

Array Processor Support in GIPSY

by

Gregg Roland Fabregas

Thesis submitted to the Graduate Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science Applications

APPROVED:

R. W. Ehrich, Chairperson

J. W. Roach

J. G. Tront

November, 1989

Blacksburg, Virginia

Array Processor Support in GIPSY

by

Gregg Roland Fabregas

Roger W. Ehrich, Chairperson

Computer Science Applications

(ABSTRACT)

The CSPI mini-MAP array processor is supported for use with a RATFOR preprocessor in the software environment defined by the Generalized Image Processing System (GIPSY). A set of interface routines presents the mini-MAP as a tightly-coupled slave processor with well-defined rules for control from the host computer. The slave is programmed by adapting host-based software, using a proscribed set of guidelines for conversion. A software protocol has been defined to allow mini-MAP data memory to be allocated dynamically. Several examples of modified GIPSY commands are examined.

o ACKNOWLEDGEMENTS

I wish to acknowledge the support of several people. Foremost, I wish to thank _____ who hired me in the first week of my tenure as a graduate student, introduced me to the field of Image Processing, showed great confidence in my capabilities, and served as my thesis committee chairperson. I regret that he left campus before I could complete my degree.

I wish to thank _____ who recognized my talents during an interview, recommended me to _____, played an important personal and technical development role early in my graduate training, and served as a thesis advisor. I also regret her exodus from campus and her absence from my oral defense.

I wish to thank Dr. Roger Ehrich who was very supportive in helping me to acclimatize to Virginia Tech and, in particular, the Computer Science department, maintained communications after I left campus to work for the Procter & Gamble Company, and recently accepted the position as committee chairperson after

departure.

I wish to thank Dr. John Roach and Dr. Joseph Tront who were always willing to field questions, demonstrated a contagious enthusiasm for their respective fields, and attended my oral defense. Their presences have been felt throughout my tenure at Virginia Tech.

I wish to thank _____ and _____ for their assistance in teaching me how to manage the GIPSY package and navigate my way through its complexities. I wish to thank _____ who smoothed over the turbulence created by _____, and hired me on the Telesign project. Finally, I wish to thank all of the people in the SDA lab who made work at the lab more pleasant and continue to demonstrate their confidence, professionalism, and good cheer.

CONTENTS

o	ACKNOWLEDGEMENTS	iii
o	LIST OF TABLES	xiv
1	INTRODUCTION	1
1.1	Slave Processor	1
1.2	Programming Environment	3
2	BACKGROUND	9
2.1	What Is An Array Processor ?	9
2.2	History Of Array Processing	10
2.3	Marketplace Niches	11
2.4	AP Architecture	13
2.4.1	Parallelism	14
2.4.2	Synchronicity Vs. Asynchronicity	15
2.5	Mini-MAP	18
2.6	Newer Technology	18
2.7	Programming An AP	20
2.8	History Of GIPSY	22
3	GIPSY	24
3.1	Code Hierarchy	24
3.2	Flow Of Execution	26
3.3	Command Driver	27

3.4	Application Subroutines	28
3.5	Conditional Execution	29
4	INTERFACE ROUTINES	30
4.1	Software Partitions	30
4.2	Abstract Description	30
4.3	Memory Management	32
4.4	Background Support	35
4.5	Group One Routines	37
4.5.1	MMINIT Routine	37
4.5.2	MMCLOS Routine	39
4.5.3	MMLOAD Routine	39
4.5.4	MMEEXEC Routine	41
4.5.5	MMWAIT Routine	44
4.5.6	MMSKIP Routine	44
4.6	Group Two Routines	46
4.6.1	GETWPM Routine	47
4.6.2	MMALOC Routine	49
4.6.3	MMSNDA Routine	49
4.6.4	MMRCVA Routine	50
4.6.5	MMSNDS Routine	51
4.6.6	MMRCVS Routine	52
4.6.7	MMSSEND Routine	53
4.6.8	MMRECV Routine	54
4.7	Group Three Routines	55
4.7.1	OSMMAP Routine	57
4.7.2	MMINTR Routine	61

5	HOST SOFTWARE	63
5.1	Using The MMINIT Routine	63
5.2	Using The MMCLOS Routine	64
5.3	Using The MMEEXEC And MMWAIT Routines	66
5.4	Using The MMSKIP Routine	67
5.5	Allocation Of Arrays	69
5.6	Transmission Of Arrays	72
5.7	Transmission Of Scalars	72
5.8	Transmission Of Arbitrary Data	73
5.9	Using The OSMMAP Routine	74
5.10	Adaptation Of A GIPSY Command	77
5.11	Using A Double Buffer	81
6	MINI-MAP SUPPORT	86
6.1	FORTTRAN Compiler	86
6.2	Parameter Passing	89
6.3	Module Length	90
6.4	Include Files	91
6.4.1	Virtual Memory Protocol	92
6.4.2	X-driver Include File	93
6.4.3	Main Loop	94
6.4.4	MAPDRV Include File	95
6.4.5	MAPID Include File	97
6.4.6	Array Allocation	100
6.4.7	Generated Code	102
6.4.8	Virtual Array Definitions	103
6.4.9	Logical Memory Protocol	105

6.4.10	Subroutine Support	108
7	VAX SUPPORT	111
7.1	DCL Symbols	112
7.2	GIPSY System Flags	113
7.3	Utilities	115
7.3.1	RATFOR Preprocessor	115
7.3.2	GENCMD Procedure	117
7.3.3	GENMAP Procedure	118
7.3.4	DEBUG Facility	119
7.4	DCL Logicals	122
7.5	GIPSY Commands	123
7.5.1	INCLMM Command	123
7.5.2	MKRAN Command	126
8	BUILDING AN APPLICATION	128
8.1	Host Driver Modifications	129
8.2	Host Subroutine Modifications	131
8.3	MAP-driver Design	140
8.4	MAP-subroutine Design	150
9	RESULTS AND DISCUSSION	154
9.1	ABSIMG Command	155
9.2	EDGE Command	159
9.3	FACET2 Command	165
9.4	Discussion	167
o	BIBLIOGRAPHY	174

A	APPENDIX A: LISTINGS OF GROUP ONE ROUTINES	176
A.1	MMINIT Routine	177
A.2	MMCLOS Routine	184
A.3	MMLOAD Routine	188
A.4	MMEEXEC Routine	196
A.5	MMWAIT Routine	209
A.6	MMSKIP Routine	215
B	APPENDIX B: LISTINGS OF GROUP TWO ROUTINES	219
B.1	GETWPM Routine	220
B.2	MMALOC Routine	227
B.3	MMSNDA Routine	232
B.4	MMRCVA Routine	237
B.5	MMSNDS Routine	242
B.6	MMRCVS Routine	247
B.7	MMSEND Routine	252
B.8	MMRECV Routine	261
C	APPENDIX C: LISTINGS OF GROUP THREE ROUTINES . . .	270
C.1	OSMMAP Routine	271
C.2	MMINTR Routine	280
C.3	MMHSLD Routine	285
D	APPENDIX D: LISTINGS OF HOST INCLUDE FILES	290
D.1	MAPDEF Include File	291
D.2	MAPOPTS Include File	293
D.3	WINDOW Include File	294
E	APPENDIX E: LISTINGS OF MINI-MAP INCLUDE FILES . .	295
E.1	MAPDRV Include File	296

E.2	MAPCHAR Include File	298
E.3	MAPID Include File	299
E.4	MAPOPTS Include File	300
E.5	MAPSUB Include File	301
F	APPENDIX F: LISTINGS OF VAX/VMS PROCEDURES	302
F.1	GIPSYM Procedure	303
F.2	GIPLDG Procedure	305
F.3	GENCMD Procedure	307
F.4	GENMAP Procedure	313
F.5	MMDUM Linker Options File For GENCMD	317
F.6	MMSYM Linker Options File For GENCMD	318
G	APPENDIX G: INCLMM COMMAND	319
G.1	INCLMM Documentation	321
G.2	INCLMM Run-file	323
G.3	INCLMM Link File	324
G.4	DINCMM Routine	325
G.5	IMMARR Routine	331
G.6	RATLIN Routine	337
G.7	LINFIL Routine	345
G.8	RATTAB Include File	349
H	APPENDIX H: MKRAN COMMAND	352
H.1	MKRAN Documentation	353
H.2	MKRAN Run-file	355
H.3	MKRAN Link File	356
H.4	DMKRAN Routine	357
H.5	MKRNC Routine	360

I	APPENDIX I: ABSIMG COMMAND	367
I.1	ABSIMG Documentation	369
I.2	ABSIMG Run-file	370
I.3	ABSIMG Host Link File	371
I.4	DABSIM Routine	372
I.5	ABSIMG Routine	376
I.6	ABSIMG Mini-MAP Link File	380
I.7	ABSIMG MAP-driver	381
I.8	ABSWRK MAP-subroutine	382
I.9	XABSIMG Include File	383
J	APPENDIX J: EDGE COMMAND	384
J.1	EDGE Documentation	386
J.2	EDGE Run-file	391
J.3	EDGE Host Link File	394
J.4	DEEDGE Routine	395
J.5	EDGENC Routine	402
J.6	EDGE Mini-MAP Link File	419
J.7	EDGE MAP-driver	420
J.8	EDGENC MAP-subroutine	421
J.9	XEDGE Include File	425
K	APPENDIX K: RIDVAL COMMAND	426
K.1	RIDVAL Documentation	428
K.2	RIDVAL Run-file	433
K.3	RIDVAL Host Link File	435
K.4	RIDDV Routine	436
K.5	RIDNC Routine	442

K.6	HD1ZER Routine	449
K.7	RIDVAL Mini-MAP Link File	456
K.8	RIDVAL MAP-driver	457
K.9	RIDNC MAP-subroutine	458
K.10	HD1ZER MAP-subroutine	465
L	APPENDIX L: FACET2 COMMAND	470
L.1	FACET2 Documentation	473
L.2	FACET2 Run-file	475
L.3	FACET2 Host Link File	476
L.4	RFACT2 Routine	477
L.5	FACET2 Routine	485
L.6	SRESER Routine	499
L.7	APLBAS Routine	503
L.8	APBASR Routine	507
L.9	APBASC Routine	509
L.10	CONVC0 Routine	511
L.11	CANTRN Routine	513
L.12	OUTBCT Routine	516
L.13	INTSQS Routine	520
L.14	SQCNV1 Routine	523
L.15	DOP Routine	526
L.16	ROTBAS Routine	529
L.17	FACET2 Mini-MAP Link File	531
L.18	FACET2 MAP-driver	532
L.19	MSKSCL MAP-subroutine	533
L.20	DOP MAP-subroutine	538

L.21	APLBAS MAP-subroutine	540
L.22	XFACET2 Include File	544
o	VITA	545

- o LIST OF TABLES

- o TABLE 9-1: ABSIMG WITH A 50 X 50 IMAGE 156

- o TABLE 9-2: ABSIMG WITH A 512 X 512 IMAGE 158

- o TABLE 9-3: EDGE WITH A 40 X 40 IMAGE 160

- o TABLE 9-4: EDGE WITH A 1000 X 1600 IMAGE 162

- o TABLE 9-5: INTERFACE ROUTINES CALLED BY EDGE . . . 164

- o TABLE 9-6: FACET2 WITH A 40 X 40 IMAGE 166

1 INTRODUCTION

1.1 Slave Processor

Many software applications require a large amount of computation time particularly in image processing. The slow execution is compounded when the task is timeshared with many other users. A suitable approach to reducing the turn-around time uses parallelism, i.e., distributing the computation over more than one processing node. Often the overhead of scheduling and even the knowledge of the existence of additional processing elements are hidden by complicated operating system software and hardware. For example, few users working on a VAX-11 computer acquaint themselves with the floating-point accelerator which is hardware dedicated to computation with floating-point numbers and is completely transparent to the user. This optional processor increases the system's speed of computation. Unfortunately this type of resource is shared and is under operating system control. Features such as multiple central processor units are also not always available in the first place.

One means around this limitation employs another computer, hereafter called the slave, subordinate to the principal host computer as an external device. There are

several advantages to this scheme:

1. The slave is single-user oriented and allocated for private use. Its use can be dictated and controlled by the need for large amounts of computation time. Applications which are computation intensive can be downloaded to the slave, relieving some of the load on the host.
2. As a dedicated device the slave can be tailored to specific applications, notably image processing. Most image processing is repetitive, generating results pixel-by-pixel. Demands on memory for both input and output data also tend to be sequential and predictable, facilitating the advanced queueing and buffering of data.
3. If the user has complete control over the slave, then a great deal of cooperation between the host and the slave can be achieved through proper programming. The host must concern itself with scheduling and resource allocation. The slave can perform computation while the host attends to system overhead and device I/O.

4. There is a question of cost. Entire systems, especially those capable of multiple-user timesharing, are infrequently procured. Existing systems are expected to serve user needs for many years. It is more feasible to attach special-purpose equipment in cases where user needs are not being met than to overhaul the entire system.

This thesis is concerned with one such slave processor.

The most notable drawback with slave computers is their infrequent use, especially by those programmers who have not received personalized assistance in constructing an application that exploits the slave's capabilities. As an important corollary it is presumed that programmers will not utilize the slave processor if it appears complex. This mandates an interface scheme that presents minimal complexity to the programmer. Furthermore it is intended that activity on the slave should be transparent to the user.

1.2 Programming Environment

At this point further design decisions require knowledge of the intended programming environment, the intended applications to be implemented on the slave, and

some of the constraints placed on the interface by the host and slave. The environment for this thesis is the Generalized Image Processing System (GIPSY). This is a large set of implemented applications in image processing using common support software for low-level operations such as I/O and string manipulation. There are several attributes of the application software in GIPSY that affect the design of the programmer interface:

1. Portability limits the structure of the procedure calls and the procedure bodies. All of the procedure names are limited to six letters and a clear majority of the operands are integers. The machine-dependent portion of the code is easily identified and clearly labeled.
2. Many of the applications are time-consuming and thus would benefit from the additional computational capability. Moreover, the slave must be tailored to each application independently.
3. GIPSY imposes a strict discipline on the order of many administrative functions performed in the course of application execution. This is advantageous to extensions to GIPSY by encouraging the imbedding of

interface routine calls in pre-existent code common to all applications, and by suggesting familiar syntax in the form of calls which must be placed in specific points in the application code. Experienced programmers can easily remember procedure calls when their respective argument lists are identical, or at least very similar to other known procedures performing analogous operations.

4. GIPSY programmers are familiar with abstract data types. The interface has been designed to encapsulate most of the complexity of the slave so that the programmer need not be concerned with the implementation details of the interface. This approach is analogous to GIPSY's encapsulation of the Standard Image Format (SIF) file definition which includes concepts such as open, close, read, write, and wait.

5. A number of programming conventions are used in GIPSY. This is evident in coding style as features such as the Integer Event Variable (IEV) and the alternate return used in procedure calls. The IEV returns a predefined code when control is returned to the label specified by the alternate return. When the failure of a procedure is not critical to the overall performance of the

application, the procedure is implemented as a function which evaluates to the IEV return code.

6. Input and Output operations are logically interleaved with computational sections of code. This engenders a need for some means to isolate interface operations from conventional file I/O, and a need for the support of asynchronous operations. The former is important in error recovery when the interface or slave fails to perform satisfactorily. The latter is particularly evident when the programmer chooses to utilize dead time caused by delays propagated in the host, slave, and disk operations.

It should be clear that many of GIPSY's potential assets and liabilities play a significant role in what is presented to the programmer.

As mentioned the primary role of GIPSY software is image processing. The slave interface is compatible with all of the applications already defined in GIPSY but may not be useful in all cases. The exceptions include situations where computation is not an issue such as device allocation. Nevertheless, the programmer can insert the interface support, and expect no errors to be generated due to missing

hardware or slave support software. Slave utilization is maximized when the slave is compute bound and not interrupted by I/O operations. The number of breakpoints requiring exchange of information should be minimized. For example, if an application loads an input buffer, does a significant amount of computation on that buffer, yielding results in some output buffer, there is no overhead besides memory consumption in regards to intermediate arrays, and limited competition with I/O. Moreover, a simple scheme such as this is amenable to optimization by implementing rotating buffers and asynchronous I/O. Efficiency decays when intermediate results must be output. This requires additional breakpoints, tending to use the slave for shorter bursts of time. Also, more overhead is needed for interface operations.

Another consideration is the potential for failure. An algorithm on the host that does not check input parameters or intermediate results does not need to check the slave for analogous conditions. This is apparent in GIPSY source code by the lack of the IEV or alternate return used in some subroutine calls, notably the generation of coefficients by the central neighborhood cubic facet operator (CNCFLN routine). This particular routine is very CPU intensive and is uninterruptable except by FORTRAN run-time conditions.

This makes the CNCFLN routine an excellent candidate for downloading to the slave.

Many of GIPSY's applications are well-suited for slave support. Probably the most frequently used of the more time-consuming applications is the least squares fit of polynomial surfaces of arbitrary degree to the graytone values of an image. For this particular class of operations, many routines meet the previously cited recommendations for procedure structure. For example, the APLBAS routine and all of its subroutines are responsible for convolving columns and rows with respective row and column basis vectors for the generalized facet operator. This routine is bounded by input and output operations but has no provision for alternate error returns. See the Appendix for a conversion of the GIPSY source code with slave support.

Explanations of the constraints placed on the design by the host and the slave allude to many architectural details. The interface is designed to isolate these details so that preferably only one or two routines are affected by a given constraint. Therefore, these constraints affecting a particular routine will be enumerated in the explanation of each interface routine later in the thesis.

.spacing 2

2 BACKGROUND

2.1 What Is An Array Processor ?

An array processor, or AP, is generally a device designed for high-speed computation with arrays of data, typically involved with scientific applications [Martin 1980]. It could be as small as a single board such as the Skymnk-C (Sky Computer's Skymicro Number Kruncher for the Chromatics's system bus) board [Kotelly 1983], or as large as a mainframe such as the ILLIAC-IV [Erickson 1975]. However, in the context of this thesis the definition supplied by Walter Karplus and Danny Cohen [Karplus 1981] defines the hardware more precisely:

The term "peripheral array processor" denotes a digital computing device with all of the following features:

- (1) It is designed as a peripheral for a conventional host computer and is intended to enhance the performance of the host in specific numerical computing tasks.
- (2) It achieves high performance through parallelism and/or pipelining.
- (3) It includes an arithmetic section containing at least one adder and one multiplier capable of operating in parallel.
- (4) It can be programmed by the user to accommodate a variety of arithmetic problems.

Thus, supercomputers such as Burroughs' BSP, CDC's Cyber 205, and the Cray-1 and the systems composed of arrays of

processors, such as the ILLIAC-IV, the Goodyear Staran array, and the Carnegie-Mellon Cm* array of microcomputers are excluded [Theis 1981]. On the small-scale end of the spectrum, arithmetic chips serving as coprocessors such as the Intel 80X87 [Cushman 1982], and single-chip systems such as the Intel 2920, AMI S2881 [Bernhard 1982], and Weitek WTL1167 [Allison 1987] are also excluded.

2.2 History Of Array Processing

In the early 1950's, serious discussions on the need for array processing were surfacing [Evans 1975]. D. J. Evans cites a typical set of operations that includes the following:

1. Move vector within memory
2. Convert vector elements from floating-point to integers
3. Multiply array by a scalar and add to another array
4. Three-array addition ($C = A + B$)
5. Sum the array elements
6. Sum the squares of the array elements

7. Compute the dot, or inner, product of two arrays
8. Square the array elements, and
9. Partial matrix multiplication for a single matrix row

Frequently, two convolution operators are also suggested.

Early systems tended towards the use of arrays of processors, such as the Soloman machine and the ILLIAC-IV in the early and mid 1960's. By the early 1970's manufacturers of large-frame computers jumped on the bandwagon with add-on processors such as the IBM 2938 and IBM 3838 [Zakharov 1984]. These systems were powerful but too uneconomical for laboratory environments or data acquisition. Minicomputers were available for a modest cost but had limited computing power and precision [Alexander 1981]. A clearly defined niche in the marketplace had developed for inexpensive, high-speed, add-on processors.

2.3 Marketplace Niches

The marketplace can be divided two ways, by data type and price/performance. By the mid 1980's some companies such as Numerix were manufacturing fixed-point processors primarily for the signal-processing environment. A system restricted to integer arithmetic has relatively narrow range

of application but offers faster speeds as compensation. The principal alternative data type was floating-point [Wilson 1982]. The mini-MAP used in experiments for this thesis is an example of such an array processor. Note that it could perform integer operations but its fastest components are dedicated to floating-point operations. The attention given to other data types, such as character, has remained small to this date.

Historically, manufacturers of array processors have historically targeted relatively narrow ranges of price and performance for their systems. Sky Computers targets the low end of the spectrum with their Skymnk series one- or two-card systems [Kotelly 1983]. These systems are unusual in that a design decision was made to limit severely the amount of data memory on the cards. Instead, host memory is used for problem storage and work space. The system does not require explicit input or output of data at all. Since the card(s) plug directly into the host's backplane, calls to the system do not tie up the host operating system, and communication is direct, through the host's I/O memory page [Wilson 1982].

Most manufacturers aim for the middle of the cost/performance curve, including Computer Signal Processing, Incorporated, or CSPI, the manufacturer of the mini-MAP. They all design their systems primarily for minicomputers such as the PDP-11 or the VAX-11, or microcomputers such as the now archaic LSI-11. Each of these systems have a large amount of dedicated data memory and thus incur the overhead of transmitting data to or from host memory. These systems are differentiated from high cost/performance machines such as the Star Technologies ST-100 or Floating Point Systems FPS-100 by an arbitrary boundary set at \$100,000 and half the machine cycle times of the mid-range systems such as forty ns for the ST-100 [Kotelly 1983].

2.4 AP Architecture

Early in the evolution of computer technology there was plenty of opportunity for increases in component speed. Demands for speed were generally answered by faster (SIMD) uniprocessors, i. e., processors that repeatedly fetch and execute a single instruction. However, once intrinsic limits such as propagation delays were approached, emphasis shifted towards the use of parallelism and asynchronicity [Cohler 1981].

2.4.1 Parallelism -

There was enough hardware in the CSPI uniprocessor to accomplish multiple tasks such as a memory write and a branch in the same instruction cycle, but the controller did not allow this combination of activities in a single instruction. According to Edmund Cohler of CSPI, "a microcoded controller presented the programmer with an instruction field for each piece of equipment in the structure. The more fields he could fill on a given line of code, the more efficient the usage of the parallel hardware."

Many applications in signal processing, for example, a Fourier transform, are composed mostly of additions, multiplications, and divisions. Array processors are generally designed to facilitate additions and multiplications. Divisions can be performed with the multiplication of reciprocals based on the Newton-Raphson algorithm. This scheme uses some initial approximation, $X(0)$, generated by an approximate reciprocal operator. CSPI uses a 256-element lookup table in its operator for a value within 2×10^{-7} of the correct reciprocal. Successive approximations, $X(n+1) = 2X(n) - RX(n)^2$, where R is the divisor, converge rapidly on the reciprocal. Thus, an

obvious design approach is to provide one or more parallel adders and multipliers.

This architecture also lends itself to another form of parallelism, pipelining. All functions are divided by the architecture into steps that can be executed in some integral number of clock cycles. Each step within a function executes independently of any other step or function. This architecture allows more than one function to be actively processing data, and if a function contains more than one step, then for each step an independent datum can be processed. Given that the above adders and multipliers have input and output buffers, and there exist data paths linking these buffers with memory and each other, then it takes a small increase in hardware complexity to chain these functional units together. This concept is not limited just to adders and multipliers. Instruction fetches and decoders can also be included, but the programming will become increasingly complicated. The Cray-1 has a variety of scalar and vector functional units which can be pipelined, and has served as a good reference in the design of smaller AP's [Karplus 1981].

2.4.2 Synchronicity Vs. Asynchronicity -

The working definition for an AP allows the functional units and addressing to run under the control of one program and one clock as in a synchronous AP, or more than one program and more than one clock as in an asynchronous AP. There are advantages and disadvantages to each architecture, and both are well represented in the marketplace.

Consider first the synchronous AP, using a Floating Point Systems, Inc. architecture. There is one control processor which decodes a multifield instruction every 167 ns. There is one adder and one multiplier. An instruction can include computation, memory addressing, and control all at the same time. Every operation is pipelined and takes an integral multiple of clock cycles to execute: two for addition and addressing, three for multiplication. Odd and even memory addresses are in separate memory banks so that consecutive memory addresses can be available every clock cycle [Cohler 1982]. This architecture eliminates the need for programmer coordination of machine elements. Only one program is needed and that program can be tested on a simulator since the timing is relatively simple. As a result program libraries tend to be larger than those for asynchronous AP's [Martin 1980]. The major drawback is the difficulty in filling and maintaining both pipelines. Also, non-linear addressing will slow down throughput, either by

forcing consecutive memory requests onto the same memory bank, or by increasing the complexity of the intermediate address calculations beyond the typically hardware-defined increment or decrement of an integer by one. Synchronization is a necessary concern in every program.

