60428901

CONTROL DATA CORPORATION

MATRIX ALGORITHM PROCESSOR III (MAP III) SYSTEM USER REFERENCE MANUAL

CDC® OPERATING SYSTEMS:
NOS
NOS/BE

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released.
(05-01-75)	
B	Manual revised. This revision adds the MDRLSE call description, modifies the FILTER macro
(06-01-76)	description, adds appendix F, and incorporates miscellaneous editorial modifications. This
	edition obsoletes all previous editions.
C	Manual revised. This revision adds three macro descriptions and incorporates miscellaneous
(11-15-76)	editorial modifications. Pages ii, iii/iv, vi, vii, viii, ix, 1-1, 1-8, 2-5, 3-2, 3-3, 3-6, 3-14,
	3-22, 3-23, 3-28, 3-29, 4-1, 4-2, 4-4, and A-1 are revised. Page 3-14.1 is added.
D	Manual revised to incorporate reference information associated with modifications to the
(07-17-78)	MAP System Software Interface. Because of extensive changes to this manual, chart tape and
	dots are not used, and all pages reflect the latest revision level. This edition obsoletes all
	previous editions.

Publication No.
60428901

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:

Control Data Corporation
Publications and Graphics
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

Printed in the United States of America

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV
Front Cover	-
Title Page	-
ii	D
iii/iv	D
v	D
vi	D
vii	D
viii	D
1-1	D
1-2	D
1-3	D
1-4	D
1-5	D
1-6	D
2-1	D
2-2	D
2-3	D
2-4	D
2-5	D
2-6	D
2-7	D
2-8	D
2-9	D
2-10	D
3-1	D
3-2	D
3-3	D
3-4	D
3-5	D
3-6	D
3-7	D
3-8	D
3-9	D
3-10	D
4-1	D
4-2	D
4-3	D
4-4	D
4-5	D
4-6	D
4-7	D
4-8	D
4-9	D
4-10	D
4-11	D
4-12	D
4-13	D
5-1	D
5-2	D
5-3	D
5-4	D
5-5	D
5-6	D
5-7	D
5-8	D
A-1	D
A-2	D
A-3	D
A-4	D
A-5	D
A-6	D
A-7	D
A-8	D
A-9	D
A-10	D
A-11	D
A-12	D
A-13	D
B-1	D
B-2	D
C-1	D
C-2	D
Index-1	D
Index-2	D
Comment Sheet	D
Back Cover	-

PREFACE

This manual provides reference information for the CDC® Matrix Algorithm Processor III (MAP III) System, which consists of a microprogrammable processor called MAP and a software package called the MAP System Software Interface (MSSI). The MAP III system provides an efficient way for user FORTRAN programs to perform lengthy array calculations.

The MAP III system interfaces with the following extended core storage - equipped computer systems running under CDC NOS or NOS/BE operating systems.

- CDC 6000 series.
- CDC CYBER 70 models 72/73/74.
- CDC CYBER 170 series.

This manual is written primarily for FORTRAN (FTN) programmers and computer operators who use the MAP III system. Refer to the following table for supplementary and related documents.

The following conventions are used in this manual.

- All numbers are decimal unless another base is indicated.
- Logical zero and one are abbreviated 0 and 1, respectively.
- Bits are numbered from right to left beginning with 0.
- [ ] enclose optional FORTRAN call arguments.
- . . . indicates preceding pattern repeats indefinitely.

The following articles describe the algorithms underlying the fast Fourier transform (FFT) macros described in section 3. The basic FFT algorithm first appeared in article 1. Article 2 discusses many details relevant to the MAP implementation of the algorithm. The time-saving and space-saving variations used for the REALFFT and INVRFFT macros are discussed on page 65 of article 3.

1. J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, vol. 19, pp. 297-301, April, 1965.
2. G-AE Subcommittee on Measurement Concepts (W. T. Cochran, et. al.), "What is the Fast Fourier Transform?". IEEE Transactions on Audio and Electroacoustics, vol. AU-15, pp. 45-55, June, 1967.
3. C. Bingham, M. D. Godfrey, and J. W. Tukey. "Modern Techniques of Power Spectrum Estimation", IEEE Transactions on Audio and Electroacoustics, vol. AU-15, pp. 56-66, June, 1967.

DISCLAIMER

The MAP III system is intended for use only as described in this manual. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTROL DATA MAP III DOCUMENTS

Document	Type†	Part No.	User	Purpose
ECL 10000 Series Circuit Description Manual	Controlled	60417700	Customer engineer	Describes MAP III logic elements and related symbology.
MAP Hardware Maintenance Manual	Controlled	60429100	Customer engineer	Provides MAP III hardware maintenance information.
MAP III Installation Handbook (NOS)	Uncontrolled	-	System programmer	Describes MSSI installation procedure for NOS.
MAP III Installation Handbook (NOS/BE)	Uncontrolled	-	System programmer	Describes MSSI installation procedure for NOS/BE.
NOS Programmer's Maintenance Aid, CYBER MAP III Software	Uncontrolled	-	System programmer	Defines MSSI internal structure for NOS.
NOS/BE Programmer's Maintenance Aid, CYBER MAP III Software	Uncontrolled	22836700	System programmer	Defines MSSI internal structure for NOS/BE.
6000 MAP III Controlware External Reference Specification (ERS)	Uncontrolled	12104400	System programmer	Defines external interface to MAP III internal controlware.
MAP III On-Line Controlware ERS	Uncontrolled	-	System programmer	Describes additional controlware macros.
6000 MAP III Assembler ERS	Uncontrolled	12104300	Controlware programmer	Describes assembler used to generate internal controlware.
MAP III Command Diagnostics ERS	Uncontrolled	12104200	Customer engineer	Describes MAP III control unit diagnostics.
MAP III Memory Test (QMM) ERS	Uncontrolled	12104100	Customer engineer	Describes MAP III data storage diagnostics.
MAP III Test Functional Units (TFU) ERS	Uncontrolled	22836600	Customer engineer	Describes MAP III functional unit diagnostics.
MAP III System Confidence Test (QM3) ERS	Uncontrolled	12103900	Customer engineer	Describes diagnostics used to confirm normal MAP III system operation.

†Control Data is not responsible for the contents of uncontrolled documents.

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

This section describes MAP and MSSI, which are the hardware and software elements, respectively, of the MAP III system.

MAP

The hardware element of the MAP III system is Matrix Algorithm Processor III (MAP), a microprogrammable array processor containing its own data storage. MAP uses a 32-bit floating-point format for arithmetic calculations and contains a numerical conversion unit to translate various computer system data formats to the 32-bit format.

Microcode that resides within MAP is called controlware. Although MSSI supplies default controlware that provides standard MAP capabilities, controlware options are available to increase these capabilities. Appendix B lists standard and optional MAP capabilities and associated macros.

The user controls MAP with macro strings, each of which is a collection of macros assembled and loaded into MAP by means of MSSI. Generally speaking, a macro is a symbol representing the microcode for performing a specific task or algorithm. For example, the HALT macro represents the microcode that stops macro execution, the LOADP32 macro represents the microcode that transfers packed data from extended core storage (ECS) to MAP, and the SUMPROD macro represents the microcode that performs a sum of products calculation. From the user's point of view, MAP executes a macro string as a computer executes a program.

MAP CABINET

Figure 1-1 shows the MAP cabinet, which includes a power bay, a logic bay, and an optional memory bay. Operating controls are in the power and logic bays, and input/output cables connect to the logic bay.

OPTIONS

Figure 1-2 shows the options that can be added to MAP.

- Data storage options permit MAP data storage to be expanded from its minimum size of 24 K† to its maximum size of 256 K.
- Additional arithmetic units reduce processing time for certain algorithms. For example, MAP performs sum-of-products calculations most rapidly when equipped with four add/subtract units and four multiply units. Controlware must be modified when arithmetic units are added.

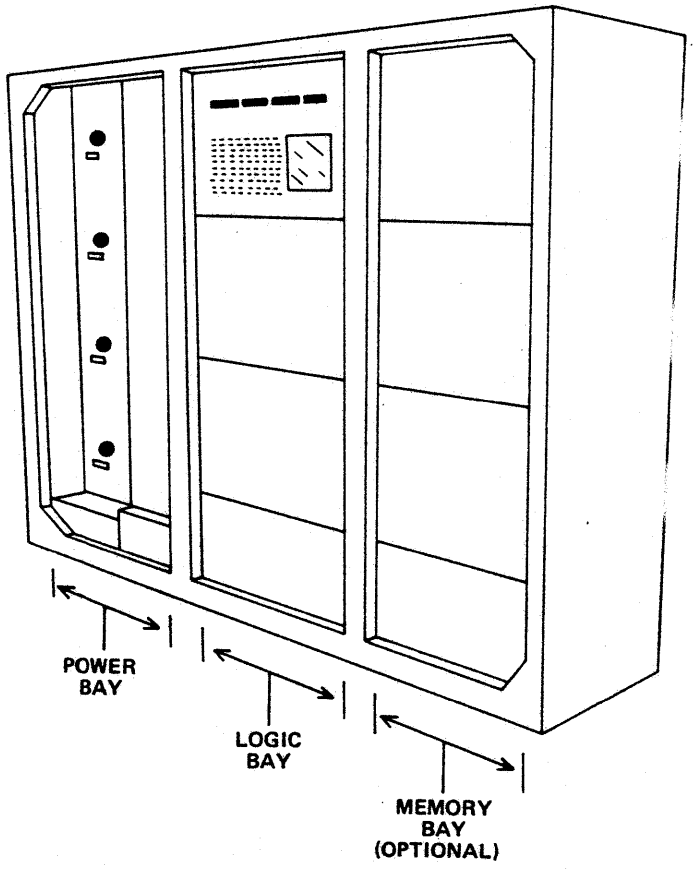


Figure 1-1. MAP Cabinet

Figure 1-3 shows the minimum MAP configuration necessary to support MSSI.

†1 K = 1024 words.

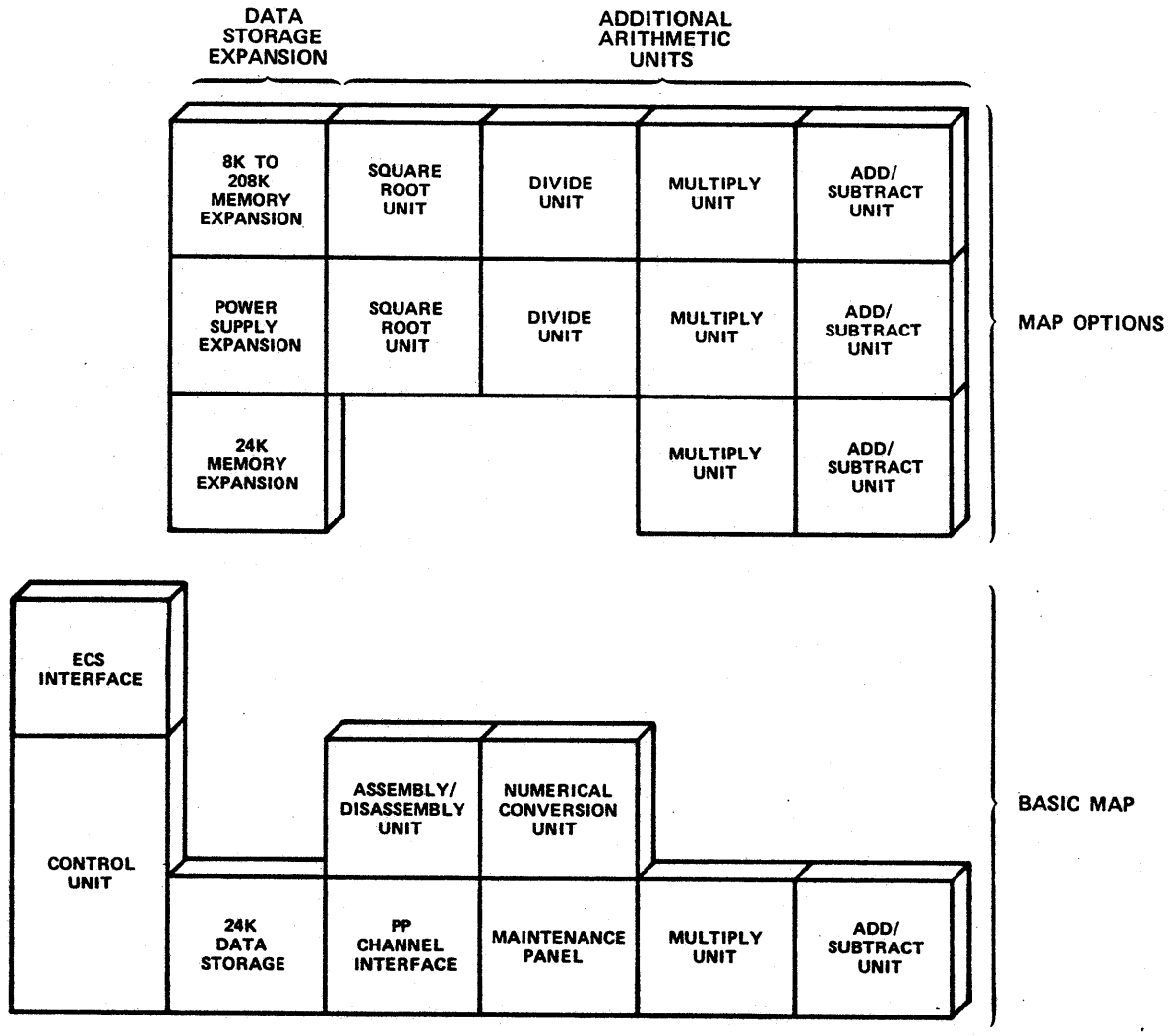


Figure 1-2. Basic MAP and Options

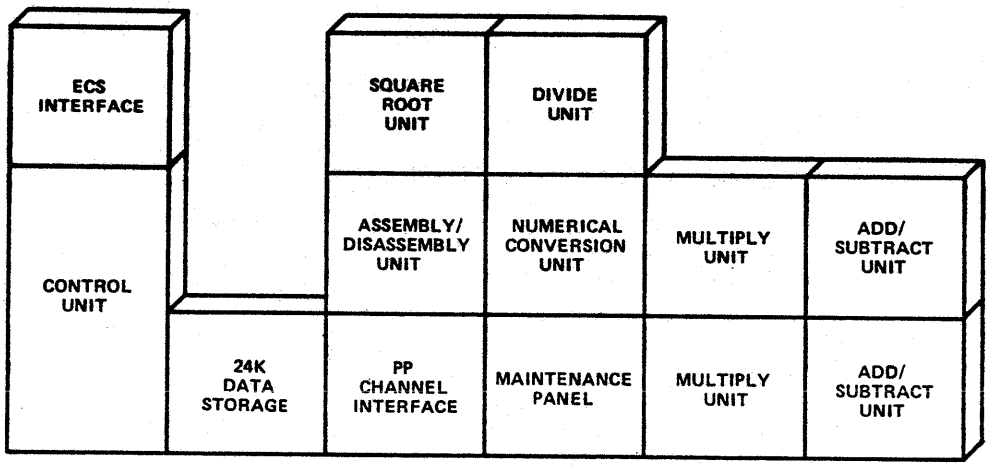


Figure 1-3. Minimum MAP Configuration for MAP III System

DATA FORMAT

Figure 1-4 shows the MAP internal data format.

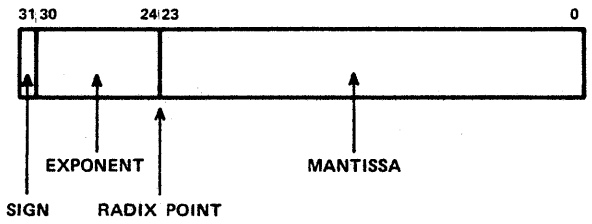


Figure 1-4. MAP Internal Data Format

The format consists of a 24-bit normalized mantissa, a 7-bit, signed twos complement exponent, and a sign bit using sign and magnitude representation. The exponent represents powers of 2.

The range for positive normalized internal floating-point format MAP numbers is from 2^{-100_8} to $0.77777777_8 \cdot 2^{77_8}$. The smallest and largest decimal numbers (7 significant digits) in this range are $0.5421011 \cdot 10^{-19}$ and $0.9223371 \cdot 10^{+19}$. All positive fixed-point numbers not exceeding 2^{24} (16,777,216) can be represented exactly in internal format.

FUNCTIONAL DESCRIPTION

Figure 1-5 shows MAP functional areas.

Input/Output

During MAP input, the input/output area accepts ECS, peripheral processor (PP), or cassette data; passes it through the assembly/disassembly unit; and places the data on the assembly/disassembly bus. From here, the data can transfer directly to the control area or indirectly to data storage through the numerical conversion unit and the result bus. During MAP output, data flow reverses except that data returns from data storage to the assembly/disassembly bus through operand bus 2 and the numerical conversion unit.

Control

The control area executes controlware and macro strings. Controlware determines how MAP processes each algorithm and resides in control memory, read-only memory (ROM), and subcontrol memory. Macro strings reside in macro memory and are used to sequence microcode execution. The user generates macro strings by using the FORTRAN subroutine calls described in section 2. The 6000 MAP III Assembler ERS listed in the preface describes the special assembler used to generate controlware.

Arithmetic

This area contains arithmetic units and the buses that transfer operands/results between data storage, the arithmetic units, and the numerical conversion unit. Figure 1-2 shows the number and types of arithmetic units that can attach to MAP.

Data Storage

Data storage contains from 24 K to 256 K of 32-bit storage in three contiguous sections labeled x, y, and z. Maximum sizes for sections x, y, and z are 96 K, 96 K, and 64 K, respectively. Hardware switches specify the boundaries between sections so that addresses are continuous from the first word of section x to the last word of section z.

Data storage is equipped with three accesses, labeled A, B, and C. Since each access can read from or write to any nonbusy section, up to three data storage operations can proceed simultaneously. Thus, two accesses can read operands while, at the same time, the third stores a result.

MSSI

The software element of the MAP III system is the MAP System Software Interface (MSSI), a collection of COMPASS programs, FORTRAN routines, and MAP controlware that coordinates with the NOS or NOS/BE operating system. MSSI allows the user to:

- Generate macro strings for calling a series of MAP-resident algorithms into execution.
- Transfer controlware and macro strings to MAP.
- Monitor MAP status.
- Print MAP dump information.

MSSI also provides a repertoire of commands and messages that allows an operator to control the MAP III system from the system console. Section 5 describes these commands and messages.

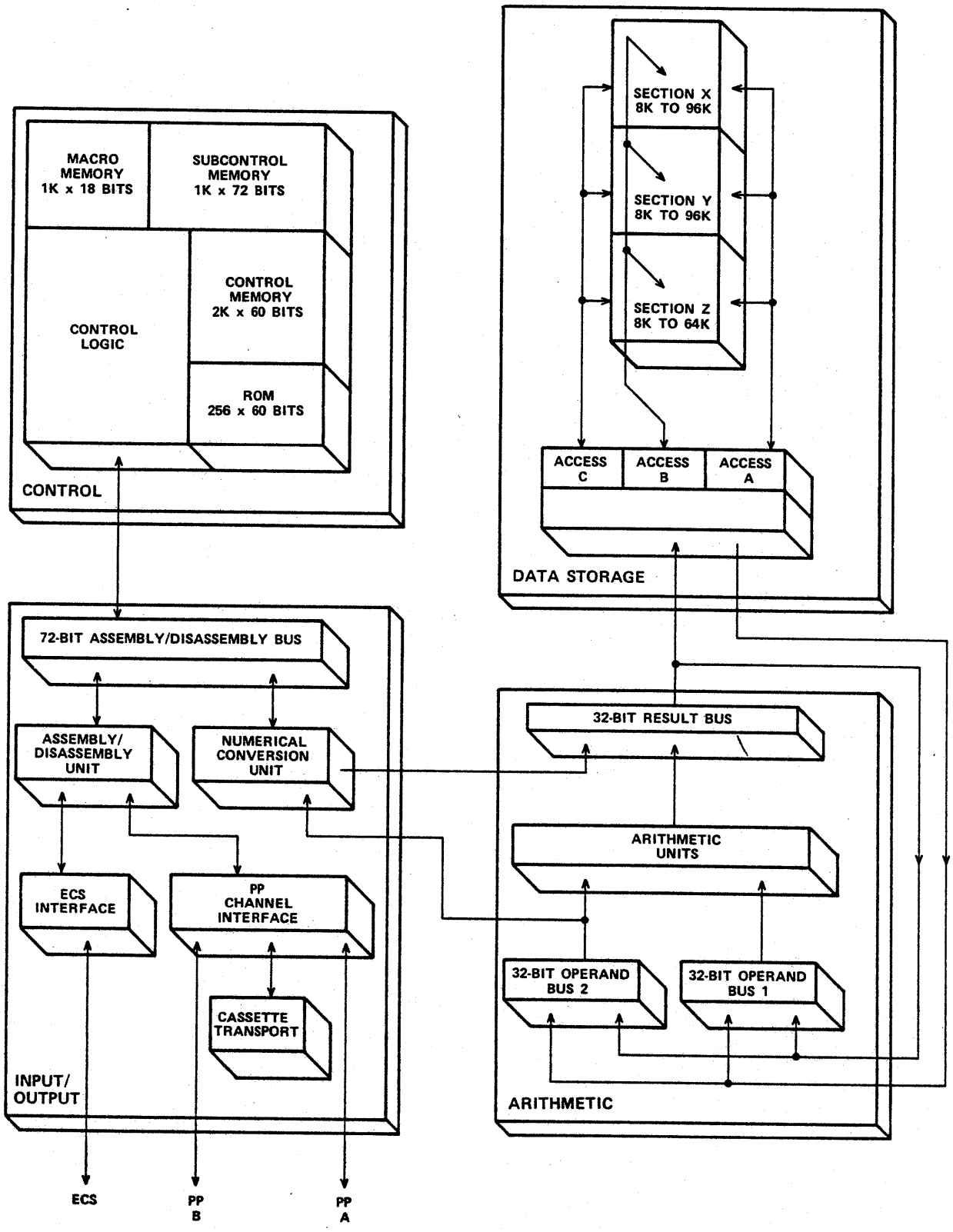


Figure 1-5. MAP Functional Entities

USER CONTROL

A user employs the MAP III system by writing a FORTRAN program that:

- Places operands in ECS.
- Generates a macro string that instructs MAP to obtain the operands from ECS, process them, and return results to ECS.
- Removes the results from ECS for output or further processing.

