALL" restart
JOBSLEEP	VIIB-8	Put job "to sleep" (inactive)

Symbol	See Page	Function
JOBWAKE	VIIB-8	Awaken job previously put to sleep
KEYRUPT1,2	IIH-5	Computations performed when DSKY, optics, or LM ROD (rate of descent) control inputs received
KILLTASK	VIIA-7	Remove task from waitlist queue
L	IIC-1	L-register (has least significant half of double precision number)
LCHAN	IIE-3	L-register information when used as channel 01
LEMONM	IID-13	Cell assigned for LM monitoring purposes
LOADTIME	VIID-3	Load $MPAC_{dp}$ with computer clock (TIME2,TIME1)
LOC	VIIB-13	Cell for keeping track of addresses in interpreter
LOCALARM	IID-14	LOC value for alarm source
LOCCTR	VIIB-6	Cell containing indexing information for location of Job Register Set when a job established or awakened
LONGBASE	VIIC-6	Time origin of information for restarting "LONGCALL"
LONGCADR	VIIA-5	Starting address of task in "LONGCALL"
LONGCALL	VIIA-5	Routine permitting use of waitlist system (for one task at a time) for a large time value
LONGTIME	VIIA-5	Time argument for "LONGCALL"
LOSCALAR	IIE-3	Channel 04
LRUPT	IID-1	Used to save L after a program interrupt
LST1	VIIA-9	List of times in waitlist system
LST2	VIIA-9	List of task starting addresses in waitlist system
LV	VID-2	Vector length (cf. 36D,37D)
LVSQUARE	VID-2	Square of vector length (cf. 34D,35D)
MAKECADR	VIID-4	Form an address in CADR format
+MAX, -MAX	Var.	Maximum values that can be accommodated by computer register
MISCJUMP	VIC-8	Transfer to Miscellaneous Operation
MNKEYIN	IIE-18	Channel 15
MODE	VIIB-13	Information on character of computations being performed (cf. DP, TP, and VC)
MPAC	VIIB-12	Multi-purpose accumulator
NAVKEYIN	IIE-18	Channel 16
NEWJOB	IID-15	Cell set to LOCCTR if current job not the one of highest priority
NEWOPS	VIC-2	Obtain new interpreter operation

Symbol	See Page	Function
NEWPHASE	VIIC-13	Update restart phase table information
NOQBRSM	VIIA-7	Resume computations after a program interrupt
NOQRSM	VIIA-7	Restore BBANK and resume computations after a program interrupt
NORMUNIT	VIID-4	Form unit vector after shifting left 13 if all components small
NORMUNX1	VIID-4	Entrance to "NORMUNIT" leaving shift amount in X1
NOVAC	VIIB-6	Establish a new job without a VAC area
NOVAC2	VIIB-6	Computations common to "FINDVAC" and "NOVAC"
ONEORTWO	VIIC-14	Check nature of variable-type phase update
OPJUMP	VIC-3	Branch to proper computations determined by prefix of interpretive order
OPJUMP2,3	VIC-3	Continue "OPJUMP" checks
OUTO	IIE-4	Channel 10
OUTLINK	IID-14	Erasable memory shifting-type cell
OVFIND	VIA-10	Overflow indicator for interpreter
P-RHCCTR	IID-7	Yaw rotational hand controller analog input (LM)
PHASCHNG	VIIC-13	Update phase table information
PHASEi i = 1-6	VIIC-3	Restart phase table selection information
PHSCHNGA	VIIC-13	Entrance to "PHASCHNG" with argument in A
PHSNAMEi i = 1-6	VIIC-5	Starting addresses stored in erasable memory for variable-type restarts
PHSPRDTi i = 1-6	VIIC-5	Priority/delay time stored in erasable memory for variable-type restarts
PIPAX,Y,Z	IID-6	Accelerometer inputs
POLISH	VIC-3	Cell loaded for Miscellaneous Operations with address information (subsequently modified in some cases). Method of writing interpretive language information is "Polish notation".
POLY	VIID-4	Routine to evaluate a polynomial
POODOO	VIID-4	Software restart for problem not expected to be recoverable
POODOO1	VIID-4	Entrance to "POODOO" to retain triggering data
POSTJUMP	VIID-4	Transfer to a step in a different bank (retaining (A, L))
POWRSERS	VIID-5	Routine to evaluate a polynomial (different calling sequence than "POLY")
PRDTTAB	VIIC-5	Entry of priority/delay time for restart table
PRIOCHNG	VIIB-9	Change priority of current job
PRIORITY	VIIB-13	Cell containing priority information on job
PUSHLOC	VIIB-13	Address where next word to be stored in push-down list