The alternative architecture, using a CSPI MAP system, has three independent processors for control, arithmetic, and host-interfacing, respectively, and three independent multiport memories, with one for programs. The processors have their own clocks, operating at 170 ns for the control unit, and may access memory at will. Memory may have speeds from 170 to 500 ns, depending upon cost. The arithmetic processor has a 250-ns adder and a 500-ns multiplier operating in parallel. To synchronize these elements one-, two-, or three-element queues have been inserted between elements. When an input queue empties or an output queue fills, the processing element halts. Inherently faster elements are not impeded by slower ones. Also, a continuous stream of data in real-time is better handled. An asynchronous AP can make good use of the multiple buses when input data activity becomes very intense. Unfortunately, coordinating all of these elements is a complex job and requires a separate program for every processor. Since the timing is very complicated, it would be very difficult, if

not impossible, to test the execution by simulation. Therefore, debugging must tie up the AP hardware.

2.5 Mini-MAP

As the name implies, the mini-MAP is a small scale version of an older system called the MAP (Macro Arithmetic Processor) and executes asynchronously using pipelining. The mini-MAP has two processors, one for addressing and one for arithmetic. The arithmetic processor has a parallel adder and multiplier. Each processor has its own memory and instruction set. Connection to the VAX-11 is through the Unibus as a peripheral device. Some features of this architecture will be studied in more depth in later chapters. Since this thesis does not focus on hardware activities within the mini-MAP, references to internal hardware will intentionally be minimal.

2.6 Newer Technology

The mini-MAP became available in the early 1980's. The research for this thesis was completed by early 1986. According to Bernard Pelon of CSPI, two new product lines have succeeded the mini-MAP thus far. The first product, called the MAP-4000, is a three-board version of the earlier MAP system designed for DEC's micro-VAX Q-bus. This system is rated at 40 Mflops for the 32-bit floating-point model,

and 20 Mflops for the 64-bit floating-point model. This compares with six to seven Mflops for the mini-MAP. Included with the system is 64 kilobytes of local memory similar to that in the mini-MAP and a choice of two or eight Megabytes of main memory, rated at a peak access rate of 125 ns per integer. This compares favorably with the 375 ns per integer access rate for the mini-MAP. Also faster is the 100 ns delay to complete two additions and two multiplications. Even faster results can be obtained with the other product line, the one-board Supercard-860 system. This card has a VME interface for Sun Microsystems workstations and is rated at 66 Mflops.

These products fit into a broader class called minisupercomputers, used as cost-effective, high-performance departmental processors, compute servers, and workstations. Array processors are now typically identified by their 32-bit floating-point word, called by host-based firmware [Allison 1987]. This new class is targeting a processing rate of ten to 100 Mflops at a cost of \$100,000 to \$1.5 million, and the market is growing. The demand for processing capacity is further evident in the introduction in 1986 of Thinking Machines Corporation's Connection Machine, which is a 65,536-processor hypercube able to yield over 2.5 Gflops. Although The Connection Machine is not a

minisupercomputer, it and a growing number of other systems identify a trend towards increasing computing power [Dugdale 1987].

Trends also indicate that a given amount of computing power is fitting into a smaller volume. Consider the 1989 entry of the Intel i860 64-bit microprocessor. This single chip contains a multiplier unit, an adder unit, and a graphics unit, all accessed by a floating-point control unit, which also executes in parallel to a RISC (Reduced Instruction Set Computer) Integer Core. Like the mini-MAP, it has the integer, floating-point, and memory control units. More striking is the presence of a reciprocal operation and the lack of a divide operation. This chip can operate at a peak rate of 80 Mflops using the instruction and data caches [Kohn 1989].

2.7 Programming An AP

Sam Martin [Martin 1980] suggests three programming alternatives for exploiting an array processor. Starting with the easiest to implement and ending with the most efficient, these are:

- o Program in FORTRAN and use a host-resident FORTRAN compiler for conversion to an array processor's assembly language.

- o Use a series of FORTRAN calls issued by the host.
- o Program directly in the array processor's assembly language.

The use of FORTRAN with a host-resident compiler is the only approach advocated in this thesis due to the ease of implementation of software and the programmer's expected familiarity with FORTRAN. CSPI provides a FORTRAN compiler, MFORT, for programming in a subset of FORTRAN-77 and an assembler, MASM, for programming in the assembly languages of the two processors. However, no provision was made for programming FORTRAN calls issued by the host. Instead, a precursor to the MFORT compiler, MCL, is used to code the calls for the mini-MAP processors using a WINDOW common to communicate the data between the host and the AP. In this thesis, an approach was taken to eliminate the need to make any direct calls to any of these facilities. Further, it will be shown that the programmer can convert existing GIPSY code for use on the mini-MAP without great difficulty.

This thesis will also attempt to refute an assertion made by Walter Karplus and Danny Cohen [Karplus 1981]:

In the case of peripheral array processors, ... every programming manual includes detailed diagrams of the machine architecture. Without them, it is impossible to program the array processor. It is necessary to know which operations share which buses because operations sharing buses cannot be executed in parallel. The programmer must also know how the registers are

interconnected and many other details on that level.

While it is reasonable to say that the most efficient programming requires an understanding of the machine architecture, it is not necessary to know the architecture to program. A software interface will be illustrated in another chapter that encapsulates essential mini-MAP functionality from the host side. A software protocol has also been defined which will be explained later and which encapsulates the host-slave interface from the slave side. The remaining mini-MAP architecture will be hidden by a RATFOR preprocessor supplied with GIPSY and the FORTRAN compiler supplied by CSPI.

2.8 History Of GIPSY

In the early 1970's a set of image processing routines were brought together under the collective name, KANDIDATS, by Robert M. Haralick and several students at the University of Kansas. This system served as the predecessor for a new system, GIPSY, developed by Scott Krusemark under Dr. Haralick's tutelage at the Virginia Polytechnic Institute and State University in the early 1980's. After Mr. Krusemark's tenure as GIPSY manager the reins were delivered into the hands of Oscar Zuniga, and then to the author in 1982. The system is composed of command modules

that may be chained together, usually using disk files as intermediate storage. Starting with 130 commands in 1981, to date, GIPSY has 371 commands. More will be said of GIPSY later in this thesis [Krusemark 1981].

3 GIPSY

3.1 Code Hierarchy

Before delving into the interface routines some background is required on features inside most GIPSY applications. GIPSY is logically separated into four levels of code. Each level contains subroutines which may only call routines at the same or lower levels. Moreover, each level is distinguished by its scope of access, knowledge, and purpose. The following table characterizes each level:

1. Control - This is the highest level in GIPSY. It is responsible for activities common to all applications, hereafter distinguished by a unique command name. There is an interactive communication facility for accepting commands from the user which activates application software to do a specific task for the user. This level is also responsible for error recovery when commands fail and setup of the run-time environment, such as the input source and default outputs for every command. This is a suitable location for interface routines responsible for slave initialization and shut-down.

2. Application - The bulk of the source code in GIPSY falls into this level. The code at this level is tailored to individual commands. All needed modifications to add slave support to individual applications, i.e., the calls to the interface routines in the next level, occur at this level. Routines within this level may communicate only if they are enumerated in the definition of the command by the GIPSY utility called GENCMD, which is used to build GIPSY commands. Much of the code executed on the slave will be adapted from code at this level.

3. Primitives - High-level programming tools such as those that format output are stored at this level. Many abstractions, including the definition of external data files and the majority of the interface routines, are programmed at this level. The code is generally command-independent and machine-independent. It is important from a development viewpoint that this and lower levels are invisible to the programmer. That is, the programmer need not be concerned about implementation details at this level, nor the specification of routines from this level in the GENCMD utility.

4. Kernel - Low-level, machine-dependent programming tools are stored at this level. This is, in fact, the lowest level in GIPSY. Most of the knowledge about the operating system is encapsulated by this level, presenting a standardized set of facilities to the above levels. The slave interface is designed to keep as much machine-dependent code at this level as feasible. The most important restriction at this level is the lack of FORTRAN common declarations. The effect of this restriction will be explained later.

There are additional libraries in GIPSY that extend the power of the Application and Primitive levels, but they are not important to the slave interface. All references to code in this thesis will be related to the above levels.

3.2 Flow Of Execution

A typical execution of a GIPSY command begins in the control level. Arguments such as file specifications are recognized syntactically and parsed. The process terminal, if present, and default interactive run-time I/O streams are defined. When a command is run in batch mode, input is from a file, or when output is being recorded by the history logging facility, then the user terminal is not the default I/O stream. If timing statistics are being monitored, an internal clock is reset. Control can now be transferred to

the application level by calling the entry routine unique to each command, called the command driver. When the command driver returns control to the control level an error trace is dumped if an error is reported by the command driver, and then the timing statistics are computed, if that feature is enabled. The user is then prompted for the next command, and the process repeats.

3.3 Command Driver

The command driver is responsible for setting up resources needed by the command and for releasing those resources when done. There are a number of facilities which may be used, but only those relevant to the slave interface will be discussed. The executable code begins with a call to PPUSH. All application level routines begin with this call. Input conditions are checked to verify that the command may continue. A work space pointer is initialized to one prior to a sequence of function calls to GETWP that allocates work space from the array space designated by the first argument in the command driver subroutine declaration. The sequence of GETWP calls is terminated by a function call to OSALOC in an IF statement. The OSALOC routine verifies that memory demands have not exceeded memory capacity. One or more calls are then made to subroutines that will execute the operation defined by the command. If successful, a call

to PPOP is done just prior to the RETURN statement, returning control to the GIPSY control level. For failures, error codes may be placed in the IEV variable, and control is returned via the RETURN 1 statement.

3.4 Application Subroutines

Subroutines below the command driver have a variety of formats and demonstrate large degrees of programming freedom. However, there are some clearly discernable operations that should be noted. Computation is bounded by calls for input and output. Input is most frequently from disk, using calls to RREAD, AREAD, PREAD, routines with "GET" embedded in their names and a file descriptor as the first argument, or routines whose names begin with "RNGET" and contain quoted text as the first argument. A file descriptor specifies a device, file specification, and status information. It is usually identified by an array of size ".FDLENGTH" which begins with the letters, "FD". These operations bring data into reach for computation, so it is expected that processing will follow shortly. Output is usually done after computation, so it is expected that processing has completed. Calls generating output corresponding to the above routines include RWRITE, PWRITE, and calls with "PUT" embedded in routine names with a file descriptor as the first argument. Thus far, all of the I/O

routines mentioned are members of the primitives level. The programmer should also be wary of any application level routines that perform I/O, especially if they include file descriptors in their argument list.

3.5 Conditional Execution

Flags and conditional flow controls such as the IF statement can complicate the issue, particularly if segments of computation targeted for slave support are involved. If the conditional control encloses only parts of code that could be host- or slave-executed, then the slave may need additional information pertaining to proper synchronization with the host. Methods of synchronization will be detailed later. If the conditional control is embedded in the computation, then both host and slave code must reflect this parity. The latter approach is preferred since the adaptation of host code to the slave is simpler and more straightforward.

Most of the interface routines discussed in the next chapter will be classified as GIPSY primitives. One kernel-based routine will handle most of the machine-dependent activities not already managed by software supplied with the array processor.

4 INTERFACE ROUTINES

4.1 Software Partitions

From the programming perspective, the interface routines can be partitioned into three groups separated by complexity and usage:

Software Partitions

Group 1: Abstract description of the slave

Group 2: Memory management capabilities

Group 3: Background support

This chapter will focus on each group, both separately and together.

4.2 Abstract Description

The first group presents the slave as a black box with a slot for programs, start and reset switches, and stop and ready indicators. Associated with the slave are five operations, hereafter referred to as initialization operations. These are termination, downloading, execution, synchronization, and status-checking. Initialization is the process of resetting the hardware and software context to a known state, enabling further activity with the slave (pushing the reset switch and permitting access to the black box). Termination is the cessation and disabling of slave activities (pushing the reset switch and hiding the black

box). Downloading is the procedure for transferring an executable sequence of instructions or program module from a host-controlled storage device such as a disk, to the instruction memory in the slave (inserting a program into the slot). Execution is the activation of the slave program (pushing the start switch). Synchronization is the suspension of host activity until the slave program has halted (wait for the stop indicator). Finally, status-checking reports the availability of the slave for execution, i.e., a successful initialization (check the ready indicator). The following list enumerates the relationships between each operation:

1. No operation may precede initialization except another initialization and status-checking.
2. No operation may follow termination except another initialization and status-checking.
3. Default downloading is attempted during initialization.

4. A valid program must be downloaded prior to execution.
5. Downloading may follow synchronization. That is, the slave may be reprogrammed if it is known to be halted.
6. Execution precedes synchronization or is done concurrently.
7. Status-checking may be done anytime.

Status-checking returns true if downloading has completed, and false otherwise.

4.3 Memory Management

The second group of routines handles memory allocation and data transmission. The slave contains its own memory, which may be accessed using a virtual, logical, or physical protocol. The virtual protocol allows the interface routines to dynamically allocate memory for the programmer, eliminating any concern on the programmer's part over how memory is managed. The programmer is required to identify explicitly each memory partition with a name, data type, memory consumption, and request allocation before referencing the partition. In addition, a number of integer and real scalars are reserved for programmer use. These

scalars are simply numbered and can be named via RATFOR definition. The virtual protocol supports specification (naming of arrays), allocation (including total deallocation), sending scalars or arrays to the slave, and receiving scalars or arrays from the slave.

The logical protocol defines slave memory as a contiguous sequence of slave-specific integers. The programmer must explicitly reference the first integer as the starting boundary of a block of data whose structure is superimposed on the integer array. Scalars and multi-dimensional arrays may arbitrarily be referenced. The first integer is numbered one. Although the memory is divided into integers, any data type defined in GIPSY may be used. The logical protocol supports sending and receiving of arbitrary data.

The physical protocol is used by the interface routines, but is not directly accessible to the programmer from this group of interface routines. More will be said about this protocol in the explanation of the MMSEND routine. It is sufficient here to say that access to slave memory under the physical protocol requires more understanding of memory-mapped I/O, the linker, and the slave than either of the other protocols.

There is some interaction between the first and second groups of interface routines. The first group of operations is closed under order of invocation. That is, for any combination of calls, the interface routines will either work within the context created by the prior calls or reject the context with a specific error message. However, there is a memory option in the status-check operation that can alter the returned value. Essentially, if memory is required, then the status-check will return true only if memory is allocated by the virtual protocol and the previously defined check condition is satisfied. The order of use of the second group routines is also well-defined:

1. No operation may precede initialization.
2. No operation may succeed termination.
3. Specification of virtual memory demands precedes allocation.
4. Allocation is cleared during initialization.

5. Allocation may be cleared anytime after initialization and before termination.
6. Allocation may not be performed twice without an intermediate clearance.
7. Allocation may succeed the downloading of a program module.
8. Transmission of data to or from virtual memory in the slave may follow allocation.
9. Transmission of data to or from logical memory may follow the downloading of a program module.

For most applications the programmer will need only the virtual memory protocol, which is much easier to use than the logical protocol. Moreover, the virtual protocol fits into GIPSY source code with much less effort. The logical protocol is included primarily for completeness.

4.4 Background Support

The third group contains code used for background GIPSY support. That is, either the code will not be used by the

programmer because its use is intended for environmental definition outside the application level, or the operations are low-level and are generally expected to be used only by the interface routines. This is the case when machine-dependent code is pushed down from the primitive level to the kernel level. This group includes an interrupt handler for trapping conditions signalled by the system which are related to slave operations.

The implementation of each of the aforementioned operations is obviously dependent upon the slave being supported. The selected hardware is the model MM-111 mini-MAP array processor distributed by CSPI. The naming of the interface routines reflects this selection by starting each routine with "MM", followed by four letters identifying the function. There are two exceptions to this scheme due to the naming conventions used in GIPSY source code. Routines which may fail without disturbing the normal flow of GIPSY commands are formatted as functions, so an IF test can be used to test the returned IEV for failure or the IEV can be discarded. The usual format is a subroutine call which uses an alternate return to break the normal flow of execution when the operation fails.

4.5 Group One Routines

The first group contains the following routines:

1. IEV = MMINIT(0)

2. IEV = MMCLOS(0)

3. CALL MMLOAD(MODULE, IEV, LABEL)

4. CALL MMEEXEC(WAIT, IEV, LABEL)

5. CALL MMWAIT(TIMER, IEV, LABEL)

6. FLAG = MMSKIP(MEMORY)

Each routine will now be considered separately.

4.5.1 MMINIT Routine

```
IEV = MMINIT( 0 )
```

The MMINIT routine is responsible for the initialization of the interface package. It takes one input argument that is currently reserved for future use, so the argument should be set to zero for future compatibility. Initialization in this case is a dynamic process, using information from the system and GIPSY to decide in what starting state the interface should be placed. From the system, information about the availability of a slave processor is essential to future operations. From GIPSY, MMINIT must determine whether the user wishes to inhibit slave operations, and if not inhibited, what the name of the command is. The command name is compared with the unique name returned by a previously downloaded program module, if one exists, to determine whether a new program must be loaded. Programs used by the slave for GIPSY applications include an identification scheme that allows repeated executions of the slave programming without automatically reloading the program. Also, the correct results serve as the first stage of synchronization, verifying the state of the slave, and proving that the host and slave are communicating properly. Along with identification, the slave sends back contextual information for the host. This context will be explained in the description of the MMLOAD routine.

4.5.2 MMCLoS Routine

```
IEV = MMCLoS( 0 )
```

The MMCLoS routine is responsible for cleanup at the end of the current execution with the slave. It takes one input argument that is currently reserved for future use, which should be set to zero for future compatibility. The cleanup includes halting any activity on the slave and returning the slave to the common resource pool. All context related to package status is lost, requiring another call to the MMINIT routine before any further activity using the slave.

4.5.3 MMLOAD Routine

```
CALL MMLOAD( MODULE, IEV, LABEL )
```

The MMLOAD routine is responsible for downloading the slave with a program module stored on a host disk. The single input argument specifies the generic file name of the load module. The full file specification is built internally from the specified file name. The name must be a GIPSY character array with a maximum of six characters

followed by a ".EOS" character. The IEV is filled with an error status if an error occurs during the operation, and control is returned at the specified label. Under normal operations, the current module in the slave is automatically replaced by the new module, and the name and context of the new module are checked.

The context consists of values with predefined significance regarding actions in the host. An array with each element set to -1 is sent to the slave. The slave replies by altering those locations with non-negative values that will override default values in the host. The array location determines the meaning of the value. The following table explains the meaning of each defined slot:

1. .OPTTIMER - Set the maximum time in seconds that the slave may execute after the call to MMEXEC before it completes its activity. A time of zero is interpreted as no limit. The default time limit is 30 seconds.
2. .OPTSIZE - Set the maximum number of elements allocated in a single array using the virtual memory protocol. A positive value sets the limit, and zero clears the limit. The default is 32767 elements.

3. .OPTDEBUG - Set the execution mode for debugger support. The default provides no support. Debugger support will be explained in the section on the MMEXEC routine.

Other options can be added at a later date. The instructions for setting these options will be explained in the discussion on mini-MAP software.

4.5.4 MMEXEC Routine

```
CALL MMEXEC( WAIT, IEV, LABEL )
```

The MMEXEC routine is responsible for activating the slave so it can execute its programming. The single input argument is the wait option. If the argument is .WAIT, a GIPSY symbol for the RATFOR preprocessor, then MMEXEC does not return to the calling routine until the slave has halted or the time-out interval has expired. If the argument is .NOWAIT, another GIPSY symbol, then MMEXEC returns immediately. Another wait option, .NOWTARR, will be explained shortly. As in the MMLOAD routine, the IEV and alternate return LABEL are used if an error occurs.

Locating and correcting errors in logic can cause considerable difficulty, especially when there is no interactive communication facility for displaying intermediate results. To remedy this difficulty, a debugging option, called .OPTDEBUG, is defined by the MMLOAD routine. This option, specified by the slave program, enables support from the MMEXEC routine so that the programmer can more easily debug slave programming at run-time. When enabled, the MMEXEC routine performs the following operations prior to slave startup:

1. Two debugging status variables are recorded for later comparison. Their purpose will become clearer as the process is explained;
2. The host attempts to open an optional lookup file, built by the MKRAN command in GIPSY from a list of translations between status values for the second status variable and user-defined statements;
3. The virtual memory scalars, I1 through I20, and R1 through R20, are also recorded; and

4. Illegal address exceptions are suppressed until after the debugging processing is completed.

A breakpoint is defined in slave code as a RATFOR macro that modifies one or both debugging status variables just prior to halting. In response, the host can distinguish between breakpoints and normal halts in slave execution. After the initial startup in the MMEXEC call, the slave advances to the next breakpoint and halts. At each breakpoint, the scalars are retrieved, and changes are reported. If the .WAIT or .NOWAIT option was specified in the call, then virtual memory arrays are also retrieved. The programmer can use the host debugging facility to monitor changes to array locations. The .NOWTARR option suppresses the retrieval of the arrays, but behaves as the .NOWAIT option in all other aspects. The .NOWTARR option also protects arrays not already sent to the slave from being overwritten. The process repeats until the slave halts without disturbing the debugging status parameters, ensuring that it is not at a breakpoint. MMEXEC then behaves as if debugging is disabled.

4.5.5 MMWAIT Routine

```
CALL MMWAIT( TIMER, IEV, LABEL )
```

The MMWAIT routine is responsible for the synchronization of the host with activity in the slave. The single input argument is the maximum number of seconds that the slave may take before it must halt. If the slave exceeds this limit, the alternate return is taken with an error code in the IEV indicating the time-out condition. The programmer can avoid using explicit calls to MMWAIT by specifying the .WAIT option in the prior call to MMEXEC. However, for asynchronous cooperation between the host and the slave, the .NOWAIT option should be specified in the call to MMEXEC, followed by the call to MMWAIT. If the timer is set to zero, then the maximum allowed time is selected.

4.5.6 MMSKIP Routine

```
FLAG = MMSKIP( MEMORY )
```

The MMSKIP routine is responsible for informing the programmer of the status of the slave. The single input argument is a logical expression indicating whether virtual memory is needed. The failure of allocation on the slave does not impair the host's ability to execute the intended operation, but it may be a factor if the slave is supposed to execute the operation. Therefore, this argument eliminates the need to carry the status returned by the allocation interface routines. MMSKIP returns a boolean result. If the slave is ready for execution and memory is allocated, then MMSKIP always returns .TRUE. regardless of the MEMORY argument. If the slave is not ready for execution, MMSKIP always returns .FALSE. regardless of the MEMORY argument. If the slave is ready for execution and memory is not allocated, MMSKIP returns the complement of the MEMORY argument.

The intended use of this routine is as the condition in an IF statement. If the condition is satisfied, then the slave interface routines are skipped and work is done on the host. Otherwise work is done on the slave. That is, operations are host-based if MMSKIP returns .TRUE., and slave-based if MMSKIP returns .FALSE.

4.6 Group Two Routines

The second group contains the following routines:

1. POINTER = GETWPM(NEXT, DATA-TYPE, SIZE, NAME)

2. IEV = MMALOC(NEXT)

3. CALL MMSNDA(NAME, IEV, LABEL)

4. CALL MMRCVA(NAME, IEV, LABEL)

5. CALL MMSNDS(VALUE, SCALAR, IEV, LABEL)

6. CALL MMRCVS(VALUE, SCALAR, IEV, LABEL)

7. CALL MMSEND(VALUE, SIZE, TYPE, ADDRESS, IEV, LABEL)

8. CALL MMRECV(VALUE, SIZE, TYPE, ADDRESS, IEV, LABEL)

As with the previous group, these routines will also be studied separately.

4.6.1 GETWPM Routine

```
POINTER = GETWPM( NEXT, DATA-TYPE, SIZE, NAME )
```

The GETWPM routine is responsible for computing memory requirements and array boundaries in slave memory. The first three arguments, NEXT, DATA-TYPE, and SIZE, are identical to the input arguments to the GETWP routine in the primitives level of GIPSY. NEXT is a work space pointer which maintains an updated subscript of the first element in an integer work array that is used by an array to be allocated from the work array. The DATA-TYPE argument specifies the format of each element of the allocated array. SIZE specifies the number of elements to be allocated in the new array. NAME is used by the interface routines to identify the array in routines where the value of NEXT is not available. Also, the slave uses this field to equivalence host arrays with slave arrays. The POINTER value returned by the GETWPM function is a copy of the value in NEXT prior to the allocation of the new array. All of the arguments, except NAME, are integer expressions. NAME is a packed-character string constant enclosed in double quotes. There may not be more than six characters inside the quotes. The returned POINTER value is also an integer.

GIPSY memory and slave memory are both regarded as continuous streams of integers, as defined in the logical memory protocol. A GIPSY command allocates memory from an arbitrarily-sized work area for arrays used in the execution of the command. For compatibility with GIPSY, GETWPM allocates slave memory in a similar manner, using the virtual memory protocol. The GETWP routine allocates memory for host operations. The GETWPM interface routine allocates memory for both host and slave operations. Therefore, if a programmer has a sequence of GETWP statements for host operations, few modifications are needed to add slave support to the host source code.

This scheme enforces the allocation of host memory whenever slave memory is allocated. There are two reasons for this approach. First, this guarantees that if the slave can execute the operation, then the host is also able to do so, at least in regard to memory requirements. GIPSY commands do not execute if the host cannot meet the memory demands of the command, but the host can execute if the demand exceeds slave capacity. Second, the ability of the slave to allocate memory outside the programmer's reach is contrary to the tenet that slave activity should be easily accessible to the programmer, especially where debugging is required. It is intended that differences between host and

slave operations are made as transparent as possible.