To simplify MAP-related programming, MSSI provides a set of execution time FORTRAN routines called the Macro String Assembly Module (MSAM). By making calls to these routines, a user program can build a macro string and also accommodate ancillary macro string requirements.

The following sequence shows tasks performed by a typical MAP-related program. Parentheses enclose MSAM-provided calls, which are described in section 2 along with the macro string format. Section 4 contains examples of MAP-related programs.

1. Declare file OUTPUT in PROGRAM statement.
2. Define parameters and data arrays to reside in central memory and ECS, respectively.
3. Open MAP environment table (MET). The MET is a table used for communication between MSSI and the operating system. (METOPEN)
4. Clear buffer area used to build macro string. (MAPSET)
5. Define MAP data storage arrays. (MALLOT, MEQUIV)
6. Define common macro parameters. (MPARAM)
7. Use MACRO calls to build macro string.
   a. Transfer operands from ECS to MAP with LOADxxx macro.
   b. Use arithmetic macros (REALFFT, ADDVEC, and so on) to perform calculations and use control/pseudo macros (TMM, JUMP, RJUMP, UPM) to perform tests, jumps, and updates.
   c. Transfer results from MAP to ECS with UNLDxxx macro.
   d. Stop macro string execution with HALT or END macro.
8. Pack macro string. (MAPNOGO)
9. Execute macro string. (MAPGO)
10. Allow for MAP-related program recall. (MAPGO call argument or MRECALL call)
11. Release ECS dump area. (MDRLSE)
12. Close MET. (MCLOSE)
13. Process or output results.
14. Terminate MAP-related program.

JOB SEQUENCE

Figure 1-6 shows a user program in the MSSI environment. A typical MAP III system job proceeds as follows:

1. The user program begins execution and calls MSAM routines to initialize tables, prepare a macro string, and request macro string execution. User program activity may suspend either immediately after the request or after additional processing.
2. After MSSI has scheduled the macro string and ensured that the correct controlware is loaded, the macro string transfers from central memory to MAP and begins execution.
3. MSSI records MAP status during macro string execution. When the macro string completes execution or a fatal error occurs, MSSI places the user program back in execution after updating timing/error tables.

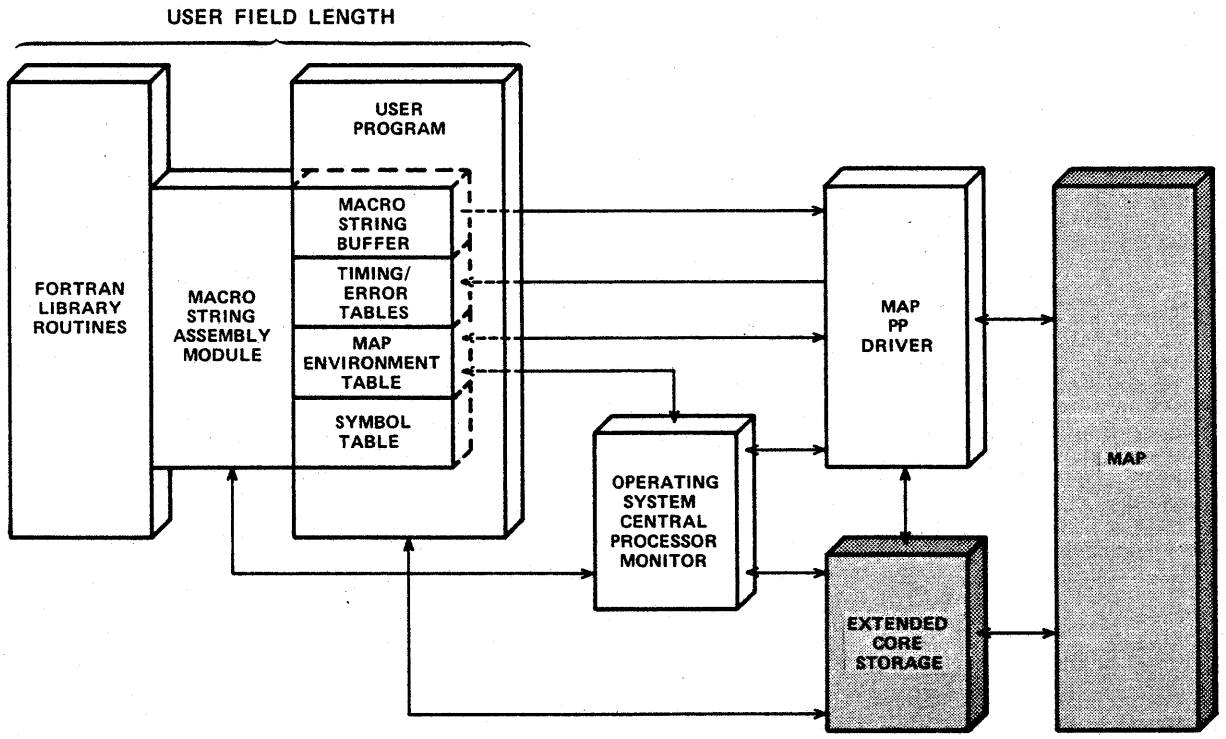


Figure 1-6. MSSI Environment

This section describes macro string formats, Macro String Assembly Module (MSAM) status, and FORTRAN calls provided by MSAM. The macro string formats are described here chiefly for reference purposes, since MSAM automatically builds and packs macro strings according to calls from user programs.

Appendix B contains an MSAM call summary. Section 4 describes the LOCE function (used to obtain the address of an ECS-resident variable) and provides programming considerations applicable to MSAM.

MACRO STRING

Figures 2-1 and 2-2 show the same macro string in unpacked and packed formats, respectively. MSAM uses the unpacked format while building a macro string, and then packs the macro string before sending it to MAP for execution.

Each macro string has a header, a macro field, and a parameter field. MSAM builds the macro field from the top down and the parameter field from the bottom up. Parameters within a block remain in user-assigned order, regardless of the position of the block in the parameter field.

HEADER

Before packing a macro string, MSAM uses the header as a scratch area for maintaining data necessary to complete the macro string. After packing the macro string, MSAM rewrites the header with the information shown in figure 2-2. When it transfers the macro string to MAP, the MAP PP driver replaces the word count and checksum in the header with RA (x), the relative address of MAP data storage section x.

MACRO FIELD

In unpacked form, each 60-bit word of the macro field contains a macro code in bits 0 through 17 and space for a tag in bits 18 through 59. A macro code is an 18-bit number associated with a macro. For example, the macro code for the NOOP macro is 020000_8. MACRO call arguments determine the placement of macro codes and tags as follows:

- A MACRO call with 0 for the tag argument places the macro code for the macname argument in bits 0 through 17 of a macro field word. When the call has a nonzero paraddr argument, the call places the parameter address for the paraddr argument in the next macro field word.
- A MACRO call with a 4L JUMP or 5LR JUMP macname argument places the appropriate macro code in bits 0 through 17 of a macro field word, and places the paraddr argument in bits 18 through 59 of the same word (figure 2-1, word 14).
- A MACRO call with a nonzero tag argument generates code in two or three words and is used to begin macro string segments that will be entered with RJUMP macros.

  The first word contains the macro code for a NOOP macro (020000_8) in bits 0 through 17 and the tag argument (with the leftmost bit unconditionally set) in bits 18 through 59 (figure 2-1, word 11). This reserves a word for storing a return address and the macro code for a JUMP macro.

  The second word contains the macro code for the macname argument in bits 0 through 17 (figure 2-1, word 12).

  A third word contains the parameter address if the MACRO call had a nonzero paraddr argument (figure 2-1, word 13).

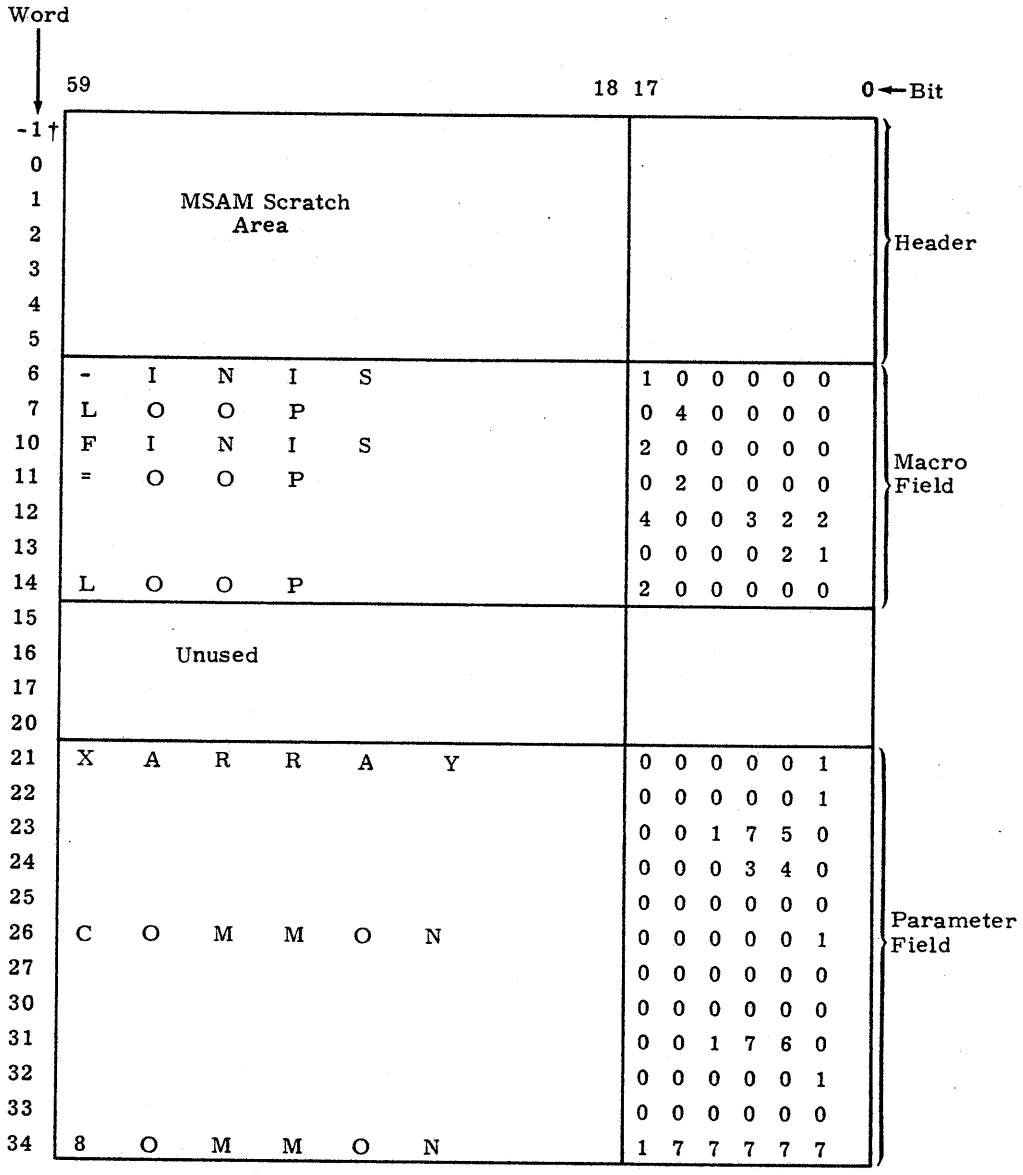
MSAM automatically loads the first word of each macro field (word 6) with a HALT macro (100000_8) labeled - INIS. The END macro places in the macro string a JUMP macro that transfers control to the HALT macro at word 6.

PARAMETER FIELD

In unpacked form, each 60-bit word of the parameter field contains a parameter value or a pointer to a parameter value in bits 0 through 17, and space for a tag in bits 18 through 59. Upon receipt of a MACRO call, MSAM examines the macname argument and determines the parameter count for the macro. MSAM then searches for the parameter location specified by the paraddr argument, transfers the parameter to the parameter field, and continues transferring subsequent parameters until the parameter count is satsified.

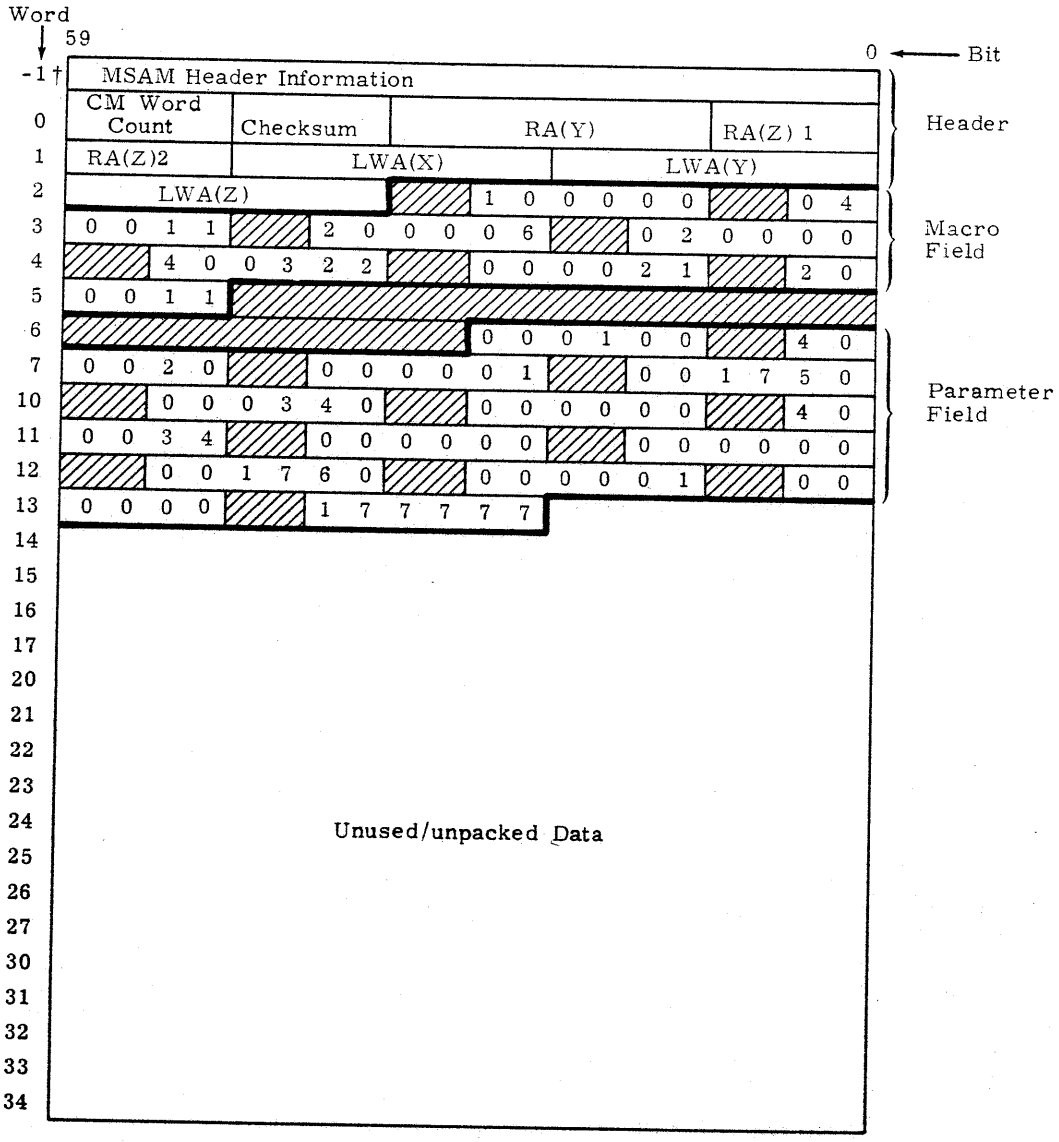
MSAM does not satisfy indirect references to parameter values (that is, values tagged by MPARAM calls) until the macro string is packed.

When it knows that a parameter specifies a data storage address, MAP examines parameter bits 10 and 11 for 0 (section x), 1 (section y), or 2 (section z). MAP determines the address by adding the contents of header word 0, 1, or 2 to the contents of the parameter field word specified by bits 0 through 9 of the parameter.



†The first word is considered -1 as it is not packed or loaded into macro memory.

Figure 2-1. Unpacked Macro String Buffer



†The first word is considered -1 as it is not packed or loaded into macro memory.

Figure 2-2. Packed Macro String Buffer

EXECUTION SEQUENCE

All macro strings begin execution at word 7. The macro string shown in figures 2-1 and 2-2 executes as follows:

1. Execution begins at word 7 with an RJUMP macro that loads a return address (10) and JUMP macro in =OOP and then transfers control to =OOP+1. =OOP is the tag argument (with the leftmost bit set) of the MACRO call that placed the LOADL32 (400322_8) macro in word 12.
2. The LOADL32 macro causes MAP to fetch the parameter block whose address is at word 13, perform the load operation, and return control to the JUMP macro at word 14.
3. The JUMP macro at word 14 transfers control to =OOP.
4. =OOP now contains a JUMP macro that transfers control to word 10.
5. Word 10 contains a JUMP macro that transfers control to -INIS.
6. -INIS contains a HALT macro that stops execution.

MSAM STATUS

Each MSAM call (except MDUMP and MDRLSE) has a status argument that specifies a location to receive MSAM status. Since MSAM does not clear the status word, the user program should clear the status word after each MSAM call unless cumulative status is desired. MSAM status bits are defined as follows:

Bit(s)	Description
0, 1	Unused.
2	Illegal array name.
3	No controlware address; assume default controlware.
4	No controlware name; assume default controlware.
5	MET undefined.
6	MET closed.
7	Macro string buffer too small.
8	Symbol defined twice.
9	Unused.
10	Array overflow into previously defined field.
11	Base array undefined.
12	Equivalent array exceeds base array field length.
13	Undefined or previously packed macro string.
14	Undefined macro.
15	Illegal data in macro parameter field.
16	MAP not available.
17	Macro string executed through different MET (informative).
18	Array overflows x data storage (informative).
19	Array overflows y data storage (informative).
20	Array overflows z data storage.
21	Undefined label.
22	MCLOSE attempted on closed MET.
23	MSAM call error.
24	Macro string buffer too large; assume macro memory size plus 1.
25	METOPEN attempted on open MET.
26	Parameter block contains indirect parameter reference or illegal parameter (informative).
27	Indirect parameter block too large for macro string buffer.
28	Illegal macro label.
29	ECS unload array last word address not multiple of 8.
30-59	Unused.

An installation parameter classifies MSAM status bits as fatal or nonfatal. The user program aborts when MSAM detects an invalid argument count or when MSAM detects more fatal errors than allowed by the errlim argument of the METOPEN call.

MSAM CALLS

The following calls allow user programs to control the MAP III system.

METOPEN

Establishes MAP environment table (MET) within user's field length. MET contains pointers, code/status values, and error information used by MAP PP driver and central processor monitor. A user program can define more than one MET, and more than one macro string can be associated with the same MET. Figure 2-3 shows MET format.

When user specifies new controlware, METOPEN transfers controlware to ECS and records ECS address of controlware in MET. Refer to 6000 MAP III Assembler ERS listed in preface for controlware structure.