Symbol	See Page	Function
PUSHUP	VIC-5	Routine to withdraw information from push-down list
PYJETS	IIE-4	Channel 05
Q	IIC-1	Return address (Q-register) information for TC machine language order
Q-RHCCTR	IID-7	Pitch rotational hand controller analog input (LM)
QCHAN	IIE-3	Q-register information when used as channel 02
QPRET	VID-2	Return address information in interpreter
QRUPT	IID-1	Used to save Q after a program interrupt
R-RHCCTR	IID-7	Roll rotational hand controller analog input (LM)
RADAR RUPT	IIH-7	Computations when radar input buffer has been filled
RESTARTS	VIIC-15	Computations performed for each active group when a restart takes place
RESUME	VIIA-7	Restore Q and BBANK, then resume computations after a program interrupt
RNRAD	IID-9	Radar input data buffer
ROLLJETS	IIE-4	Channel 06
S1, S2	VID-2	Step registers (used with interpreter TIX)
S3, S4	IIB-2	Superbank settings
SAMPTIME	IID-1	Sampled value of computer clock
SETLOC	VIIB-7	Check for whether NEWJOB must be set
SGNAGREE	VIID-5	RTB order to force $MPAC_{tp}$ sign agreement
SHORTMP	VIID-5	Multiply $MPAC_{tp}$ and A
SIGNMPAC	VIID-5	Set $MPAC_{dp}$ to $\pm$ MAX depending on MPAC+0 sign
SPCOS	VIID-5	Single precision cosine routine
SPECTEST	VIIB-7	Special computations if no active jobs are running and one becomes activated
SPSIN	VIID-5	Single precision sine routine
SPVAC	VIIB-6	Special entrance to "FINDVAC" with priority information already loaded (into NEWPRIO) and job starting address in (A, L)
SR	IID-3	Shift right register
STORE	VIC-7	Store information in erasable memory
SUPDACAL	VIID-6	Obtain data from another bank, including switch of SUPERBNK
SUPDXCHZ	VIID-6	Transfer to address in (A, L) including setting of SUPERBNK (format as for 2CADR)
SUPERBNK	IIE-4	Channel 07
SUPERSW	VIID-6	Routine to load bits 7-5 of A into channel 07

Symbol	See Page	Function
SVCT3	VIIA-6	Dummy task, serving primarily for waitlist control purposes
SWCALL	VIID-6	Perform routine in another bank (CADR given in A), and then return
SWRETURN	VIID-6	Return from "BANKCALL" or "SWCALL"
T3RUPT	VIIA-6	Computations when TIME3 program interrupt processed
TiRUPT i = 4,5,6	IIH-5	Computations when TIME4, TIME5, and TIME6 program interrupts processed
TASKOVER	VIIA-7	Terminate a waitlist task
TBASEi i = 1-6	VIIC-6	Time origin information for restarting waitlist tasks
THRUST	IID-13	Cell containing number of pulses to be sent to change throttle setting of descent engine(LM)
TIME1	IID-4	Least significant half of computer clock
TIME2	IID-4	Most significant half of computer clock
TIME3,4	IID-4	Cells controlling T3RUPT and T4RUPT
TIME5,6	IID-5	Cells controlling T5RUPT and T6RUPT
TIMETEST	VIIC-17	Check for nature of time information for a waitlist restart
TP	VIA-4	Setting of MODE if triple precision computations are being performed (is +1)
TPAGREE	VIID-6	Force sign agreement of $MPAC_{tp}$
TPMODE	VIID-6	Set interpreter MODE cell for triple precision
TVCPITCH	IID-12	SPS pitch control ("steering")
TVCYAW	IID-12	SPS yaw control ("steering")
TWIDDLE	VIIA-3	Waitlist entry for task in same BBANK
UNAJUMP	VIC-9	Transfer to Unary Operation
UPFLAG	VIID-6	Set flagword bit to one
UPRUPT	IIH-6	Computations when uplink interrupt received
USPRCADR	VIC-9	Routine to permit use of interpreter subroutines in variable-fixed memory
VACIUSE I = 1-5	VID-1	Control cell indicating, if zero, that the associated VAC area is assigned
VARDELAY	VIIA-4	Delay a waitlist task the amount of time in A, then return to computations
VC	VIA-4	Setting of MODE if vector computations are being performed (is -1)