4.6.2 MMALOC Routine

```
IEV = MMALOC( NEXT )
```

The MMALOC routine is responsible for allocating array space for arrays declared by the GETWPM routine. The single input argument is the same argument as the first argument in the GETWPM routine, or zero. If the value of NEXT is positive, then memory is allocated in the slave; otherwise the virtual memory facility is initialized for new allocations. The first MMALOC(0) function is executed by the MMINIT routine. In most cases, the programmer will call MMALOC once with a positive value in any given GIPSY command. Once the call is validated, the array information is passed to the slave as the second stage of host synchronization. The slave halts when it has processed the information and is ready for continuous execution.

4.6.3 MMSNDA Routine

```
CALL MMSNDA( NAME, IEV, LABEL )
```

The MMSNDA routine is responsible for sending arrays specified by the GETWPM routine to the slave. The only input value is the name of the array as indicated in the last argument of one of the calls to GETWPM. This argument is specified as a double-quoted character string with no more than six characters between the quotes. The name is translated to an array slot number determined by the order of the GETWPM calls. The array is then transmitted from the host to the corresponding array in the slave. Arrays may be transmitted while the slave is executing a program. The IEV is loaded with a return code and the alternate return is taken if an error occurs in this operation.

4.6.4 MMRCVA Routine

```
CALL MMRCVA( NAME, IEV, LABEL )
```

The MMRCVA routine is responsible for retrieving arrays specified by the GETWPM routine from the slave. This routine is modeled after the MMSNDA routine and behaves identically, except that the direction of array transfer is reversed. That is, the specified array is transferred from slave memory to the corresponding host array. As with the MMSNDA routine the slave may be executing a program when the MMRCVA routine is called.

4.6.5 MMSNDS Routine

```
CALL MMSNDS( VALUE, SCALAR, IEV, LABEL )
```

The MMSNDS routine is responsible for sending a scalar value to a specified reserved slot in slave memory. Two input arguments are required. VALUE contains the integer or real expression to be evaluated and stored in the location in slave memory specified by the SCALAR argument. The value of SCALAR is a positive integer if an integer value is transmitted, and negative if a real value is transmitted. The value is translated to a specific variable defined by the virtual memory protocol in the slave's memory. When SCALAR contains a value, 'x', the 'Ix' integer variable is addressed in slave memory. When SCALAR contains a negative value, '-x', the 'Rx' real variable is addressed in slave memory. The programmer can name both the value assigned to SCALAR and the reserved variable in slave memory with a RATFOR definition. Currently, twenty integer variables, named I1 through I20, and twenty real variables, named R1 through R20, are reserved for programmer use. MMSNDS may only reference these variables. The programmer must note that these variables have fixed locations in slave memory. If the programmer mixes calls to this routine with calls to logical memory protocol interface routines, the integrity of

these variables cannot be guaranteed. This call is compatible with the GETWPM, MMALOC, MMSNDA, and MMRCVA routines. This routine has the freedom of the logical memory protocol, so it may be called without prior calls to the GETWPM or MMALOC routines. The IEV and LABEL arguments are used only if the operation fails.

4.6.6 MMRCVS Routine

```
CALL MMRCVS( VALUE, SCALAR, IEV, LABEL )
```

The MMRCVS routine is responsible for retrieving scalar values from locations in slave memory reserved by the virtual memory protocol. The VALUE argument receives the integer or real value returned by the routine. It is the programmer's responsibility to specify a variable with the correct data type for the result. The single input argument is SCALAR, which specifies which reserved variable should be read. The call to MMRCVS has the same format as the MMSNDS routine, except for the processing of the VALUE argument and the direction of data transfer. That is, the meaning of SCALAR is interpreted identically with MMSNDS, but VALUE is written by MMRCVS, not read as in the MMSNDS routine. The referenced value falls within the scope of the logical memory protocol, so the programmer should use discretion

when mixing protocols.

4.6.7 MMSEND Routine

```
CALL MMSEND( VALUE, SIZE, DATA-TYPE, ADDRESS, IEV, LABEL )
```

The MMSEND is responsible for defining the sending of data to the slave using the logical memory protocol. VALUE is interpreted as a source array of arbitrary length. SIZE specifies the number of elements read from VALUE. Each element is interpreted according to the GIPSY-defined data mode in DATA-TYPE. The most likely modes are .INTMODE and .REALMODE. ADDRESS indicates which integer in slave memory will receive the first transmitted bits. A single element from the host may occupy several integers in the slave and may not align properly with the boundary separating integers in the slave if the number of bits in a slave integer (16) does not divide evenly into the number of bits in the specified data mode.

A major factor in the design of the logical memory protocol is the host's implementation of the physical memory protocol, which is a fixed access method between host and slave. In the current implementation, eight kilobyte segments of slave memory are accessible through a memory

window. That is, using memory-mapped I/O four thousand two-byte integers in the host are mapped to fixed locations in the window. The window may be moved to any portion of valid addressing space in the slave, provided that the first integer in the window falls on a four-kilobyte boundary. The first four kilobytes of memory are reserved for FORTRAN support on the slave and are inaccessible via the logical or virtual memory protocol.

Some newer systems support Direct Memory Access (DMA) between host memory and slave memory. The selection of window access or DMA is made by the MMINIT routine. An external flag is available to the user to indicate to the MMINIT routine which mode of access to enable. All models of the slave support the window access to slave memory. The DMA protocol is a faster option for large amounts of data. However, DMA transfers are performed by the slave and may only be initiated while the slave is not running.

4.6.8 MMRECV Routine

```
CALL MMRECV( VALUE, SIZE, DATA-TYPE, ADDRESS, IEV, LABEL )
```

The MMRECV routine is responsible for defining the receipt of data from the slave using the logical memory protocol. The arguments have the same meaning as in the MMSSEND routine, except that elements in the VALUE array are overwritten with data from slave memory. Bits in the VALUE array that are not modified with valid data from slave memory are left unchanged.

4.7 Group Three Routines

The third group contains the following routines:

1. IEV = OSMMAP(OP, ARG1, ARG2, ...)
2. CALL MMINTR(ERROR, IEV, LABEL)
3. CALL CHKMM(ARG1, ARG2, IEV)
4. CALL CLOSMM(IEV)
5. CALL LOADMM(ARG1, IEV)

6. CALL OPENMM(ARG1, ARG2, ARG3, IEV)

7. CALL OUTMM(IEV)

8. CALL READMM(ARG1, ARG2, ARG3)

9. CALL SETMMR(ARG1, IEV)

10. CALL SETPAR(ARG1, IEV)

11. CALL SIZEMM(ARG1)

12. CALL STOPMM(IEV)

13. CALL STRTMM(ARG1, IEV)

14. CALL STSGMM(ARG1)

15. CALL STTOMM(ARG1, IEV)

16. CALL WAITMM(IEV)

17. CALL WRITMM(ARG1, ARG2, ARG3)

Except for the first two routines, OSMMAP and MMINTR, which will be studied separately, the calls in group three are provided by the vendor as host support with the mini-MAP slave processor. If GIPSY commands are generated on a system without mini-MAP slave support, then dummy routines are substituted which automatically fail, indicating that slave support is inactive.

It is expected that the programmer will primarily use the first group of abstraction routines, with the virtual memory supports from the second group. In the next chapter, discussion will focus on how these routines are intended to be used, with examples.

4.7.1 OSMMAP Routine

IEV = OSMMAP(OP, ARG1, ARG2, ...)

The OSMMAP routine is responsible for most of the machine-dependent functions required by the other interface routines. The value in OP selects the operation to perform. The arguments after OP are generally optional and can be omitted from the call. Each defined operation may require its own additional arguments. This routine is the only interface routine that is situated in the kernel level of GIPSY. The following list enumerates the defined functions by GIPSY-defined name and operation:

1. .LOADMODULE - Load a new executable module into the slave programming space. ARG1 provides the generic file name of the module as a ".EOS"-terminated unpacked character string.
2. .INTMODE - Convert a slave integer in a host variable to a host integer. ARG1 contains the slave integer and ARG2 receives the converted integer.
3. .MAPINTMODE - Convert a host integer to a slave integer in a host variable. ARG1 contains the host integer and ARG2 receives the converted integer.

4. .REALMODE - Convert a slave real in a host variable to a host real. ARG1 contains the slave real and ARG2 receives the converted real.

5. .MAPREALMODE - Convert a host real to a slave real in a host variable. ARG1 contains the host real and ARG2 receives the converted real.

6. .CHARMODE - Convert a slave integer in a host variable to a host character. ARG1 contains the slave integer and ARG2 receives the converted character.

7. .ADDARRAY - Add the contents of the first two-element, slave integer array to the second two-element array, and store the results in the second array.

8. .MULTARRAY - Multiply the contents of the first two-element, slave integer array with the second two-element array, and store the results in the second array.

9. .ARRAYADDR - Convert the first slave integer, two-element array to subscripts for a two-dimensional, slave work array in the second two-element slave integer array.

10. .COMPARE - Return -1 in the third argument if the first slave integer, two-element array is less than the contents of the second slave integer, two-element array.

11. .MAPMEMSIZE - Return the number of slave integers in slave memory in ARG1.

12. .ALLOCATE - Allocate a slave processor from the public resource pool and return the name in ARG1 as a ".EOS"-terminated unpacked character string.

13. .DEALLOCATE - Return the slave processor to the public resource pool. No optional arguments are used.

14. .DMAENABLED - Return .TRUE. in the first argument if the environmental GIPSY symbol, GIPSY\$APDMA, indicates that DMA transfers are enabled. See the chapter on VAX interfacing for more information.

Other operations may be added as the need arises.

4.7.2 MMINTR Routine

```
CALL MMINTR( ERROR, IEV, LABEL )
```

The MMINTR routine is responsible for reacting to interruptions related to slave activity. The ERROR argument is a machine-dependent parameter containing information describing the condition causing an interrupt. There are three classes of conditions that MMINTR currently recognizes:

1. Informative signals indicating state changes in the host protocols with the slave, including initialization and termination, for example. These conditions are always discarded. If the user sets a trace flag in GIPSY, suitable messages indicating the conditions are displayed.
2. Device time-out is a condition that warrants further analysis. Any device can cause the time-out condition to be signalled. It is safe to assume the slave caused the time-out condition if MMEXEC has started the slave,

a time limit is defined, and the time that has elapsed exceeds the time limit. If this is the case, a flag is set for later examination by the MMWAIT routine.

3. Normally, illegal references to slave memory cause immediate termination of slave execution. However, if the interface routines request these errors to be permitted, then the MMINTR routine will continue execution after decrementing a count of permitted errors.

This facility uses a number of machine-dependent references, but is associated with the condition-handler used in GIPSY. Currently, only the GPINTR routine, a GIPSY primitive, calls the MMINTR routine, so it should be located in the primitive level. Also, information is passed to other interface routines through a common area which is not available in kernel routines.

5 HOST SOFTWARE

Now that each interface routine has been explained, attention can be focused on the application of these routines to new and existing commands, as well as on GIPSY-supported applications that execute without the default GIPSY environment. It must be noted that the interface routines exploit routines included in the primitives and kernel levels of GIPSY, but this does not limit the use of these routines to GIPSY commands. Some internal features such as tracing will be disabled without the GIPSY environment, but the crux of each operation will be unimpeded. The emphasis of the applications will be in GIPSY commands.

5.1 Using The MMINIT Routine

Each routine will first be considered separately, and afterwards together. The MMINIT routine is the first to consider since no other operation should precede it. In the general case, the GPMAIN routine calls MMINIT just prior to the call to the command driver:

```
IF ( SFLAG( .X ) )  
    IF ( .OK -= OSTIME( TM, 2 ) ) GOTO 9000  
IGNORE = MMINIT( 0 )  
CALL DRIVER( WORK, %9000 )
```

The call to OSTIME is for initialization of command timing, enabled by the GIPSY system-X flag. The slave is initialized, ignoring any errors. This scheme supports the tenet that the slave is an optional and limited resource. Further execution of a GIPSY command at this point should not depend upon the slave. If the call to MMINIT fails, then the other interface routines will fail, and the MMSKIP routine will return true, thus eliminating the need to save the error. The command driver is then called to execute command-specific operations. If the programmer is preparing to execute an operation specific to the slave or is not building a GIPSY command, then the call to MMINIT must succeed:

```
IF ( .OK == MMINIT( 0 ) ) GOTO 9000
```

The 9000 label is reserved for error condition-handling and follows the GIPSY convention that any labels at or above 9000 are reserved for such purpose.

5.2 Using The MMCLoS Routine

The call to MMCLoS is handled in a manner much like MMINIT. The GPMAIN routine also calls MMCLoS, but in this case, after the call to the command driver:

```
CALL DRIVER( WORK, %9000 )
```

```
IGNORE = MMCLOS( 0 )  
  
GOTO 9900  
  
9000 CONTINUE  
  
IGNORE = MMCLOS( 0 )  
  
CALL GPTRC  
  
9900 CONTINUE
```

The issue is complicated by the alternate return, requiring one call to MMCLOS for the normal return and another call for the error return. There is no need to cause a failure after the command has executed. The call to GPTRC, a GIPSY primitive, is responsible for the display of error messages and the dump of the subroutine trace stack. If the call to MMCLOS must succeed, as in the case of the MMINIT routine, then the call should be coded as follows:

```
IF ( .OK == MMCLOS( 0 ) ) GOTO 9000
```

It is important that the programmer maintains a proper pairing of calls to MMINIT and MMCLOS. The exception occurs in GPMAIN, which is why the IEV returned from MMCLOS is ignored in GPMAIN.

5.3 Using The MMEEXEC And MMWAIT Routines

The programmer may have one or more executable modules for the slave. The first module is assumed to have a name matching the command name, and the first call to the MMLOAD routine is performed by the MMINIT routine. If the load fails, then the MMINIT routine will fail, but this does not prevent another call to MMLOAD. The programmer may choose any name for any additional modules. It is recommended that these names do not conflict with names of other GIPSY commands. Also, the names should not be longer than six characters or digits since module identification uses a maximum of six characters.

The next module out of the first group is the MMEEXEC routine. Since most of the applications in GIPSY use synchronized I/O, the most likely call to MMEEXEC will use synchronized execution:

```
CALL MMEEXEC( .WAIT, IEV, %9000 )
```

This is the simplest form of execution used. The call behaves as if all of the work is done by the MMEEXEC routine. The programmer will probably use a single-buffer scheme. As a direct consequence of the .WAIT option, there is no need for a call to MMWAIT.

The situation grows in complexity as soon as the WAIT option is changed to .NOWAIT. The programmer must now include the call to MMWAIT even if nothing else is changed:

```
CALL MMEEXEC( .NOWAIT, IEV, %9000 )  
CALL MMWAIT( 10, IEV, %9000 )
```

The programmer gains the freedom of selecting the time limit for slave execution. This is useful if the programmer suspects that the slave is running into an infinite loop. The MMWAIT routine may be called prior to the call to MMEEXEC to set the time limit, but no wait is done. Calls to MMWAIT after the call to MMWAIT that synchronized the host and slave will have no effect, except to set the time limit.

5.4 Using The MMSKIP Routine

The MMSKIP routine separates host operations from slave operations and calls to interface routines. Using this routine in an IF statement, the consequence in the .TRUE. case is to perform the operation on the host and to perform the operation on the slave otherwise:

```
IF ( MMSKIP( .FALSE. ) )  
    CALL ONHOST  
ELSE  
    CALL MMEEXEC( .WAIT, IEV, %9000 )
```

In this example virtual memory is not needed, and the ONHOST routine performs the desired operation on the host. An interesting variation on this scheme takes advantage of the alternate return in the MMEEXEC routine:

```
                IF ( MMSKIP( .FALSE. ) )
1000            CALL ONHOST
                ELSE
                    CALL MMEEXEC( .WAIT, IEV, %1000 )
```

Now, if the slave or the MMEEXEC routine fails, the work is attempted on the host. A problem with this code is the breakdown in the structuring of the IF statement because of the unorthodox use of the alternate return. However, this code can be rewritten to perform the task in a structured manner:

```
                LOGICAL HOST
                ...
                HOST = MMSKIP( .FALSE. )
                IF ( ~ HOST )
                    $(
                        CALL MMEEXEC( .WAIT, IEV, %1000 )
                        HOST = .TRUE.
1000            CONTINUE
                    HOST = ~ HOST
```

```
    $)  
    IF ( HOST )  
        CALL ONHOST
```

This example produces the same effect as the previous example and is structured, as long as the MMEEXEC call is considered an operation followed by a test. The CONTINUE statement closes the body of the test.

The MMSKIP routine is intended to spare the programmer the hassle of carrying around IEV's returned by earlier calls to interface routines. This problem and the MMSKIP routine can be ignored when the application is intended to run solely on the slave, with the host processing any I/O. In this case any error returned by the interface routines in the application level software is fatal. This simplifies the structure of the code.

5.5 Allocation Of Arrays

In the analysis of the second group of interface routines, the most significant application is the allocation of arrays using the virtual memory protocol. The implementation was designed to fit easily into existing GIPSY commands, and because of the similarities, the programmer has little more to learn to use the virtual

memory protocol effectively. Consider the following example:

```
NXT = 1  
PTR1 = GETWP ( NXT, .INTMODE, NPPL )  
PTR2 = GETWP ( NXT, .REALMODE, NPPL )  
PTR3 = GETWP ( NXT, .REALMODE, NPPL )  
IF ( .OK -= OSALOC( NXT ) ) GOTO 9000
```

In this example the programmer is allocating three arrays, indexed in the total work space by the "PTR" variables. All variables and functions are declared integer. The first allocated array is for integers and the other two will contain real values. All of the arrays will hold "NPPL" values. Now, suppose the programmer does not wish to have the slave work on the integer arrays but does wish the slave to work on the real arrays. First, names must be assigned to each array for later reference by the MMSNDA and MMRCVA routines and by the slave. For this example, the first real array will be called "INPUT", and the other real array will be called "OUTPUT". Thus, the above example with slave support becomes:

```
NXT = 1  
PTR1 = GETWP ( NXT, .INTMODE, NPPL )  
PTR2 = GETWPM( NXT, .REALMODE, NPPL, "INPUT" )
```

```
PTR3 = GETWPM( NXT, .REALMODE, NPPL, "OUTPUT" )  
IF ( .OK == OSALOC( NXT ) ) GOTO 9000
```

Now the two real arrays are specified by the GETWPM routine which performs the function of the GETWP routine and declares access to these arrays by the slave and interface routines. The final step is to add the allocation call to MMALOC. Note that the syntax of the call matches the GIPSY allocation routine, OSALOC:

```
IF ( .OK == OSALOC( NXT ) ) GOTO 9000  
IF ( .OK == MMALOC( NXT ) ) GOTO 9100
```

The present form of the example requires that GIPSY memory is allocated and slave memory is allocated. This is only required in cases where the slave is a required part of the operation. The more likely use of MMALOC is as an optional facility where failure does not impede normal GIPSY operations:

```
IF ( .OK == OSALOC( NXT ) ) GOTO 9000  
IGNORE = MMALOC( NXT )
```

In those GIPSY commands that have several calls to OSALOC, place the call to MMALOC after the last call to OSALOC. This difference is due to OSALOC's capability in allocating

incrementally. MMALOC may only be called once when the allocation in "NXT" is positive. When the argument is zero, the OSALOC and MMALOC may both be called repeatedly.

5.6 Transmission Of Arrays

After allocation the arrays in the above example can be used with the MMSNDA and MMRCVA routines. The following example sends input to the slave and retrieves output from the other array:

```
CALL MMSNDA( "INPUT", IEV, %9000 )  
CALL MMRCVA( "OUTPUT", IEV, %9000 )
```

These routines do not distinguish between arrays used for input and those used for output. It is possible to use the same array for input and output, provided that the programmer does not try to reference the array while the slave is working on it. In this case, the results are unpredictable.

5.7 Transmission Of Scalars

For the reserved scalars, it is recommended that the programmer uses RATFOR definitions for each referenced slot. Names should be selected for each used scalar, and the same name should be used in the slave programming. The following example illustrates the calls to MMSNDS and MMRCVS:

```

DEFINE(SLNPPL,3)

DEFINE(SLRESULT,4)

CALL MMSNDS( NPPL, SLNPPL, IEV, %9000 )

CALL MMRCVS( RESULT, SLRESULT, IEV, %9000 )

```

Two points should be noted in this example. First, the name of the scalar must not conflict with any host variable. If this occurs, the programmer loses access to that variable because RATFOR will make the substitution defined in the DEFINE statement in every occurrence of the variable. The substitution will always yield an integer constant in this case. Second, the name may be longer than six letters since RATFOR will make substitutions in both the host and slave code, yielding FORTRAN-compatible results in both cases.

5.8 Transmission Of Arbitrary Data

Switching from virtual to logical memory protocols, the next example demonstrates the calls to MMSEND and MMRECV:

```

REAL R( 10 )

...

CALL MMSEND( R, 10, .REALMODE, 1, IEV, %9000 )

CALL MMRECV( R( 5 ), 1, .REALMODE, 9, IEV, %9000 )

```

The slave will receive ten real values from the R array. The host then gets a copy of the fifth element of R written

back into the fifth element of R. Therefore, no apparent change should be seen in the host. Since the mini-MAP requires two integers to store a host real, the fifth real occupies the ninth and tenth integers, with the ninth being the lowest addressable integer.

5.9 Using The OSMMAP Routine

From the third group of subroutines, only the OSMMAP routine will be illustrated. The MMINTR routine is not supposed to be called, except by a condition-handler, and the arguments and return IEV are machine-dependent. The other routines in group three are documented in the technical manuals supplied by CSPI.

The .ALLOCATE and .DEALLOCATE functions are used to reserve and free, respectively, a slave in a public pool of available devices. The operation of reserving a device determines its availability and removes it from public access. If the programmer wishes to reserve a slave ahead of time, this call should be used. Unfortunately, the device must be deallocated prior to the call to MMINIT for proper execution since the MMINIT routine also tries to allocate a slave. Thus, the two calls should be used together, or not at all:


```
IF ( .OK ~= OSMMAP( .ALLOCATE, NAME ) ) GOTO 9000
...
IF ( .OK ~= OSMMAP( .DEALLOCATE ) ) GOTO 9000
```

If the need arises, some cooperation between device aliases may be implemented to support the pre-allocation of slave devices.

If the programmer is concerned about the size of slave memory, especially in cooperation with the MMSSEND and MMRECV routines, then a call will need to be made to OSMMAP with the .MAPMEMSIZE option:

```
IF ( .OK ~= OSMMAP( .MAPMEMSIZE, SIZE ) )
    GOTO 9000
```

SIZE is an integer whose value is the number of slave-sized integers in slave memory. The logical memory protocol uses this value in determining the highest address available to the programmer. This function is executed by calling MMLOAD with a zero argument.

The other listed functions of OSMMAP are related to data type conversions. The definition of an integer or real, for example, may differ from machine to machine. There is one important difference between the VAX-11 and the mini-MAP. The integer on the VAX is 32 bits long, while it is only 16 bits long in the mini-MAP. This causes some problems in the processing of integers. Fortunately, reals are implemented identically on both systems. The programmer should let the slave do as much of the conversion as possible. Conversions in the host are required when the slave is unavailable for the task, as when the user is referencing slave scalars while the slave is performing some other task, or when the slave is in debugging mode. The following statements are examples of the conversions supported:

```
INTEGER I
MAPINTEGER M
REAL A
MAPREAL Z
CHARACTER C
...
IF ( .OK == OSMMAP( .INTMODE, M, I ) )
    GOTO 9800
IF ( .OK == OSMMAP( .MAPINTMODE, I, M ) )
    GOTO 9800
```

```

IF ( .OK -= OSMMAP( .REALMODE, Z, A ) )
    GOTO 9800
IF ( .OK -= OSMMAP( .MAPREALMODE, A, Z ) )
    GOTO 9800
IF ( .OK -= OSMMAP( .CHARMODE, M, C ) )
    GOTO 9800

```

Slave integers are declared MAPINTEGER in the host. Slave reals are declared MAPREAL in the host. The above example converts a MAPINTEGER to an INTEGER and back, a MAPREAL to a REAL and back, and an ASCII value in a MAPINTEGER to a CHARACTER, respectively.