Sequence:

CALL METOPEN (met, symtable, controlware, conaddr, status [, errlim])

met	MET name.
symtable	Symbol table name.
controlware	0 (use default controlware) or left-justified, zero-filled name of local file containing user-supplied controlware.
conaddr	0 (default controlware) or destination ECS address (within user's field length) for user-supplied controlware.

status	Location to receive MSAM status.	Examples:
errlim	Number of fatal assembly errors allowed before aborting job.	CALL METOPEN (MET1, SYM, 0, 0, STAT (1), 2) CALL METOPEN (MET2, ARRAYS, 5LMYLIB, LIBLOC, ST)

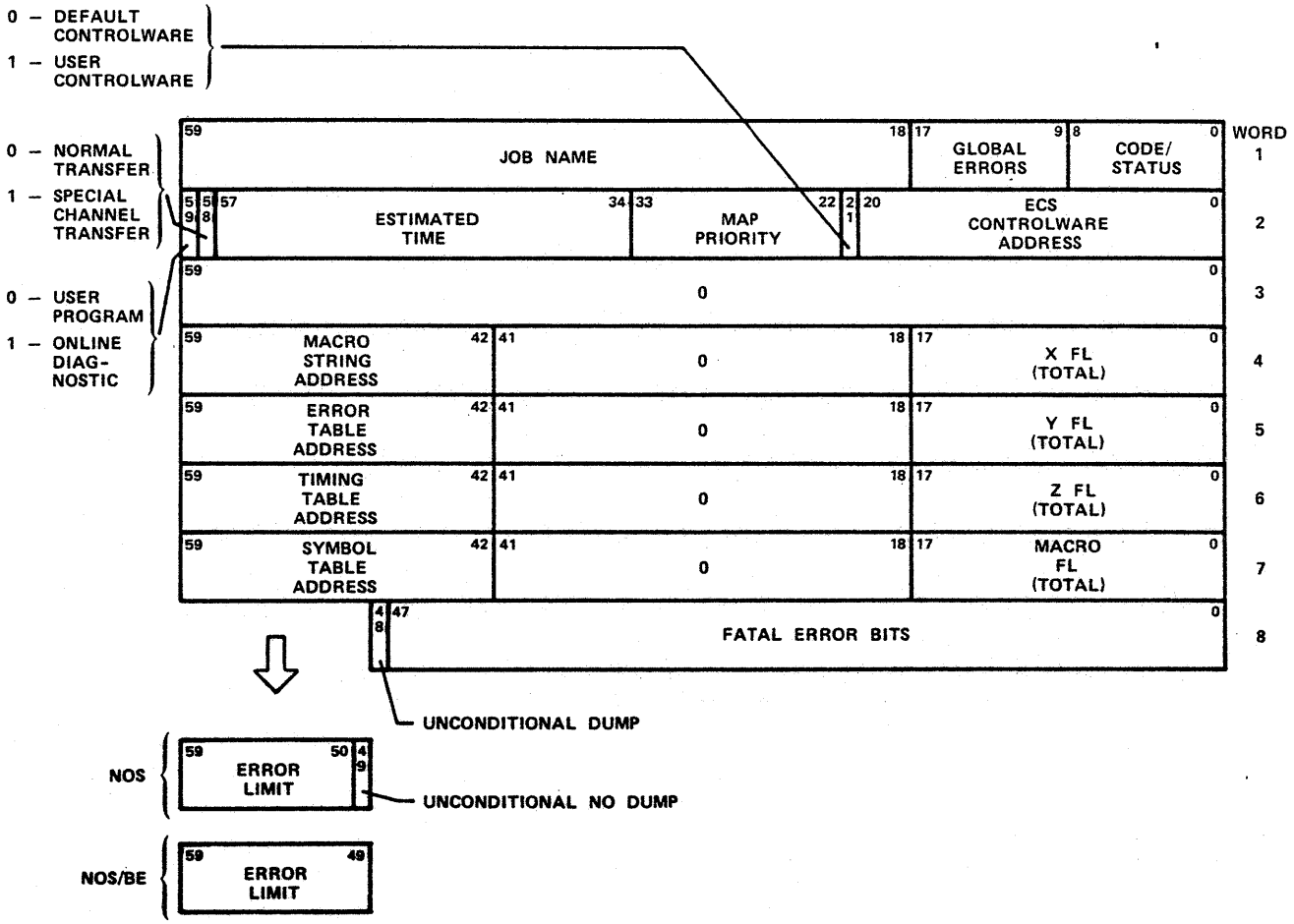


Figure 2-3. MET Format

MAPSET

Zero-fills macro string buffer and restores header information.

Sequence:

CALL MAPSET (met, macbuf, length, status)

met	MET for this macro string.
macbuf	Macro string buffer name.
length	Number of words in macro string buffer.
status	Location to receive MSAM status.

Example:

CALL MAPSET (METBUF, STRING, 100, STAT)

MALLOT

Defines and assigns name to MAP data storage array, thereby enabling macro string parameters to reference array by name. Individual array elements may be referenced by subscript in same manner as individual parameters in tagged parameter block (refer to MPARAM call description).

Sequence:

CALL MALLOT (met, aryname, maplen, 0, mem, status)

met	MET name.
aryname	Left-justified, zero-filled array name (one to seven characters, first must be alphabetic).
maplen	Number of elements in array.
0	Required, but unused.
mem	1Ln where n is section (x, y, z) of MAP data storage to store first word of array.
status	Location to receive MSAM status.

Examples:

CALL MALLOT (MET2, 6LARRAY1, 1000, 0, 1LX, STAT(6))

CALL MALLOT (MYMET, 6LFILTER, FTLEN, 0, 1LZ, ERRORS)

MEQUIV

Defines and assigns name to array within previously defined MAP data storage array. New array must not extend beyond last word of previously defined array.

Sequence:

CALL MEQUIV (met, equivname, maplen, basearray, offset, status)

met	MET name.
equivname	Left-justified, zero-filled, new array name (one to seven characters, first must be alphabetic).
maplen	Number of elements in new array.
basearray	Previously defined array name.
offset	Number of locations between first element of previously defined array and first element of new array.
status	Location to receive MSAM status.

Examples:

CALL MEQUIV (JOBMET, 3LIAA, 200, 2LIA, 800, STATUS)

CALL MEQUIV (MYMET, 6LCOSINE, 512, 4LSINE, 256, STAT(12))

MACRO

Places macro and associated parameters into macro string.

Sequence:

CALL MACRO (macstr, tag, macname, paraddr, status)

macstr	Macro string name.
tag	Unused (0) unless macname begins a macro sequence entered by an RJUMP or JUMP macro, in which case tag is a left-justified, zero-filled label (one to seven characters, first must be alphabetic) for a NOOP macro used to reserve the location immediately preceding the location to hold macname.

macname	Left justified, zero-filled macro mnemonic.
paraddr	Unused (0) for NOOP, HALT, and END macros. Jump address for JUMP or RJUMP macros. Name of first parameter for other macros.
status	Location to receive MSAM status.

Examples:

```
CALL MACRO (MACS, 0, 6LFILTER, PS(1),
  ST(10))

CALL MACRO (STRING1, 4LLOOP, 7LLOADP32,
  PARS(1), ERRS)

CALL MACRO (STRING1, 0, 5LRJUMP, 4LLOOP,
  ERRS)

CALL MACRO (MACSTR, 0, 3LEND, 0, ST)
```

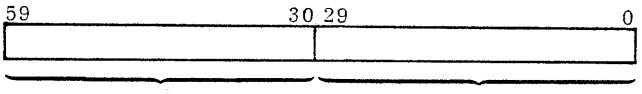
MPARAM

Assigns name to and places common parameter block in macro string. When block contains more than one parameter, second through last parameters may be referenced by subscripting parameter name using .OR. expression. For example, second parameter in block tagged 6LCOMMON can be referenced as 6LCOMMON.OR.2.

Sequence:

```
CALL MPARAM (macstr, tag, value, length,
  status[, loc])
```

macstr	Macro string name.
tag	Left-justified, zero-filled label (one to seven characters, first must be alphabetic) by which parameter block is referenced.
value	First value in parameter block. value must not be an array name or the name of a parameter block.
length	Number of parameters in block.
status	Location to receive MSAM status.
loc	(Optional) Location to receive position of first parameter in packed macro string. Interpret loc contents as follows:



Right-justified, end-around, left-shift count that right-justifies first parameter in 60-bit word. When left-shift count is 12, parameter bits 0 through 11 are in bits 48 through 59 of the word and parameter bits 12 through 18 are in bits 0 through 5 of the previous word.	Number of 60-bit words between macro string word 0 and word containing all or part of first parameter.

Examples:

```
CALL MPARAM (MAC1, 7LADMASK1, 177777B,
  1, STAT(15))

CALL MPARAM (MS2, 7LNPOINTS, NPTS, 1,
  ERRS, PACKADD)

CALL MPARAM (STRING, 6LCOMMON, COM(1),
  6, ST)
```

MAPNOGO

Satisfies macro string tag references and packs macro string, but does not request macro string execution.

Sequence:

```
CALL MAPNOGO (met, macstr, status[, ref])
```

met	MET name.
macstr	Macro string name.
status	Location to receive MSAM status.
ref	(Optional) Argument (any value) that returns symbol table and data storage map in program listing (refer to Example Programs in section 4).

Examples:

```
CALL MAPNOGO, (MET3, STRING, ERR)

CALL MAPNOGO, (MYMET, MAC, STAT, 1)
```

MAPGO

Requests macro string execution. If macro string is not packed, MAPGO satisfies macro string tag references and packs macro string before requesting execution.

Sequence:

CALL MAPGO (met, macstr, timtable, errtable, recall, estime, status[, febits])

met	MET name.
macstr	Macro string name.
timtable	Timing table name (refer to section 4) or 7LNOTABLE if no table is desired.
errtable	Error table name (refer to section 4) or 7LNOTABLE if no table is desired.
recall	Nonzero: Suspend program execution until macro string completes execution. 0: Continue program execution.
estime	Estimated macro string execution time in milliseconds. estime must not exceed maximum time set by installation parameter.
status	Location to receive MSAM status.
febits	(Optional) 48-bit pattern plus dump bit corresponding to four MAP status words described in table 2-1. MAP status bits having corresponding febits bits set are defined as fatal error bits. Bit 48 of febits causes MAP PP driver to transfer MAP data storage to ECS dump area after macro string completes execution. Default febits pattern defines each MAP status bit as fatal and specifies that no dump occur.

Examples:

```
CALL MAPGO (MET1, MS2, TIM, ERR, 0,
  ESTIME, STAT)

CALL MAPGO (MET, STRING, 7LNOTABLE,
  7LNOTABLE, 1LR, EST, ST(17),
  1777701040617000 7B)
```

TABLE 2-1. MAP STATUS WORDS

MAP Status Word	febits Bit	Description
Memory error	0	Data storage access A parity error.
	1	Data storage access B parity error.
	2	Data storage access C parity error.
	3	Subcontrol memory parity error.
	4	Control memory parity error.
	5	Macro memory parity error.
	6	Access A address out of range.
	7	Access B address out of range.
	8	Access C address out of range.
	9	ECS parity error.
	10	ECS abort.
	11	ECS field length error.
Arithmetic error 3	12	Negative square root (unit 1).
	13	Negative square root (unit 2).
	14	Numerical conversion unit overflow.
	15	Numerical conversion unit underflow.
	16	
	17	
	18	
	19	
	20	Unused.
	21	
	22	
	23	
Arithmetic error 2	24	Overflow in multiply unit 1.
	25	Overflow in multiply unit 2.
	26	Overflow in multiply unit 3.
	27	Overflow in multiply unit 4.
	28	Underflow in multiply unit 1.
	29	Underflow in multiply unit 2.
	30	Underflow in multiply unit 3.
	31	Underflow in multiply unit 4.
	32	Overflow in divide unit 2.
	33	Underflow in divide unit 2.
	34	Divide 0 by 0, divide unit 2.
	35	Divide by 0, divide unit 2.
Arithmetic error 1	36	Overflow in add/subtract unit 1.
	37	Overflow in add/subtract unit 2.
	38	Overflow in add/subtract unit 3.
	39	Overflow in add/subtract unit 4.
	40	Underflow in add/subtract unit 1.
	41	Underflow in add/subtract unit 2.
	42	Underflow in add/subtract unit 3.
	43	Underflow in add/subtract unit 4.
	44	Overflow in divide unit 1.
	45	Underflow in divide unit 1.
	46	Divide 0 by 0, divide unit 1.
	47	Divide by 0, divide unit 1.

TABLE 2-1. MAP STATUS WORDS (Contd)

MAP Status Word	febits Bit	Description
-	48	Unconditional dump flag.
	49	
	50	
	51	
	52	
-	53	Unused.
	54	
	55	
	56	
	57	
	58	
	59	

MODIFY

Replaces first value of MPARAM-defined common parameter block with new value, then recomputes checksum and stores new checksum in header.

Sequence:

CALL MODIFY (macstr, loc, value, status)

macstr	Macro string name.
loc	Parameter position returned to location specified by loc argument of MPARAM call.
value	New value for parameter. Relative ECS limit addresses for ECS load/unload macros should be adjusted as follows: ECS load macro: Add 48 to value and increment result until result is multiple of 8. ECS unload macro: Add 1 to value and increment result until result is multiple of 8.
status	Location to receive MSAM status.

Examples:

CALL MODIFY (CHGMAC, WORD, NEWVAL, ST)

CALL MODIFY (MAC, RET, 177777B, STAT(6))

MCLOSE

Ensures that MAP activity associated with MET is complete, closes MET, and clears schedule table entry associated with MET.

Sequence:

CALL MCLOSE (met, status)

met	MET name.
status	Location to receive MSAM status.

Example:

CALL MCLOSE (MET2, ST(25))

MRECALL

Suspends program execution until MAP finishes processing macro string.

Sequence:

CALL MRECALL (met, status)

met	MET name.
status	Location to receive MSAM status.

Example:

CALL MRECALL (MYMET, ERRS)

MRESET

Reinitializes symbol table for MET so that previously packed and newly generated macro strings can execute from same MET.

Sequence:

CALL MRESET (met, status)

met	MET name.
status	Location to receive MSAM status.

Example:

CALL MRESET (MET, ST)

MDUMP

Transfers ECS dump information to file OUTPUT for printing. Transferred information includes contents of MAP status registers, register files, and macro memory as well as information specified by MDUMP call arguments. (Refer also to MDUMP Control Card in section 4.)

Sequence:

CALL MDUMP (x, y, z, c)

x, y, z, c	0: Do not include following MAP information in dump. nonzero: Include following MAP information in dump. x; data storage section x contents. y; data storage section y contents. z; data storage section z contents. c; control and subcontrol memory contents.

Example:

CALL MDUMP (X, Y, Z, 0)

MDRLSE

Releases MAP ECS dump area without printing it and clears dump area interlock word.

Sequence/Example:

CALL MDRLSE

This section describes standard and optional macros that may be placed in a macro string by means of the MACRO call described in section 2. A macro may be placed in a macro string only if the microcode for the macro is part of the controlware specified by the METOPEN call that establishes the MET for the macro string.

Macros are referenced in this section by their mnemonics. Appendix B contains a summary that provides the full name for each macro.

MACRO CATEGORIES

Macros are mnemonics that represent and call into execution MAP microcode sequences. Macros are divided into the following categories.

- Control/pseudo macros determine control flow within a macro string.
- ECS input/output macros transfer data between ECS and MAP data storage.
- Arithmetic macros perform various array calculations, many of which involve two operand arrays and a result array. Generally, MAP operates most efficiently when each of the three arrays resides in a separate section of data storage. Refer to individual macro descriptions for more specific information.

MACRO PARAMETERS

Appendix A contains tables that define parameters required for those macros that use parameters.

CONTROL/PSEUDO MACROS

NOOP

Transfers control to next macro.

JUMP

Transfers control to macro memory location specified by paraddr argument of MACRO call.

RJUMP

Allows control to transfer from original macro sequence to secondary sequence, then back to original sequence. Secondary sequence entry location is specified by paraddr argument of MACRO call.

RJUMP adds 1 to current macro memory program address, places result in JUMP macro, stores JUMP macro at address paraddr, and transfers control to address paraddr+1. Sequence starting at address paraddr+1 returns control to original sequence by jumping to address paraddr.

HALT

Stops macro execution. All macro strings should terminate with HALT.

END

Causes jump to HALT macro (labeled-INIS) automatically stored by MSAM at first location of macro field (location 6). User can terminate macro string without END by inserting HALT macro(s) where appropriate.

UPM

Replaces or adds value to common parameter(s) defined by MPARAM call. This allows parameters to be modified during macro string execution. Table A-2 defines UPM parameters.

TMM

Compares value to common parameter defined by MPARAM call. Transfers control to next location when values are not equal. Transfers control to next location plus one when values are equal. Table A-2 defines TMM parameters.

XMM2DM/XDM2MM

Allows integer values between 131 071 and -131 072 to be exchanged between MAP macro memory and data storage as follows:

Macro	Transfer	FLAG Parameter
XMM2DM	Macro memory-to-data storage.	0
XDM2MM	Data storage-to-macro memory.	1

Macro memory-to-data storage transfer can be part of sequence for transferring macro memory data to ECS.

Table A-12 defines parameters for these macros. Each macro automatically selects FLAG parameter. AU parameter points to first macro memory location to supply or receive data (use MPARAM call to link AU parameter to first transfer location). D FWA, D OFF, and D IF parameter values must each be 0.

ECS INPUT/OUTPUT MACROS

LOADP32

Transfers data from ECS to MAP using format shown in figure 3-1. Table A-3 defines LOADP32 parameters.

UNLDP32

Transfers data from MAP to ECS using format shown in figure 3-1. Table A-4 defines UNLDP32 parameters.

LOADP30

Transfers data from ECS to MAP according to following format.

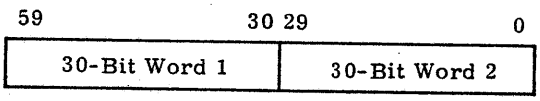


Table A-3 defines LOADP30 parameters.

UNLDP30

Transfers data from MAP to ECS according to following format.

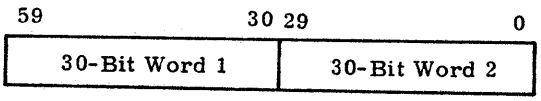


Table A-4 defines UNLDP30 parameters.

LOADL32

Transfers data from ECS to MAP according to following format.

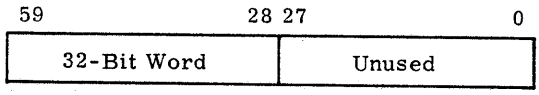


Table A-3 defines LOADL32 parameters.

NOTE

The following UNLDL32 and UNLDR32 macros require approximately twice as much execution time per word as their LOADL32 and LOADR32 counterparts. When possible, use the UNLDP32 or UNLDP30 macro to transfer data from MAP to ECS.

UNLDL32

Transfers data from MAP to ECS according to following format.

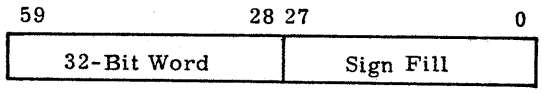


Table A-4 defines UNLDL32 parameters.

LOADR32

Transfers data from ECS to MAP according to following format.

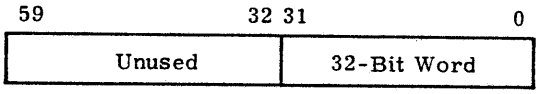


Table A-3 defines LOADR32 parameters.

UNLDR32

Transfers data from MAP to ECS according to following format.

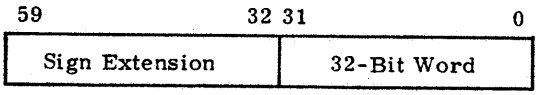
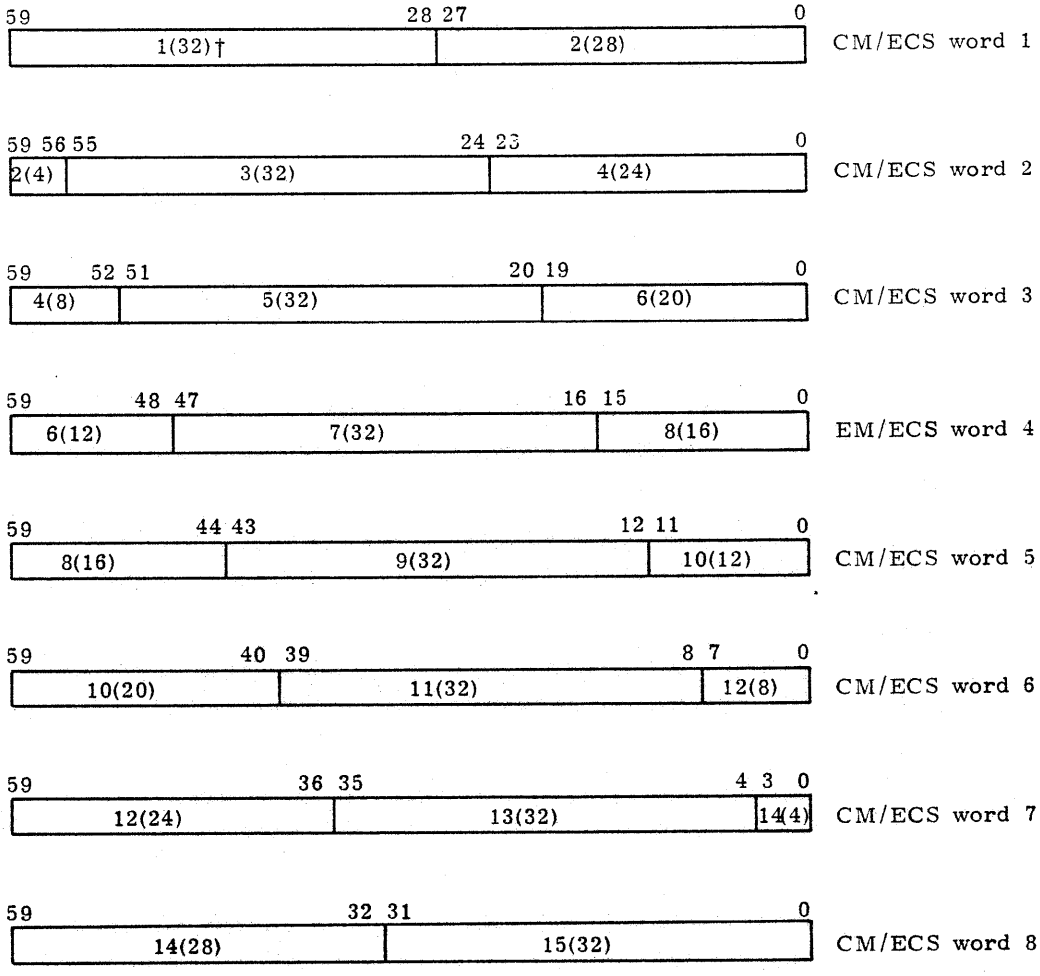


Table A-4 defines UNLDR32 parameters.