Symbol	See Page	Function
WAITLIST	VIIA-3	Entrance to waitlist system to place a task in queue for execution after a specified time has elapsed
X1, X2	VID-2	Subtractive index registers for interpretive language
Z	IIC-2	Z-register (program counter)
ZRUPT	IID-2	Saves value of Z (is loaded and restored by hardware means) when program interrupt occurs

Alphabetical Listing of Terms

This list includes those terms which have a special definition in the previous sections of this document. Most computer-oriented terms have been defined in accordance with a glossary compiled by the U. S. Bureau of the Budget as an official reference (and reprinted by the magazine "Datamation").

Term	See Page	Comments
Accumulator	IIC-1 VIA-7	In machine language, register used to hold the result of most arithmetic operations. In the interpretive language, is set of 7 erasable memory cells (MPAC) performing an analogous function.
Assembler	III-1	A general-purpose computer program ("assembly program") which produces memory information for a computer from symbolic inputs.
Bank number	IIF-2	
Bank register	IIB-2	There are separate ones for fixed and erasable memory.
CADR	VC-3	Term meaning "complete address", used for information expressed in this format.
Calling address	---	Address of cell causing transfer to a "called" program: return generally would be to the step after this address.
CDU	IID-5	Coupling Data Unit, consisting of an analog-to-digital side and a digital-to-analog side. Used in association with IMU gimbal angle determination (and other functions).
Centi-second	A-15	0.01 second.
Channel	IIE-1	
Checksum	IIF-5	Sum (formed in a special way) of the contents of a fixed memory bank.

Term	See Page	Comments
COLOSSUS	I-1	Generic identification of manned CSM earth orbital and lunar capability program. Numerical values for first two flights were assembly revision numbers; subsequently, the numerical value indicates a major revision, and the letter (if any) a baseline update (although not necessarily a flight program). Same first letter as "CSM".
COMANCHE	I-1	Code name for programs of the "COLOSSUS 2x" series, with subsequent serial giving the unique assembly revision number.
DSKY	IIJ-1	
Dummy job	VIIB-5	Routine performed if no jobs need be done.
Dummy task	VIIA-6	Task performed every 81.93 sec., primarily for waitlist control.
Error counter	IID-10	
Executive system	VIIB-1	
Extended order	IVA-1	
Field	III-12	An assigned area on a card.
Fixed-fixed memory	IIB-2	Special term used to refer to that portion of fixed memory which can be uniquely addressed by S-register alone, regardless of FBANK value.
GAP	III-4	General Assembly Program.
Hardware restart	IIH-9	Causes program to go back to a certain planned point (based on information in restart tables), due to various hardware-discovered problems (some can be induced by the software).
IMU	IID-5	Inertial Measurement Unit.
Index register	VIA-5	Cell containing a quantity which may be used to modify addresses: there are two associated with the interpretive language, stored in VAC area. Should not be confused with the machine language INDEX order, although net effect is similar.
Indexable Operation	VIA-6	
Interpreter	VIA-2	A routine which, as the computation progresses, translates operation and address parameter information into machine language orders and executes these orders.