5.10 Adaptation Of A GIPSY Command

The interface routines are designed to work together, so emphasis is now shifted to more complicated interactions. First, consider the adaptation of a GIPSY command that is a point operator which uses a single buffer and synchronous I/O. The allocation appears as follows:

```

NXT = 1
INPUT = GETWP( NXT, .INTMODE, NPPL )
OUTPUT = GETWP( NXT, .INTMODE, NPPL )
IF ( .OK -= OSALOC( NXT ) ) GOTO 9000

```

This example allocates enough points in an input buffer

array and an output buffer array for the storage of one line of a line-formatted input image. The modified version looks as follows:

```
NXT = 1
INPUT = GETWPM( NXT, .INTMODE, NPPL, "INPUT" )
SLAVE = GETWPM( NXT, .MAPINTMODE, NPPL, "SLAVE" )
OUTPUT = GETWPM( NXT, .INTMODE, NPPL, "OUTPUT" )
IF ( .OK -= OSALOC( NXT ) ) GOTO 9000
IGNORE = MMALOC( NXT )
```

Both of the original arrays are made accessible to the slave. In addition, some work space has been allocated for mode conversions in the slave. The host will not use the "SLAVE" array. The original operator works on the image one line at a time:

```
DO LINE = 1, NLINS
$(
CALL RREAD( FDI, INPUT, BAND, LINE,
IDENT, .WAIT, IEV, %9000 )
CALL ONHOST( INPUT, OUTPUT, NPPL )
CALL RWRITE( FDO, OUTPUT, BAND, LINE,
ODENT, .WAIT, IEV, %9000 )
$)
```

The image contains "NLINS" lines and "NPPL" points per line. The GIPSY primitives, RREAD and RWRITE, perform random-access disk reads and writes, respectively. The application level routine, ONHOST, reads values from the "INPUT" array, performs some transformation on each value, and writes values into the "OUTPUT" array. The arguments in the calls to the primitives are not important to this discussion, except the synchronization argument, ".WAIT". This indicates that control will not be returned to the calling routine until I/O has completed, behaving identically as MMEEXEC. Adding the interface routines produces the following code:

```

DO LINE = 1, NLINS
  $(
    CALL RREAD( FDI, INPUT, BAND, LINE,
                IDENT, .WAIT, IEV, %9000 )
    IF ( MMSKIP( .TRUE. ) )
      CALL ONHOST( INPUT, OUTPUT, NPPL )
    ELSE
      $(
        CALL MMSNDA( "INPUT", IEV, %9000 )
        CALL MMEEXEC( .WAIT, IEV, %9000 )
        CALL MMRCVA( "OUTPUT", IEV, %9000 )
      $)
    CALL RWRITE( FDO, OUTPUT, BAND, LINE,

```

ODENT, .WAIT, IEV, %9000)

\$)

The call to MMSKIP uses the ".TRUE." argument because virtual memory must be allocated. The MMSNDA routine sends the input buffer to the slave. The slave executes the operation while the host waits. Finally, the MMRCVA routine picks up the output buffer so RWRITE can write the next line of the output image.

The slave-supported code has several features that are worth noting. First, considerable time is spent waiting for I/O to complete. Asynchronous I/O would allow the overlapping of computation and disk services in the same time span. Second, the host and the slave are synchronized. Obviously, nothing can be done about the host, but the slave could do its task while the host spends time on other aspects of the operator that the slave cannot handle, such as moving data between slave and disk. There is no direct path between the slave and the disk. Finally, the single-buffer scheme may be relatively simple, but it does not support the notion of parallelism.

5.11 Using A Double Buffer

The timing situation can be greatly improved with a double-buffer scheme. This scheme requires two input buffers and two output buffers. The slave can use the same intermediate array for both buffer pairs. The allocation of array space becomes the following:

```
NXT = 1
INPUT1 = GETWPM( NXT, .INTMODE, NPPL, "INPUT1" )
INPUT2 = GETWPM( NXT, .INTMODE, NPPL, "INPUT2" )
SLAVE = GETWPM( NXT, .MAPINTMODE, NPPL, "SLAVE" )
OUTPT1 = GETWPM( NXT, .INTMODE, NPPL, "OUTPT1" )
OUTPT2 = GETWPM( NXT, .INTMODE, NPPL, "OUTPT2" )
IF ( .OK ~= OSALOC( NXT ) ) GOTO 9000
IGNORE = MMALOC( NXT )
```

"INPUT1" and "INPUT2" are the input buffers, and "OUTPT1" and "OUTPT2" are the output buffers. Everything else stays the same. Unfortunately, the body of the operator becomes significantly more complicated:

```
CALL RREAD( FDI, INPUT1, BAND, 1,
           IDENT, .WAIT, IEV, %9000 )
IF ( ~ MMSKIP( .TRUE. ) )
  $(
```

```

CALL MMSNDA( "INPUT1", IEV, %9000 )
CALL MMEEXEC( .NOWAIT, IEV, %9000 )
$)

CALL RREAD( FDI, INPUT2, BAND, 2,
            IDENT, .NOWAIT, IEV, %9000 )

IF ( MMSKIP( .TRUE. ) )
    CALL ONHOST( INPUT1, OUTPT1, NPPL )
ELSE
    $(
    CALL MMWAIT( 30, IEV, %9000 )
    CALL MMRCVA( "OUTPT1", IEV, %9000 )
    $)

NLIN2 = NLINS - 2
DO LINE = 1, NLIN2, 2
    $(
    IF ( .OK == OSWAIT( FDI, .WAIT ) ) GOTO 9100
    IF ( ~ MMSKIP( .TRUE. ) )
        $(
        CALL MMSNDA( "INPUT2", IEV, %9000 )
        CALL MMEEXEC( .NOWAIT, IEV, %9000 )
        $)

    IF ( .OK == OSWAIT( FDO, .WAIT ) ) GOTO 9100
    CALL RWRITE( FDO, OUTPT1, BAND, LINE,
                ODENT, .NOWAIT, IEV, %9000 )
    CALL RREAD( FDI, INPUT1, BAND, LINE + 2,
                IDENT, .NOWAIT, IEV, %9000 )

```



```

IF ( MMSKIP( .TRUE. ) )
    CALL ONHOST( INPUT2, OUTPT2, NPPL )
ELSE
    $(
        CALL MMWAIT( 30, IEV, %9000 )
        CALL MMRCVA( "OUTPT2", IEV, %9000 )
    $)
IF ( .OK == OSWAIT( FDO, .WAIT ) ) GOTO 9100
CALL RWRITE( FDO, OUTPT2, BAND, LINE + 1,
            ODENT, .NOWAIT, IEV, %9000 )
IF ( .OK == OSWAIT( FDI, .WAIT ) ) GOTO 9100
IF ( - MMSKIP( .TRUE. ) )
    $(
        CALL MMSNDA( "INPUT1", IEV, %9000 )
        CALL MMEEXEC( .NOWAIT, IEV, %9000 )
    $)
IF ( LINE + 3 <= NLINS )
    CALL RREAD( FDI, INPUT2, BAND, LINE + 3,
              IDENT, .NOWAIT, IEV, %9000 )
IF ( MMSKIP( .TRUE. ) )
    CALL ONHOST( INPUT1, OUTPT1, NPPL )
ELSE
    $(
        CALL MMWAIT( 30, IEV, %9000 )
        CALL MMRCVA( "OUTPT1", IEV, %9000 )
    $)

```

```

        SAVE = LINE + 2

        $)

    IF ( .OK /= OSWAIT( FDO, .WAIT ) ) GOTO 9100

    CALL RWRITE( FDO, OUTPT1, BAND, SAVE,
                OIDENT, .WAIT, IEV, %9000 )

    IF ( ( NLINS / 2 ) * 2 == NLINS )

        $(

        IF ( .OK /= OSWAIT( FDI, .WAIT ) ) GOTO 9100

        IF ( MMSKIP( .TRUE. ) )

            CALL ONHOST( INPUT2, OUTPT2, NPPL )

        ELSE

            $(

            CALL MMSNDA( "INPUT2", IEV, %9000 )

            CALL MMEEXEC( .WAIT, IEV, %9000 )

            CALL MMRCVA( "OUTPT2", IEV, %9000 )

            $)

            CALL RWRITE( FDO, OUTPT2, BAND, SAVE + 1,
                        OIDENT, .WAIT, IEV, %9000 )

            $)

```

The above example processes two lines at a time but includes provision for an unpaired line. This approach requires at least three lines of input. The first and last lines are processed synchronously to properly initialize and terminate the operation. The second argument of the RREAD and RWRITE routines indicates the buffer that is to be transmitted.

The fourth argument is the image line being processed. Aside from the wait option, which is now ".NOWAIT" for asynchronous operations, the undefined arguments of the RREAD and RWRITE routines may be ignored. The only additional routine added to the double-buffering scheme is the OSWAIT routine, which is responsible for I/O synchronization in GIPSY.

In a later chapter, significantly more complicated examples of GIPSY commands adapted for use with the slave will be presented. This chapter limited the scope of the discussion to illustrating the use of the interface routines in a fundamentally uncomplicated setting or in the actual setting in the Control level in GIPSY. To date, no GIPSY command takes advantage of the double-buffering scheme presented earlier or of any scheme using asynchronous disk access.

6 MINI-MAP SUPPORT

Until now attention has been focused on the host. The mini-MAP has its own RATFOR support, FORTRAN compiler, and linker for building executable modules. Moreover, there are a number of facilities provided with the host interface routines for easing the programmer's task when dealing with the mini-MAP.

6.1 FORTRAN Compiler

The most difficult aspect of dealing with the mini-MAP is the construction of the programs. The optimal approach is to simply take the host code and compile it for the mini-MAP. Unfortunately, there are fundamental differences between host and mini-MAP FORTRAN's. It is these differences which will complicate the adaptation process.

The mini-MAP currently supports three data types: INTEGER*2, LOGICAL*2, and REAL*4. Integers are represented internally in two's-complement format. Logical constants, .TRUE. and .FALSE., are represented by -1 and 0, respectively. Real values match the format used by the VAX-11. Currently, the VAX-11 and PDP-11 are the only systems supported by the mini-MAP. One of the first obstacles caused by this limitation is the lack of any character data types or constants. Although RATFOR

translates GIPSY's data types into other host-supported data types, it is not sensitive to character constants, such as "'a'". For identification purposes, RATFOR tokens are defined for all letters and digits in integer form, and the OSMMAP routine converts the integers to GIPSY characters.

Another major difference is the lack of I/O. The FORTRAN statements, READ and WRITE, are not supported. The debugging facility in the MMEEXEC routine is implemented to get around this restriction when trying to get diagnostic information about program execution and variable contents. For other I/O applications, the programmer may not use the mini-MAP.

Programs may be written only with uppercase letters. This is not a serious problem because all of GIPSY, except prompt text, is written in uppercase for unfortunate reasons related to IBM compatibility. Moreover, there is no text to be referenced for I/O purposes so that any conflicts in text construction can be avoided.

Several statements that appear in GIPSY code are not supported in the mini-MAP FORTRAN, notably the PROGRAM, BLOCK DATA, and EQUIVALENCE statements. Every slave program

is considered a subroutine, so the PROGRAM statement is of no consequence. Data initialized in the host can be sent to the mini-MAP, or the programmer may use assignment statements in the slave code as an initialization stage. The lack of an EQUIVALENCE statement is not serious for two reasons. First, the EQUIVALENCed references in GIPSY are almost exclusively used for the identification arrays used by I/O primitives or for the verification of initial conditions, such as command flag settings and image characteristics. This level of operation should not be delegated to the mini-MAP. Second, EQUIVALENCed references may not have been passed through a subroutine call. The programmer can pass an element of an array, for example, and have the subroutine give it a unique name. Alternatively, the desired value can simply be assigned to another variable.

There is one important statement that is supported by the mini-MAP FORTRAN that is different from the host FORTRAN: the PAUSE statement. This instruction stops the mini-MAP, but allows execution to continue by a call to the MMEXEC routine. The wait condition in the MMWAIT routine is satisfied by this statement's execution. Breakpoints also use this instruction embedded in a RATFOR macro.

6.2 Parameter Passing

A very important difference between VAX-11 and mini-MAP FORTRAN's is the parameter-passing mechanism. In both cases, the parameter list is composed of an parameter count, N, the number of passed parameters, followed by N four-byte words, each containing the location of one parameter. Unfortunately, the similarity ends here. For all VAX-11 FORTRAN-declared variables, the location is the virtual address of the lowest-order byte occupied by the passed parameter, regardless of its data type, or whether it is a scalar or an array. If the call is a function, the return value is stored in hardware registers, R0 and R1, except in the case of character values. Since characters are not supported in the mini-MAP FORTRAN, they will be henceforth ignored for this discussion.

Mini-MAP FORTRAN employs a significantly different addressing scheme. First, memory is available as a single contiguous allocation. For an integer scalar, memory is viewed as a sequence of two-byte integers. For a real scalar, memory is viewed as a sequence of four-byte real values. Zero is the index of the first scalar at the start of mini-MAP Data Memory. The location of a scalar is an index expressed in the number of units of the same data type as the variable. For example, an address of 100 for an

integer scalar is at byte address 200, while an address of 100 for a real scalar refers to byte address 400. In addition, addresses for integer scalars have bit 23 set to 1. For mini-MAP arrays the location is the index of an array descriptor, with memory viewed as four-byte values. Consequently, addresses of integer scalars are not easily interchanged with addresses of real scalars or of arrays. Also, references to elements in arrays are handled as scalars. In the case of functions, the first parameter in the parameter list defines the return value of the function.

The IDENT routine, illustrated later, is provided as a builtin operation that allows the programmer to redefine the interpretation of a scalar or array. There are other supports that are supplied with the interface routines that assist the programmer in this regard. These supports will be detailed later.

6.3 Module Length

The last noteworthy difference in mini-MAP code is its length. There is a maximum of 4096 assembly-language instructions that any slave module may contain. This translates to between 400 and 500 FORTRAN instructions. With respect to GIPSY host code, there is no explicit size limit on the size of executable images. The programmer

should not be concerned about this limit until the condition surfaces. If the condition should occur, the programmer can break the operation into smaller modules, and add additional calls to the MMLOAD routine. A Scientific Subroutine Library is provided as a collection of preprogrammed and optimized operations that may assist the programmer in cutting down the number of instructions used.

6.4 Include Files

Many potential difficulties in programming have been hidden by RATFOR macros in predefined include files. The following list identifies those include files which are used in mini-MAP RATFOR routines:

1. MAPDRV - support for the entry routine
2. X || command name - array information for the virtual memory protocol
3. MAPCHAR - character support

4. MAPID - module identification

5. MAPOPTS - module options

6. MAPSUB - support for subroutines

The include files will be explained further throughout this chapter.

A standard protocol has been defined for the construction of the mini-MAP entry routine, called the MAP-driver. Subroutines are given a different kind of support with more freedom since they do not usually require the synchronization and verification controls used in the MAP-driver. All PAUSE statements, aside from those used in debugging support, are restricted to the MAP-driver so the overall flow of control is apparent by looking at one routine.

6.4.1 Virtual Memory Protocol

The MAP-driver must conform to one of two formats. The first format supports the virtual memory protocol and the second format supports the logical memory protocol. The following example fits the first format:

```

INCLUDE MAPDRV
INCLUDE XABSIMG
REPEAT
    $(
        I1 = INT( ARRAYLEN( IOLINE ) )
        CALL ABSWRK( IOLINE, I1, I2, I3,
                    I4, I5, DEBUG )
        PAUSE
    $)
END

```

All MAP-drivers using the virtual memory protocol must include the MAPDRV definitions as the first line. This include file contains definitions for identification, allocation, debugging in the MAP-driver, and initial code generation. Routines using the logical memory protocol do not need the MAPDRV include file, but some of its components may be useful.

6.4.2 X-driver Include File

The XABSIMG include file is generated by the INCLMM command for equivalencing the names of arrays in the GETWPM calls in the command driver with the same array names in the MAP-driver. The programmer does not need to declare explicitly any of the routines allocated by the host when using this facility. The INCLMM command scans the command

driver for the GETWPM statements to be able to build the include file. The body of the MAP-driver is an unconditional repetition of statements enclosed by the REPEAT construct. The driver is never supposed to reach a termination statement, such as END, STOP, or RETURN, since this would invalidate any further calls to the MMEXEC routine.

6.4.3 Main Loop

The code inside the REPEAT loop indicates the actions performed between the call to MMEXEC in the host and the terminating PAUSE. The PAUSE statement should be the last statement in the REPEAT loop. No computation should be done in the MAP-driver except that which sets up parameters to subroutines. In this example, the ARRAYLEN macro returns the length of the IOLINE array in REAL format. The purpose of using REAL format is to give the programmer the opportunity to operate on more than 32767 elements in an array. Since the mini-MAP uses INTEGER*2 subscripts, there is an explicit maximum of 32767 elements in a one-dimensional array. For those applications which do not need to exceed this limit, converting the array length to an integer with the INT builtin function is adequate. To handle more than 32767 elements, the array length can be divided into increments smaller than 32768 and processed in pieces.

The DEBUG parameter in the ABSWRK call is used to pass tracing information from the subroutine back to the MAP-driver, possibly for transfer to the host. It should be standard procedure to include the DEBUG parameter as the last parameter in all routines, giving the programmer the freedom to "turn on" the tracing feature without having to modify all of the mini-MAP subroutines. The tracing facility will be explained more fully later.

6.4.4 MAPDRV Include File

The MAPDRV include file is expanded into RATFOR definitions and support code. A subsampled version of the expansion can be used to highlight major features in the include file:

```
INCLUDE MAPCHAR
INCLUDE MAPID
...
DEFINE(GETARRAYS,... PAUSE;)
...
<debugging definitions>
...
DEFINE(ARRAYLEN(X),ARRLEN(X L))
DEFINE(IDESC1(N),
    CALL XGADDR( ARRLOC( N ), ARRWK1, ARRWK2 );
    CALL IDENT( IAR N, WORK( ARRWK1, ARRWK2 ),
```

```

1, ARRLEN( N ) );
IFDEF .ENBTRACE PAUSE ENDIF )
...
SUBROUTINE DRIVER
INTEGER I1, I2, ..., I20
INTEGER IAR1( 1, 1 )
INTEGER IAR2( 1, 1 )
...
INTEGER IAR20( 1, 1 )
REAL R1, R2, ..., R20
REAL RAR1( 1, 1 )
REAL RAR2( 1, 1 )
...
REAL RAR20( 1, 1 )
REAL CAR1( 1, 1 )
REAL CAR2( 1, 1 )
...
REAL CAR20( 1, 1 )
PARAMETER( IAR1L = 1 )
PARAMETER( IAR2L = 2 )
...
PARAMETER( IAR20L = 20 )
PARAMETER( RAR1L = 1 )
PARAMETER( RAR2L = 2 )
...
PARAMETER( RAR20L = 20 )

```

```

PARAMETER( CAR1L = 1 )

PARAMETER( CAR2L = 2 )

...

PARAMETER( CAR20L = 20 )

INTEGER ARRWK1, ARRWK2, WORK, INT, JUNK1, JUNK2

REAL ARRLOC, ARRLen

COMMON / WINDOW / ARRLOC( 20 ), ARRLen( 20 ),

        ARRWK1, ARRWK2, I1, I2, ..., I20,

        R1, R2, ..., R20, WORK( 32767, 513 )

```

Many features are distinguished at this level of expansion. The first include file, MAPCHAR, provides ASCII definitions for A through Z and 0 through 9, using RATFOR definitions, .A through .Z and .D0 through .D9, respectively. These definitions are required for the module identification stage. They are isolated in a separate include file for the logical memory protocol and for some minimal character support for the programmer.

6.4.5 MAPID Include File

The next include file, MAPID, contains the RATFOR support for module identification and context definition. Identification is defined by a set of six name definitions:

```

DEFINE(NAME6(A,B,C,D,E,F),

        WORK(11,1)=A

```

```
WORK(12,1)=B
WORK(13,1)=C
WORK(14,1)=D
WORK(15,1)=E
WORK(16,1)=F
PAUSE)
```

The six name definitions, NAME1 through NAME6, take one to six arguments, respectively. The above example for NAME6 is representative of all six definitions. Predefined locations in the WORK array are assigned values from the MAPCHAR include file. For example, the macro for the ABSIMG command uses the following construction:

```
NAME6(.A,.B,.S,.I,.M,.G)
```

The generated code is executed only by the MMINIT and MMLOAD routines to

1. Verify that the module is the correct module chosen for execution,

2. Ensure that the module is at a known point for subsequent host operations, and
3. Validate the positioning of the memory window.

This macro is normally the first executable code in the module after the MAPDRV include statement, since after the PAUSE is executed the host may swap modules. Moreover, every module compatible with GIPSY must use this construct to get by the MMLOAD routine.

The only variation in the placement of the NAME macro is in the use of module options. Options are specified in the following way:

```
OPTION( name, value )
```

where "name" is one of the slot names recognized by the MMLOAD routine, and "value" is the new value for the option. Recall that a value of -1 selects the default value defined in the MMLOAD routine for that slot. All option statements must be placed after the MAPDRV or MAPID INCLUDE statements, and before the NAME macro. After the NAME macro, the OPTION statement is not operational. For the .OPTDEBUG option, any non-negative value enables debugging.

6.4.6 Array Allocation

The next set of definitions in the MAPDRV include file deal with array allocation. Actually, most of the work is done by the host, but the slave must complete the process by adjusting for offsets unknown to the interface routines. The decoupling of this knowledge makes it easier to identify slave activity and to make changes without necessarily disturbing the interface routines. The GETARRAYS macro completes the allocation process:

GETARRAYS

It is supposed to be executed only by the MMALOC routine. Each call to MMALOC with a positive argument must be paired with a call to GETARRAYS in the mini-MAP. Typically, there is only one call of this type in a command, so the MAP-driver should place it just after the NAME macro. The MMALOC routine waits for the GETARRAYS to PAUSE before it returns control to the calling routine.

There are several definitions for the support of debugging on the mini-MAP. Breakpoints are specified by the TRACE and TRACE2 statements:

TRACE(sequence number)

TRACE2(sequence number,message index)

The first, or only, argument is any integer different from the previously executed TRACE or TRACE2 statement. This sequence number will be reported to the programmer by the MMEXEC routine. The second argument in the TRACE2 statement is an arbitrary integer that the programmer associates with a particular message in an external message file. If the programmer defines a lookup table with the MKRAN command, the message index is used as a key for text retrieval. After the MAPDRV include statement, the programmer inserts the DEBUGON statement to enable the effects of TRACE and TRACE2 statements. Throughout the mini-MAP code the programmer may insert DEBUGOFF and DEBUGON statements to control the scope of the debugging support. Finally, the DEBUG parameter expands to two integers, ARRWK1 and ARRWK2 if DEBUGON was last executed, and JUNK1 and JUNK2 if DEBUGOFF was last executed. Writing to ARRWK1 or ARRWK2 causes debugging activity in MMEXEC if DEBUGON was executed prior to the NAME statement. The JUNK1 and JUNK2 integers are not known to the host so that spurious activity is not inadvertently transferred to the host. For any subroutines that require the debugging facility, a path must be enabled from each routine to the MAP-driver via the DEBUG parameter by executing the DEBUGON statement in each of the intermediate subroutines and the MAP-driver.

6.4.7 Generated Code

Before explaining the virtual array definitions in the next section of the MAPDRV include file, it is necessary to have knowledge of the code that will also be generated in the last section of the include file. The first line of code is the subroutine declaration. Every MAP-driver is named DRIVER without any arguments. Subsequently, there are declarations for the reserved scalars, I1 through I20, and R1 through R20, dummy arrays IAR1 through IAR20, RAR1 through RAR20, and CAR1 through CAR20, and parameters IAR1L through IAR20L, RAR1L through RAR20L, and CAR1L through CAR20L. The scalars are addressable by the MMSNDS and MMRCVS routines. Since the mini-MAP passes array arguments by descriptor instead of by reference there must be pre-defined descriptors of the correct data type for each of the allocated arrays. There may be a maximum of two dimensions for each allocated array. All of the descriptors are represented by single-element, two-dimensional arrays of type INTEGER, REAL, or REAL x 2, for complex operations. These dummy arrays are then redefined to match the dimensions of the allocated arrays. The IARn arrays are for INTEGER arrays, RARn arrays are for REAL arrays, and CARn arrays are for complex arrays, where "n" is the number of the slot, from one to twenty. The parameters are defined so that the array name can be translated to the slot number.

They are only used by the ARRAYLEN macro to make a proper translation between array name and actual slot used. The starting address and length of each allocated array is stored in the REAL arrays, ARRLOC and ARRLEN, respectively. The ARRWK1 and ARRWK2 variables are used by GETARRAYS and the debugging definitions. The WORK array is analogous to the WORK array in the host. JUNK1 and JUNK2 replace ARRWK1 and ARRWK2, respectively, when debugger support is disabled. Memory is transmitted through the WINDOW common, whether or not the module is GIPSY-supported.

6.4.8 Virtual Array Definitions

The virtual array definition section includes the ARRAYLEN macro and the array descriptor macros. ARRAYLEN translates to an element from the ARRLEN array which is passed to the mini-MAP by the MMALOC routine. For example, ARRAYLEN(IOLINE) translates first to ARRAYLEN(IAR1), which then translates to ARRLEN(IAR1L). IAR1L is a parameter with a value of one, so the final reference is ARRLEN(1). The translation of array name, IOLINE in the example, to IAR1, the slot for the first integer array, is done by a RATFOR definition in the XABSIMG include file.

There are six array descriptor macros named IDESC1, RDESC1, CDESC1, IDESC2, RDESC2, CDESC2. The first letter of each macro defines the data type: I, R, or C for INTEGER, REAL, or COMPLEX (REAL x 2), respectively. The digit at the end is the number of dimensions. The argument is a slot number that needs redefinition. For a two-dimensional array, the size is automatically 32767 in the leftmost subscript. The IDENT routine supplied with the min-MAP is used to redefine the array descriptor.