†aa(bb) = bb bits of 32-bit word aa

Figure 3-1. Packed 32-Bit Format

ARITHMETIC MACROS

SUMPROD

Performs correlation or convolution operation on filter array $A_0, A_1, \ldots, A_{LA-1}$ and trace array $B_0, B_1, \ldots, B_{LB-1}$, yielding result array $C_0, C_1, \ldots, C_{LC-1}$ defined by:

$$C_j = \sum_{i=0}^{LA-1} A_i B_{j+si+shift}$$

LA, LB, and LC are positive integers and $LB \geq LA > 10$.

Correlation occurs when s is 1 and convolution occurs when s is -1. shift is a positive, negative, or zero integer that specifies an initial shift of array A with respect to array B as shown in tables 3-1 and 3-2.

SUMPROD assumes that arrays A and B are zero outside their domains of definition, so pad zeros do not have to be included.

MAP executes SUMPROD most efficiently when each array resides in a separate section of data storage.

Table A-5 defines SUMPROD parameters.

TABLE 3-1. SUMPROD WITH POSITIVE shift

s, shift, j	Array Relationship For Products	Cj
+1,0,0	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 0 0 0 0 A_0 A_1 A_2 A_3 A_4 A_5	$A_0B_0+A_1B_1+A_2B_2+\cdots$
+1,0,1	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 0 0 0 0 0 A_0 A_1 A_2 A_3 A_4	$A_0B_1+A_1B_2+A_2B_3+\cdots$
-1,0,0	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 A_4 A_3 A_2 A_1 A_0 0 0 0 0 0	A_0B_0
-1,0,1	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 A_5 A_4 A_3 A_2 A_1 A_0 0 0 0 0	$A_0B_1+A_1B_0$
+1,2,0	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 0 0 0 0 0 0 A_0 A_1 A_2 A_3	$A_0B_2+A_1B_3+A_2B_4+\cdots$
+1,2,1	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 0 0 0 0 0 0 0 A_0 A_1 A_2	$A_0B_3+A_1B_4+A_2B_5+\cdots$
-1,2,0	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 A_6 A_5 A_4 A_3 A_2 A_1 A_0 0 0 0	$A_0B_2+A_1B_1+A_2B_0$
-1,2,1	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0 0 0	$A_0B_3+A_1B_2+A_2B_1+A_3B_0$

LA-1

$$C_j = \sum_{i=0} A_i B_{j+si+shift}$$

TABLE 3-2. SUMPROD WITH NEGATIVE shift

s, shift, j	Array Relationship For Products	Cj
+1,-2,0	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 0 0 A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7	$A_2B_0+A_3B_1+A_4B_2+\cdots$
+1,-2,1	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 0 0 0 A_0 A_1 A_2 A_3 A_4 A_5 A_6	$A_1B_0+A_2B_1+A_3B_2+\cdots$
+1,-2,2	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 0 0 0 0 A_0 A_1 A_2 A_3 A_4 A_5	$A_0B_0+A_1B_1+A_2B_2+\cdots$
-1,-2,0	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 A_2 A_1 A_0 0 0 0 0 0 0 0	0
-1,-2,1	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 A_3 A_2 A_1 A_0 0 0 0 0 0 0	0
-1,-2,2	0 0 0 0 B_0 B_1 B_2 B_3 B_4 B_5 A_4 A_3 A_2 A_1 A_0 0 0 0 0 0	A_0B_0

$$C_j = \sum_{i=0}^{LA-1} A_i B_{j+si+shift}$$

STKMOVE

Performs stack operation (array sum) or move operation (array relocation) on arrays $A_0, A_1, \ldots,$ A_{LC-1} and $B_0, B_1, \ldots, B_{LC-1}$, yielding result array $C_0, C_1, \ldots, C_{LC-1}$ defined by:

$C_i=A_i+B_i$ (stack)

$C_i=A_i$ (move)

LC is a positive integer.

User need not supply zero-filled B array for move operation. MAP executes STKMOVE most efficiently when each array resides in a separate section of data storage.

NOTE

Some moves that overlap old and new list locations require negative increment factors. For example, to transfer a list from locations 300 through 399 to locations 350 through 449, set array first word addresses to 399 and 449, and set the increment factor for each array to -1.

Table A-6 defines STKMOVE parameters.

CPLXFFT

Performs fast Fourier transformation (FFT) on complex series $B_0, B_1, \ldots, B_{N-1}$, yielding complex series $C_0, C_1, \ldots, C_{N-1}$ defined by:

$$C_k = \sum_{j=0}^{N-1} B_j W^{-jk}$$

N is 2^n and n is an integer not less than 3.

i is $\sqrt{-1}$.

W is exp $(2 \pi i/N)$.

k is $0, 1, \ldots, N-1$.

Real parts of B_j, imaginary parts of B_j, and sine/cosine table must each reside in separate sections of MAP data storage. After CLPXFFT executes, real and imaginary parts of each C_k occupy locations initially occupied by real and imaginary parts of B_k.

Table A-7 defines CPLXFFT parameters.

ICPXFFT

Inverses process performed by CPLXFFT. ICPXFFT performs FFT on complex series $C_0, C_1, \ldots, C_{N-1}$, yielding complex series $B_0, B_1, \ldots, B_{N-1}$ defined by:

$$B_j = (1/N) \sum_{k=0}^{N-1} C_k W^{jk}$$

N, W, k are as defined for CPLXFFT and j is $0, 1, \ldots, N-1$.

Real parts of C_k, imaginary parts of C_k, and sine/cosine table must each reside in separate sections of MAP data storage. After ICPXFFT executes, real and imaginary parts of each B_j occupy locations initially occupied by real and imaginary parts of C_j.

Table A-7 defines ICPXFFT parameters.

REALFFT

Performs FFT on real series $R_0, R_1, \ldots, R_{N-1}$, yielding data to form complex series $C_0, C_1, \ldots, C_{N-1}$ defined by:

$$C_k = \sum_{j=0}^{N-1} R_j W^{-jk}$$

N is 2^n and n is an integer not less than 4.

M (figure 3-2) is N/2.

i is $\sqrt{-1}$.

W is exp $(2 \pi i/N)$.

k is $0, 1, \ldots, N-1$.

Figure 3-2 shows MAP data storage placement of real series before execution and first M+1 points of complex series after execution. Sine/cosine table resides in remaining section of data storage. Following relationship may be used to obtain remaining points of complex series.

$$C_k = \overline{C_{N-k}}$$

$M < k < N$.

$\overline{}$ Denotes complex conjugate (if C is x+iy, $\overline{C}$ is x-iy).

Table A-8 defines REALFFT parameters.

INVRFFT

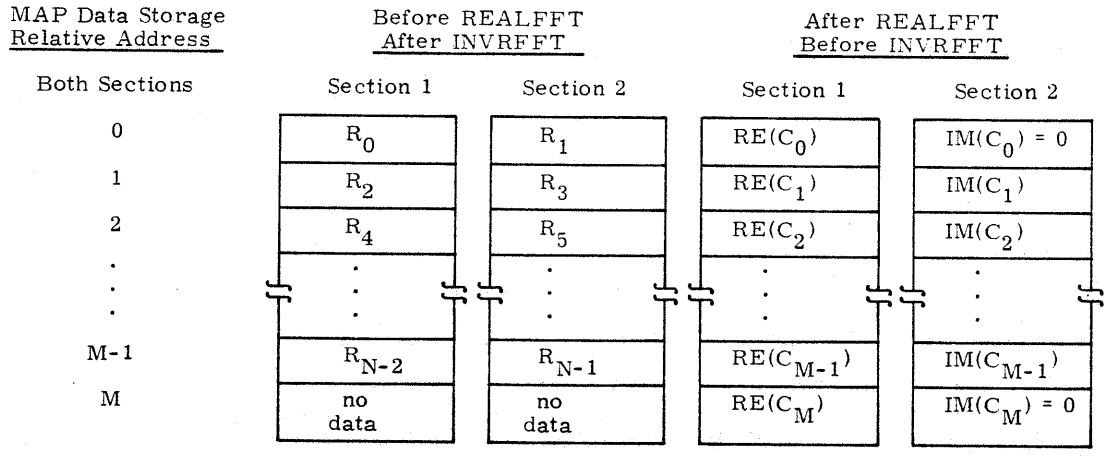
Inverses process performed by REALFFT. INVRFFT performs FFT on complex series $C_0, C_1, \ldots, C_{N-1}$, yielding real series $R_0, R_1, \ldots, R_{N-1}$ defined by:

$$R_k = (1/N) \sum_{j=0}^{N-1} C_j W^{jk}$$

N, W, k are as defined for REALFFT.

C_0 and C_M must be real and user should supply only complex numbers $C_0, C_1, \ldots, C_M$. Figure 3-2 shows MAP data storage placement of first M+1 points of complex series before execution and all N points of real series after execution. Sine/cosine table resides in remaining section of data storage.

Table A-9 defines INVRFFT parameters.



NOTES: R_i = i^{th} real point
$RE(C_i)$ = real part of i^{th} complex point
$IM(C_i)$ = imaginary part of i^{th} complex point

Figure 3-2. REALFFT/INVRFFT Data Storage Use

FILTER

Uses Weiner-Levinson algorithm to solve Weiner-Hopf equation

$$\begin{bmatrix} r_0 & r_1 & r_2 & r_3 & \cdots & r_n \\ r_1 & r_0 & r_1 & r_2 & \cdots & r_{n-1} \\ r_2 & r_1 & r_0 & r_1 & \cdots & r_{n-2} \\ r_3 & r_2 & r_1 & r_0 & \cdots & r_{n-3} \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ r_n & \cdot & \cdot & \cdot & \cdots & r_0 \end{bmatrix} \begin{bmatrix} F_0 \\ F_1 \\ \cdot \\ \cdot \\ \cdot \\ F_n \end{bmatrix} = \begin{bmatrix} G_0 \\ G_1 \\ \cdot \\ \cdot \\ \cdot \\ G_n \end{bmatrix}$$

for array F_i where arrays r_i and G_i are given. To solve Weiner-Hopf equation, FILTER first solves auxiliary equation

$$\begin{bmatrix} r_0 & r_1 & r_2 & r_3 & \cdots & r_n \\ r_1 & r_0 & r_1 & r_2 & \cdots & r_{n-1} \\ r_2 & r_1 & r_0 & r_1 & \cdots & r_{n-2} \\ r_3 & r_2 & r_1 & r_0 & \cdots & r_{n-3} \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ r_n & \cdot & \cdot & \cdot & \cdots & r_0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_n \end{bmatrix} = \begin{bmatrix} a_n \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$$

for array a_i where array a_i is prediction error operator for unit prediction distance with $a_0=1$, and a_n is expected error for n+1 element operator. FILTER then uses array a_i from second equation to solve first equation for array F_i.

FILTER parameters accommodate following processing options.

- When IFSPIKE parameter is zero, FILTER stops when second equation is solved. User program may then examine array a_i to determine whether FILTER should solve first equation.

  When IFSPIKE parameter is nonzero, FILTER solves both equations.

- When IFSTABL parameter is zero, FILTER checks array a_i during generation to determine whether array F_i is stable. If array F_i is unstable, FILTER returns code and length of stable portion of array F_i in parameter-specified locations. Arrays F_i and A_i contain only elements corresponding to stable portion of array F_i.

  When IFSTABL parameter is nonzero, FILTER does not check stability.

- Length MS and length MD can be used to save FILTER execution time after trial execution. For example, assume user has 200-element F_i array to calculate but elects to calculate first 20 elements, perform check, and then conditionally calculate remaining 180 elements. For first calculation, user loads 200-element r_i and G_i arrays, sets MD parameter to 20, and sets MS parameter to zero. For second calculation, user ensures that all arrays and three-word ALPHA buffer are intact, sets MD parameter to 200, and sets MS parameter to 20. Upon receipt of second request, FILTER starts calculating at twenty-first element using previously calculated elements.

FILTER also checks for conditions indicating that array F_i is singular. When it detects singular condition, FILTER returns code in parameter-specified location.

Table A-10 defines FILTER parameters.

NMO

Accepts seismic trace and produces output trace corrected for normal moveout. NMO reports apparently muted (zero) values in input trace, allows for muting front portion of output trace, and operates in alternate mode to produce output trace for later input to velocity analysis programs.

NMO transfers to the output trace input trace samples selected by addressing equation

$$I_j = \frac{\sqrt{(D*V_i) + (j*T_r)^2} - T_1}{T_r}$$

T_r is input trace sample rate (nominally milliseconds).

T_1 is time of first input trace sample (same unit as T_r).

V_i is sample i (i and j may differ) of array containing squared inverse velocity values. (If D is in feet and T_r is in milliseconds, array V is in milliseconds per foot squared.)

D is squared offset distance corresponding to surface position of input trace.

I_j is unrounded j^{th} address of input trace array. NMO converts I_j to integer by rounding up, then stores sample from I_j at location j of output trace array.

User can employ parameter K2 and BETA array value THRESH to mute (clear) beginning of output trace. THRESH determines index J_{THRESH}, which is last output trace index where two consecutive arrival times differ by less than THRESH. Arrival time, T, is computed by:

$$T = \sqrt{(D*V_i) + (j*T_r)^2}.$$

NMO mutes output trace by clearing first K samples.

$$K = \max (K2, J_{THRESH}).$$

Table A-11 defines NMO parameters, velocity function list, and BETA array.

CVEC/NVEC/MVEC/NMVEC

Perform following calculations.

Macro	Calculation	FLAG Parameter	Equivalent FORTRAN Statement
CVEC	Copy vector	0	D(I)=A(I)
NVEC	Negate vector	1	D(I)=-A(I)
MVEC	Magnitude vector	2	D(I)=ABS[A(I)]
NMVEC	Negative magnitude vector	3	D(I)=-ABS[A(I)]

Table A-12 defines parameters for these macros. Each macro automatically selects appropriate FLAG parameter. AU parameter value of 1, 2, 3, 4 selects corresponding MAP add/subtract unit.

ADDVEC/SUBVEC/MULVEC/DIVVEC

Perform following calculations.

Macro	Calculation	FLAG Parameter	Equivalent FORTRAN Statement
ADDVEC	Add vectors	0	D(I)=A(I)+B(I)
SUBVEC	Subtract vectors	1	D(I)=A(I)-B(I)
MULVEC	Multiply vectors	2	D(I)=A(I)*B(I)
DIVVEC	Divide vectors	3	D(I)=A(I)/B(I)

Table A-13 defines parameters for these macros. Each macro automatically selects appropriate FLAG parameter. AU parameter value of 1, 2 selects corresponding MAP add/subtract/multiply/divide unit. AU parameter value of 3, 4 selects corresponding add/subtract/multiply unit.

IPVEC

Performs inner product vector calculation defined by:

$$D(final)=D(initial)+ \Sigma\ A(I)*B(I).$$

Table A-13 defines IPVEC parameters. D IF parameter must be set to 0 to produce scalar D(final). FLAG parameter value of 0 clears D(initial). FLAG parameter value of 1 leaves D(initial) undisturbed. AU parameter value of 1, 2, 3, 4 selects corresponding MAP add/subtract and multiply units.

SUMRVEC

Performs sum reduction vector calculation defined by:

$$D(final)=D(initial)+\Sigma A(I).$$

Table A-12 defines SUMRVEC parameters. D IF parameter must be set to 0 to produce scalar D(final). FLAG parameter value of 0 clears D(initial). FLAG parameter value of 1 leaves D(initial) undisturbed. AU parameter value of 1,2,3,4 selects corresponding MAP add/subtract unit.

ZEROVEC/BCASVEC

Perform following calculations.

Macro	Calculation	FLAG Parameter	Equivalent FORTRAN Statement
ZEROVEC	Zero array	0	D(I)=0
BCASVEC	Broadcast scalar	1	D(I)=A

Table A-12 defines parameters for these macros. Each macro automatically selects appropriate FLAG parameter. A IF parameter value must be 0 for BCASVEC. ZEROVEC ignores A FWA, A OFF, and A IF parameters. AU parameter value of 1, 2, 3, 4 selects corresponding MAP add/subtract unit.

MINE/MAXE

Perform following calculations.

Macro	Calculation	FLAG Parameter	Equivalent FORTRAN Statement
MINE	MIN elements	0	D(I)AMIN1[A(I), B(I)]
MAXE	MAX elements	1	D(I)=AMAX1[A(I), B(I)]

Table A-13 defines parameters for these macros. Each macro automatically selects appropriate FLAG parameter. AU parameter value must be 0 for each macro.

SQRTVEC

Performs square root vector calculation equivalent to FORTRAN statement D(I)=SQRT(A(I)).

MAP provides positive root and declares square root error for each negative element of A(I).

Table A-12 defines SQRTVEC parameters. FLAG parameter value must be 0. AU parameter value of 1, 2 selects corresponding MAP square root unit.

MAVVS/MAVSV/MAVVV

Perform following calculations.

Macro	Calculation	FLAG Parameter	Equivalent FORTRAN Statement
MAVVS	Multiply add vector, vector, scalar	0	D(I)=(A(I)*B(I))+C
MAVSV	Multiply add vector, scalar, vector	1	D(I)=(A(I)*C)+B(I)
MAVVV	Multiply add vector, vector, vector	2	D(I)=(A(I)*B(I)) +C(I)

For MAVVS/MAVSV and MAVVV, arrays A, B, D and A, B, C, respectively, should reside in separate sections of MAP data storage.

Table A-14 defines parameters for these macros. Each macro automatically selects appropriate FLAG parameter. For MAVVS and MAVSV, C IF parameter value must be 0. AU parameter value of 1, 2, 3, 4 selects corresponding MAP add/subtract and multiply units.

TVEC

MAP automatically rounds up when converting data storage values to external fixed-point values. TVEC counters this roundup by preprocessing values according to following equivalent FORTRAN statements.

If A(I) .GE. 1.0, then D(I)=A(I)-.5

If A(I) .LE. -1.0, then D(I) = A(I)+.5

If A(I) .LT. 1.0 and A(I) .GT. -1.0, then D(I)=0.0

Table A-12 defines TVEC parameters. FLAG parameter value must be 0. AU parameter must point to macro memory location containing 200_8 (use MPARAM call to link AU parameter to location containing 200_8).

COMVEC

Tests corresponding elements in two arrays, records number of test failures, and records offsets for first elements that fail test. Table A-12 defines COMVEC parameters. FLAG parameter value selects test as follows:

FLAG Parameter	Equivalent FORTRAN Test Statement
0	A(I) . EQ. D(I)
1	A(I) . LE. D(I)
2	A(I) . LT. D(I)

AU parameter points to first of three macro memory locations to hold test results (use MPARAM call to link AU parameter to first test result location). Test result locations are defined as follows:

Location	Contents
First	Number of times test failed.
Second	Absolute offset from A of first A element to fail test.
Third	Absolute offset from D of first D element to fail test.

This section contains miscellaneous programming information related to the MAP III system and provides example programs.

FILE DECLARATION

A user program making MSAM calls should declare file OUTPUT on the PROGRAM card.

FIELD LENGTH ALLOCATION

Because MSAM consists of object-time routines, the user should allocate an additional 3100_8 words of central memory to accommodate these routines.

Also, the user program must dimension buffer areas for MSSI elements that reside in the user field length. The following list provides approximate sizes for these elements. To determine the exact buffer length for a particular macro string, refer to the symbolic reference table and memory map generated by the ref argument of the MAPNOGO call.

MSSI Element	60-Bit Word Requirement
MET	8 per MET.
Symbol table	4 plus 2 for each MALLOT and MEQUIV call.
Macro string buffer	8 plus 14 for each MACRO call plus 2 for each MPARAM call plus an additional 3 for each macro requiring type 3 parameters (table A-14).
Timing table (optional)	64.
Error table (optional)	64.

MDUMP CONTROL CARD

This control card transfers part or all of the ECS dump area to an output file and then prints the file.

MSSI transfers the contents of MAP status registers, register files, macro memory, control memory, subcontrol memory, and data storage to the ECS dump area when requested by the febits argument of a MAPGO call or when a fatal MAP execution error occurs. The user can then copy part or all of the dump information from ECS to an output file with an MDUMP control card (abort situation) or an MDUMP call (unconditional dump).

Once it has loaded the ECS dump area, MSSI prevents writing into the dump area until one of the following occurs.

- The user job issues an MDUMP call or executes an MDUMP control card.
- The user program issues an MDRLSE call.
- The operator issues a MAP, NODUMP. command.