Term	See Page	Comments
Interpretive language	VIA-1	The notation used for operations and addresses that can be understood by the interpreter.
Interrupt	IIH-1	Temporarily disrupt normal computations because of a special signal (computer has both counter and program interrupts).
Jask	VII-2	"A job to other tasks and a task to other jobs".
Job	VIIB-1	Any computation being performed under jurisdication of the executive system. If the computer is not performing a task/jask or the dummy job, it is performing a job (by definition).
Job Register Set	VIIB-12	
Left justified	III-5	Arranged so that the left-most digit (whether 0 or not) is in a standard column.
LM-1		First (unmanned) Lunar Module flight.
Location counter	III-8	Assembler counter used to determine assignment of binary memory information to absolute machine addresses.
Log section	III-4	
LUMINARY	I-1	Generic identification of manned LM earth orbital and lunar capability program (see COLOSSUS). Also used as the code name for the "LUMINARY 1x" series, with subsequent serial giving the unique assembly revision number. Same first letter as "LM".
Machine language	IVA-1	A language which, after processing by the assembler, can be performed directly by the hardware, as contrasted with "Interpretive language".
MCT	IIA-3	Memory Cycle Time, the quantum of execution time for a machine language order. One MCT is 11.71875 microsec., the time for 12 pulses at a prf of 1.024 mc.
Miscellaneous Operation	VIA-6	
Ones Complement	A-5	
Overflow	A-8	
Parameter	VIA-2	Quantity processed by interpreter to produce an instruction or an address.

Term	See Page	Comments
Parity	IIA-3	A check bit indicating (for the "odd parity" system used) that the total number of binary ones, including itself, should be odd.
PIPA	IID-6	Pulsed Integrating Pendulous Accelerometer.
Precision	VI	The degree of exactness with which a quantity is stated. Single precision means that the basic computer word size is used; double precision means two such words; triple precision means three such words. Computer word length is 14 magnitude bits (plus a sign bit).
Prefix	VIA-5	
Priority	VIIB-2	
Pseudo-Operation	VA-1	A symbolic operation which is not one of the standardized machine language codes, or not part of the hardware design, but which can be suitably analyzed by the assembler.
Push-down list	VID-3	List in which the last item that is inserted is first item withdrawn.
Regular order	IVA-1	
Relative address	VID-1	An address to which a base address must be added to find the machine address.
Restart tables	VIIC-5	Tables containing job and task information used if a restart is encountered.
Revolution	A-11	360 degrees.
Ropes	IIF-2	
S_1, S_2	IIG-2	Accumulator sign bits. Should not be confused with S1 and S2, which are interpretive language index stepping cells.
S-register	IIB-2	
Scale factor	A-11	
Software restart	IIH-9	Causes program to go back to a certain planned point (based on information in restart tables), due to various software-discovered problems (if recoverable, is a BAILOUT type, if not, a POODOO type).

Term	See Page	Comments
Subroutine	III-15	Program segments read by SUBRO assembler control cards, and identified in this document by the capitalized first letter.
Superbank	IIB-2	
Switched erasable	IIB-2	Special term used to refer to portion of erasable memory which must use EBANK to help determine cell reference.
Tag	III-8	Symbolic identification for a cell location (so it can be referenced by other steps).
Task	IIH-4	Computations performed in the "interrupt mode" when a program interrupt is being acted upon (distinguished from a "Job").
True address	IIB-3	Address determined from combination of S-register and appropriate bank register.
Twos complement	A-5	
Unary Operation	VIA-7	
Unswitched erasable	IIB-2	Special term used to refer to portion of erasable memory which can be uniquely addressed by S-register, regardless of EBANK.
VAC area	VID-1	One of five working areas which can be assigned to a job.
Variable-fixed memory	IIB-2	Special term used to refer to portion of fixed memory which must use FBANK (and perhaps SUPERBNK) to help determine cell reference.
Waitlist system	VIIA-1	
YUL System		G&N contractor software package for general-purpose computer to aid in preparation of programs for guidance computer (comprised of assembler and other programs). So named because of development schedule of an early computer (December 1959). The assembly program in Section III is called "GAP", and superseded YUL.