The XABSIMG include file is constructed by the INCLMM command. This command builds the NAME, GETARRAYS, and array descriptor macros for the programmer, so that the programmer does not reference the arrays by slot number. Instead, the programmer uses the quoted names from the command driver's GETWPM statements. This scheme allows the programmer to change the order of the GETWPM statements or add and remove statements without having to modify mini-MAP code. The include file is easily regenerated and the MAP-driver recompiled and linked.

If the programmer wishes to add other declarations to the MAP-driver, they must be inserted after the INCLUDE MAPDRV statement and before the INCLUDE XABSIMG statement. The choice of include file names for the output of the

INCLMM command is at the programmer's discretion, but the default is an "X" followed by the command name used in the NAME macro. The XABSIMG include file looks as follows:

```
NAME6(.A,.B,.S,.I,.M,.G)

GETARRAYS

IDESC1( 1 )

DEFINE(IOLINE,IAR1)
```

The IOLINE array is equivalenced with the first slot, in integer mode, by the DEFINE statement, and the slot is properly sized by the IDESC1 statement. More information on the INCLMM command and the include file it generates will be provided in a later chapter on VAX-11 and GIPSY commands.

6.4.9 Logical Memory Protocol

The second format for the MAP-driver requires much less conformance with any protocol and less support. A sample MAP-driver looks as follows:

```
INCLUDE MAPCHAR

INCLUDE MAPID

SUBROUTINE DRIVER

INTEGER A, B, I

COMMON / WINDOW / A( 100 ), B( 100 )

INTEGER IABS
```

```

DEFINE(WORK,A)
NAME3(.A,.B,.S)
REPEAT
    $(
        DO I = 1, 100
            B( I ) = IABS( A( I ) )
        PAUSE
    $)
END

```

The above example will execute in GIPSY with the logical memory protocol. The MAPCHAR and MAPID include files support the NAME3 statement. This example takes 100 integers in locations 1 through 100 and returns their absolute values in locations 101 through 200. Note that the arrays are declared scalar but are dimensioned in the common. IABS is a builtin function that computes the absolute value of an integer. These addresses are referenced in the host by the MMSSEND and MMRECV routines:

```

MAPINTEGER A( 100 ), B( 100 )
...
CALL MMSSEND( A, 100, .MAPINTMODE, 1,
             IEV, %9000 )
CALL MMEXEC( .WAIT, IEV, %9000 )
CALL MMRECV( B, 100, .MAPINTMODE, 101,

```



```
IEV, %9000 )
```

```
...
```

The reference to the WORK array in the NAME3 macro are 'rerouted' to the A array with the RATFOR definition. Alternately, the programmer can make space for the work array and shift the addresses by .OPTEND:

```
INTEGER WORK, A, B, I  
COMMON / WINDOW / WORK( .OPTEND ), A( 100 ),  
B( 100 )
```

The DEFINE is removed. While in host:

```
CALL MMSSEND( A, 100, .MAPINTMODE, 1 + .OPTEND,  
IEV, %9000 )  
CALL MMEXEC( .WAIT, IEV, %9000 )  
CALL MMSSEND( B, 100, .MAPINTMODE, 101 + .OPTEND,  
IEV, %9000 )
```

The .OPTEND symbol is defined in the MAPOPTS include file, which is referenced in the MAPID include file. The MAP-driver is simpler in concept in this new version at the expense of the host software.

6.4.10 Subroutine Support

For subroutines below the MAP-driver, the structure is much simpler. The subroutine, ABSWRK, is called by the MAP-driver for the ABSIMG command:

```
INCLUDE MAPSUB

SUBROUTINE ABSWRK( IOLINE, LEN, INDEX, PTLOW,
                  PTHIGH, IDXH, DEBUG )

INTEGER LEN, IOLINE( LEN ), INDEX, PTLOW
INTEGER PTHIGH, IDXH, DEBUG

DO INDEX = 1, LEN, 2
  $(
    IDXH = INDEX + 1
    PTLOW = IOLINE( INDEX )
    PTHIGH = IOLINE( IDXH )
    IF ( PTHIGH < 0 )
      $(
        IF ( PTLOW == 0 )
          $(
            IOLINE( IDXH ) = -PTHIGH
          $)
        ELSE
          $(
            IOLINE( INDEX ) = -PTLOW
            IOLINE( IDXH ) = -PTHIGH - 1
          $)
        $)
  $)

```

```
        $)
    $)
$)
RETURN
END
```

The only support for subroutines is the debugger support in the MAPSUB include file. This include file can be included in subroutines with MAP-drivers that use the virtual memory protocol or those which reserve addresses 41 and 42 for the debugging parameters when using the logical memory protocol. The debugging support has the same definitions as in the MAPDRV include file, DEBUGON, DEBUGOFF, TRACE, TRACE2, and DEBUG, but the variables currently used are DBGTR1 and DBGTR2. This subroutine operates on INTEGER*4 data using INTEGER*2 arithmetic. This design is compatible with integers on the VAX-11. No code is generated by the MAPSUB include file.

For test purposes, other GIPSY commands besides the ABSIMG command have been adapted to use the mini-MAP. In a later chapter, the details of the conversion will be more fully examined. The software conventions used in this chapter were applied in the other commands so that further explanation of the conversion fundamentals should not be

necessary.

7 VAX SUPPORT

The last mode of interaction between the programmer and the mini-MAP is the command level. This includes instructions submitted to the Digital Command Language interpreter (DCL) of the VAX/VMS operating system and the command level of GIPSY. This chapter is concerned with the following:

1. DCL symbols;
2. GIPSY system flags which influence the interface routines;
3. Compilation and command-building utilities;
4. DCL logicals; and
5. GIPSY commands which assist in building mini-MAP support for other GIPSY commands.

For clarity, all instructions to DCL are preceded by a dollar sign (\$) and all GIPSY commands are preceded by the GIPSY prompt, "G:".

7.1 DCL Symbols

Several symbols are defined by the GIPSY environment that require the selection of a particular configuration of the GIPSY package. These symbols affect how commands are built, stored, and executed. All of the symbols begin with "GIPSY\$" and maintain a value of zero or one for the duration of the login session. The mini-MAP support in GIPSY is affected by many of these symbols, but the most influential is named GIPSY\$MINIMAP. This symbol determines whether a GIPSY command can access the mini-MAP or its supports outside the GIPSY package. If zero, the programmer may not build any modules, compile any mini-MAP code, or execute any activities on the mini-MAP with GIPSY commands built while GIPSY\$MINIMAP was zero. This switch is defined when the programmer signs onto the system:

```
$ GIPSY$MINIMAP == 1
```

In this example the mini-MAP support is enabled. The programmer may change the setting of the symbol, but results are not predictable. Changing the symbol value will not have effect on existing commands until they are regenerated with the GENCMD utility, explained later in this chapter. At installation time, the symbols are examined to determine if any of the symbols are to be modified. If so, the

commands will probably need regeneration.

Another notable symbol that influences enabled mini-MAP activity is GIPSY\$APDMA. This symbol determines whether data may be transferred to and from the mini-MAP using Direct Memory Access (DMA). If zero, all transfers must use a moving memory window, which aligns host memory with mini-MAP memory. This switch is also defined when the user signs onto the system:

```
‡ GIPSY$APDMA == 1
```

The default setting is to enable DMA transfers if mini-MAP support is also enabled by using the GIPSY\$MINIMAP symbol. Unlike the GIPSY\$MINIMAP symbol, the user may change this setting anytime to influence command execution. The impact of modifying this symbol will be studied later in the analysis of actual test cases.

7.2 GIPSY System Flags

In GIPSY there are twenty six user-level and twenty six system-level command flags, each assigned a letter of the alphabet. Of the system-level flags, four have specific effects on the mini-MAP interface routines:

1. The A-flag enables tracing of the calls to the interface routines, and major state changes inside these routines.

2. The D-flag enables tracing of debugging activity for modules which have also enabled tracing with the DEBUGON instruction.

3. The F-flag forces the MMINIT routine to attempt a load of the default module into the mini-MAP. This bypasses the check of any programming that may already exist in mini-MAP memory.

4. The P-flag prohibits any execution activity using the mini-MAP. When the flag is clear, by default, the commands which have the mini-MAP support attempt to use the mini-MAP.

There is no flag which forces execution onto the mini-MAP. This condition must be explicitly programmed into the host code.

7.3 Utilities

7.3.1 RATFOR Preprocessor

Three important utilities are provided for the programmer in the GIPSY package, and one VAX/VMS utility is provided for building GIPSY commands. The first, RATFOR, is a DCL command procedure which invokes the RATFOR compiler to convert the source code to VAX-11 FORTRAN and then invokes the FORTRAN compiler to translate the FORTRAN to object code. Files with object code can be linked together to form an executable module. The command may be abbreviated to as few as three letters:

```
  $ RAT source-file
```

This utility has been enhanced to distinguish between host code and mini-MAP code, both written in RATFOR. Experienced GIPSY programmers do not need to learn new commands to compile mini-MAP code. The file type of host code defaults to ".RAT", while that of mini-MAP code is ".MMR". The utility is programmed to assume the input file is host code unless shown or instructed otherwise. In most cases it is enough to specify the file name, letting the utility check the file type. When the file type is not recognized or both file types are used, the programmer can either include the file type with the file name and/or add the "/MAP" switch

§ RAT source-file/MAP

to force compilation as mini-MAP code. Alternatively, the MMRATFOR command can be substituted to produce the same effect:

§ MMR source-file

As with RATFOR, the command name may be abbreviated to three letters.

When GIPSY\$MINIMAP is zero, compilation of mini-MAP code stops after the RATFOR translator completes its part of the task. The FORTRAN conversion requires a mini-MAP FORTRAN compiler, called MFORT, whereas RATFOR is a component of the GIPSY package. When the translation of host code fails, there is usually an intermediate FORTRAN source file left behind as a failure indicator and debugging aid. The same condition holds for mini-MAP code. When the utility executes to completion, the intermediate files are normally deleted.

For mini-MAP code, the MFORT compiler translates to assembler code. A separate assembler is invoked to build the object file. The object file for the VAX is not compatible with the object file for the mini-MAP. The file types, ".OBJ" and ".RO", respectively, identify the target system.

7.3.2 GENCMD Procedure

The second utility, GENCMD, is used to build a GIPSY command. This DCL command procedure links a list of object files and libraries into an executable module for the VAX with a name that is used by the GIPSY command interpreter:

```
‡ GENCMD  command  filenames  libraries
...
G: command  operands  (flags)
```

Whenever an object file does not exist or is older than the corresponding source file, GENCMD automatically invokes the RATFOR compiler to build a new object file prior to the linkage process. The resultant executable image has a ".EXE" file type.

As mentioned earlier in this chapter, the GIPSY\$MINIMAP symbol affects the linkage process. This is accomplished by controlling access to some of the Group Three interface routines enumerated in chapter four. Accompanying the mini-MAP is a Host Support Library, MMHSL, which is used by the other interface routines for enabled access to the mini-MAP, i.e., when the GIPSY\$MINIMAP switch is on. A set of dummy routines with the same names as the MMHSL routines resides in the GIPSY library, GIPLIB, with the other interface routines. If the GIPSY\$MINIMAP switch is off, then these dummy routines, are linked in place of the MMHSL routines to inhibit access to the mini-MAP. This approach was taken to handle the case where the MMHSL library does not exist.

7.3.3 GENMAP Procedure

An analogous approach to the GENCMD scheme has been taken for the mini-MAP support. To prevent the programmer from mixing VAX and mini-MAP code in the same GENCMD instruction, no mini-MAP code may be used in the GENCMD instruction. Instead, a parallel GENMAP instruction accepts only mini-MAP code, and uses different standard-support libraries than GENCMD. The syntax of GENMAP is identical to GENCMD, but the output is a load module that executes only on the mini-MAP. Moreover, the file type is ".LO", which differs from GENCMD. When GIPSY\$MINIMAP is zero, GENMAP

immediately aborts.

7.3.4 DEBUG Facility

To assist the programmer in the development of software, a debugging facility, DEBUG, is provided with the VAX/VMS operating system. To access DEBUG for a given subroutine with mini-MAP support, the programmer must execute the following steps:

1. Modify the MAP-driver routine to enable the debugging support in the MMEXEC routine by adding the following instruction just before the NAME macro or the X-command INCLUDE statement:

```
OPTION(.OPTDEBUG,0)
```

Also add TRACE or TRACE2 statements to the MAP-driver or MAP-subroutines as breakpoints for the MMEXEC routine;

2. Compile and build the mini-MAP executable module:

```
§ GENMAP command filenames libraries
```

The modified routines will automatically be recompiled if the GIPSY\$CHECK switch is on, which is the default;

3. Compile the VAX-11 source program using the mini-MAP with debugging support:

```
    $ RAT source-file/DEBUG
```

This will give the debugger information on all of the programs symbols and suppress the optimization phase of the FORTRAN compiler;

4. Build the command with debugging support:

```
    $ GENCMD command/DEBUG filenames libraries
```

This causes the resulting GIPSY command to enter DEBUG automatically when the command is invoked;

5. Invoke the GIPSY command:

```
G: command operands (flags)
```

```
..
```

```
DBG>
```

Instead of proceeding as usual, the command will display the announcement associated with the debugger and display the DEBUG prompt. If the system D-flag is set

in the invocation, then the scalars will be monitored by the MMEXEC routine; and

6. Synchronize the debugger with the MMEXEC routine:

```
DBG> SET BREAK MMBRAK
```

```
DBG> GO
```

When the DEBUG prompt is redisplayed, the program has either completed execution, aborted, or reached the MMBRAK breakpoint.

The MMBRAK reference is a pre-defined breakpoint inside the MMEXEC routine for the explicit use by the debugger. MMBRAK was added as a global reference whose effective position is never changed if future modifications are made to the MMEXEC routine. The programmer also would not need to recompile the MMEXEC routine with debugging support. This breakpoint is reached after the scalars and arrays have been retrieved but before the mini-MAP has commenced execution after a breakpoint. If the .OPTDEBUG option is not specified or no TRACE or TRACE2 statements are used, then the MMBRAK breakpoint will never be reached.

7.4 DCL Logicals

DCL logicals are used to define two types of information for the interface software, the identification of the mini-MAP as a VAX-11 device, and the locations of files stored on VAX-11 disks. All of these logicals are defined in the file, GIPSY_MGR:GIPLOG.COM. The physical device name of the mini-MAP is stored in the logical, GIPSY_MINIMAP, and is referenced by the OSMMAP interface routine. Another logical referenced by the OSMMAP routine is GIPSY_MAPPATH, which identifies a list of directories which may be searched for executable modules with a given name. This list defaults to the current directory and GIPSY_MAPLOAD, which will be defined shortly.

The remaining logicals identify directories inside GIPSY's directory tree that contain GIPSY command support for the mini-MAP. The following list identifies these logicals and their contents:

1. GIPSY_MAP - list of directories in use, i.e., the other logicals in this list;

2. GIPSY_MAPLINK - files with GENMAP instructions for each command with mini-MAP support;
3. GIPSY_MAPLOAD - files generated by the GENMAP utility;
4. GIPSY_MAPOBJ - object files generated by the MFORT utility which are not deleted; and
5. GIPSY_MAPSRC - source files for GENMAP instructions.

The GIPSY_MAPOBJ directory is used in place of an object library, such as MMHSL, since to date, no librarian utility has been supplied for the mini-MAP.

7.5 GIPSY Commands

7.5.1 INCLMM Command

There are two GIPSY commands which can assist the programmer in the development of mini-MAP support for other GIPSY commands. The first is called INCLMM. This command, already introduced in the previous chapter, builds an include file for the entry routine of the mini-MAP executable module. The generated file contains all of the necessary code for identifying the module to the MMLOAD routine and getting array information for the virtual memory

protocol. The command has the following GIPSY syntax:

```
G: INCLMM driver > command, include
```

where "driver" is the name of the GIPSY command driver in a given GIPSY command, "command" is the name of the command, and "include" is the name of the created include file. Note that the include file and preceding comma may be left out of the command invocation. By default, the created include file has the name of the command, preceded by an "X". For example, using the EDGE command, the INCLMM command invocation would look as follows:

```
G: INCLMM GIPSY_SRC:DEDGE.RAT > EDGE
```

The name of the GIPSY command driver can be obtained from the GENCMD instruction in the link file for the EDGE command, GIPSY_LNK:EDGE.LNK. The created include file would be named XEDGE., with a null file type, and have the following components:

1. The module identifier - a NAMEi macro, where "i" is replaced with a digit, one to six, specifying the number of arguments, with "i" MAP-character arguments specifying the command name associated with that module,

and is checked by the MMLOAD routine;

2. The array allocator - a GETARRAYS macro without arguments, which reads array information defined by the GETWPM routine and is transmitted to the mini-MAP by the MMALOC routine;
3. The array descriptor - zero or more xDESCi macros, where "x" is replaced with "I", "R", or "C", for INTEGER*2, REAL*4, or COMPLEX*8 arrays, respectively, "i" is replaced with a one or two, for the number of array dimensions, and a single numeric argument specifying the index of the array to so designate; and
4. The array definition - zero or more RATFOR DEFINE statements, where the first argument is an array name defined by a call to the GETWPM routine, and the second argument specifies an array xARi, where "x" and "i" are interpreted as in the array descriptor.

For every GETWPM found in the GIPSY command driver, there is exactly one array descriptor, followed immediately by exactly one array definition. Since the mini-MAP uses INTEGER*2 arithmetic, a choice of array dimensions was provided. One dimension provides for a maximum of 32767

elements, typically sufficient for most applications. However, for completeness, the programmer may setup for two dimensions, which would permit the addressing of more than 32767 elements. The left-most dimension will always be 32767 in this case. The INCLMM command was designed to read any GIPSY command driver following standard GIPSY software conventions for driver construction.

7.5.2 MKRAN Command

The other GIPSY command useful to the mini-MAP programmer is called MKRAN. It is optionally used by the mini-MAP programmer who is debugging a mini-MAP module. See the description of the MMEXEC command for more information on debugging. This command has the following syntax:

```
G: MKRAN list > B-tree
Enter key length ( D = 10, 1 to 40 ) --
Enter tree depth ( D = 4, 1 to 10 ) --
```

where "list" is a text file with a list of single line entries of the form, key||text, and "B-tree" is a random-access file in B-plus tree format. Each line of the input file has a user-selected key field of fixed size, followed by text to be associated with that key. The keys are the character representations of the second values in all of the TRACE2 statements in a mini-MAP program and must

be in alphabetical, not numeric, order. The maximum line width is eighty characters. The B-tree is a linked-list of tree nodes, where internal nodes are clusters of keys, and leaf nodes contain the text associated with each key.

The user is prompted for two parameters. The first is the maximum key length, in characters. Since the second debug variable is declared INTEGER*2, the maximum key length would be six characters. The tree depth parameter needs to be computed from the key length and maximum number of keys. For a maximum key length of five characters, fifteen keys can fit into a seventy-five-character internal node. Each level increases the tree-capacity by a factor of fifteen. Thus, two levels can accommodate $15 \times 15 = 225$ input lines. More information on this command is included in GIPSY documentation.

In the next chapter, conversions of pre-existing GIPSY commands will be considered. The use of the facilities in this chapter are further illustrated.

8 BUILDING AN APPLICATION

The construction of a mini-MAP executable module is obviously dependent upon the intended application. To demonstrate this construction, four pre-existing GIPSY commands were adapted for use with the mini-MAP:

1. ABSIMG - constructs an output image from the absolute value of the pixels from an input image;
2. EDGE - identifies edge pixels from a least squares fit of a third-degree surface to a square neighborhood centered at each pixel of the original input image;
3. RIDVAL - identifies "ridge" and "valley" pixels from the same least squares fit used by the EDGE command; and
4. FACET2 - generates the coefficients of a least squares fit of a surface of any degree to a square picture neighborhood centered at each pixel of the input image.

This list is arranged in order of increasing modification complexity. The adaptation of these commands will be divided into four steps:

1. Host driver modifications
2. Host subroutine modifications
3. MAP-driver design
4. MAP-subroutine design

For each step, all of the commands mentioned above will be compared.

8.1 Host Driver Modifications

The alterations to the host drivers were similar in all of the commands. These alterations may be divided into four actions:

1. Added a call to the MMINIT routine after the call to the PPUSH routine:

```
CALL PPUSH( "driver name" )  
CALL MMINIT( 0 )
```

2. Added calls to the MMCLoS routine before the call to the PPOP routine, and before the alternate return statement:

```
CALL MMCLoS( 0 )  
  
CALL PPOP  
  
RETURN  
  
...  
  
CALL MMCLoS( 0 )  
  
RETURN 1  
  
END
```

3. Converted the GETWP function assignment statements to use the GETWPM function, where applicable. This step was not attempted until after the other routines were modified when the arrays used by the mini-MAP were identified; and
4. Added a call to the MMALOC routine. If there were conditions when the mini-MAP should not have been invoked, the MMALOC statement was enclosed in the body of an IF statement:

```
IF ( -AFLAG & -BFLAG & ... )  
  
CALL MMALOC( NXT )
```


where NXT is the variable used for the allocation pointers.

In all cases, the commands were altered to use the mini-MAP if there were no user flags set.

The first two actions were identical in all four commands. For the ABSIMG command, memory allocation was not implemented in standard GIPSY format, which necessitated a modification to the code prior to adding the mini-MAP support. Also, this was the only command without user flags that affect the application design. Thus, the call to the MMALOC routine was unconditional. For all commands except FACET2, the selected GETWPM arrays were the input array from the call to the RREAD primitive and the output array from the call to the RWRITE primitive. For the FACET2 command, there was considerable activity with intermediate arrays necessitating many more calls to GETWPM.

8.2 Host Subroutine Modifications

The modification of host subroutines was straightforward in all but the FACET2 command. All commands used synchronous, line-oriented I/O. That is, after the initial setup, the commands read in one line of input, processed that line, and finished by writing out one line of output. This procedure was repeated for every subsequent

line in the input image. As a result, the entire processing step was moved to the slave, except for the FACET2 command. For the EDGE and RIDVAL commands, some scalars were transmitted just prior to the nested loops with the first call to the RREAD primitive. For the ABSIMG command, the only required scalar was the array size, which was automatically passed to the slave by the MMALOC routine.

The situation was significantly more complicated in the FACET2 command. To illustrate the implementation of the software changes, several lines of code have been extracted from the software, and simplified, before and after modification. For subroutine calls, only those arguments pertinent to the discussion have been included. The simplified code before the modification looks as follows:

```
CALL MSKSCL( RBASIS )
CALL BASCAN( RBASIS [IN], SCALE, MTRXBC )
CALL PDSCR( SCALE [IN] )
CALL AREAD( BUFFER )
CALL INTSQS( BUFFER [IN], SQSBUF )
DO LINE = ...
  $(
    CALL AREAD( IOBUF )
    CALL APLBAS( IOBUF [IN], RCCONV )
```

```

DO BAND = 1, NBANDS
    $(
        CALL OUTBCT( SCALE [IN], MTRXBC [IN],
                    RCCONV [IN], IOBUF )
        CALL RWRITE( IOBUF [IN], BAND [IN],
                    LINE [IN] )
    $)

CALL SQCNV1( SQSBUF [IN], IOBUF )
CALL RWRITE( IOBUF [IN], NBANDS + 1 [IN],
            LINE [IN] )

CALL ROTPTU

$)

```

To clarify the flow of information, input arguments to the subroutines are suffixed with "[IN]". All of the subroutines were candidates for slave support, except for the I/O primitives, PDSCR (Put DeScriptor Real variable), AREAD (Augmented random file READ), and RWRITE (Random file WRITE), and the user-supplied BASCAN routine, which called mathematical library primitives.

Unlike the other commands, there was significant computation (MSKSCL and BASCAN routines) prior to the descriptor record processing (PDSCR routine), between the output (RWRITE routine) of multiple bands for each line

(OUTBCT and SQCNV1 routines), and after the final output for the line (ROTPTU routine). Once the positions of the ineligible routines, the DO loops, and the demand on input array data were known, the remaining code was enclosed in MMSKIP tests. Consider the above code after modification:

```
MAPINTEGER MAPIS( 8 )
IF ( MMSKIP( .TRUE. ) )
    CALL MSKSCL( RBASIS )
ELSE
    $(
        CALL MMSEND( MAPIS, 8, .MAPINTMODE,
                    .MAPIS, IEV, %9998 )
        CALL MMEEXEC( .WAIT )
        CALL MMRCVA( "RBASIS" )
    $)
CALL BASCAN( RBASIS [IN], SCALE, MTRXBC )
IF ( MMSKIP( .TRUE. ) )
ELSE
    $(
        CALL MMSNDA( "SCALE" )
        CALL MMEEXEC( .NOWAIT )
        CALL MMSNDA( "MTRXBC" )
    $)
CALL PDSCR( SCALE [IN] )
CALL AREAD( BUFFER )
```

```

IF ( MMSKIP( .TRUE. ) )
    CALL INTSQS( BUFFER [IN], SQSBUF )
ELSE
    $(
    CALL MMWAIT
    CALL MMSNDA( "BUFFER" )
    CALL MMEEXEC( .WAIT )
    CALL MMRCVA( "SQSBUF" )
    $)
DO LINE = ...
    $(
    CALL AREAD( IOBUF )
    IF ( MMSKIP( .TRUE. ) )
        CALL APLBAS( IOBUF [IN], RCONV )
    ELSE
        $(
        CALL MMSNDA( "IOBUF" )
        CALL MMEEXEC( .WAIT )
        CALL MMRCVA( "RCONV" )
        $)
    DO BAND = 1, NBANDS
        $(
        IF ( MMSKIP( .TRUE. ) )
            CALL OUTBCT( SCALE [IN], MTRXBC [IN],
                RCONV [IN], IOBUF )
        ELSE

```

```

        $(
        CALL MMWAIT
        CALL MMRCVA( "IOBUF" )
        CALL MMEEXEC( .NOWAIT )
        $)

        CALL RWRITE( IOBUF [IN], BAND [IN],
                    LINE [IN] )

        $)

        IF ( MMSKIP( .TRUE. ) )
            CALL SQCNV1( SQSBUF [IN], IOBUF )
        ELSE
            $(
            CALL MMWAIT
            CALL MMRCVA( "IOBUF" )
            $)

            CALL RWRITE( IOBUF [IN], NBANDS + 1 [IN],
                        LINE [IN] )

            IF ( MMSKIP( .TRUE. ) )
                CALL ROTPTU
            $)

```

Several features in the modified software were unique to the FACET2 command. The most obvious is the use of numerous calls to the MMSKIP routine. This implies that the slave software will contain many sections which will not execute on a given call to the MMEEXEC routine. This enables the use

of concurrency even when the host software is not concurrent. This prospect will be explored further, shortly.