When a user program is likely to result in a MAP dump, the user should ensure that the ECS dump area is released after the dump either by including an MDUMP or MDRLSE call in the program or by including an MDUMP control card in the control card deck.

The MDUMP control card format is:

MDUMP[(X, Y, Z, C)]

All arguments are optional.

X, Y, Z	Print indicated section of MAP data storage.
C	Print MAP control and subcontrol memories.

Transferred information includes the contents of MAP status registers, register files, and macro memory as well as the information specified by MDUMP control card arguments.

MET/MACRO STRINGS

As long as the user allocates sufficient field length, there is no restriction on the number of METs or macro strings employed by a program. A MAPGO call may have any open MET† listed as the met argument. MSAM status bit 17 (informative status) sets when the met argument of a MAPGO call is not the same as the met argument used when building the macro string.

LOCE FUNCTION

The LOCE function may be used to obtain the ECS address of an ECS-resident variable.

†MET referenced must specify proper system library for macro string.

NOTE

Bit 59 of LOCE-returned ECS address is 0.

Format:

loc = LOCE (variable)

loc	Location to receive ECS address of ECS-resident variable.
variable	ECS-resident variable.

Example:

LOCA = LOCE (IBC)

MAP REQUESTS

MSSI uses an ECS-resident schedule table to handle MAP requests. The schedule table contains a fixed number of elements, called requests. A request can reside in one of three chains called empty, inactive, and active. A METOPEN call moves a request from the empty to the inactive chain and assigns the request to the MET identified in the call. A MAPGO call moves the request from the inactive to the active chain for MAP processing. When MAP processing completes, the request returns to the inactive chain.

When a MET is no longer needed, the user program should close the MET with an MCLOSE call to move the associated request from the inactive to the empty chain.

PROGRAM RECALL

MSSI provides two methods of suspending program execution while a macro string executes. The first method is to provide a nonzero value as the recall argument of the MAPGO call that requests macro string execution. This suspends program execution until the macro string completes execution.

The second method is to provide 0 as the recall argument of the MAPGO call, continue program execution as desired, and then issue an MRECALL call to suspend program execution until the macro string completes execution.

MET CODE/STATUS VALUES

MSAM communicates with the CP monitor and MAP PP driver by means of a 9-bit MET code/status value stored in bits 0 through 8 of the first 60-bit word in a MET. Rules for interpreting MET code/status values are:

- An even value less than 200_8 indicates that MSAM has requested action, but the action is not yet complete.
- An odd value less than 200_8 indicates that the last MSAM-requested action is complete.
- A $2xx_8$ value indicates a CP monitor error return.
- A $3xx_8$ value indicates a MAP PP driver error return.

MET code/status values are described as follows:

MSAM REQUEST VALUES

Octal Value	Description
120	Request copy of schedule table.
122	Read ECS dump table.
124	Read ECS partition table.
130	Clear dump table interlock.
140	MET open.
150	MET close.
160	Request active entry.
170	Lock MAP.
172	Unlock MAP (on-line diagnostics only).
174	Down MAP (on-line diagnostics only).
176	Up MAP.

CP MONITOR ERROR RETURN VALUES

Octal Value	Description
211	Illegal MET.
213	MAP locked.
215	Unknown schedule table entry.
217	MAP down.
221	Requested equipment not available.
231	Open/close sequence error.
241	Schedule table full.
251	MET close while active.
261	Illegal function code.
271	Illegal dump request.

MAP PP DRIVER ERROR RETURN VALUES

Octal Value	Description
311	Macro string time limit exceeded.
321	Fatal MAP execution error.
323	Nonfatal MAP execution error.
331	Control point error.
341	User CM or ECS address out of range.
351	Macro string checkword error.
361	MP3-detected hardware error.

TIMING/ERROR TABLES

The MAPGO call allows a user program to define two 64-word tables where MSSI records cumulative timing and error information for each type of macro. When cumulative results are not desired, the user should initialize each table before issuing the MAPGO call defining the tables. Either or both of the tables may be deleted by using 7LNOTABLE as the timtable/errtable argument of the MAPGO call.

Table entries are arranged according to the macro codes listed in table B-1. To obtain the address corresponding to the entry for a macro type, add the macro code to the first address of the table.

For example, if TIME is the timing table first word address, the entry for the ICPXFFT (05_8) macro is at TIME+5. If TITAB is the FORTRAN array name for the timing table array, the entry for the ICPXFFT (05_8) macro is TITAB(6).

The timing table entry format is:

59 — 36	35 — 0
Number of times this macro has executed	Total number of milliseconds (NOS) or quarter-milliseconds (NOS/BE) spent executing this macro

The error table entry format is:

59 — 48	47 — 0
Unused	MAP status as described in table 2-1

EXAMPLE PROGRAMS

The following programs show how MSAM calls are used to generate and execute macro strings.

PROGRAM SOP

This program reads two arrays from separate tapes, performs a sum-of-products calculation on the arrays, and then returns the result to a third tape. Figure 4-1 shows data flow for program SOP.

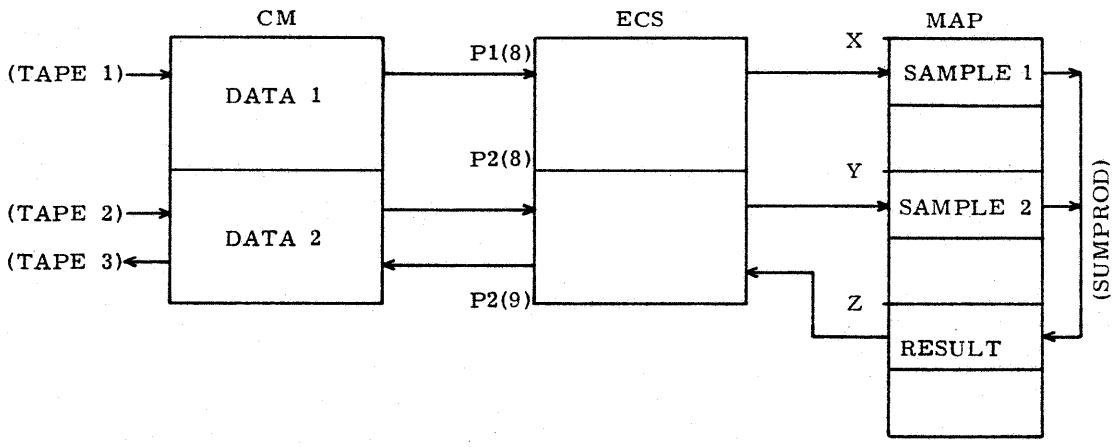


Figure 4-1. SOP Data Flow

```
      PROGRAM SDP (OUTPUT,TAPE1,TAPE2,TAPE3)
C
C     THIS PROGRAM READS 100 SAMPLES EACH FROM TAPE1 AND TAPE2. THEY ARE
C     LOADED INTO MAP DATA MEMORY WHERE A SUM OF PRODUCTS IS PERFORMED
C     AND THE RESULT IS WRITTEN TO ECS.  RESULTS ARE THEN WRITTEN TO
C     TAPE3 PRIOR TO PROCESSING THE NEXT SET OF SAMPLES.
C
C
C     RESERVE BUFFER SPACE FOR TABLES AND ARRAYS

      DIMENSION MET(88), MAC(80), ISYM(10), ITIM(64), IERR(64)
      DIMENSION DATA1(100), DATA2(100), P1(11), P2(11), P3(11), P4(9)
      INTEGER P1, P2, P3, P4
      TST = 0
C
C     DEFINE PARAMETERS FOR LOAD LEFT-JUSTIFIED 32 BIT DATA
C
      P1(1) = 7LSAMPLE1
      P2(1) = 7LSAMPLE2
      P1(2) = P2(2) = 1
      P1(3) = P2(3) = 100
      P1(4) = P2(4) = 16043
      P1(5) = P2(5) = 603
      P1(6) = P2(6) = 1777773
      P1(7) = P2(7) = 0
      P1(8) = 0
      P1(9) = P1(8) + 100
      P2(8) = P1(9) +1
      P2(9) = P2(8) + 100
      P1(10) = P2(10) = 1
      P1(11) = P2(11) = 0
C
C     SET UP PARAMETERS FOR SUM OF PRODUCTS
C
      P3(1) = 7LSAMPLE1
      P3(2) = P3(4) = P3(6) = 1
      P3(3) = 7LSAMPLE2
      P3(5) = 6LRESULT
      P3(7) = P3(8) = P3(9) = 100
      P3(10) = 1
      P3(11) = 0
C
C     SET UP PARAMETERS FOR UNLOAD LEFT-JUSTIFIED 32 BIT DATA
C
      P4(1) = 6LRESULT
      P4(2) = 1
      P4(3) = 100
      P4(4) = 14048
      P4(5) = 77208
      P4(6) = 1777778
      P4(7) = 0
      P4(8) = P2(8)
      P4(9) = (P2(8) + 101)/8 *8 +8
C
C     SET UP MET AND DEFINE ARRAYS
C
      CALL METOPEN (MET, ISYM, 0, 0, IST)
      CALL MALLOT (MET, 7LSAMPLE1, 100, 2LNS, 1LX, IST)
      CALL MALLOT (MET, 7LSAMPLE2, 100, 2LNS, 1LY, IST)
      CALL MALLOT (MET, 6LRESULT, 100, 2LNS, 1LZ, IST)
C
C     CREATE MACRO STRING AND OBTAIN SYMBOLIC REFERENCE TABLE.
C     IF ERRORS IN STATUS, CALL USER-DEFINED ERROR HANDLING ROUTINE.
C
      CALL MAPSET (MET, MAC, 80, IST)
      CALL MACRO (MAC, 0, 7LLOADL32, P1, IST)
      CALL MACRO (MAC, 0, 7LLOADL32, P2, IST)
      CALL MACRO (MAC, 0, 7LSUMPROD, P3, IST)
      CALL MACRO (MAC, 0, 7LUNLOL32, P4, IST)
      CALL MAPNOGO (MET, MAC, IST, 1)
      IF (IST .NE. 0) CALL ERRORS (IST)
C
C     READ INPUT, XFER TO ECS, AND EXECUTE SDP
C
```

```
   10 READ (1,100) (DATA1(I), I = 1,100)
      IF (EOF(1)) 99, 5
    5 CONTINUE
      WRITE (3,200)
      WRITE (3,500) (DATA1(I), I = 1,100)
      READ (2,100) (DATA2(I), I = 1,100)
      WRITE (3,300)
      WRITE (3,500) (DATA2(I), I = 1,100)
      CALL WRITEC (DATA1, P1(8), 100)
      CALL WRITEC (DATA2, P2(8), 100)
C
      CALL MAPGO (MET, MAC, ITIM , IERR, 1, 8, IST)
C
      CALL READEC (DATA2, P4(8), 100)
      WRITE (3,400)
      WRITE (3,500) (DATA2(I), I = 1,100)
C
      GO TO 10
   99 STOP
  100 FORMAT (5(F10.5))
  200 FORMAT (1H1,4X,11HDATA1 INPUT /)
  300 FORMAT (// 5X,11HDATA2 INPUT /)
  400 FORMAT (1H1,4X,7HRESULTS /)
  500 FORMAT (5(2X,F10.5))
      END


      SUBROUTINE ERRORS (IST)
C
C     PRINT ERROR MESSAGE IF ERROR STATUS RECEIVED
C
      PRINT 100,IST
  100 FORMAT(/// 10X,37H*** MAP STATUS ERROR ***   STATUS IS  ,O20 ///)
      RETURN
      END


MACRO STRING REFERENCE TABLE

  18(D) UNUSED MACRO STRING BUFFER WORDS

INSTRUCTION BLOCK SYMBOLS

      LABEL     LOC      REFERENCES
     FINIS      0006   000017


PARAMETER BLOCK SYMBOLS

      LABEL     LOC      VALUE     REFERENCES
     SAMPLE2    0042   000000   000060   000071
     SAMPLE1    0043   000000   000056   000104
     RESULT     0044   000000   000045   000062


DATA MEMORY MAP

     X FL (SAVE) = 000000      X FL (TOTAL) = 000144

     Y FL (SAVE) = 000000      Y FL (TOTAL) = 000144

     Z FL (SAVE) = 000000      Z FL (TOTAL) = 000144

      ARRAY     TYPE  MEM   MAP RA
     SAMPLE1     NS    X    000000
     SAMPLE2     NS    Y    000000
     RESULT      NS    Z    000000

SYMBOL TABLE LENGTH 10(D) WORDS
```

DATA1 INPUT

2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000
2.00000	2.00000	2.00000	2.00000	2.00000

DATA2 INPUT

.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000
.50000	.50000	.50000	.50000	.50000

RESULTS

100.00000	99.00000	98.00000	97.00000	96.00000
95.00000	94.00000	93.00000	92.00000	91.00000
90.00000	89.00000	88.00000	87.00000	86.00000
85.00000	84.00000	83.00000	82.00000	81.00000
80.00000	79.00000	78.00000	77.00000	76.00000
75.00000	74.00000	73.00000	72.00000	71.00000
70.00000	69.00000	68.00000	67.00000	66.00000
65.00000	64.00000	63.00000	62.00000	61.00000
60.00000	59.00000	58.00000	57.00000	56.00000
55.00000	54.00000	53.00000	52.00000	51.00000
50.00000	49.00000	48.00000	47.00000	46.00000
45.00000	44.00000	43.00000	42.00000	41.00000
40.00000	39.00000	38.00000	37.00000	36.00000
35.00000	34.00000	33.00000	32.00000	31.00000
30.00000	29.00000	28.00000	27.00000	26.00000
25.00000	24.00000	23.00000	22.00000	21.00000
20.00000	19.00000	18.00000	17.00000	16.00000
15.00000	14.00000	13.00000	12.00000	11.00000
10.00000	9.00000	8.00000	7.00000	6.00000
5.00000	4.00000	3.00000	2.00000	1.00000

PROGRAM NUMBERS

This program demonstrates the use of common parameters established by the use of the MPARAM call and testing and updating macro memory locations with the TMM and UPM macros. These features are used first to loop within the macro string to repeat a stack move macro 10 times. Then UPM is used to redefine the beginning and ending ECS addresses for a load left-justified, 32-bit data transfer from ECS. UPM and TMM are then used to repeat a second stack move macro 10 times.

```
      PROGRAM NUMBERS (OUTPUT,        TAPE2=OUTPUT)
      INTEGER MET(8), MAC(150), SYM(15), IA(100), IB(100), IC(100)
      INTEGER PA(11), PB(11), PC(11), PD(4), PE(4),PF(4), PG(4)
      INTEGER PH(4), PI(9), TIM(64), ERR(64)
      INTEGER STAT
      COMMON IAC(100), IBC(100), ICC(100)
      LEVEL 3, IAC, IBC, ICC
      STAT = 0
C
      LOCA = LOCE(IBC)
      LOCB =  (LOCA + 99 + 608) /8 *8 +8
      LOCC = LOCE(IAC)
      LOCD = (LOCC + 99) /8 *8 +8
      LOCG = LOCE(ICC)
      LOCH = (LOCG + 100 + 608) /8 *8 +8
C
      DATA PA /6LBUFFER,1,100,2378,378,1777778,3*0,1,0/
      DATA PB / 5LINPUT,1,100,2378,378,1777773,0,5LSTART,5LLIMIT,1,0/
      DATA PC /6LBUFFER,1,5LINPUT,1,6LBUFFER,1,2*0,100,1,0/
       DATA PD /1,4LTEST,2*1/
      DATA PE /1,4LTEST,10,0/
      DATA PF /1,5LSTART,2*0/
      DATA PG /1,5LLIMIT,2*0/
      DATA PH /1,4LTEST,20,0/
      DATA PI /6LBUFFER,1,100,2*378, 1777778,3*0/
C
      DATA IA /100*0/
      DATA IB /100*200100000038/
      DATA IC /100*400100000038/
C
      PA(8) = LOCC
      PA(9) = LOCD
      PF(3) = LOCG
      PG(3) = LOCH
      PI(8) = LOCG
      PI(9) = LOCH
C
```

```
      CALL MOVLEV (IA,IAC,100)
      CALL MOVLEV (IB,IBC,100)
      CALL MOVLEV (IC,ICC,100)
C
      CALL METOPEN (MET, SYM, 0, 0, STAT)
      CALL MAPSET (MET, MAC, 150, STAT)
      CALL MALLOT (MET, 6LBUFFER, 100, 2LNS, 1LY, STAT)
      CALL MALLOT (MET, 5LINPUT, 100, 2LNS, 1LX, STAT)
      CALL MPARAM (MAC, 4LTEST, 0, 1, STAT)
      CALL MPARAM (MAC, 5LSTART, LOCA, 1, STAT)
      CALL MPARAM (MAC, 5LLIMIT, LOCB, 1, STAT)
      CALL MACRO (MAC, 0, 7LLOADL32, PA, STAT)
      CALL MACRO (MAC, 0, 7LLOADL32, PB, STAT)
      CALL MACRO (MAC, 4LLOOP, 7LSTKMOVE, PC, STAT)
      CALL MACRO (MAC, 0, 3LUPM, PC, STAT)
      CALL MACRO (MAC, 0, 3LTMM, PE, STAT)
      CALL MACRO (MAC, 0, 4LJUMP, 4LLOOP, STAT)
      CALL MACRO (MAC, 0, 3LUPM, PF, STAT)
      CALL MACRO (MAC, 0, 3LUPM, PG, STAT)
      CALL MACRO (MAC, 0, 7LLOADL32, PB, STAT)
      CALL MACRO (MAC, 5LLOOP2, 7LSTKMOVE, PC, STAT)
      CALL MACRO (MAC, 0, 3LUPM, PD, STAT)
      CALL MACRO (MAC, 0, 3LTMM, PH, STAT)
      CALL MACRO (MAC, 0, 4LJUMP, 5LLOOP2, STAT)
      CALL MACRO (MAC, 0, 7LUNLOL32, PI, STAT)
      CALL MACRO (MAC, 0, 3LEND, 0, STAT)
      CALL MAPNOGO (MET, MAC, STAT, 1)
C
      CALL MAPGO (MET, MAC, TIM, ERR, 1, 500, STAT, 0)
C
      CALL MOVLEV (ICC, IC, 100)
      DO 10 K = 1,101,5
      IF (IC(K).NE. 7400000000008) GO TO 99
      IK = K+4
   10 PRINT 20,(IC(I),I=K,IK)
   20 FORMAT (5022)
C
      CALL REMARK (16H TEST SUCCESSFUL)
      STOP
   99 PAUSE *TEST FAILED*
      END
```

```
MACRO STRING REFERENCE TABLE

  26(D) UNUSED MACRO STRING BUFFER WORDS

INSTRUCTION BLOCK SYMBOLS

     LABEL     LOC      REFERENCES
    FINIS      0006    000043
    LOOP       0013    000022
    LOOP2      0031    000040

PARAMETER BLOCK SYMBOLS

     LABEL     LOC      VALUE     REFERENCES
    INPUT      0076    000000    000121    000132    000163    000174
    BUFFER     0077    000000    000103    000117    000123    000161    000165    000207
    LIMIT      0222    000370    000142    000145    000204
    START      0223    000144    000141    000150    000203
    TEST       0224    000000    000111    000114    000153    000156

DATA MEMORY MAP

    X FL (SAVE) = 000000      X FL (TOTAL) = 000144

    Y FL (SAVE) = 000000      Y FL (TOTAL) = 000144

    Z FL (SAVE) = 000000      Z FL (TOTAL) = 000000

     ARRAY     TYPE   MEM    MAP BA
    BUFFER      NS     Y     000000
    INPUT       NS     X     000000

SYMBOL TABLE LENGTH   9(D) WORDS
```