Starting at the top of the modified code, at first glance two lines seem out of place:

```
MAPINTEGER MAPIS( 8 )  
...  
CALL MMSEND( MAPIS, 8, .MAPINTMODE,  
             .MAPIS, IEV, %9998 )
```

The use of the MMSEND routine is intended for use with the logical, not virtual, memory protocol. The FACET2 command passes many MAPINTEGER and REAL scalar variables to the slave. Rather than sending these variable to the slave individually, an array of scalars can be transmitted, provided the location of the scalars are known. This serves two purposes. First, copying the host integers to elements in the MAPIS array automatically converted the elements from VAX-sized INTEGER*4 variables to MAP-sized INTEGER*2 variables. This makes the host code less dependent on the intrinsic meaning of the INTEGER declaration in RATFOR. Second, the transfer time will be reduced. The .MAPIS symbol defines the starting address of the reserved

MAPINTEGERS. It and many other useful symbols are defined in the MAPDEF include file automatically included with the MACAL include file.

Earlier, the concept of concurrency was mentioned. Moving half-a-page down in the code, the opportunity for concurrency first appeared:

```
CALL BASCAN( RBASIS [IN], SCALE, MTRXBC )  
...  
CALL MMSNDA( "SCALE" )  
CALL MMEEXEC( .NOWAIT )  
CALL MMSNDA( "MTRXBC" )
```

The SCALE array being sent to the slave is also needed for the next step, the writing of descriptor records. No other array or processing is needed from the slave at this point. This frees the slave for future actions, justifying the .NOWAIT argument in the call to the MMEEXEC routine. At the same time, the MTRXBC array is ready for further processing, but the step using it, the OUTBCT routine, will not be executed until just before the calls to the RWRITE routine. That is, the slave will halt before it reaches the code using the MTRXBC array. This allowed the safe transmission of the MTRXBC array without collision with the active data

in the slave.

The other occasion for concurrency interleaves the slave processing with the output of individual bands in a given line:

```
DO BAND = 1, NBANDS
  $(
    ...
    CALL MMWAIT
    CALL MMRCVA( "IOBUF" )
    CALL MMEEXEC( .NOWAIT )
    ...
    CALL RWRITE( IOBUF [IN], BAND [IN],
                LINE [IN] )
  $)
  ...
  CALL MMWAIT
```

This is a straightforward optimization using a single buffer with asynchronous computation. The slave computes the next output buffer while the host is performing the I/O synchronously. The cost of this approach is the two extra calls to the MMWAIT routine, the first being non-operational since the slave is already known to be halted from the

previous operation.

The last observation about this implementation involves the final processing after the last output for the line:

```
IF ( MMSKIP( .TRUE. ) )  
CALL ROTPTU
```

It is unusual for the host code to be bypassed through the use of the MMSKIP routine, without a corresponding call to the MMEEXEC or MMWAIT routines. This situation arose when the results of post-processing were not needed outside of the slave. The actions of the ROTPTU routine were concatenated with the prior execution step. This effort traded the host calls to the MMEEXEC and MMWAIT routines for the extra time taken to perform the work in the slave. In this case, the extra time taken was negligible.

8.3 MAP-driver Design

With the host code in place for slave support, attention was then focussed on the slave software. There is a MAP-driver for each command, with a file name matching the command name. All of the drivers were less than a page, not counting include files, and contained one repeat loop with a call to a work routine, as in the host driver, followed by

the PAUSE statement. However, each MAP-driver had some unique features. First, consider ABSIMG.MMR, the MAP-driver for the ABSIMG command:

```
INCLUDE MAPDRV

INCLUDE XABSIMG

C      NAME6(.A,.B,.S,.I,.M,.G)

C      GETARRAYS

C      IDESC1( 1 )

C      DEFINE(IOLINE,IAR1)

REPEAT

  $(

TRACE( 1 )

I1 = INT( ARRAYLEN( IOLINE ) )

TRACE( 2 )

CALL ABSWRK( IOLINE, I1, I2, I3, I4, I5, DEBUG )

PAUSE

  $)

END
```

The contents of the XABSIMG include file, generated by the INCLMM GIPSY command, are listed below the INCLUDE instruction as comments. This convention will also be used with the other MAP-drivers. The ABSIMG driver only worked with one array of integers, IOLINE, for both input and output. The ARRAYLEN(IOLINE) macro returns the length of

the IOLINE array as a REAL*4 value. Provisions were made for debugging this module, as indicated by the TRACE statements and the DEBUG argument. However, without the .OPTDEBUG option specified before the INCLUDE XABSIMG statement and a DEBUGON statement, debugging is disabled. This module used only pre-defined scalar integers, I1 through I5, so that during a debugging session, all memory locations used by the slave could be monitored. The DEBUG argument permits TRACE statements in the ABSWRK routine to also be effective, if enabled.

A more complicated MAP-driver was needed for the EDGE command:

```
INCLUDE MAPDRV
DEFINE(NPPL,I1)
DEFINE(RADIUS,R1)
DEFINE(SIZE,R2)
DEFINE(GRADTH2,R3)
DEFINE(F4,R4)
DEFINE(F5,R5)
DEFINE(F6,R6)
DEFINE(F7,R7)
DEFINE(F8,R8)
DEFINE(F9,R9)
```

```

DEFINE(F10,R10)

DEFINE(F11,R11)

DEFINE(F12,R12)

DEFINE(F13,R13)

REAL EPS, XLOW, XHIGH, ANG, CST, SNT

REAL COS, SIN

DATA EPS / 0.001 /

DATA XLOW / 0.342702 /

DATA XHIGH / 2.59168 /

DATA ANG / 0.26 /

INCLUDE XEDGE

C      NAME4(.E,.D,.G,.E)

C      GETARRAYS

C      DEFINE(IIN,IAR1)

C      IDESC1(1)

C      DEFINE(IOUT,IAR2)

C      IDESC1(2)

CST = COS( ANG )

SNT = SIN( ANG )

REPEAT

$(

CALL EDGENC( IIN, IOUT, NPPL * 2, RADIUS, SIZE,

            GRADTH2, F4, F5, F6, F7, F8, F9,

            F10, F11, F12, F13, EPS, XLOW,

            XHIGH, CST, SNT, NPPL * 20 )

PAUSE

```

\$)

END

To be consistent with the variable names used in the host code, the pre-defined scalars were renamed using DEFINE statements. EQUIVALENCE statements could have also been used, but this would have required declaration statements. There were declarations made in this MAP-driver for constants, such as EPS, for local variables, such as CST, and for functions, such as COS. Note that SIN and COS are builtin trigonometric functions. They will be called once as a result of the first call to the MMEXEC routine in the host code. Also note that the definitions and declarations were placed between the MAPDRV and XEDGE include files. As a reminder, the MAPDRV include file must be the first line in the MAP-driver, and since there are executable statements in the XEDGE include file, the INCLUDE XEDGE statement must follow all declaration statements. The EDGENC routine will be called for each line in the input image.

Further complications occurred in the construction of the MAP-driver for the RIDVAL command, RIDVAL.MMR:

```
INCLUDE MAPDRV  
DEFINE(IIN,IAR1)
```

```

DEFINE(IOUT, IAR2)

DEFINE(NPPL, I1)

DEFINE(RADIUS, R1)

DEFINE(CRVTHR, R2)

DEFINE(ANGTHR, R3)

DEFINE(SCALE(X), R X)

DEFINE(DEPTHR, R13)

DEFINE(FRATIO, R14)

DEFINE(GRADTH, R15)

DEFINE(GTHETA, R16)

DEFINE(RVTHET, R17)

DEFINE(SIZE, R18)

NAME6(.R, .I, .D, .V, .A, .L)

GETARRAYS

IDESC1(1)

IDESC1(2)

REPEAT
  $(
    CALL RIDNC( IIN, IOUT, NPPL * 18, NPPL * 2,
               SCALE( 4 ), SCALE( 5 ), SCALE( 6 ),
               SCALE( 7 ), SCALE( 8 ), SCALE( 9 ),
               SCALE( 10 ), SCALE( 11 ),
               SCALE( 12 ), RADIUS,
               CRVTHR, ANGTHR, DEPTHR, FRATIO,
               GRADTH, GTHETA, RVTHET, SIZE )
  )
PAUSE

```

§)

END

This MAP-driver is like the EDGE MAP-driver except for two features. First, due to the fact that the host driver accepts a variable data mode for its arrays, the INCLMM command could not process the host driver's GETWPM statements. As a result, there is no XRIDVAL include file. If the programmer changes the order of the GETWPM statements in the host driver, then the DEFINE statements for the IIN and IOUT arrays will need to be modified. For the other commands, changing the order only requires a new invocation of the INCLMM command, prior to recompiling the MAP-driver. The other feature was the transmission of the SCALE array in the pre-defined REAL*4 scalars, supporting the addition of the SCALE array definition.

The most complicated MAP-driver was FACET2.MMR, the MAP-driver for the FACET2 command:

```
INCLUDE MAPDRV
DEFINE(NR,I1)
DEFINE(NC,I2)
DEFINE(NUMPPL,I3)
DEFINE(NBBND,I4)
```



```
DEFINE(MXDG1,I5)

DEFINE(MAX0,I6)

DEFINE(MAXI,I7)

DEFINE(NDIM5,I8)

INTEGER NDIM1, NDIM6, NDIM2, PNT, LL

REAL FLOAT

INCLUDE XFACET2

C      NAME6(.F,.A,.C,.E,.T,.D2)

C      GETARRAYS

C      IDESC1( 1 )

C      DEFINE(IOBUF,IAR1)

C      IDESC1( 2 )

C      DEFINE(IPT,IAR2)

C      RDESC1( 3 )

C      DEFINE(RBASIS,RAR3)

C      RDESC1( 4 )

C      DEFINE(CBASIS,RAR4)

C      RDESC1( 5 )

C      DEFINE(BUFFER,RAR5)

C      RDESC1( 6 )

C      DEFINE(RCONV,RAR6)

C      RDESC1( 7 )

C      DEFINE(RCCONV,RAR7)

C      RDESC1( 8 )

C      DEFINE(SCALE,RAR8)

C      RDESC1( 9 )
```

```

C      DEFINE(SQSBUF,RAR9)
C      RDESC1( 10 )
C      DEFINE(SQSSUM,RAR10)
C      RDESC1( 11 )
C      DEFINE(MTRXBC,RAR11)
C      IDESC1( 12 )
C      DEFINE(CTINDX,IAR12)
C      IDESC1( 13 )
C      DEFINE(PTRNCN,IAR13)

I20 = NR * MXDG1

I19 = NC * MXDG1

I18 = NR * NC

I17 = NBBND + 1

I16 = MXDG1 + 1

NDIM1 = NR * ( NUMPPL + NC )

NDIM6 = NR * NUMPPL

I15 = NR - 1

I14 = NUMPPL + NC

I13 = NR * 2

R19 = FLOAT( I18 )

CALL MSKSCL( RBASIS, CBASIS, BUFFER, SCALE, I20,
I19,

I18, NBBND, NR, MXDG1, NC, MAXI,
MAXO,

R20, CTINDX, PTRNCN, I17, I16,
NDIM5,

```

```

                SQSBUF, SQSSUM, R19, I15, NUMPPL,
I14,

                NDIM1, NDIM6, IPT, I13 )

NDIM2 = MXDG1 * I14

I12 = NUMPPL * 2

I11 = NC - 1

I10 = NUMPPL * NBBND

I9 = NBBND * NBBND

PAUSE

REPEAT

$(

CALL APLBAS( BUFFER, RCONV, RCCONV, RBASIS,
            CBASIS, NR, NC, NUMPPL, PNT,
            LL, I14 - 1, I14, MXDG1, NBBND,
            NDIM1, NDIM2, I20, I19, IOBUF,
            IPT, I13, SCALE, MTRXBC, CTINDX,
            PTRNCN, I17, I10, I9, NDIM5,
            SQSBUF, SQSSUM, I10, NDIM6,
            R20, I12, I11 )

PAUSE

$)

END

```

Unlike the other MAP-drivers, the FACET2 MAP-driver does significant processing, including the MSKSCL routine, and the PAUSE statement, prior to the main loop. There needed

to be a PAUSE statement for every host call to the MMWAIT routine. Also unlike the other commands, several of those PAUSEs will be outside of the main loop and in subroutines. This will be further studied in the next section. As with the ABSIMG MAP-driver, an attempt was made to keep many, if not all, of the scalar values in the pre-defined scalars for debugging purposes.

8.4 MAP-subroutine Design

The significant application work in a mini-MAP module should be performed by the subroutines. In most cases, sections of the host code were extracted without modification and placed in a MAP-subroutine. The notable exception was the ABSWRK routine, which contained most of the application code for the ABSIMG command. Unlike the other commands, the ABSIMG command worked only with integers and yielded integer results. Since the VAX worked with four-byte integers and the mini-MAP worked with two-byte integers, the MAP-subroutine had to be implemented with double-precision, integer arithmetic to be able to duplicate host activities accurately. Thus, the ABSWRK routine was implemented independently, without host code.

There were three kinds of alterations needed in the MAP-subroutines. The first kind was the removal of calls used in tracing, notably the calls to the PPUSH and PPOP routines. Tracing would have been accomplished by the TRACE macros, if desired. The second kind satisfied differences between the VAX FORTRAN compiler and the mini-MAP MFORT compiler. The biggest incompatibility occurred in the use of two-dimensional arrays. On the VAX, there was no problem declaring a one-dimensional array in the driver routine and then calling a subroutine which declared the same array as a two-dimensional array. This implementation was not permissible in the MAP-subroutines. As a consequence, all of the two-dimensional references were replaced by address calculations for one-dimensional arrays. Two other minor modifications were the expansion of integer exponents ("** N", where N is an integer) into repeated multiplications and the replacement of statement function definitions with RATFOR macro definitions.

The third kind of alteration was much more serious in nature. The mini-MAP architecture violates Von Neumann characteristics in several ways. The memory for instructions is not interchangeable with memory for data. Moreover, the mini-MAP has two memory partitions for instructions, one for the IPU (Integer Processing Unit)

which is used primarily for address calculations, and the other for the APU (Arithmetic Processing Unit) which is primarily used for floating-point computations. The crux of the problem is the fixed size of these two partitions. The IPU and APU each have space for 4096 instructions. This contrasts sharply with the data memory size measured in megabytes.

These characteristics engendered a problem in the construction of the FACET2 support. Originally, all of the subroutines were converted to MAP-subroutines with minor alterations. The linker used by the GENMAP facility, MLINK, was unable to fit all of the routines into the available space. Since the mini-MAP did not use a stack for the storage of call frames as the VAX does, space could be reclaimed by concatenating the subroutines in the same order they were called. The final MSKSCL routine contains code from the original MSKSCL, CANTRN, and INTSQS routines. The final APLBAS routine contains code from the original APLBAS, APBASR, CONVCO, APBASC, SQCNV1, SRESER, ROTPTU, and ROTBAS routines. More space was reclaimed by expanding calls to intrinsic functions, such as MIN and MAX. The programmer could use the link map (GENMAP cmd/MAP) to determine where space can be reclaimed and how much space needs to be reclaimed.

The time and effort needed to add the mini-MAP support was minimal for the EDGE and RIDVAL commands. Since the ABSIMG command was the first to be supported, intermediate developments in the host and slave supports slowed the rate of progress and increased the total development time. The FACET2 command required significantly more time and effort, mostly due to the size and complexity of the original code.

9 RESULTS AND DISCUSSION

As suggested in an earlier chapter, the purpose of adding array processor support to a GIPSY command was, and is, to increase the speed of the application. In this chapter several timing metrics will be considered, and the usefulness of the support will be discussed. The GIPSY commands, ABSIMG, EDGE, and FACET2, studied in the previous chapter, will receive further attention. For brevity, some shorthand notation will be adopted in this chapter:

1. "No support" - The state of the software prior to modification to employ the mini-MAP;
2. "Suppressed" - The state of the software after the addition of the mini-MAP support but with the support inhibited by setting the system P-flag in GIPSY;
3. "Active" - The state of the software with the mini-MAP support, enabled for normal operations;
4. "DMA" - Use DMA data transfers instead of memory-window transfers;

5. "Force" - Always load a new copy of the executable module without checking the state of the mini-MAP software;
6. "Trace" - Tracing enabled in the host via the system A-flag in GIPSY; and
7. "Debug" - Debugging display enabled in the host via the system D-flag in GIPSY.

These terms will be used in tables, possibly in combination, and throughout the chapter text.

9.1 ABSIMG Command

Consider first the ABSIMG command. This is a simple command, with a small amount of computation for every processed line:

Table 9-1: ABSIMG with a 50 x 50 image

G: ABSIMG NEGATIVE.SIF > POSITIVE.SIF (,WX)

No support	2.66 CPU seconds
Suppressed	2.69
Active	3.32
Active + Debug	96.86

Table 9-1 was measured using the specified ABSIMG invocation, including any additional system flags indicated by the shorthand labels. Several observations can be made by comparing the table entries pairwise. The difference between having no support and suppressed support identifies the overhead in the main loop processing and in a portion of the initialization and termination steps. In this case, the overhead amounted to 0.03 CPU seconds, or 1.1 % of the no-support time. This is reasonable for a repetitive operation. A comparison of the first and third values reveals an unexpected result. The addition of the mini-MAP support slowed the execution of the command by 0.66 CPU seconds, or 25 %. This indicates that the command is not complex enough to justify the addition of the support. More attention will be given to this phenomenon later in the chapter. The final pair of values demonstrates the cost of adding the .OPTDEBUG option and TRACE statements to the MAP-software and permitting the host software to report on slave activity. Obviously, debugging is costly in time, and, once unneeded, should be disabled.

Another set of tests on the ABSIMG command was performed using a larger, 512 x 512 image. This would increase the duration of the processing, and the proportion of computation to I/O.

Table 9-2: ABSIMG with a 512 x 512 image

G: ABSIMG NEGATIVE.SIF > POSITIVE.SIF (,X)

No support	16.42 CPU seconds
Suppressed	16.61
Active	30.98

The results presented in table 9-2 support the observations derived from table 9-1. The overhead of the suppressed support amounted to 0.19 CPU seconds, or, again, 1.1 %. However, the active support added 14.56 CPU seconds, or 89 % of the unsupported time consumption. This implies that as the array size increased, so did the processing time for the array. Since the array processing consists solely of data transfers, the support timing will need to be better studied. The array was transferred using a memory window rather than by DMA. This will be explored with the EDGE command, which has significantly more computation.

9.2 EDGE Command

The EDGE command was tested with the mini-MAP data memory capacity set to the basic 64 kilobytes and then with an additional megabyte of data memory. This forced the first test image to be limited to a forty pixel line width. Adding a megabyte of memory permitted the line width to be increased to 1600 pixels.

Table 9-3: EDGE with a 40 x 40 image

G: EDGE FACET.SIF > EDGE.SIF (,X)

No support	1.96 CPU seconds
Suppressed	2.07
Active	2.29
Active + Force	2.41
Active + DMA	2.06
Active + Force + DMA	2.19

Two additional variables have been tested in this set of metrics, the time needed to load a MAP-executable module ("Force") and the time needed to transfer arrays using DMA instead of a memory window ("DMA"). First consider the overhead of inactive support, 0.11 CPU seconds, or 5.6 % of the unsupported command time, which is still a small overhead. The use of the mini-MAP added 0.33 CPU seconds, or 17 % over the unsupported command. This result still makes the slave support cost-prohibitive, but not as discouraging as the ABSIMG results. The remaining test results were attempts to improve the active time by trading I/O for computation ("Force") and by using non-processor I/O ("DMA"). Table 9-3 shows that forcing a module load does not improve the results but using DMA transfers does. Unfortunately, the "Active + DMA" result is still greater than the unsupported result.

To confirm the observations made about the ABSIMG command, a larger 1000 x 1600 pixel image was substituted for the 40 x 40 pixel image. In addition, the tracing facility was employed to determine where time was consumed during execution:

Table 9-4: EDGE with a 1000 x 1600 image

G: EDGE FACET.SIF > EDGE.SIF (,X)

No support	347.44 CPU seconds
Suppressed	348.92
Active	372.30
Active + DMA	117.86
Active + DMA + Trace	999.21

The results started out as predicted. The overhead of the suppressed support added 1.48 CPU seconds, or 0.43 % of the unsupported command. As expected, the increased application complexity reduced the percentage of impact. Active support using a memory window for data transfers added 24.86 CPU seconds, or 7.2 % of the unsupported command. Again, the command complexity compensated for the overhead, as compared to the ABSIMG command. The next figure, "Active + DMA", was a pleasant surprise, reducing the CPU demand by 229.58 CPU seconds, or 66 % of the unsupported command. Thus, time can be saved by using the mini-MAP, but the application must be substantially CPU-intensive, with infrequent I/O interruptions. As indicated by the last item in table 9-4, tracing adds considerable overhead to the command execution.

As a result of the trace, the CPU demand of each step was recorded. Table 9-5 shows the routine names used in the execution of the EDGE command, adding "DMA" if applicable, the number of calls used in the statistic, the mean CPU time demand in seconds, and the 95 % confidence interval:

Table 9-5: Interface routines called by EDGE

G: EDGE FACET.SIF > EDGE.SIF (,AX)

<u>Routine</u>	<u>Calls</u>	<u>Time</u>	<u>95% CI</u>
MMINIT	1	1.20	--
GETWPM	2	0.035	0.064
MMALOC	1	0.39	--
MMSKIP	30	0.079	0.0016
MMSNDS	14	0.13	0.0038
MMSNDS + DMA	14	0.17	0.0053
MMSNDA	29	0.72	0.0071
MMSNDA + DMA	30	0.21	0.0036
MMEXEC	30	0.15	0.0018
MMRCVA	29	0.25	0.0043
MMRCVA + DMA	30	0.21	0.0034
MMCLOS	1	< 0.01	--

Of the nearly 1000 seconds needed for traced command execution with DMA, 4.1 seconds were needed outside the main loop for slave setup, and 650 seconds were consumed inside the main loop for interface activity. The largest segment of activity, 420 of those 650 seconds, were used solely for array transfers. Arrays required a minimum of 0.20 seconds to be read or written via DMA. Without DMA, the transfer time climbed by 240 % for 32000-byte writes and by 19 % for 3200-byte reads. However, for scalar transmission, DMA was actually slower by 31 %. The time for DMA transfer remains relatively constant, but the constant is large. Memory window transfers follow a pattern whose complexity is roughly linearly proportional to the number of shifts in the placement of the transfer window. The window was moved eight times for the transmitted array, once for the received array, and not at all for the scalars. Note that the scalar transfer time is the time needed for the smallest portion of memory, i. e., one byte. One byte took the same or more time, statistically, as two bytes. More discussion on transfer modes is forthcoming.

9.3 FACET2 Command

As a final example, the FACET2 command was also checked for time consumption. Unfortunately, even with a megabyte of memory, the mini-MAP could handle only a file with no more than forty pixels per line:

Table 9-6: FACET2 with a 40 x 40 image

G: FACET2 BARS.SIF > FACET.SIF (,X)

No support	4.71 CPU seconds
Suppressed	4.73
Active	4.94
Active + Force	5.01
Active + Force + Trace	45.15

The results were analogous to those obtained with ABSIMG or EDGE, operating on a small image.

9.4 Discussion

Reviewing the advantages of adding a slave processor, as cited in the introduction, several points can be made. First, the software interface makes the slave easily accessible for a variety of applications. What was not originally anticipated were the limits inherent with the slave, notably instruction capacity, and the demand for add-on data memory. The basic mini-MAP came with only 64 kilobytes of data memory and without DMA support. Memory limits can make the otherwise available slave unusable for the larger applications. Even though the DMA transfers are managed by the mini-MAP, the VAX still incurred a heavy CPU demand for initiating the transfer. The memory window was not a time-effective means for transferring arrays.