PROGRAM FOURIER

This program demonstrates use of the real fast Fourier transform macro.

```
      PROGRAM FOURIER(OUTPUT=513, TAPE5)
C     ORIGINAL CALCULATION ARRAYS
      COMMON TRAC(1025), SINC(1024), RESUL(1026), RESUK(1026)
      INTEGER ROW
      LOGICAL COMP
C     ECS ARRAYS FOR MAP DATA
      COMMON /IECS/ INPR(1025), SINT(1024)
      REAL INPR
      LEVEL 3, INPR, SINT
      COMMON /OECS/ OUTR(1026), OUTS(1026)
      LEVEL 3, OUTR, OUTS
C     MAP REQUIRED ARRAYS
      INTEGER MET(8), SYMB(30), MAC(200)
      INTEGER LOT(11), MVO(11), MVE(11), LOS(11), RFT(6), MVR(11),
     1        MVI(11), UNR(9)
      INTEGER STAT, ETAB(64)
      DATA LOT/4LLCAD,1,-0,16048,608,177777B,0,-0,-0,1,0/
      DATA MVO/-0,2,0,0,4LIMAG,1,0,0,-0,0,0/
      DATA MVE/4LLCAD,2,0,0,4LREAL,1,0,0,-0,0,0/
      DATA LOS/3LSIN,1,-0,16048,608,177777B,0,-0,-0,1,0/
      DATA RFT/4LREAL,4LIMAG,-0,3LSIN,3LCOS,1/
      DATA MVR/4LREAL,1,0,0,4LDUMP,2,0,0,-0,0,0/
      DATA MVI/4LIMAG,1,0,0,-0,2,0,0,-0,0,0/
      DATA UNR/4LDUMP,1,-0,14048,7720B,177777B,0,-0,-0/
C     PROGRAM ASSOCIATED PARAMETERS
      DATA LEV/10/
      DATA PI/3.14159265/
C     STATEMENT FUNCTION
      LOC8(IA)=LOCE(IA) / 8 * 8 + 8
C     INITIALIZE THE TRACE ARRAYS
      LIM=2 ** LEV
C     PERFORM THE MAP RELATED CALLS
      CALL METOPEN(MET, SYMB, 0, 0, STAT)
      CALL MAPSET(MET, MAC, 200, STAT)
      CALL MALLOT(MET, 4LREAL, LIM+1, 2LNS, 1LX, STAT)
      CALL MALLOT(MET, 4LIMAG, LIM+1, 2LNS, 1LY, STAT)
      CALL MALLOT(MET, 3LSIN, LIM+2, 2LNS, 1LZ, STAT)
      CALL MALLOT(MET, 4LLCAD, LIM+2, 2LNS, 1LZ, STAT)
```

```
CALL MALLOT(MET, 4LCOMP, LIM+2, 2LNS, 1LZ, STAT)
CALL MALLOT(MET, 4LREL2, LIM+1, 2LNS, 1LX, STAT)
CALL MALLOT(MET, 4LIMG2, LIM+1, 2LNS, 1LY, STAT)
CALL MEQUIV(MET, 3LCOS, LIM/2, 3LSIN, LIM/4, STAT)
LOT(3)=LIM
LOT(8)=LOCE(INPR)
LOT(9)=LOC8(INPR(LIM))
CALL MACRO(MAC, 0, 7LLOADL32, LOT, STAT)
MVC(1)=4LLOAD .OR. 2
MVC(9)=LIM / 2
CALL MACRO(MAC, 0, 7LSTKMOVE, MVC, STAT)
MVC(5)=4LIMG2
CALL MACRO(MAC, 0, 7LSTKMOVE, MVC, STAT)
MVE(9)=LIM / 2
CALL MACRO(MAC, 0, 7LSTKMOVE, MVE, STAT)
MVE(5)=4LREL2
CALL MACRO(MAC, 0, 7LSTKMOVE, MVE, STAT)
LOS(3)=LIM/2 + LIM/4
LOS(8)=LOCE(SINT)
LOS(9)=LOC8(SINT(LIM/2 + LIM/4))
CALL MACRO(MAC, 0, 7LLOADL32, LOS, STAT)
RFT(3)=LEV
CALL MACRO(MAC, 0, 7LREALFFT, RFT, STAT)
RFT(1)=4LREL2
RFT(2)=4LIMG2
CALL MACRO(MAC, 0, 7LREALFFT, RFT, STAT)
MVR(9)=LIM/2 + 1
CALL MACRO(MAC, 0, 7LSTKMOVE, MVR, STAT)
MVI(5)=4LOLMP .OR. 2
MVI(9)=LIM/2 + 1
CALL MACRO(MAC, 0, 7LSTKMOVE, MVI, STAT)
UNR(3)=LIM + 2
UNR(8)=LOCE(OUTR)
UNR(9)=LOC8(OUTR(LIM + 2))
CALL MACRO(MAC, 0, 7LUNLOL32, UNR, STAT)
MVR(1)=4LREL2
CALL MACRO(MAC, 0, 7LSTKMOVE, MVR, STAT)
MVI(1)=4LIMG2
```

```
      CALL MACRO(MAC, 0, 7LSTKMOVE, MVI, STAT)
      UNR(8)=LOCE(OUTS)
      UNR(9)=LOC8(OUTS(LIM + 2))
      CALL MACRO(MAC, 0, 7LUNLOL32, UNR, STAT)
      CALL MACRO(MAC, 0, 3LEND, 0, STAT)
      CALL MAPNOGO(MET, MAC, STAT, REF)
C     GENERATE SIN / COS TABLE
      A=2. * PI / LIM
      DO 20 I=1, LIM
      SINC(I)=SIN((I-1) * A)
   20 CONTINUE
      CALL WRITEC(SINC, SINT, LIM)
      ROW=0
C     READ THE TEST DATA
   30 CONTINUE
      ROW=ROW + 1
      DO 35 I=1,LIM
      R=ROW
   35 TRAC(I)=RANF(R)
C     MOVE DATA TO ECS
      CALL WRITEC(TRAC, INPR, LIM)
C     FIRE UP THE MAP
      CALL MAPGO(MET, MAC, 7LNCTABLE, ETAB, 1, 50, STAT,
     1           1000000000000009)
C     PRINT NON-ZERO ERROR TABLE ENTRIES
      PRINT 2, ROW
    2 FORMAT(*1NON-ZERO ERROR TABLE - ROW *,I2,/)
      DO 40 I=1, 64
      IF(ETAB(I) .NE. 0) PRINT 3, (I-1), ETAB(I)
    3 FORMAT(1X,O2,2X,O16)
   40 CONTINUE
C     CHECK THE DATA
      CALL READEC(RESUL, OUTR, LIM+2)
      CALL READEC(RESUK, OUTS, LIM+2)
      LST=LIM + 2
      COMP=.TRUE.
      DO 50 I=1, LST
      IF(RESUL(I) .EQ. RESUK(I)) GO TO 50
      PRINT 4, I, RESUL(I), RESUK(I)
    4 FORMAT(* COMPARISON FAILURE - *,I4,2(2X,E16.8))
      COMP=.FALSE.
   50 CONTINUE
      IF(.NOT.COMP) CALL MDUMP(1, 1, 1, 1)
      IF(.N.COMP) PAUSE *TEST FAILED*
      CALL MORLSE
C     TEST IF CYCLE COMPLETED
      IF(ROW .LT. 24) GO TO 30
C     GET OUT
      CALL MCLOSE(MET, STAT)
      END
```

```
MACRO STRING REFERENCE TABLE

  13(D) UNUSED MACRO STRING BUFFER WORDS

INSTRUCTION BLOCK SYMBOLS

     LABEL     LOC      REFERENCES
     FINIS     0006     000043

PARAMETER BLOCK SYMBOLS

     LABEL     LOC      VALUE     REFERENCES
     LOAD      0061     002003    000246    000261
     LOAD      0062     002002    000220    000233    000274
     COS       0063     000400    000175    000203
     SIN       0064     000000    000174    000202    000205
     REAL      0065     000000    000156    000177    000237
     IMAG      0066     000000    000143    000209    000265
     REL2      0067     002001    000117    000171    000224
     DUMP      0070     004005    000110    000147
     IMG2      0071     002001    000104    000172    000252
     DUMP      0072     004004    000073    000123    000132    000162

DATA MEMORY MAP

     X FL (SAVE) = 000000      X FL (TOTAL) = 004002

     Y FL (SAVE) = 000000      Y FL (TOTAL) = 004002

     Z FL (SAVE) = 000000      Z FL (TOTAL) = 006006

     ARRAY     TYPE   MEM    MAP RA
     REAL       NS     X     000000
     IMAG       NS     Y     000000
     SIN        NS     Z     000000
     LOAD       NS     Z     002002
     DUMP       NS     Z     004004
     REL2       NS     X     002001
     IMG2       NS     Y     002001
     COS        E      Z     000400

SYMBOL TABLE LENGTH 206(D) WORDS
```

This section describes MAP III system operator commands and console, dayfile, and error log messages provided by MSSI.

OPERATOR COMMANDS

MAPINIT.

n. XMAPINIT. (NOS/BE)
X. MAPINIT. (NOS)

Initializes MAP for user programs running on this computer, provided that MAP is operational and turned on in equipment status table (EST), and that MAP PP driver (MP3) is not already executing in this computer.

MAP,IDLE.

Disables MSAM calls from user programs not holding schedule table entries. Programs holding schedule table entries run to completion. After MAP, IDLE., MSAM responds to METOPEN calls with MET code/status 221_8 (requested equipment not available).

MAP,ABORT.

Causes central processor (CP) monitor to ignore OPEN and EXECUTE functions, aborts user programs having schedule table entries for this computer, and turns MAP off in EST.

MAP,CHECKPOINT. †

Suspends processing of MAP jobs while leaving MAP logically on. MAP processing resumes when operator issues MAPINIT command.

MAP,CLEAR. ††

Clears active/inactive schedule table entries for other computer and clears MAP bits in ECS flag register. If macro string from other computer is executing, MAP, CLEAR. terminates this execution.

MAP,NODUMP.

Clears ECS dump interlock word. Since interlock word normally clears during end-of-job processing, use MAP, NODUMP. only when job that set interlock word is hung.

MAP,UNLOCK.

Clears lock word in ECS. MAP, UNLOCK. may be required if an on-line diagnostic hangs after making MLOCK call.

MAP,DIAG.

Schedules an on-line diagnostic for immediate execution.

MAP,DIAG,XXXXX.

Sets an on-line diagnostic execution interval to $xxxxx_8$ seconds.

MAP,DOWN.

Identifies MAP as nonoperational. After MAP, DOWN., MSAM responds to MAPGO calls by sending

MAP DOWN. TYPE GO OR DROP.

to B display. MSSI returns MET code/status 217 (MAP down) to jobs that had active schedule table entries when operator issued MAP, DOWN.

MAP,UP.

Identifies MAP as again being operational.

MSSI CONSOLE MESSAGES

The following messages may appear on the B display, and in some cases, in user job dayfiles. Unless otherwise indicated, notify the system analyst when one of these messages appears.

Console Message/Description	Routine/Command
BAD CHANNEL XFER LOADING CONTROL MEMORY Checkword error occurred or MAP did not respond after MPI loaded control and subcontrol memory from PP channel.	MPI/MAPINIT

†NOS only.
††Dual computer installation, NOS/BE only.

Console Message/Description	Routine/Command
CANT LOAD MAPLIB Controlware not properly structured.	MAPINIT/ MAPINIT
DRIVER REQUIRED FOR MAP CLEAR† MAP,CLEAR. command entered while MAP not in operation.	1MP/MAP,CLEAR.
ECS DUMP INTERLOCK WAS job/id†† Displays interlock word just cleared. job Job name. id 0 - access A mainframe. 1 - access B mainframe.	1MP/ MAP,NODUMP.
ECS LOCK INTERLOCK WAS job/id†† Displays interlock word just cleared. job Job name. id 0 - access A mainframe. 1 - access B mainframe.	1MP/MAP/ UNLOCK.
FATAL MAP ERRORS LOADING CONTROLWARE Error detected in MAP status word during default controlware load.	MPI/MAPINIT
FULL INITIALIZATION This computer controls MAP PP channel interface. MSSI has initialized ECS common area and loaded MAP with default controlware. No action required.	MPI/MAPINIT
INTERLOCK = job/id/jdt † Displays interlock word just cleared. job Job name. id 0 - access A mainframe. 1 - access B mainframe jdt Job descriptor table ordinal.	1MP/ MAP,NODUMP. 1MP/ MAP,UNLOCK.

Console Message/Description	Routine/Command
MAP CHANNEL ALREADY RESERVED MAPINIT command issued while MP3 active.	MPI/MAPINIT
MAP DOWN†† MSSI declared MAP non-operational. Notify customer engineer.	CP monitor
MAP DOWN. TYPE GO OR DROP. † MSAM determined that MAP is nonoperational and suspended processing. To resume processing, verify that MAP is operational, type MAP,UP., and type n.GO. To terminate processing, type n.DROP.	MSAM
MAP DUMP I/L - CYB (id) (jdt)† MAP DUMP IL - job id†† MP3 is waiting for displayed job to clear ECS dump interlock word. Type MAP,NODUMP. if job appears hung. id 0 - access A mainframe. 1 - access B mainframe. jdt Job descriptor table ordinal. job Job name.	MP3
MAP HUNG, STATUS RETURN ERROR MAP returned incomplete status or did not respond to status function. Notify customer engineer.	MPI
MAP INITIALIZATION REQUIRED FOR ABORT † MAP,ABORT. command entered while MAP was not initialized.	1MP/MAP,ABORT.
MAP INTERLOCK CP monitor cannot modify ECS flag register. Issue MAP,CLEAR. command.	CP Monitor

†NOS/BE only.
††NOS only.

Console Message/Description	Routine/Command
MAP IS DISABLED OR NOT READY†	MPI/MAPINIT
Status received from MAP during initialization does not have ready bit set.	
MAP IS HUNG OR DOWN	MPI/MAPINIT
MAP does not respond or MAP returned error status in response to MAPINIT. command. Notify customer engineer.	
MAP IS NOT AVAILABLE	MAPINIT/ MAPINIT
CP monitor returned requested equipment not available status during initialization. Check MAP status in EST.	
MAP IS OFF	MPI/MAPINIT
MAP is turned off in EST. Turn MAP on and reenter MAPINIT command.	
MAP JOBS ARE CHECKPOINTED†	1MP/MAP, CHECKPOINT. or
MAP jobs have been checkpointed in preparation for MSSI or system recovery.	CHECKPOINT SYSTEM
MAP NOT AVAILABLE. TYPE GO OR DROP.	MSAM
MSAM is attempting to open schedule table entry while MAP is turned off in EST, or MAP has not been initialized via MAPINIT command. Turn on MAP in EST and type n.GO, or enter MAPINIT and type n.GO, or type n.DROP.	
MAPGO(S) ISSUED AFTER MAP EXECUTION ERR	MSAM
MAPGO call issued while associated MET contained code/status 321 (fatal execution error).	
MAPIII NOT INITIALIZED†	1MP
MAP,... command entered while MAP was not in operation.	
MAPINIT FIELD LENGTH OUT OF RANGE	MPI/MAPINIT
MAPINIT object code not properly structured.	

Console Message/Description	Routine/Command
MAP3-ECS I/O ERROR††	CP Monitor
ECS failure occurred while MP3 was attempting to read or write. Notify customer engineer.	
MP3 ACTIVE	MPI/MAPINIT
MAPINIT command entered while MAP was already active.	
MP3 ACTIVE BEFORE FUNCT XXXX	MP3
MP3 found PP channel already active before issuing last function (XXXX). Notify customer engineer.	
MP3 CHECKWORD ERR XXXX	MP3
MP3 received checkword error status from MAP. XXXX was last function issued by MP3. Notify customer engineer.	
MP3 EMPTY BEFORE INPUT XXXX	MP3
MP3 found PP channel empty before attempting input. XXXX was last function issued by MP3. Notify customer engineer.	
MP3 FATAL MAP/SYSTEM ERR XXXX	MP3
MAP unable to complete macro string normally for reason displayed in previous dayfile message. XXXX was last function issued by MP3. Notify customer engineer.	
MP3 FULL AFTER OUTPUT XXXX	MP3
MAP failed to accept last word output to PP channel. XXXX was last function issued by MP3. Notify customer engineer.	

†NOS only.
††NOS/BE only.

Console Message/Description	Routine/Command
MP3 FULL BEFORE OUTPUT XXXX MP3 found PP channel already full before output. XXXX was last function issued by MP3. Notify customer engineer.	MP3
MP3 FUNCTION BUSY TIMEOUT XXXX Function busy bit failed to drop from MAP status. XXXX was last function issued by MP3. Notify customer engineer.	MP3
MP3 INPUT CHECKWORD ERROR XXXX MP3 detected invalid input checkword for on-line diagnostic PP channel transfer. XXXX was last function issued by MP3. Notify customer engineer.	MP3
MP3 NO RESPONSE FUNCTION XXXX PP channel failed to go inactive after MP3 issued function XXXX to MAP. Notify customer engineer.	MP3
MP3 TIMEOUT ON INPUT XXXX MP3 found PP channel inactive after IAM (block input) instruction. XXXX was last function issued by MP3. Notify customer engineer.	MP3
MP3 TIMEOUT ON OUTPUT XXXX MP3 found channel inactive after OAM (block output) instruction. XXXX was last function issued by MP3. Notify customer engineer.	MP3
NO COMMON PARTITION FOUND IN ECS PARTITION TABLE†	MAPINIT/ MAPINIT
MAPINIT did not find common ECS partition named COMMON. Check deadstart procedure.	

Console Message/Description	Routine/Command
NO DDP† Distributive data path (DDP) device not defined in EST or logically off. No dump produced.	MP3
NO MAP ENTRY IN EST MAP not defined in EST.	MPI/MAPINIT
NO MAP EQUIPMENT PRESENT MAP not defined in EST.	1MP/MAP, IDLE.
NOT ENOUGH ECS FOR MAP TABLES† ECS common area too small for MAP tables. Check deadstart procedure.	MAPINIT/ MAPINIT
PARTIAL INITIALIZATION Other computer controls MAP PP channel interface. MSSI initialized only schedule table in ECS and did not reload MAP with controlware. No action required.	MPI/MAPINIT
RECOVERY/FULL INITIALIZATION†† MSSI initialized.	MPI/MAPINIT
RECOVERY/PARTIAL INITIALIZATION†† MSSI recovered from ECS.	MPI/MAPINIT
REQUESTED CHANNEL NOT ASSIGNED - TRY AGAIN CP monitor error occurred while trying to reserve MAP PP channel. Try again.	MPI/MAPINIT
WAIT ACTIVE STE MAP, ABORT. command waiting for all schedule table entries to become inactive.	1MP/MAP, ABORT.
WAITING FOR LOCKED MAP MAP locked by another on-line diagnostic job when this job issued MLOCK call. MAP will be locked for this job when current reservation released.	MSAM

†NOS/BE only.
††NOS only.

Console Message/Description	Routine/Command
WAITING FOR MAP ACTIVITY†	1MP
User program terminated before MAP completed processing associated macro string. No operator action required.	
WAITING FOR SCHEDULE TABLE ENTRY	MSAM
Schedule table full when MSAM issued OPEN function. MSAM periodically tries again. No operator action required.	
1SO--SWAPOUT SUSPENDED BY MAP ACTIVITY - TRY LATER†	1SO/n. LOCKOUT
Operator attempted to swap out MAP job via n. LOCKOUT. command. Swapout has not occurred because job is currently using MAP or has locked MAP. If necessary to swap out job, operator must retry swapout later (system will not automatically retry swapout).	

MSSI DAYFILE MESSAGES

Dayfile Message/Description	Routine
DDP ERROR, NO DUMP	MP3
DDP error occurred during MAP dump. No dump produced.	
DPxx, Cyy, PE, RNR, S0000, Azzzzzzz††	MP3
DDP error occurred while writing MAP dump buffer to ECS.	
xx DDP EST ordinal.	
yy DDP channel number.	
zzzzzzz ECS buffer address being written into.	
ERROR IN LIBRARY ECS XFER	MSAM
Error occurred while transferring controlware from central memory to ECS. Notify system analyst.	
ERROR IN LIBRARY LOAD	MSAM
MSAM detected error while loading controlware. Notify system analyst.	

Dayfile Message/Description	Routine
FATAL MAP ERROR, MACRO = nn	MP3
MP3 detected fatal error during macro string execution.	
JOB ABORTED. ERRORS DETECTED BY MSAM.	MSAM
MSAM fatal error limit exceeded.	
JOB ABORTED. MET CODE/ STATUS = nnn	MSAM
MET contains code/status that precludes further processing.	
MAP DOWN. TYPE GO OR DROP. †	MSAM
MSAM determined that MAP is non-operational and suspended processing. To resume processing, verify that MAP is operational, type MAP, UP., and type n. GO. To terminate processing, type n. DROP.	
MAP NOT AVAILABLE. TYPE GO OR DROP.	MSAM
MSAM attempting to open schedule table entry while MAP turned off in EST, or MAP has not been initialized via MAPINIT command. Turn on MAP in EST and type n. GO, or enter MAPINIT and type N. GO, or type n. DROP.	
MAP TIME LIMIT	MP3
Macro string execution time exceeded limit specified by MAPGO call or installation parameter.	
MAPGO(S) ISSUED AFTER MAP EXECUTION ERR	MSAM
MAPGO call issued while associated MET contained code/status 321 (fatal execution error).	
MET FIELD O. R	MP3
MET central memory address out of range.	
MP3 ACTIVE BEFORE FUNCT XXXX	MP3
MP3 found PP channel already active before issuing last function (XXXX).	
MP3 BAD CHANNEL TRANSFER REQUEST	MP3
MP3 received unrecognizable PP channel transfer request from an on-line diagnostic.	

†NOS/BE only.
††NOS only.

Dayfile Message/Description	Routine
MP3 CHECKWORD ERR XXXX MP3 received checkword error status from MAP. XXXX was last function issued by MP3.	MP3
MP3 EMPTY BEFORE INPUT XXXX MP3 found PP channel empty before attempting input. XXXX was last function issued by MP3.	MP3
MP3 FATAL MAP/SYSTEM ERR XXXX MAP unable to complete macro string normally for reason displayed in previous dayfile message. XXXX was last function issued by MP3.	MP3
MP3 FULL AFTER OUTPUT XXXX MAP failed to accept last word output to PP channel. XXXX was last function issued by MP3.	MP3
MP3 FULL BEFORE OUTPUT XXXX MP3 found PP channel already full before output. XXXX was last function issued by MP3.	MP3
MP3 FUNCTION BUSY TIMEOUT XXXX Function busy bit failed to drop from MAP status. XXXX was last function issued by MP3.	MP3
MP3 INPUT CHECKWORD ERROR XXXX MP3 detected invalid input checkword for an on-line diagnostic PP channel transfer. XXXX was last function issued by MP3.	MP3
MP3 NO RESPONSE FUNCTION XXXX PP channel failed to go inactive after MP3 issued function xxxx to MAP.	MP3
MP3 TIMEOUT ON INPUT XXXX MP3 found PP channel inactive after IAM (block input) instruction. XXXX was last function issued by MP3.	MP3
MP3 TIMEOUT ON OUTPUT XXXX MP3 found PP channel inactive after OAM (block output) instruction. XXXX was last function issued by MP3.	MP3

Dayfile Message/Description	Routine
MSAM ARGUMENT COUNT ERROR DETECTED BY xxxxxxx CALLED FROM yyyyyyy LINE zzzz MSAM routine call had incorrect number of parameters. xxxxxxx — Name of routine that detected error. yyyyyyy — Name of routine that called xxxxxxx. zzzz — Call line number on yyyyyyy program listing.	MSAM
MSAM STATUS WORD = xxxxxxxxxx yyyyyyy Issued when MSAM aborts job. xxxxxxxxxx MSAM status word. yyyyyyy Macro or MSAM routine that triggered abort.	MSAM
NO DDP† DDP not defined in EST or logically off. No dump produced.	MP3
OPERATOR MAP ABORT†† Operator aborted MAP III system thereby aborting jobs with pending MAP requests.	1MP

MSSI ERROR LOG MESSAGES††

When one of these messages appears, notify the customer engineer.

Error Log Message/Description	Routine
DPxx, Cyy, PE, RNR, S0000, Azzzzzzz†† DDP error occurred while writing MAP dump buffer to ECS. xx — DDP EST ordinal. yy — DDP PP channel number. zzzzzzz — ECS buffer address being written into.	MP3

†NOS/BE only.
††NOS only.

Error Log Message/Description	Routine
MPxx, Cyy, 1S, EC, AAAA, BBBB, CCCC, DDDD	MP3
MPxx, Cyy, 2S, EC, EEEE, FFFF, GGGG, HHHH	MP3

MAP hardware error.

xx MAP EST ordinal.

yy MAP PP channel number.

1S Signifies this is first line of two-line message.

2S Signifies this is second line of two-line message.

EC MP3 error code.

1	No response to function code.
2	Fatal system/ MAP error.
3	Checkword error.
4	PP channel failed to go empty.
5	Deadman timeout on input.
6	Deadman timeout on output.
7	PP channel full before output.
8	PP channel active before function.
9	Timeout exceeded waiting for function busy to clear.
10	PP channel empty before input.

Error Log Message/Description		Routine
AAAA	MAP status word 0.	
BBBB	MAP status word 1.	
CCCC	MAP status word 2.	
DDDD	MAP status word 3.	
EEEE	MAP status word 4.	
FFFF	MAP status word 5.	
GGGG	MAP status word 6.	
HHHH	MAP status word 7.	

MSSI CERFILE ENTRY FORMAT†

Figure 5-1 shows the format of the CERFILE entry made by MSSI for some MAP errors.

Error codes for word 3, bits 48 through 53, and word 4, bits 48 through 59, are defined as follows:

Error Code	Definition
1	No response to function.
2	Fatal MAP/system error.
3	Checkword error.
4	Channel full after output.
5	Timeout on channel output.
6	Timeout on channel input.
7	Channel full before output.
8	Channel active before function.
9	Function busy timeout.
10	Channel empty before input.

†NOS/BE only.

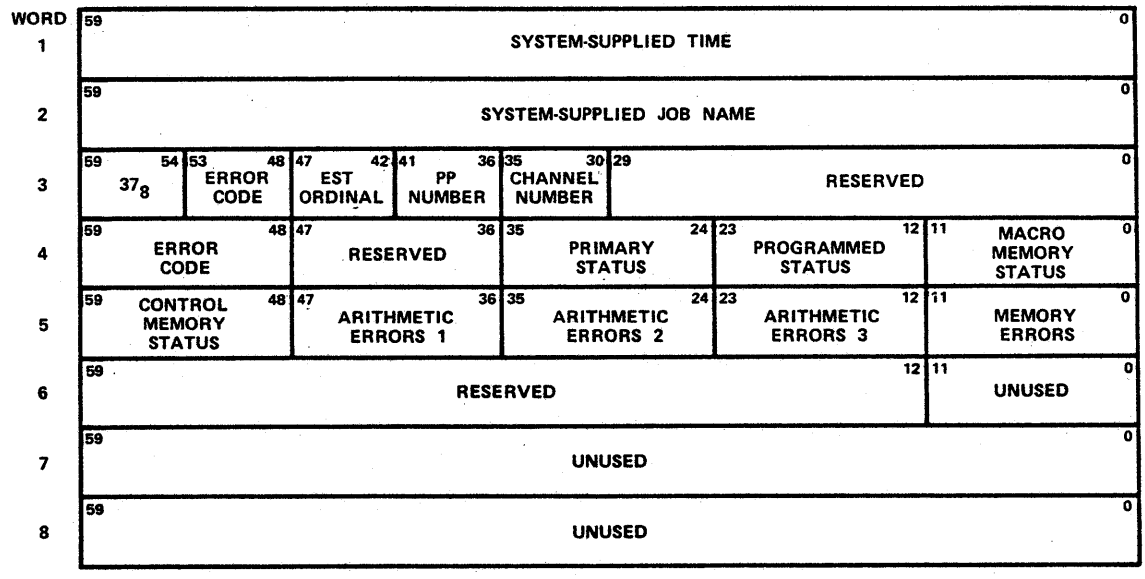


Figure 5-1. MAP CERFILE Entry Format

This appendix describes common types of macro parameters, describes the sine/cosine table used with FFT macros, and provides a parameter table for each macro that requires parameters.

PARAMETER DESCRIPTIONS

The following paragraphs describe types of parameters used with more than one macro.

FIRST WORD ADDRESS (FWA)

Specifies MAP data storage address to contain first element of array.

INCREMENT FACTOR (IF)

Determines spacing of array elements in MAP data storage. First element has address FWA and i^{th} element (i=0, 1, 2, . . .) has address FWA+i*(IF). IF must be an integer. Figure A-1 shows array loaded into data storage with IF set to +2.

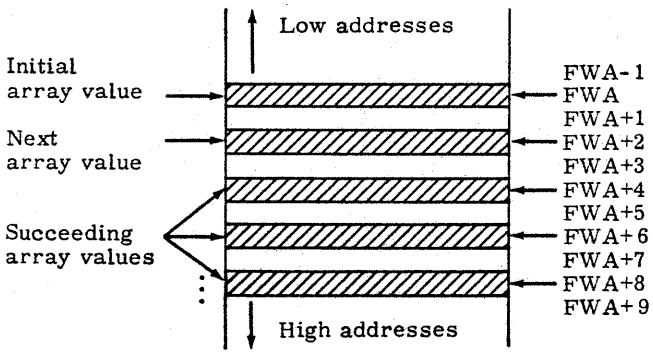


Figure A-1. +2 Increment Factor

OFFSET (OFF)

Number of data storage locations between first element of array and first element of array to be processed.

FORMAT CONVERSION PARAMETERS

Numerical conversion and assembly/disassembly units within MAP convert various external data formats to the MAP internal data format, and vice versa. These units extract the following four words from the parameters accompanying an input/output macro and use these words to perform the specified conversion. Table A-1 lists format conversion parameter values for several external formats.

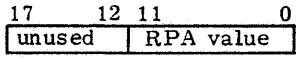
Numerical Conversion Control (NCC) Word

Identifies external format, specifies conversion direction, and provides for sign inversion or conversion disable. Figure A-2 shows bit fields in NCC word.

Radix Point Adjust (RPA) Word

Contains 12-bit twos complement number that is added to unbiased external exponent to adjust floating-point formats having radix point other than immediately to left of mantissa. RPA word can also be used to enable MAP to process numbers exceeding allowable MAP range.

RPA word format is as follows:



Appendix C contains instructions for using RPA word.

TABLE A-1. FORMAT CONVERSION PARAMETERS

External Format	Macro	Value (1)			
		NCC Word	RPA Word	Word 1 Mask	Word 2 Mask
CDC CYBER/ 6000 32-bit floating point (2)	LOADL32 (3)	001604	000060 (48)	177777	0
	UNLDL32	001404	007720 (-48)	177777	0
CDC CYBER/ 6000 30-bit floating point (4)	LOADP30 (3)	001604	000060 (48)	177776 (5)	0
	UNLDP30	001404	007720 (-48)	177777	0
CDC CYBER/ 6000 32-bit fixed point (6)	LOADR32	000237	000037 (31)	177777	0
	UNLDR32	000037	000037 (31)	177777	0
CDC CYBER/ 6000 30-bit fixed point (7)	(8) UNLDP30	000037	000037 (31)	377777 (9)	0
MAP format: full 32-bit	LOADP32	000340	0	177777	0
	UNLDP32	000140	0	177777	0

(1) Decimal values are in parentheses; others are octal.

(2) Uses most significant 32 bits of 60-bit floating-point word (12-bit exponent, 20-bit coefficient).

(3) Numbers exceeding allowable MAP range (refer to section 1) cause numerical conversion unit overflow errors.

(4) Packs most significant 30 bits (12-bit exponent, 18-bit coefficient) of each of two 60-bit floating-point words into one 60-bit word.

(5) Allows sign fill in lower 2 bits.

(6) Expects 32-bit signed integer in lower 32 bits of 60-bit word.

(7) Packs two 30-bit signed integers into one 60-bit word.

(8) MAP does not accept 30-bit packed fixed-point data.

(9) Bit 29 is highest-order R3 bit transferred to ECS.

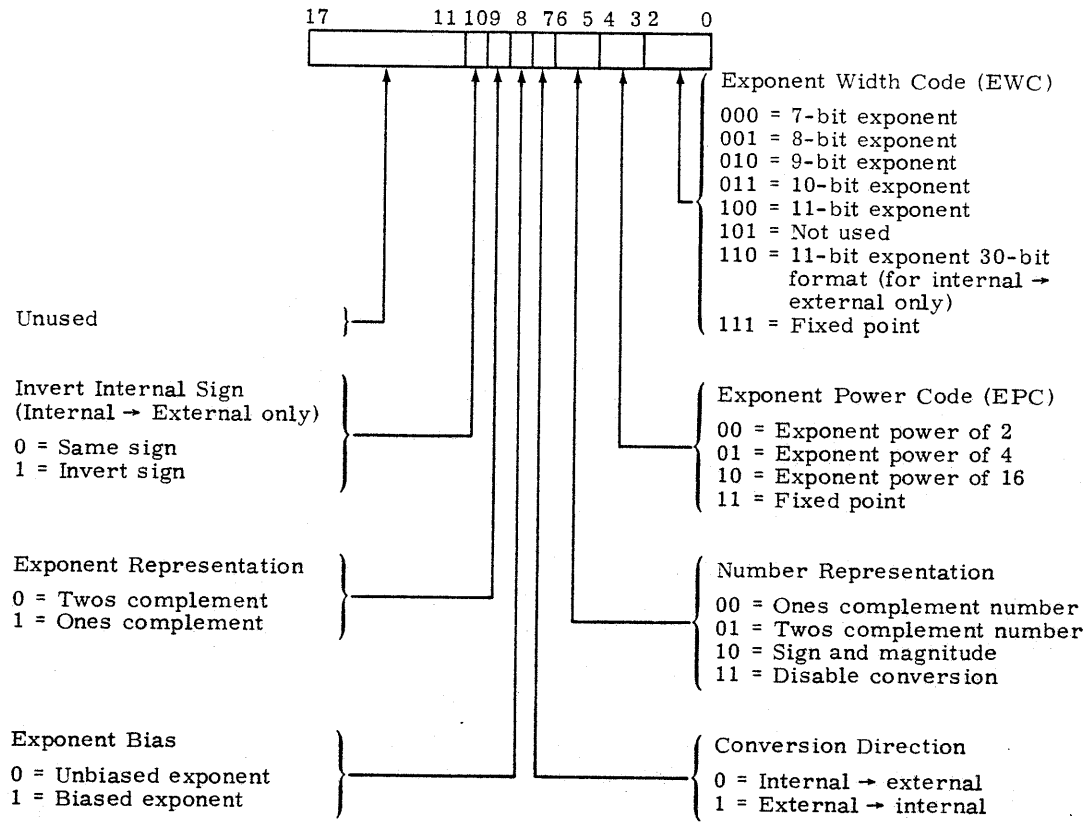


Figure A-2. NCC Word

Assembly/Disassembly (A/D) Mask Words

Control register R3 in MAP A/D unit. R3 is 72-bit left-shift register used for buffering and format modification during MAP input/output operations. Figure A-3 shows A/D mask words.

For mask bits 0 through 15, mask bit n controls R3 bits 2n and 2n+1. Rules for bits 0 through 15 are:

- Each 1 in word 2 mask enters 0's in controlled R3 bit positions.
- Each 0 in word 2 mask lets corresponding word 1 mask control R3 entry.
- Each 1 in word 1 mask enters data in controlled R3 bit positions, providing corresponding word 2 mask bit is 0.
- Each 0 in word 1 mask enters sign in controlled R3 bit position, providing corresponding word 2 mask bit is 0. Sign is defined as higher-order bit of 2 R3 bits controlled by highest-order 1 in word 1 mask.

Bits 16 and 17 of each mask, when taken together, form a 4-bit, half-pad count. MAP doubles this count to determine number of bit positions between R3 bit 31 and highest-order R3 bit (between bits 1 and 31) to be transferred to ECS.

DESTINATION LIST PARAMETERS

LOADP32 and LOADL32 each can simultaneously transfer data to multiple MAP data storage buffers called destination lists. Parameters specify number of destination lists, first location of first list, spacing of data words in lists, and spacing of lists in data storage. Figure A-4 shows destination list parameters for a three-list load. For a single-list load, list count and list increment factor parameter values should be 1 and 0, respectively.

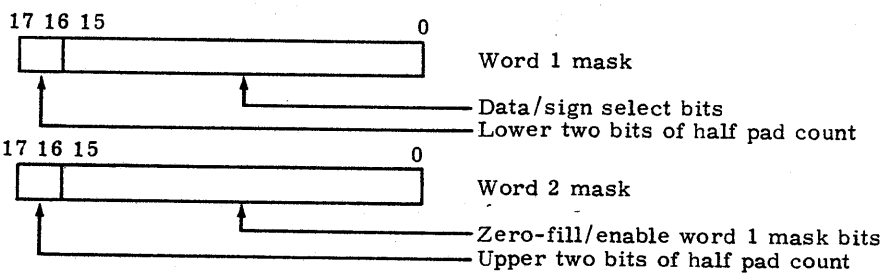


Figure A-3. A/D Mask Words

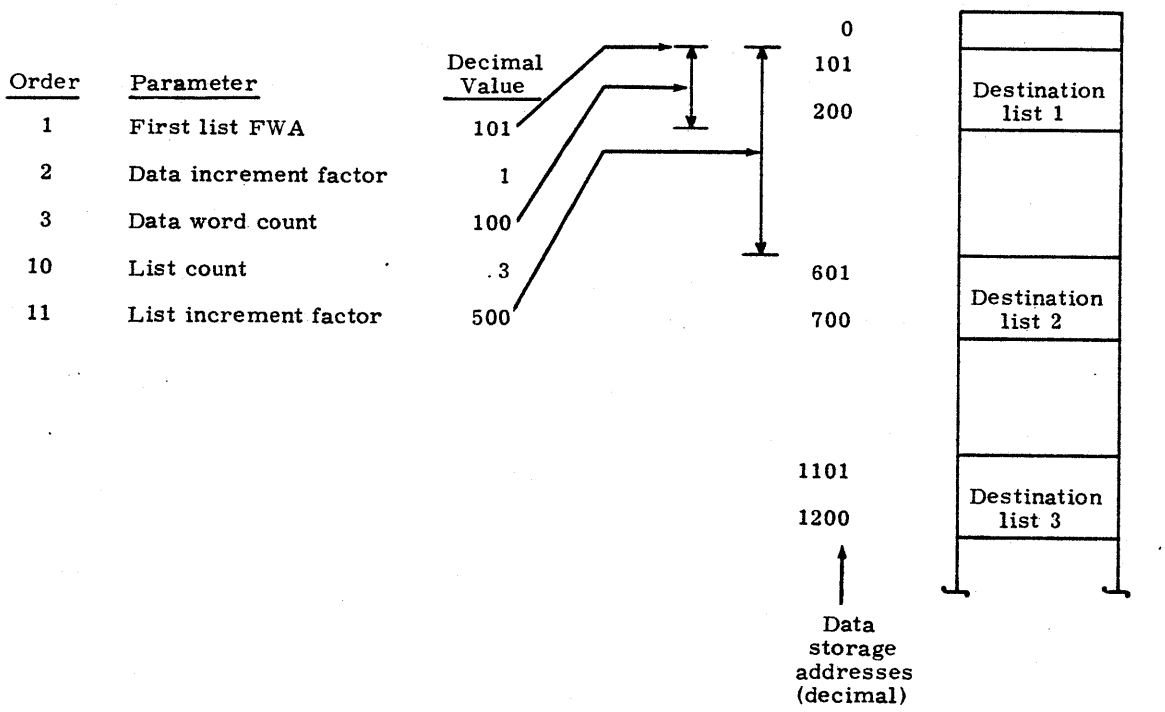


Figure A-4. Destination List Parameters

SINE/COSINE TABLES

Each FFT macro requires both a sine table and a cosine table to be resident together in one section of MAP data storage. The tables must include sines/cosines for the angles:

$0, A, 2A, 3A, \ldots, \pi\text{-}A$

A $2\pi/N$

N Number of points in series to be transferred

Also, N is 2^n where n is the level count parameter for each FFT macro.

The user can conserve MAP data storage by taking advantage of the sine/cosine symmetry [cos(a)=sin(a+π/2)] to overlap the last half of the sine table and the first half of the cosine table as shown in figure A-5.

Tables for large series can be used for smaller series by increasing the sine/cosine increment factor. For example, tables used for a 2048-point transform with an increment factor of +1 can be used for a 1024-point transform by setting the increment factor to +2.

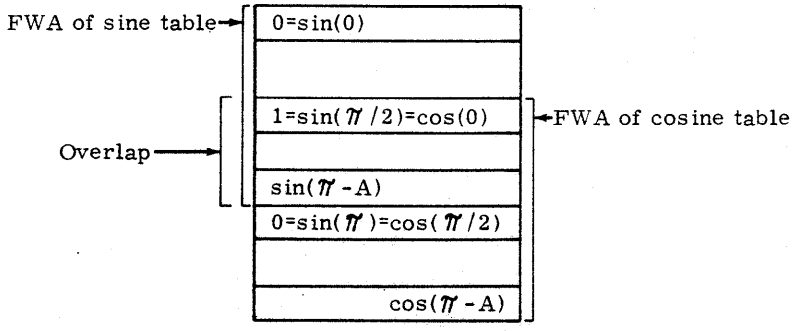


Figure A-5. Sine/Cosine Table Overlap

PARAMETER TABLES

Tables A-2 through A-14 define parameters for the macros described in section 3.

TABLE A-2. UPM/TMM PARAMETERS

Order	Parameter	Description		Notes
1	N	UPM	Number of macro memory locations to be changed.	-
		TMM	Must be 1.	
2	M address	UPM	First macro memory location to be changed.	-
		TMM	Macro memory location to be tested.	
3	Value	UPM	Update value.	-
		TMM	Test value.	
4	UPM flag	0	Replace contents of each location to be changed with update value.	①
		Non-zero	Add update value to contents of each location to be changed.	

① UPM only; unused for TMM.

TABLE A-3. LOAD FROM ECS PARAMETERS

Order	Parameter	Description	Notes
1	FWA of first list	First location of first destination list in data storage.	(1)
2	Data IF	Spacing of data words for all destination lists.	(1) (2)
3	Data word count	Number of data words to be loaded into each destination list.	(1)
4	NCC word	Data format conversion words. Refer to table A-1 for common values for these words.	-
5	RPA word		-
6	Word 1 mask		-
7	Word 2 mask		-
8	Relative ECS FWA	Relative ECS address which, when added to user's absolute ECS RA, specifies first 60-bit word of transfer.	(3)
9	Relative ECS limit address	Relative ECS address which, when added to user's absolute ECS RA, specifies last legal 60-bit word of the transfer. Transfer may terminate before this address.	(3)
10	List count	Number of destination lists.	(1) (4)
11	List IF	Spacing of corresponding data words in adjacent destination lists.	(1) (4) (5)

(1) Refer to Destination List Parameters in this appendix.

(2) Must be nonzero integer.

(3) If this value is to be changed by MODIFY call, user must observe restrictions listed in section 2.

(4) Omit for LOADP30 and LOADR32 macros.

(5) Must be positive integer. Address of word i in destination list j ($i \geq 0$, $j \geq 1$) is (first list FWA) + (j-1) · (List IF) + i · (Data IF).