Second on the list of advantages was the ability to tailor applications. Test cases to date support this assertion. The applications were repetitive, spending a small time initializing the environment, and most of the time in one principal loop. The conversion of algorithms to the mini-MAP was possible with the MFORT compiler, which was made available while the interface routines were being

designed. The precursor compiler, called MCL (Mini-MAP Control Language), was not compatible enough with FORTRAN to make the conversion reasonably straightforward.

Third on the list of advantages was the degree of cooperation that could be achieved between the VAX and the mini-MAP. Some parallel processing was possible with the FACET2 command. The implementation of GIPSY I/O does not take advantage of any of the asynchronous capabilities available on the VAX. As a result, the only available option, short of implementing asynchronous I/O in GIPSY, was to interleave the processing of multiple bands with I/O, as was the case with FACET2, or to first modify the commands to work with multiple buffers. Unfortunately, the implementation of multiple buffers is costly in terms of software complexity, memory, and development time. The final advantage of monetary cost was not explored in this thesis.

Much attention has been given to the most notable drawback with slave computers, the programmer's resistance to use them. It has been demonstrated that the programmer requires little knowledge about the mini-MAP to be able to add slave support to pre-existing GIPSY commands. The conversion of many routines can be accomplished in less than

ten lines of code. Unfortunately, in the most complicated commands, such as FACET2, the conversion can turn out to be very difficult and time-consuming. Moreover, the benefits can not be measured until the conversion is completed and may not be great enough to justify the effort.

In actual test cases, the interface routines have demonstrated themselves to be easy to use and easy to insert into the existing software. All of the Group 1 routines, i. e., those which define the "black box", were directly called, except the MMLOAD routine. The MMINIT and MMCLAS routines were not permanently placed in the routine above the command drivers, GPMAIN, since there was a noticeable delay caused by testing the mini-MAP and looking for replacement mini-MAP executable modules. As illustrated with the EDGE command, the use of the MMLOAD routine is a costly venture, in terms of time, for small applications. An alternative solution to the FACET2 space problem could have been to simply split the code into two or more modules. The concept of redefining array space after the initial allocation was not explored, and, to date, is only performed by the ARCSEG and CURPRP commands in GIPSY.

Timing results with different images suggest that for each command there is a line width which will produce nearly identical timing results to set a minimum on the number of pixels required to justify the activation of the mini-MAP support. This value, or an expression using this value, if determined, can be stored as a module option, such as .OPTMIN, and read in with the other options by the MMLoad routine. This minimum can then be used by the GETWPM routine to validate the minimum array size needed for successful array allocation.

The Group 2 routines, which manage memory and data transmission, proved to be very useful. All of the test commands used the virtual memory management scheme and their supporting interface routines. Except for the FACET2 command, no other direct calls were made to any other interface routines. The FACET2 command demonstrated how to use the logical memory management routine, MMSEND, to gain more control and efficiency without causing harm to the virtual arrays. Although possibly useful, the programmer has not been given the freedom to reference portions of virtual arrays, such as a single line in a multi-line input buffer. With DMA support, the transmission of 3200 bytes took the same amount of CPU time, statistically, as ten times that amount of data. The mini-MAP incurred the CPU

delay in executing the transmission.

The timing results for the interface routines suggest that there is an array length that will require equal transmission time with the memory window or DMA. Currently, the MMSEND and MMRECV routines use either the memory window or DMA, but not both, for all data transmissions. A possible optimization of these routines could include a check on the array size, followed by the selection of the means of transmission, assuming DMA was enabled as an option.

The construction of the MAP-drivers was simple and straightforward for every command. The INCLMM command in GIPSY had no trouble reading GETWPM statements out of the GIPSY command drivers and, except for the RIDVAL command, generated the supporting include files as appropriate. Debugging support was occasionally necessary and made the repair of MAP-software possible. TRACE statements were easily shuffled around, and the returned scalars and arrays were examined with the VAX DEBUG utility. A possibility for future development would be a simplified scheme for building a B-tree file from a MAP-driver or MAP-subroutine. Currently, it can be done with EDT and MKRAN, using ABSIMG.MMR as an example:

```

$ EDIT ABSIMG.MMR
*   SET NUMBER
*   PRINT ABSIMG.PRT
*   QUIT
$ EDIT ABSIMG.PRT
*   DELETE 1 THRU 3
*   EXIT
$ GIPSY
G: MKRAN ABSIMG.PRT > ABSIMG.MSG
Enter key length ( D = 10, 1 to 40 ) -- 5
Enter tree depth ( D = 4, 1 to 10 ) -- 1
G: EXIT

```

This sequence of instructions prefixes the lines in ABSIMG.MMR with line numbers, removes the useless header lines, and builds a B-tree with 5-digit keys. Since there are less than sixteen lines in the ABSIMG MAP-driver, all of the keys could fit into one 80-character, root node of the tree. The resulting message file is readable by the MMEXEC routine and can be referenced by TRACE2 statements. The second number in the TRACE2 statement could be a line number in the message file. The above lines were tested, but the response by the MMEXEC routine was not.

In conclusion, the mini-MAP can be a useful tool in speeding up large applications with large images. It should also be equipped with DMA capability and at least one megabyte of extra data memory. Even more memory would be useful for the largest applications. The support of an array processor can be provided invisibly to the GIPSY user. The programmer can add array processor support without knowledge of the architecture of the mini-MAP. Unfortunately, this does not guarantee that programmers will use the mini-MAP. From an industrial perspective, the mini-MAP's worth will greatly depend on how fast, in real-time, not CPU time, results can be obtained.

o BIBLIOGRAPHY

Alexander, P., "Array Processor Design Concepts", Computer Design, December 1981, pp. 163 - 172.

Allison, A., "Powerful New Machines Find a Niche", Mini-Micro Systems, October 1987, pp. 121 - 131.

Bernhard, R., "Giants in Small Packages", IEEE Spectrum, February 1982, pp. 39 - 44.

Cohler, E. U. and Storer, J. E., "Functionally Parallel Architecture for Array Processors", Computer, September 1981, pp. 28 - 36.

Cohler, J., "Array Processors Enhance Minicomputer Performance", Mini-Micro Systems, April 1982, pp. 179 - 188.

Cushman, R. H., "Arithmetic Chips Assume Greater Importance as Micro-computer Users Demand Faster Response", EDN, April 14, 1982, pp. 61 - 69.

Dugdale, D. A., "Megaflops for Minibucks", Defense Electronics, August 1987, pp. 68 - 76.

Erickson, D. B., "Array Processing on an Array Processor", SIGPLAN Notices, March 1975, pp. 17 - 24.

Evans, D. J., "Problem Formulation Using Array Processing Techniques", SIGPLAN Notices, March 1975, pp. 153 - 163.

Karplus, W. J. and Cohen, D., "Architectural and Software Issues in the Design and Application of Peripheral Array Processors", Computer, September 1981, pp. 11 - 17.

Kohn, L., "Architecture of the Intel i860 64-bit Microprocessor", Proceedings of the Compcon Spring conference, March 1989.

Kotelly, G., "Floating-point Array Processors Evolve into Tailorable Systems", EDN, May 12, 1983, pp. 43 - 60.

Krusemark, S., "A Transportable Image Processing System Called GIPSY", M. S. Thesis, May, 1981, pp. 2 - 4.

Martin, S., "Programmable Array Processors Crunch Numbers Effortlessly", EDN, February 20, 1980, pp. 107 - 115.

Theis, D. J., "Array Processor Architecture", Computer, September 1981, pp. 8 - 9.

Wilson, D., "Array Processors: The New Crays?", Digital Design, December 1982, pp. 22 - 28.

Zakharov, V., "Parallelism and Array Processing", IEEE Transactions on Computers, January 1984, pp. 45 - 78.

A APPENDIX A: LISTINGS OF GROUP ONE ROUTINES

What follows is the RATFOR source code for the MMINIT, MMCLOS, MMLOAD, MMEXEC, MMWAIT, and MMSKIP routines. They are located in the GIPSY_PRIM directory in the GIPSY package.

A.1 MMINIT Routine

```
#--MMINIT          INITIALIZATION FOR MINIMAP SUPPORT SOFTWARE
#
# IDENTIFICATION
#
#       TITLE           MMINIT
#       AUTHOR          GREGG R FABREGAS
#       VERSION         A.01
#       DATE            26-MAY-1984
#       LANGUAGE        RATFOR
#       SYSTEM          VAX 11/780
#
#
# UPDATE
#
#       UPDATE          # 1
#       AUTHOR          GREGG R FABREGAS
#       DATE            06-OCT-1985
#       VERSION         A.01
#       PURPOSE         ADD CALLS TO STARTTIMER AND
#                       CHECKTIMER TO MONITOR ELAPSED
#                       CPU TIME.
#
#       UPDATE          # 2
#       AUTHOR          GREGG R FABREGAS
#       DATE            13-OCT-1985
#       VERSION         A.01
#       PURPOSE         ADD CALL TO OSMMAP TO DETERMINE IF
#                       MMSSEND AND MMRECV MAY TRANSMIT DATA
#                       VIA DMA MECHANISM.
#
# PURPOSE
#
#       THIS IS ONE OF THE GIPSY PRIMITIVES IN THE MINIMAP
#       SUPPORT SOFTWARE.  THE MINIMAP OPERATING ENVIRONMENT
#       IS DEFINED AND INITIALIZED HERE.  FOR EACH GIPSY
#       COMMAND, A MINIMAP IS ACQUIRED IF ONE IS AVAILABLE,
#       AND THE DEFAULT MODULE IS LOADED INTO THE MINIMAP.
#       THE DEFAULT MODULE HAS THE SAME NAME AS THE GIPSY
#       COMMAND.
#
# ENTRY POINT
#
#       IEV = MMINIT( DUMMY )
#
# ARGUMENT LISTING
#
#       IEV      INTEGER      INTEGER EVENT VARIABLE.
#                       RETURNS THE FOLLOWING ERROR
```

```

#          CONDITION CODES:
#          -200 IF THE SYSTEM P-FLAG IS
#          SET, PREVENTING THE
#          COMMAND FROM USING THE
#          MINIMAP SUPPORT
#          SOFTWARE, AND
#          -201 IF A MINIMAP COULD NOT
#          BE OPENED, OR ALLOCATED
#          DUMMY ARGUMENT. RESERVED FOR
#          FUTURE USE.
#          DUMMY  INTEGER
#
# INCLUDE FILES/COMMONS
#
#          GIPCOM  COMMON  GIPSY COMMAND LINE COMMON DATA
#          TTCOM  COMMON  FD'S FOR TERMINAL & RUNFILE IO
#          MINIMAP INCLUDE COMMON FOR THE ADMINISTRATION
#          OF MINIMAP SUPPORT SOFTWARE
#
# HARDWARE REQUIRED
#
#          A CSPI MINIMAP ARRAY PROCESSOR MUST BE AVAILABLE FOR
#          ALLOCATION.
#
# LANGUAGE FEATURES REQUIRED
#
#          ALTERNATE RETURNS ARE NEEDED.
#
# PROGRAM ENVIROMENT
#
#          THIS ROUTINE IS NORMALLY EXECUTED BY THE GPMAIN
#          PRIMITIVE TO DEFINE THE GIPSY ENVIRONMENT, PRIOR TO
#          THE CALL TO THE GIPSY COMMAND DRIVER. THE
#          PROGRAMMER IS FREE TO RE-EXECUTE THIS COMMAND, BUT
#          THE PREVIOUS MINIMAP IS FREED PRIOR TO ALLOCATION.
#
# ALGORITHM
#
#          (1) TERMINATE PRIOR MINIMAP ACTIVITY,
#          (2) INITIALIZE THE ADMINISTRATION COMMON,
#          (3) TEST THE SYSTEM P-FLAG FOR OVERRIDE,
#          (4) ALLOCATE AND OPEN THE MINIMAP,
#          (5) GET THE CURRENT MODULE NAME AND CONTEXT FROM
#             THE MINIMAP, AND
#          (6) IF NECESSARY, LOAD IN THE MODULE ASSOCIATED
#             WITH THE CURRENT GIPSY COMMAND.
#
# ROUTINES CALLED
#
#          PPUSH  PUSH PROGRAM NAME ONTO ERROR STACK
#                (GIPSY PRIMITIVE)
#          PUTF   MACHINE INDEPENDENT FORMATTED OUTPUT

```



```

#           (GIPSY PRIMITIVE)
#   MMCLOS  CLOSE THE MINIMAP FOR THE GIPSY SUPPORT
#           SOFTWARE (GIPSY PRIMITIVE)
#   MMALOC  ALLOCATE MINIMAP MEMORY FOR THE GIPSY
#           SUPPORT SOFTWARE. (GIPSY PRIMITIVE)
#   OSINFO  GET SYSTEM INFORMATION
#           (GIPSY OS KERNEL)
#   STSGMM  HSL ROUTINE: SET SIGNAL STATUS FOR
#           HSL ROUTINES. (GIPSY LIBRARY)
#   OSMMAP  PERFORM MACHINE-DEPENDENT FUNCTIONS
#           FOR THE MINIMAP SUPPORT SOFTWARE.
#           (GIPSY OS KERNEL)
#   OPENMM  HSL ROUTINE: OPEN THE MINIMAP.
#           (GIPSY LIBRARY)
#   OSSIEV  SAVE EVENT VARIABLE.
#           (GIPSY OS KERNEL)
#   MMLoad  LOAD MINIMAP MODULE FOR GIPSY SUPPORT
#           SOFTWARE. (GIPSY PRIMITIVE)
#   STRCMP  COMPARE TWO .EOS-TERMINATED STRINGS
#           (GIPSY OS KERNEL)
#   PRESET  RESET ERROR STACK TO CURRENT ROUTINE
#           (GIPSY PRIMITIVE)
#   PPOP    POP PROGRAM NAME FROM THE ERROR STACK
#           (GIPSY PRIMITIVE)
#   OSGIEV  GET EVENT VARIABLE (IEV) FROM OS SAVE
#           AREA (GIPSY OS KERNEL)
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
#   INCLUDE GIPSY
#
#   INTEGER FUNCTION MMINIT( DUMMY )
#
#
#   INCLUDE GIPCOM
#   INCLUDE TTCOM
#   INCLUDE MINIMAP
#
#   INTEGER DUMMY, IEV, I, RETNAM( EVAL(.OPTNAMELEN,+,1) ), DOT
#   LOGICAL TRACE, DOLOAD, OVRIDE, FORCE
#   CHARACTER NAME( .FILENAMELENGTH )
#   MAPINTEGER HSVIRT, LUN, MIEV
#   INTEGER OSINFO, OSSIEV, STRCMP, OSGIEV
#
#   EQUIVALENCE ( TRACE, SFLAG( .A ) ), ( OVRIDE, SFLAG( .P ) )
#   EQUIVALENCE ( FORCE, SFLAG( .F ) )
#
#   DATA MMCHKD / .FALSE. /
#   DATA DOT / '.' /

```

```

DATA HSVIRT / 0 /
DATA LUN / -1 /
#
#           PUSH PROGRAM NAME ONTO ERROR TRACE STACK
#
CALL PPUSH( "MMINIT" )
IF ( TRACE )
    $(
    CALL PUTF( FDRUNO, "AP trace: MMINIT called%N" )
    STARTTIMER
    $)
#
#           DETERMINE IF MMINIT HAS BEEN CALLED AND CLOSE
#           THE MINIMAP.
#
CALL MMCLOS( 0 )
#
#           INITIALIZE COMMON, SETTING MMCHKD TO INDICATE
#           THAT THIS ROUTINE HAS BEEN CALLED. ALLOCATION
#           PARAMETERS ARE RESET BY MMALOC, WHICH REQUIRES
#           AN OPEN MINIMAP, SO CALL MMALOC AFTER THE
#           OPENMM ROUTINE.
#
MMCHKD = .TRUE.
MMOPND = .FALSE.
MMACTV = .FALSE.
MMREDY = .FALSE.
MMDOLO = .FALSE.
MMEXCD = .FALSE.
MMTOST = .FALSE.
IF ( .OK == OSMMAP( .DMAENABLED, MMDMAE ) ) GOTO 9800
MMILAT = 0
DO I = .MODEMIN, .MODEMAX
    IF ( .OK == OSINFO( I, MMMODE( I ) ) ) GOTO 9800
#
#           SETTING THE SYSTEM P-FLAG SIMULATES AN
#           INITIALIZATION FAILURE SO THAT NO MINIMAP
#           PROCESSING IS DONE.
#
IF ( OVRIDE )
    $(
    IF ( TRACE )
        CALL PUTF( FDRUNO, "AP trace: override in effect%N" )
    GOTO 9000
    $)
#
#           TURN OFF SIGNALLING OF INFORMATIONAL MESSAGES.
#           THIS PROCEDURE IS VAX-SPECIFIC. SIGNALLING OF
#           MESSAGES IS TRAPPED BY THE MMINTR ROUTINE.
#
IFDEF VAXVMS

```

```

        CALL STSGMM( .FALSE. )
    ENDIF
#
#       ALLOCATE THE MINIMAP AND RESET.  THIS SHOULD
#       BE THE ONLY PLACE WHERE OPENMM IS CALLED.  SET
#       MMOPND AFTER THE CALL TO OPENMM SUCCEEDS FOR
#       USE BY MMCLOS.  PREALLOCATION ELIMINATES SOME
#       OF THE INTERRUPTS THAT CAN BE GENERATED BY THE
#       OPENMM ROUTINE.
#
    IF ( .OK ~= OSMMAP( .ALLOCATE, NAME ) ) GOTO 9010
    IF ( TRACE )
        CALL PUTF( FDRUNO, "AP trace: %S allocated%N", NAME )
    CALL OSMMAP( .DEALLOCATE )
    MMPHYS( 1 ) = 0
    MMPHYS( 2 ) = 0
    CALL OPENMM( HSVIRT, LUN, MMPHYS, MIEV )
    IF ( .OK ~= OSMMAP( .INTMODE, MIEV, IEV ) ) GOTO 9800
    IF ( .OK ~= OSSIEV( IEV, "OPENMM called" ) )
        GOTO 9010
    MMOPND = .TRUE.
    MMACTV = .TRUE.
    CALL MMALOC( 0 )
    MMACTV = .FALSE.
#
#       START THE MINIMAP TO GET THE PROGRAM NAME, AND
#       THEN COMPARE THE NAME TO THE HOST COMMAND NAME.
#
    MMSADR = .MAPBEGIN
    DOLOAD = .TRUE.
    FOR ( I = 1; I < EVAL(.OPTNAMELEN,+,1); I = I + 1 )
        IF ( CMDNM( I ) ~= DOT )
            MMNAME( I ) = CMDNM( I )
        ELSE
            BREAK
    MMNAME( I ) = .EOS
    MMACTV = .TRUE.
#
#       IF THE SYSTEM F-FLAG IS SET, FORCE THE LOAD
#       OF A NEW MODULE.
#
    IF ( ~ FORCE )
        $(
            CALL MMLOAD( RETNAM, IEV, %1000 )
            IF ( STRCMP( MMNAME, RETNAM, .OPTNAMELEN ) == 0 )
                DOLOAD = .FALSE.
        $)
#
#       IF (1) MMEXEC FAILS, (2) MMWAIT FAILS, OR (3)
#       THE MODULE NAME DIFFERS FROM THE COMMAND NAME,
#       THEN LOAD A NEW MODULE INTO THE MINIMAP.

```

```

#
1000 CONTINUE
IF ( TRACE & IEV /= .OK )
    CALL PUTF( FDRUNO, "AP trace: error IEV = %I%N", IEV )
CALL PRESET( "MMINIT" )
MMDOLO = .TRUE.
IF ( DOLOAD )
    $(
    IF ( TRACE )
        CALL PUTF( FDRUNO, "AP trace: load required%N" )
    CALL MMLOAD( MMNAME, IEV, %9900 )
    $)
ELSE
#
#           THE PREVIOUS MODULE IS THE INTENDED MODULE, SO
#           FLAG THE READINESS OF THE MODULE AND SET FOR
#           CONTINUOUS EXECUTION.
#
    $(
    MMREDY = .TRUE.
    MMSADR = .MAPCONTINUE
    IF ( TRACE )
        CALL PUTF( FDRUNO, "AP trace: no load done%N" )
    $)
#
#           INITIALIZATION IS COMPLETE.
#
CALL OSSIEV( .OK, "MMINIT succeeded" )
CHECKTIMER
CALL PPOP
RETURN ( .OK )
#
#           ERROR CONDITIONS:
#
#           OVERRIDE IN EFFECT BY SYSTEM P-FLAG
#
9000 CONTINUE
IEV = -2020
GOTO 9900
#
#           CALL TO OPENMM FAILED.
#
9010 CONTINUE
IEV = -2021
GOTO 9900
#
#           ERROR OCCURRED IN LOWER GIPSY ROUTINE.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#

```

```
#          RETURN ERROR IEV.  
#  
9900 CONTINUE  
IEV = OSSIEV( IEV, "MMINIT failed" )  
CALL PRESET( "MMINIT" )  
CHECKTIMER  
CALL PPOP  
RETURN ( IEV )  
#  
#  
END
```

A.2 MMCLOS Routine

```
#--MMCLOS  TERMINATE ACTIVITY FOR MINIMAP SUPPORT SOFTWARE
#
#  IDENTIFICATION
#
#      TITLE           MMCLOS
#      AUTHOR          GREGG R FABREGAS
#      VERSION         A.01
#      DATE            18-JUL-1984
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#
#  UPDATE
#
#      UPDATE          # 1
#      AUTHOR          GREGG R FABREGAS
#      DATE            06-OCT-1985
#      VERSION         A.01
#      PURPOSE         ADD CALLS TO STARTTIMER AND
#                     CHECKTIMER TO MONITOR ELAPSED
#                     CPU TIME.
#
#  PURPOSE
#
#      EVERY CALL TO MMINIT, ANOTHER ROUTINE IN THE
#      MINIMAP SUPPORT SOFTWARE, SHOULD BE PAIRED
#      WITH A CALL TO MMCLOS, SO THAT THE MINIMAP
#      AND ITS SUPPORT SOFTWARE IS RETURNED TO A
#      CONSISTENT STATE.  THIS ROUTINE IS USED
#      WITHIN MMINIT SO THAT THE MINIMAP AND THE
#      SOFTWARE ARE USED IN A LOGICAL SEQUENCE.
#      ACTIVITY ON THE MINIMAP IS TERMINATED, AND
#      ANY EXECUTION OF OTHER ROUTINES IN THE
#      SUPPORT SOFTWARE, EXCEPT FOR MMINIT, WILL
#      FAIL AFTER A CALL TO THIS ROUTINE.
#
#  ENTRY POINT
#
#      IEV = MMCLOS( DUMMY )
#
#  ARGUMENT LISTING
#
#      IEV      INTEGER      INTEGER EVENT VARIABLE.
#                                     RETURNS ONE OF THE FOLLOWING
#                                     STATUS CODES WHEN DONE:
#                                     .OK   IF THE CALL SUCCEEDS,
#                                     -2023 IF MMINIT HAS NOT BEEN
#                                     PREVIOUSLY CALLED,
```

```

#           -2038 IF THE MINIMAP HAS NOT
#           BEEN OPENED
#           BY THE OPENMM ROUTINE
#           IN MMINIT, AND
#           -2039 IF THE CALL TO CLOSMM
#           FAILED.
#           DUMMY   INTEGER   DUMMY ARGUMENT, WHICH IS NOT
#           USED.
#
# INCLUDE FILES/COMMONS
#
#           GIPSY   INCLUDE   GIPSY GENERAL SYMBOL DEF'NS
#           GIPCOM  COMMON    GIPSY COMMAND LINE COMMON DATA
#           TTCOM   COMMON    FD'S FOR TERMINAL & RUNFILE IO
#           MINIMAP COMMON    COMMON FOR THE ADMINISTRATION
#                               OF THE MINIMAP
#                               SUPPORT SOFTWARE.
#
# HARDWARE REQUIRED
#
#           A CSPI MINIMAP ARRAY PROCESSOR MUST HAVE BEEN
#           ALLOCATED.
#
# PROGRAM ENVIROMENT
#
#           THIS ROUTINE IS EXPECTED TO BE CALLED AFTER A CALL
#           TO THE MMINIT
#           ROUTINE.  IN GIPSY, THIS ROUTINE IS MOST LIKELY
#           CALLED ONLY BY
#           THE GPMAIN ROUTINE, FOLLOWING THE CALL TO THE DRIVER
#           ROUTINE FOR EACH GIPSY COMMAND.
#
# ROUTINES CALLED
#
#           PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#                     (GIPSY PRIMITIVE)
#           PUTF       MACHINE INDEPENDENT FORMATTED OUTPUT
#                     (GIPSY PRIMITIVE)
#           CLOSMM     HSL ROUTINE: CLOSE THE MINIMAP.
#                     (GIPSY LIBRARY)
#           OSMMAP     MACHINE-DEPENDENT CODE FOR THE
#                     MINIMAP SUPPORT SOFTWARE.
#                     (GIPSY OS KERNEL)
#           OSSIEV     PUT EVENT VARIABLE (IEV) INTO SAVE
#                     AREA (GIPSY OS KERNEL)
#           PPOP       POP PROGRAM NAME FROM THE ERROR STACK
#                     (GIPSY PRIMITIVE)
#           OSGIEV     GET SAVED IEV FROM SAVE AREA.
#                     (GIPSY OS KERNEL)
#
#
#