TABLE A-4. UNLOAD TO ECS PARAMETERS

Order	Parameter	Description	Notes
1	FWA of unload list	First MAP data storage location in list to be transferred to ECS.	-
2	Data IF	Spacing of data words in unload list.	(1)
3	Data word count	Number of data words to be sent to ECS.	(2)
4	NCC word	Data format conversion words. Refer to table A-1 for common values for these words.	-
5	RPA word		-
6	Word 1 mask		-
7	Word 2 mask		-
8	Relative ECS FWA	Relative ECS address which, when added to user's absolute ECS RA, specifies first ECS location to receive 60-bit word from MAP.	(3)
9	Relative ECS limit address	Relative ECS address which, when added to user's absolute ECS RA, specifies last ECS location that can legally receive 60-bit word from MAP. Transfer may terminate before this address.	(3) (4)

(1) Must be nonzero integer.
(2) Single word transfer from MAP to ECS is illegal.
(3) If this value is to be changed by MODIFY call, user must observe restrictions listed in section 2.
(4) ECS limit address must be at least one greater than desired limit and evenly divisible by 8.

TABLE A-5. SUMPROD PARAMETERS

Order	Parameter	Description	Notes
1	A FWA	Location of A_0 in data storage (filter FWA).	-
2	A IF	Spacing of elements in (Aj) array.	(1)
3	B FWA	Location of B_0 in data storage (trace FWA).	-
4	B IF	Spacing of elements in (Bj) array.	(1)
5	C FWA	Location of C_0 in data storage.	-
6	C IF	Spacing of elements in (Cj) array.	(1)
7	LA	Number of elements in (Aj), LA> 10.	-
8	LB	Number of elements in (Bj), LB≥ LA.	-
9	LC	Number of elements in (Cj) array (number of results).	-
10	s	0 Enable convolution. 1 Enable correlation.	(2)
11	shift	Initial shift of (Aj) array.	(3)

(1) Positive integer.
(2) MAP microcode characteristics require that 0 rather than -1 be used to select convolution.
(3) Positive, negative, or zero integer.

TABLE A-6. STKMOVE PARAMETERS

Order	Parameter	Description	Notes
1	A FWA	Location of A_0 in data storage.	-
2	A IF	Spacing of elements in (Aj) array.	①
3	B FWA	Location of B_0 in data storage.	②
4	B IF	Spacing of elements in (Bj) array.	① ②
5	C FWA	Location of C_0 in data storage.	-
6	C IF	Spacing of elements in (Cj) array.	①
7	Pad	Zero.	③
8	Pad	Zero.	③
9	LC	Number of elements in (Cj) array (number of results).	-
10	s	0 Enable move. 1 Enable stack.	-
11	Pad	Zero.	③

① Nonzero integer except as specified in following note.
② 0 for move operations.
③ Must be included to satisfy MAP microcode requirements.

TABLE A-7. CPLXFFT/ICPXFFT PARAMETERS

Order	Parameter	Description	Notes
1	R FWA	Location of real part of first complex input point.	①
2	I FWA	Location of imaginary part of first complex input point.	①
3	Level count	$Log_2(N)$ where N is number of complex input points.	②
4	FWA of sine table	Location of first sine table entry.	③
5	FWA of cosine table	Location of first cosine table entry.	③
6	Sine/cosine table IF	Spacing between consecutive table entries. Same spacing must be used for both tables.	③ ④

① Before transform, arrays R and I contain real and imaginary parts, respectively, for N complex points of series to be transformed. Elements in both R and I arrays must be contiguous (implied increment factor of +1). After transform, arrays R and I contain real and imaginary parts, respectively, for complex points of transformed series.
② For example, when 1024 complex points are being transformed, level count is 10.
③ Refer to Sine/Cosine Tables in this appendix.
④ Positive integer.

TABLE A-8. REALFFT PARAMETERS

Order	Parameter	Description	Notes
1	R FWA	Location of first even-numbered real input point.	①
2	I FWA	Location of first odd-numbered real input point.	①
3	Level count	$Log_2(N)$ where N is number of real input points.	②
4	FWA of sine table	Location of first sine table entry.	③
5	FWA of cosine table	Location of first cosine table entry.	③
6	Sine/cosine table IF	Spacing between consecutive table entries. Same spacing must be used for both tables.	③ ④

① Before transform, array R contains even-numbered real input points and array I contains odd-numbered real input points. Elements in each of these arrays must be contiguous (implied increment factor of +1). Although initially arrays R and I each contain N/2 points, (N/2) + 1 contiguous locations must be allocated for storage of complex results. After transform, arrays R and I contain real and imaginary parts, respectively, for first (N/2) + 1 complex points of transformed series.

② For example, when 1024 real points are being transformed, level count is 10.

③ Refer to Sine/Cosine Tables in this appendix.

④ Positive integer.

TABLE A-9. INVRFFT PARAMETERS

Order	Parameter	Description	Notes
1	R FWA	Location of real part of first complex input point.	①
2	I FWA	Location of imaginary part of first complex input point.	①
3	Level count	$Log_2(N)$ where N is number of real output points.	②
4	FWA of sine table	Location of first sine table entry.	③
5	FWA of cosine table	Location of first cosine table entry.	③
6	Sine/cosine table IF	Spacing between consecutive table entries. Same spacing must be used for both tables.	③ ④

① Before transform, arrays R and I contain real and imaginary parts, respectively, for first (N/2) + 1 complex points of series to be transformed. First and last imaginary parts must be zero. Elements in both R and I arrays must be contiguous (implied increment factor of +1). After transform, array R contains even-numbered real result points and array I contains odd-numbered real result points.

② For example, when 1024 real output values are expected, level count is 10.

③ Refer to Sine/Cosine Tables in this appendix.

④ Positive integer.

TABLE A-10. FILTER PARAMETERS

Order	Parameter	Description	Notes
1	r FWA	Location of r_0 in data storage.	-
2	r IF	Spacing of elements in list for array r_i.	①
3	G FWA	Location of G_0 in data storage.	-
4	G IF	Spacing of elements in array G_i.	①
5	F FWA	Location of F_0 in data storage.	②
6	a FWA	Location of a_0 in data storage.	②
7	Length MD	Number of elements of arrays a_i and F_i to calculate. (Arrays r_i and G_i must each have at least length MD elements.)	③
8	ALPHA address	First location in data storage of three-word buffer used by FILTER to calculate each element of array F_i. Word 1 — Expected error for an a_i array of length MA (parameter 10). Word 2 — E operator for next element of array a_i. Word 3 — H operator for next element of array F_i.	-
9	Length MS	Element of arrays a_i and F_i at which to start calculation.	③
10	MA	Number of elements of arrays a_i and F_i calculated. (When array F_i is unstable or singular, MA < length MD; otherwise, MA = length MD.)	③ ④
11	IFSTABL flag	When IFSTABL flag is zero, it selects stability check on array F_i. If array F_i is unstable, IANS address (parameter 13) contains -1, and MA address contains number of stable elements in arrays a_i and F_i.	③
12	IFSPIKE flag	When IFSPIKE flag is zero, it selects generation of array a_i only; array G_i is not required and array F_i is not generated. When IFSPIKE flag is nonzero, FILTER requires array G_i and generates arrays a_i and F_i.	③
13	IANS	FILTER macro status code. 0 — Normal return; MA = length MD. -1 — Array F_i unstable; MA < length MD. -2 — Array F_i singular; MA < length MD.	④ ⑤

① Positive integer.

② FILTER stores elements of arrays a_i and F_i in consecutive data storage locations (implied increment factor of +1).

③ Refer to FILTER description for more information.

④ Use MPARAM call to specify this common parameter for use by other macros.

⑤ When IFSTABL flag is zero and array F_i is singular, IANS code is -1.

TABLE A-11. NMO PARAMETERS

Order	Parameter	Description		Notes
1	IN FWA	Location of first input trace sample.		-
2	V FWA	Location of first velocity function value.		(1)
3	OUT FWA	Location of first output trace sample.		-
4	N	Number of input trace sample (N > 0).		-
5	K1	Starting value of velocity function index (K1 > 0).		-
6	K2	Preexecution	Number of output trace samples to mute.	(2) (3)
		Postexecution	Number of output trace samples muted.	
7	K3	Postexecution	Index to first nonmuted output trace sample after sample identified by parameter K2.	(2) (4)
8	K4	Number of samples to move per velocity function sample (K4 = 1 for NMO and K4 > 1 for velocity analysis computation).		-
9	BETA	Location of first parameter in BETA auxiliary array.		(5)

(1) Each value in this list must be squared inverse of velocity function value in distance/millisecond units.

(2) NMO returns value to this macro memory location. Use MPARAM call to specify this common parameter for use by other macros.

(3) When nonzero upon entry, NMO interprets this value as number of output samples to mute.

(4) When input trace has already been muted beyond sample specified by K2 or THRESH, this value is index from last specified muted sample to first nonmuted output trace sample.

(5) BETA array contains following floating-point values.

BETA (1) = T_r = time between input trace samples in milliseconds.

(2) = TMAX = input trace last sample time in milliseconds.

(3) = D = squared offset distance for this input trace.

(4) = THRESH = front end mute parameter (refer to NMO description for use).

(5) = T_{1v} = velocity function first sample time in milliseconds.

(6) = T_1 = input trace first sample time in milliseconds.

(7) = T_{rv} = time between velocity function samples in milliseconds.

TABLE A-12. TYPE 1 PARAMETERS (1)

Order	Parameter	Description	Notes
1	A FWA	Location of A_0 in data storage.	-
2	A OFF	Offset of array A_j from A_0.	-
3	A IF	Spacing of elements in array A_j.	-
4	LA	Number of elements in arrays.	-
5	D FWA	Location of D_0 in data storage.	-
6	D OFF	Offset of array D_j from D_0.	-
7	D IF	Spacing of elements in array D_j.	-
8	FLAG	Option flag.	(2)
9	AU	Arithmetic unit selection.	(2)

(1) Following macros use this parameter block.

CVEC	NMVEC	ZEROVEC	TVEC
NVEC	SUMRVEC	SQRTVEC	XMM2DM
MVEC	BCASVEC	COMVEC	XDM2MM

(2) Refer to appropriate macro description for use of this parameter.

TABLE A-13. TYPE 2 PARAMETERS (1)

Order	Parameter	Description	Notes
1	A FWA	Location of A_0 in data storage.	-
2	A OFF	Offset of array A_j from A_0.	-
3	A IF	Spacing of elements in array A_j.	-
4	LA	Number of elements in arrays.	-
5	B FWA	Location of B_0 in data storage.	-
6	B OFF	Offset of array B_j from B_0.	-
7	B IF	Spacing of elements in array B_j.	-
8	D FWA	Location of D_0 in data storage.	-
9	D OFF	Offset of array D_j from D_0.	-
10	D IF	Spacing of elements in array D_j.	-
11	FLAG	Option flag.	(2)
12	AU	Arithmetic unit selection.	(2)

(1) Following macros use this parameter block.

ADDVEC	IPVEC
SUBVEC	MAXE
MULVEC	MINE
DIVVEC	

(2) Refer to appropriate macro description for use of this parameter.

TABLE A-14. TYPE 3 PARAMETERS (1)

Order	Parameter	Description	Notes
1	A FWA	Location of A_0 in data storage.	-
2	A OFF	Offset of array A_j from A_0.	-
3	A IF	Spacing of elements in array A_j.	-
4	LA	Number of elements in arrays.	-
5	B FWA	Location of B_0 in data storage.	-
6	B OFF	Offset of array B_j from B_0.	-
7	B IF	Spacing of elements in array B_j.	-
8	C FWA	Location of C_0 in data storage.	-
9	C OFF	Offset of array C_j from C_0.	-
10	C IF	Spacing of elements in array C_j.	-
11	D FWA	Location of D_0 in data storage.	-
12	D OFF	Offset of array D_j from D_0.	-
13	D IF	Spacing of elements in array D_j.	-
14	FLAG	Option flag.	(2)
15	AU	Arithmetic unit selection.	(2)

(1) Following macros use this parameter block.

MAVVS
MAVSV
MAVVV

(2) Refer to appropriate macro description for use of this parameter.

Table B-1 summarizes macros available with the MAP III system. The following list shows MSAM call sequences.

CALL METOPEN(met, symtable, controlware, conaddr, status[, errlim])
CALL MAPSET(met, macbuf, length, status)
CALL MALLOT(met, aryname, maplen, 0, mem, status)
CALL MEQUIV(met, equivname, maplen, basearray, offset, status)
CALL MACRO(macstr, tag, macname, paraddr, status)
CALL MPARAM(macstr, tag, value, length, status[, loc])
CALL MAPNOGO(met, macstr, status[, ref])
CALL MAPGO(met, macstr, timtable, errtable, recall, estime, status[, febits])
CALL MODIFY(macstr, loc, value, status)
CALL MCLOSE(met, status)
CALL MRECALL(met, status)
CALL MRESET(met, status)
CALL MDUMP(x, y, z, c)
CALL MDRLSE

TABLE B-1. MACRO SUMMARY

Category	Group	Macro	Octal Macro Code	Name	Parameter Table
Control/ pseudo	Standard	NOOP	N/A	No operation	N/A
		JUMP	N/A	Jump	N/A
		RJUMP ①	N/A	Return jump	N/A
		HALT	N/A	Halt	N/A
		END	N/A	Terminate macro execution	N/A
		UPM	31	Update parameter	A-2
		TMM	32	Test macro memory	A-2
		XMM2DM	54	Transfer macro memory to data storage	A-12
		XDM2MM	54	Transfer data storage to macro memory	A-12
ECS input/ output	Standard	LOADP32	20	Load packed 32-bit words from ECS	A-3
		UNLDP32	24	Unload packed 32-bit words to ECS	A-4
		LOADP30	21	Load packed 30-bit words from ECS	A-3
		UNLDP30	25	Unload packed 30-bit words to ECS	A-4
		LOADL32	22	Load left-justified 32-bit words from ECS	A-3
		UNLDL32	26	Unload left-justified 32-bit words to ECS	A-4
		LOADR32	23	Load right-justified 32-bit words from ECS	A-3
		UNLDR32	27	Unload right-justified 32-bit words to ECS	A-4

① These macros execute from the read-only section of control memory and are not affected by controlware changes.

TABLE B-1. MACRO SUMMARY (Contd)

Category	Group	Macro	Octal Macro Code	Name	Parameter Table
Arithmetic	Signal processing	SUMPROD	00	Sum of products	A-5
		STKMOVE	01	Stack/move	A-6
	Fast Fourier transform	CPLXFFT	04	Complex fast Fourier transform	A-7
		ICPXFFT	05	Inverse complex fast Fourier transform	A-7
		REALFFT	02	Real fast Fourier transform	A-8
		INVRFFT	03	Inverse real fast Fourier transform	A-9
	Signal processing	FILTER	07	Filter design	A-10
		NMO	33	Normal moveout	A-11
	Vector	CVEC	40	Copy vector	A-12
		NVEC	40	Negate vector	A-12
		MVEC	40	Magnitude vector	A-12
		NMVEC	40	Negative magnitude vector	A-12
		ADDVEC	41	Add vectors	A-13
		SUBVEC	41	Subtract vectors	A-13
		MULVEC	41	Multiply vectors	A-13
		DIVVEC	41	Divide vectors	A-13
		IPVEC	42	Inner product vectors	A-13
		SUMRVEC	43	Sum reduction	A-12
		BCASVEC	44	Broadcast scalar	A-12
		ZEROVEC	44	Zero array	A-12
		MAXE	45	MAX elements	A-13
		MINE	45	MIN elements	A-13
		SQRTVEC	46	Vector square root	A-12
		COMVEC	47	Compare vectors	A-12
		MAVVS	52	Multiply add vector, vector, scalar	A-14
		MAVSV	52	Multiply add vector, scalar, vector	A-14
		MAVVV	52	Multiply add vector, vector, vector	A-14
		TVEC	53	Pretruncate vector	A-12

This appendix contains a detailed explanation of how to use the radix point adjust (RPA) word parameter for MAP floating-point ECS I/O operations. The term radix point is a general version of the more specific binary point and the more well-known decimal point.

COMPARISON OF CDC CYBER AND MAP FLOATING POINT FORMATS

Let F = EMMMM be a CDC CYBER floating-point number with E denoting the 12-bit sign-and-exponent byte and MMMM the 4 bytes of mantissa. The mantissa is considered to be a 48-bit integer MMMM.0 between 0 and $2^{48}-1$. That is, the radix point is at the right or least-significant-bit end of the mantissa. The integer mantissa, when multiplied by 2 raised to the power represented by E, yields the floating-point number represented by EMMMM. Now let f = emmm be a MAP floating-point number with e denoting the 8-bit sign-and-exponent byte and mmm the 3 bytes of mantissa. The mantissa is considered to be a 24-bit positive fraction less than 1. That is, the radix point is at the left or most-significant-bit end of the mantissa. The fractional mantissa, when multiplied by 2 raised to the power represented by e, yields the floating-point number represented by emmm.

NORMAL RPA USE FOR LOADING DATA

Unless the MAP is informed otherwise, it assumes that each number it loads has a fractional mantissa. In particular, EMMMM is treated as the fraction 0.MMMM times 2 raised to the power represented by E. This amounts to dividing the number that EMMMM represents by 2^{48} during the loading process. Since the MAP adds the RPA value to the exponent of the external number during loading, this division by 2^{48} can be compensated for by specifying RPA = 48 ($=60_8$) to request multiplication by 2^{48}.

USING RPA FOR LOADING SCALED DATA

If data to be loaded is outside the range of numbers representable in the MAP (approximately, numbers with absolute value between 10^{-19} and 10^{+18}), or if it is suspected that operations performed on data in that range will produce results out of that range, portions of the data should be altered by scaling during the loading process. As RPA = 48 multiplies data by 2^{48} to compensate for the difference between the CDC CYBER and MAP radix point conventions, external data may be transferred to the MAP multiplied by 2^n by using RPA = 48+n. For example, RPA = 48+10 = 72_8 multiplies input by 1024 and RPA = 48-10 = 46_8 divides input by 1024. Since RPA is a 12-bit two's complement value, special care may be necessary if 48+n is negative. For instance, multiplication by 2^{-49} during a load requires RPA = 48-49= -1 = 7777_8. This is not a 12-bit representation of 0.

USING RPA FOR UNLOADING DATA

The general rule is that if MAP internal data is to be unloaded with some radix point adjustment, use the two's complement (negative) of the RPA value used to load the unloaded numbers back into the MAP exactly as they were. To unload standard MAP numbers into standard CDC CYBER format with no scaling, use the two's complement of 60_8 ($= 7717_8+1 = 7720_8$) since 60_8 is the no scaling RPA for loading. To divide the numbers by 1024 on the way out of the MAP, use RPA = 7706_8 which is the two's complement of 72_8. Similarly, multiplication by 1024 during unloads requires RPA = 7732_8.

TYPICAL EXAMPLE OF SCALING

Suppose a macro string is to be constructed for the task of computing the autocorrelation of a sequence $T_0, T_1, \ldots, T_{999}$, where the terms in the sequence are all in the range 10^8 to 10^{10}. The terms $A_0, A_1, \ldots, A_{999}$ of the resulting sequence are defined by:

$$A_k = \sum_{i=0}^{999-k} T_i \, T_{k+i}$$

Each A-value is a sum of up to 1000 numbers in the range 10^{16} to 10^{20}, and so must be in the range 10^{16} to 10^{23}. This only partially overlaps the upper range of the MAP. One solution would be to load the sequence $t_0, t_1, \ldots, t_{999}$ defined by $t_i = T_i/1024$. Then the SUMPROD macro for the autocorrelation would produce the result sequence $a_0, a_1, \ldots, a_{999}$ defined by:

$$a_k = \sum_{i=0}^{999-k} t_i \, t_{k+i} = (1024)^{-2} \sum_{i=0}^{999-k} T_i \, T_{k+i}$$

Since $A_k = (1024)^2 a_k$ (approximately $10^6 a_k$) for each k, the a-values are in or slightly below the range 10^{10} to 10^{17}. The correctly scaled A-values can then be unloaded to ECS by unloading the a-values multiplied by 2^{20}. Therefore, one loads the

T-values with an RPA of $60_8-12_8 = 46_8$ and unloads the autocorrelation results with an RPA of $-(60_8-24_8) = -34_8 = 7743_8+1 = 7744_8$. Any possibilities of arithmetic unit overflow are thereby avoided. Underflow problems can be handled similarly.

GENERAL REMARKS ON USING RPA WORD

Two cautionary remarks should be made at this point. First, overcompensate by scaling if the range of the data to be loaded is not known exactly. Second, think very carefully about what will happen to the data inside the MAP. Theoretically follow through the entire requested MAP procedure (as in the previous example) to ensure that the chosen scaling of loaded data will not produce any unwanted underflows or overflows for intermediate results, and also to determine what scaling factor to use for unloading the final results.

INDEX

COMMENT SHEET

MANUAL TITLE CDC MAP III System User Reference Manual

PUBLICATION NO. 60428901 REVISION D

FROM: NAME: _____

BUSINESS ADDRESS: _____

COMMENTS:

This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number references and fill in publication revision level as shown by the last entry on the Revision Record page at the front of the manual. Customer engineers are urged to use the TAR.

CUT ALONG LINE

AA3419 REV. 6/78 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Publications and Graphics Division
ARH219
4201 North Lexington Avenue
Saint Paul, Minnesota 55112

CUT ALONG LINE

FOLD

FOLD