```

```

#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
#   INCLUDE GIPSY
#
#   INTEGER FUNCTION MMCLOS( DUMMY )
#
#
#   INCLUDE GIPCOM
#   INCLUDE TTCOM
#   INCLUDE MINIMAP
#
#   INTEGER DUMMY, IEV
#   LOGICAL TRACE
#   MAPINTEGER MIEV
#   INTEGER OSSIEV
#
#   EQUIVALENCE ( TRACE, SFLAG( .A ) )
#
#           PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
#   CALL PPUSH( "MMCLOS" )
#   IF ( TRACE )
#       $(
#           CALL PUTF( FDRUNO, "AP trace: MMCLOS called%N" )
#           STARTTIMER
#       $)
#
#           THE MINIMAP MUST BE OPEN BEFORE CLOSING. THIS
#           REQUIRES THAT (1) MMINIT HAS BEEN CALLED, AND
#           (2) OPENMM HAS EXECUTED SUCCESSFULLY. IF
#           MMINIT HAS BEEN CALLED, IMMEDIATELY CLEAR
#           MMCHKD BEFORE ANY FURTHER ERROR CHECKING.
#
#   IF ( ~ MMCHKD )
#       GOTO 9000
#   MMCHKD = .FALSE.
#   IF ( ~ MMOPND )
#       GOTO 9010
#
#           CLOSE THE MINIMAP. TRACE MESSAGES WILL BE
#           PICKED UP BY MMINTR.
#
#   IEV = .OK
#   CALL CLOSMM( MIEV )
#   IF ( .OK == OSMMAP( .INTMODE, MIEV, IEV ) ) GOTO 9800
#   IF ( .OK == OSSIEV( IEV, "CLOSMM called" ) )
#       GOTO 9020
#
#           NORMAL RETURN IEV.
#
#

```



```

CALL OSSIEV( .OK, "MMCLOS succeeded" )
CHECKTIMER
CALL PPOP
RETURN ( .OK )
#
#           ERROR CONDITIONS:
#
#           MMINIT ROUTINE HAS NOT BEEN EXECUTED YET.
#
9000 CONTINUE
IEV = -2023
GOTO 9900
#
#           MINIMAP NOT OPEN BY OPENMM ROUTINE (IN MMINIT).
#
9010 CONTINUE
IEV = -2038
GOTO 9900
#
#           CALL TO CLOSMM FAILED.
#
9020 CONTINUE
IEV = -2039
GOTO 9900
#
#           RETRIEVE ERROR FROM LOWER ROUTINE.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#
#           RETURN ERROR IEV.  DEALLOCATE MINIMAP IF
#           ALLOCATION SUCCEEDS BUT THE OPENMM OR CLOSMM
#           ROUTINES FAIL.
#
9900 CONTINUE
CALL OSMMAP( .DEALLOCATE )
IEV = OSSIEV( IEV, "MMCLOS failed" )
CHECKTIMER
CALL PPOP
RETURN ( IEV )
#
#
END

```

A.3 MMLOAD Routine

```
#--MMLOAD  LOAD MODULE FOR THE MINIMAP SUPPORT SOFTWARE
#
# IDENTIFICATION
#
#     TITLE           MMLOAD
#     AUTHOR          GREGG R FABREGAS
#     VERSION         A.01
#     DATE            26-JUL-1984
#     LANGUAGE        RATFOR
#     SYSTEM          VAX 11/780
#
#
# UPDATE
#
#     UPDATE          # 1
#     AUTHOR          GREGG R FABREGAS
#     DATE            07-OCT-1985
#     VERSION         A.01
#     PURPOSE         ADD CALLS TO STARTTIMER
#                     AND CHECKTIMER TO
#                     MONITOR ELAPSED CPU TIME.
#
# PURPOSE
#
#     THIS ROUTINE LOADS A PROGRAM MODULE INTO THE MINIMAP
#     PROGRAM
#     MEMORY, AND ESTABLISHES ANY HOST CONTEXT REQUIRED BY
#     THE MODULE.
#     THE MODULE IS GENERATED BY THE MLINK FACILITY
#     PROVIDED WITH THE
#     MINIMAP.  THERE IS A SPECIAL FORMAT USED BY MODULES
#     COMPATIBLE
#     WITH GIPSY'S MINIMAP SUPPORT SOFTWARE.
#
# ENTRY POINT
#
#     CALL MMLOAD( MODULE, IEV, ERRET )
#
# ARGUMENT LISTING
#
#     MODULE  CHAR ARR      NAME OF THE MODULE TO BE
#                          LOADED INTO THE
#                          MINIMAP.  THE NAME IS PASSED
#                          AS AN .EOS-TERMINATED
#                          STRING WITH NO MORE THAN
#                          SIX CHARACTERS IN THE NAME.
#     IEV     INTEGER       INTEGER EVENT VARIABLE.
#                          RETURNS THE FOLLOWING
```

```

# ERROR CODES WHEN DONE:
# .OK IF THE CALL SUCCEEDS,
# -2023 IF THE MMINIT ROUTINE
# HAS NOT BEEN
# PREVIOUSLY CALLED,
# -2024 IF THE MMINIT ROUTINE
# FAILED,
# -2025 IF AN UNKNOWN OR
# ILLEGAL CONTEXT
# HAS BEEN SPECIFIED BY
# THE MODULE, AND
# -2045 IF THE MMINTR ROUTINE
# TRAPPED AN ILLEGAL
# ADDRESS EXCEPTION.
# ERRET LABEL LABEL IN CALLING ROUTINE FOR
# ALTERNATE ERROR RETURN.
#
# INCLUDE FILES/COMMONS
#
# GIPSY INCLUDE GIPSY GENERAL SYMBOL DEF'NS
# GIPCOM COMMON GIPSY COMMAND LINE COMMON DATA
# TTCOM COMMON FD'S FOR TERMINAL & RUNFILE IO
# MINIMAP COMMON COMMON FOR THE ADMINISTRATION
# OF THE MINIMAP
# SUPPORT SOFTWARE.
#
# HARDWARE REQUIRED
#
# A CSPI MINIMAP ARRAY PROCESSOR MUST BE OPENED.
#
# LANGUAGE FEATURES REQUIRED
#
# ALTERNATE RETURNS ARE NEEDED.
#
# PROGRAM ENVIROMENT
#
# THIS ROUTINE MAY BE CALLED ANY NUMBER OF TIMES AFTER
# THE MMINIT
# ROUTINE EXECUTES CORRECTLY. THE MODULES MUST BE IN
# GIPSY FORMAT.
#
# ALGORITHM
#
# (1) IF THE MMDOLO FLAG IN THE MINIMAP COMMON IS SET,
# VERIFY THE
# FORMAT OF THE MODULE NAME, AND LOAD THE MODULE
# INTO THE MINIMAP,
# (2) INITIALIZE THE MEMORY IN THE MINIMAP USED TO
# RETRIEVE THE MODULE CONTEXT,
# (3) START THE MINIMAP TO GET THE CONTEXT,
# (4) WAIT FIVE SECONDS FOR THE CONTEXT AND MODULE

```

```

#           NAME TO BE RETURNED,
#           (5) EXAMINE AND ACT ON EACH DEFINED OPTION SLOT,
#           (6) IF TRACE ON OR NO LOAD DONE, CONSTRUCT AN .EOS-
#           TERMINATED STRING FOR OUTPUT AND/OR RETURN, AND
#           (7) FLAG THE COMMON FOR CONTINUOUS EXECUTION FOR THE
#           MODULE.
#
#

```

```

# DATA FORMAT
#

```

```

#           THE MODULE IS EXPECTED TO LOAD
#           AN ARRAY WITH ANY OPTIONS FOR
#           HOST CONTROL, AND THE NAME OF
#           THE MODULE, AND THEN PAUSE. THE
#           APPLICATION ALGORITHM SHOULD
#           THEN FOLLOW. THE ARRAY OFFSETS
#           FOR THE MODULE OPTIONS AND THE
#           MODULE NAME ARE DEFINED IN THE
#           MAPOPTS INCLUDE FILE.
#

```

```

# LIMITATIONS
#

```

```

#           VALUES DETECTED IN UNDEFINED
#           OPTION SLOTS IN THE RETRIEVED ARRAY
#           ARE TREATED AS UNKNOWN OPTIONS.
#

```

```

# ROUTINES CALLED
#

```

```

#           PPOUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#                       (GIPSY PRIMITIVE)
#           PPUTF      MACHINE INDEPENDENT FORMATTED OUTPUT
#                       (GIPSY PRIMITIVE)
#           OSMMAP     PERFORM MACHINE-DEPENDENT OPERATIONS
#                       FOR THE MINIMAP SUPPORT SOFTWARE.
#                       (GIPSY OS KERNEL)
#           MMSSEND    SEND SCALAR TO MINIMAP FOR MINIMAP
#                       SUPPORT SOFTWARE. (GIPSY PRIMITIVE)
#           MMEXEC     EXECUTE MINIMAP PROGRAM FOR MINIMAP
#                       SUPPORT SOFTWARE. (GIPSY PRIMITIVE)
#           MMWAIT     WAIT FOR MODULE FOR MINIMAP SUPPORT
#                       SOFTWARE. (GIPSY PRIMITIVE)
#           MMRECV     RECEIVE SCALAR FROM MINIMAP FOR
#                       MINIMAP SUPPORT SOFTWARE.
#                       (GIPSY PRIMITIVE)
#           OSSIEV     PUT EVENT VARIABLE (IEV) INTO SAVE
#                       AREA (GIPSY OS KERNEL)
#           STRCPY     COPY .EOS-TERMINATED STRING
#                       (GIPSY OS KERNEL)
#           PPOPOP     POP PROGRAM NAME FROM THE ERROR STACK
#                       (GIPSY PRIMITIVE)
#           OSGIEV     GET EVENT VARIABLE (IEV) FROM OS SAVE
#                       AREA (GIPSY OS KERNEL)
#

```

```

#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
#   INCLUDE GIPSY
#
#   SUBROUTINE MMLOAD( MODULE, IEV, * )
#
#
#   INCLUDE GIPCOM
#   INCLUDE TTCOM
#   INCLUDE MINIMAP
#
#   CHARACTER MODULE( 7 ), NAME( 7 )
#   INTEGER IEV, I
#   LOGICAL TRACE
#   INTEGER OSMMAP, OSSIEV, OSUNPK, OSGIEV
#   MAPINTEGER MOPTS( .OPTMAX ), MNAME( .OPTNAMELEN )
#   MAPINTEGER RESET( .OPTEND )
#
#   EQUIVALENCE ( TRACE, SFLAG( .A ) )
#
#   DATA RESET / .OPTEND * -1 /
#
#           PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
#   CALL PPUSH( "MMLOAD" )
#   IF ( TRACE )
#       $(
#           CALL PUTF( FDRUNO, "AP trace: MMLOAD called%N" )
#           STARTTIMER
#       $)
#
#           THIS REQUIRES INITIALIZATION TO BE COMPLETE,
#           UNLESS THE CALL IS MADE INSIDE MMINIT.
#
#   IF ( - MMCHKD )
#       GOTO 9000
#   IF ( - MMACTV )
#       GOTO 9010
#   MMREDY = .TRUE.
#
#           RESET MODULE CONTEXT TO DEFAULT VALUES.
#
#   MMTIME = 30
#   MMMAXP = 32767
#   MMDEBUG = .DEBUGDISABLED
#
#           IF MMDOLO IS SET, LOAD THE MODULE INTO THE
#           MINIMAP.  MMDOLO IS CLEAR ONLY IN MMINIT TO

```

```

#           CHECK THE MODULE ALREADY LOADED INTO THE
#           MINIMAP.
#
IF ( MMDOLO )
    $(
    IF ( .OK -= OSMMAP( .LOADMODULE, MODULE ) ) GOTO 9800
    IF ( TRACE )
        CALL PUTF( FDRUNO, "AP trace: module loaded%N" )
    MMPHYS( 1 ) = 0
    MMPHYS( 2 ) = 0
    $)

#
#           GET THE NAME OF THE MODULE AND ITS OPTIONS.
#           ALLOW TWO ILLEGAL ADDRESS EXCEPTIONS.
#
CALL MMSEND( RESET, .OPTEND, .MAPINTMODE,
            .MAPWORKBASE, IEV, %9900 )
MMSADR = .MAPBEGIN
MMILAT = 2
CALL MMEXEC( .NOWAIT, IEV, %9900 )
CALL MMWAIT( 5, IEV, %9900 )

#
#           CLEAR MMILAT IF IT WAS > 0, ENABLING ADDRESS
#           EXCEPTIONS FROM THE MINIMAP. IF LESS THAN TWO,
#           THEN THE MMINTR ROUTINE TRAPPED AN ILLEGAL
#           ADDRESS.
#
IF ( MMILAT == 2 )
    MMILAT = 0
ELSE
    GOTO 9030

#
#           CHECK THE OPTIONS TABLE.
#
CALL MMRECV( MOPTS, .OPTMAX, .MAPINTMODE,
            .MAPWORKBASE, IEV, %9900 )
DO I = 1, .OPTMAX
    IF ( MOPTS( I ) >= 0 )
        CASE ( I )
            $(

#
#           .OPTTIMER OPTION - SET A TIME LIMIT ON THE
#           REAL TIME NEEDED FOR THE MODULE TO COMPLETE
#           EXECUTION AND GENERATE AN INTERRUPT. THE
#           VALUE IS STORED IN MMTIME, AND INITIALIZED
#           BY THE MMINIT ROUTINE.
#
            CONDITION .OPTTIMER:
            IF ( MOPTS( .OPTTIMER ) > 0 )
                $(
                    IF ( .OK -= OSMMAP( .INTMODE,

```

```

MOPTS( .OPTTIMER ),
MMTIME ) )

GOTO 9800
$)
ELSE
    MMTIME = .LARGEINTEGER

#
# .OPTSIZE OPTION - SET A LIMIT ON THE NUMBER OF
# POINTS THAT MAY BE PROCESSED AT ONE TIME. THIS
# TRANSLATES TO A MAXIMUM ARRAY SIZE THAT MAY BE
# SPECIFIED BY THE GETWPM ROUTINE.
#

    CONDITION .OPTSIZE:
    MMAXP = 0
    IF ( MOPTS( .OPTSIZE ) > 0 )
        $(
            IF ( .OK -= OSMMAP( .INTMODE, MOPTS( .OPTSIZE ),
                MMAXP ) )

GOTO 9800
            IF ( TRACE )
                CALL PUTF( FDRUNO,
                    "AP trace: maximum points = %I%N",
                    MMAXP )

            $)

#
# .OPTDEBUG OPTION - ACTIVATE THE DEBUG MODE.
#

    CONDITION .OPTDEBUG:
    MMDEBUG = .DEBUGACTIVE
    IF ( TRACE )
        CALL PUTF( FDRUNO,
            "AP trace: module uses debug mode%N" )

#
# ADD NEW OPTIONS BEFORE THIS COMMENT.
#

    OTHERWISE:
    GOTO 9020
    $)

#
# IF TRACE IS ON, DISPLAY THE MODULE NAME.
# IF MMDOLO CLEAR, RETURN THE MODULE NAME.
#

IF ( TRACE ! ( - MMDOLO ) )
    $(
        CALL MMRECV( MNAME, .OPTNAMELEN, .MAPINTMODE,
            .OPTNAME + .MAPWORKBASE - 1, IEV, %9900 )
        FOR ( I = 1; I < 7; I = I + 1 )
            IF ( MNAME( I ) == 0 )
                BREAK
        ELSE
            IF ( .OK -= OSMMAP( .CHARMODE, MNAME( I ),

```

```

NAME( I ) ) ) GOTO 9800
NAME( I ) = .EOS
IF ( TRACE )
    CALL PUTF( FDRUNO, "AP trace: module name = %S%N",
              NAME )
IF ( - MMDOLO )
    CALL STRCPY( NAME, MODULE, 1 )
$)

#
#     SET THE STARTING ADDRESS (MMSADR) FOR
#     CONTINUOUS EXECUTION AND RETURN.
#
CALL MMWAIT( MMTIME, IEV, %9900 )
MMSADR = .MAPCONTINUE
CHECKTIMER
CALL PPOP
CHECKTIMER
RETURN

#
#     ERROR CONDITIONS:
#
#     MMINIT ROUTINE HAS NOT BEEN EXECUTED YET.
#
9000 CONTINUE
IEV = -2023
GOTO 9900

#
#     MINIMAP INITIALIZATION HAS FAILED.
#
9010 CONTINUE
IEV = -2024
GOTO 9900

#
#     OPTION SPECIFIED IN UNDESIGNATED SLOT - REPORT
#     AS UNKNOWN OPTION.
#
9020 CONTINUE
IEV = -2025
GOTO 9900

#
#     MMINTR TRAPPED AN ILLEGAL ADDRESS EXCEPTION.
#
9030 CONTINUE
IEV = -2045
MMILAT = 0
GOTO 9900

#
#     ERROR OCCURRED IN LOWER ROUTINE - RETRIEVE IT.
#
9800 CONTINUE
IEV = OSGIEV( IEV )

```



```
#
#           CLEAR MMREDY AND TAKE THE ALTERNATE RETURN.
#
9900 CONTINUE
MMREDY = .FALSE.
CHECKTIMER
RETURN 1
#
#
END
```

A.4 MMEEXEC Routine

#--MMEEXEC EXECUTE LOAD MODULES FOR MINIMAP SUPPORT SOFTWARE

#

IDENTIFICATION

#

#	TITLE	MMEEXEC
#	AUTHOR	GREGG R FABREGAS
#	VERSION	A.01
#	DATE	30-JUL-1984
#	LANGUAGE	RATFOR
#	SYSTEM	VAX 11/780

#

#

UPDATE

#

#	UPDATE	# 1
#	AUTHOR	GREGG R FABREGAS
#	DATE	06-OCT-1985
#	VERSION	A.01
#	PURPOSE	ADD CALLS TO STARTTIMER AND CHECKTIMER TO MONITOR ELAPSED CPU TIME.

#

#	UPDATE	# 2
#	AUTHOR	GREGG R FABREGAS
#	DATE	12-NOV-1985
#	VERSION	A.01
#	PURPOSE	(1) ADD .NOWTARR OPTION TO SUPPRESS ARRAY TRANSFERS AT MINI-MAP TRACEPOINTS, AND (2) ADD MMBRAK ENTRY POINT TO DEFINE A BREAKPOINT FOR EASIER DYNAMIC DEBUBBING.

#

#	UPDATE	# 3
#	AUTHOR	GREGG R FABREGAS
#	DATE	06-MAR-1986
#	VERSION	A.01
#	PURPOSE	CORRECT ALIGNMENT OF SCALAR TRACES AND HEADINGS.

#

PURPOSE

#

THIS MODULE STARTS EXECUTION
OF THE MINIMAP PROVIDED IT HAS A
GIPSY-COMPATIBLE MODULE LOADED.

```

# THE WAIT ARGUMENT IS USED FOR
# SYNCHRONIZING HOST ACTIVITY
# WITH THE MINIMAP. IF THE WAIT
# ARGUMENT HAS THE VALUE ".WAIT",
# THEN CONTROL RETURNS TO THE
# CALLING ROUTINE AFTER THE
# MINIMAP HALTS, OTHERWISE CONTROL
# RETURNS IMMEDIATELY. THE DEBUG
# MODULE OPTION IS HANDLED BY
# THIS ROUTINE. SETTING A BREAKPOINT
# AT THE MMBRAK ENTRY POINT
# ENABLES THE PROGRAMMER TO EXAMINE
# THE STATE OF THE HOST AND
# MINI-MAP USING AN ON-LINE DYNAMIC DEBUGGER.
#
# ENTRY POINT
#
# CALL MMEEXEC( WAIT, IEV, ERRET )
# CALL MMBRAK
#
# ARGUMENT LISTING
#
# WAIT INTEGER DETERMINES WHETHER THIS
# ROUTINE SHOULD
# WAIT FOR THE MINIMAP
# TO STOP OR RETURN
# IMMEDIATELY. VALUES
# ARE ".WAIT",
# ".NOWAIT", AND ".NOWTARR".
# THE ".WAIT"
# AND ".NOWAIT" OPTIONS
# ENABLE THE TRANSFER
# OF ALL MINI-MAP ARRAYS
# WHENEVER DEBUGGING
# IS ENABLED AND A TRACEPOINT
# IS ENCOUNTERED.
# THE ".NOWTARR" OPTION
# DISABLES THE
# TRANSFER AND, OTHERWISE,
# BEHAVES AS THE
# ".NOWAIT" OPTION.
#
# IEV INTEGER INTEGER EVENT VARIABLE.
# THE FOLLOWING
# CODES ARE RETURNED WHEN DONE:
# .OK IF THE CALL SUCCEEDS,
# -2023 IF THERE IS NO PRIOR
# CALL TO THE
# MMINIT ROUTINE,
# -2024 IF THE MMINIT ROUTINE
# FAILED,
# -2030 IF THE MINIMAP IS NOT

```

```

#          READY FOR EXECUTION,
#          -2031 IF THE WAIT OPTION
#          IS NOT ".WAIT",
#          ".NOWAIT",
#          OR ".NOWTARR".
#          -2036 IF THE CALL TO
#          STRTMM HAS FAILED,
#          -2044 IF MMEXEC IS CALLED
#          TWICE BEFORE
#          THE MMWAIT ROUTINE
#          IS CALLED, AND
#          -2045 IF THE MINIMAP DETECTED
#          AN ILLEGAL ADDRESS.
#          ERRET    LABEL    LABEL IN CALLING ROUTINE FOR
#                               ALTERNATE ERROR RETURNS.
#
# INCLUDE FILES/COMMONS
#
#          GIPSY    INCLUDE    GIPSY GENERAL SYMBOL DEF'NS
#          GIPCOM    COMMON    GIPSY COMMAND LINE COMMON DATA
#          TTCOM     COMMON    FD'S FOR TERMINAL & RUNFILE IO
#          MINIMAP   COMMON    COMMON FOR THE ADMINISTRATION
#                               OF THE MINIMAP
#                               SUPPORT SOFTWARE.
#
# HARDWARE REQUIRED
#
#          A CSPI MINIMAP ARRAY PROCESSOR MUST BE OPENED AND
#          LOADED WITH A GIPSY-COMPATIBLE LOAD MODULE.
#
# LANGUAGE FEATURES REQUIRED
#
#          ALTERNATE RETURNS ARE NEEDED.
#
# PROGRAM ENVIROMENT
#
#          ANY NUMBER OF CALLS TO THE ROUTINE
#          MAY FOLLOW A SUCCESSFUL CALL
#          TO THE MMINIT AND MMLOAD ROUTINES,
#          PROVIDED THE PROGRAMMER
#          (1) USES THE .WAIT OPTION, OR
#          (2) ISSUES A CALL TO MMWAIT PRIOR
#          TO THE NEXT CALL TO THE MMEXEC
#          ROUTINE.  MMLOAD IS CALLED
#          INSIDE THE MMINIT ROUTINE.
#
# ALGORITHM
#
#          (1) IF NEEDED, OPEN THE MODULE SYMBOL
#          FILE, IF ONE EXISTS,
#          (2) UNTIL A BREAKPOINT IS REACHED

```

```

#           WHERE THE DEBUGGING SEQUENCE
#           PARAMETER IS UNCHANGED,
#           CYCLE THROUGH SYNCHRONIZED
#           EXECUTIONS ON THE MINIMAP,
#           (3) IF TRACE DEBUGGING, REPORT BREAKPOINTS, AND
#           (4) IF DEBUGGING AND ENABLED, RETRIEVE ALL
#           ARRAYS ON EACH BREAKPOINT.

```

```

# ROUTINES CALLED

```

```

#           PPUSH           PUSH PROGRAM NAME ONTO ERROR STACK
#                           (GIPSY PRIMITIVE)
#           PUTF           MACHINE INDEPENDENT FORMATTED OUTPUT
#                           (GIPSY PRIMITIVE)
#           OSFNAM        CONSTRUCT SYSTEM DEPENDENT FILE
#                           NAME. (GIPSY OS KERNEL)
#           OSINFD        OPEN FILE DESCRIPTOR GIVEN FILE.
#                           (GIPSY OS KERNEL)
#           RDKINL        INITIALIZE AND ACCESS A STANDARD
#                           IMAGE FILE. (GIPSY PRIMITIVE)
#           OSSMAP        MACHINE-DEPENDENT CODE FOR
#                           MINIMAP SUPPORT SOFTWARE.
#                           (GIPSY OS KERNEL)
#           OSTIME        GET CURRENT TIME. (GIPSY OS KERNEL)
#           STRTMM        HSL ROUTINE: START MINIMAP.
#                           (GIPSY LIBRARY)
#           MMWAIT        WAIT FOR MINIMAP FOR THE MINIMAP SUPPORT
#                           SOFTWARE. (GIPSY PRIMITIVE)
#           ICCVT         CONVERT INTEGER TO CHARACTERS.
#                           (GIPSY PRIMITIVE)
#           BTRSCH        B-TREE SEARCH FOR TEXT GIVEN KEY.
#                           (GIPSY PRIMITIVE)
#           PUTEOL        PUT END OF LINE TO OUTPUT.
#                           (GIPSY PRIMITIVE)
#           MMRCVA        RETRIEVE ARRAY FROM MINIMAP FOR
#                           MINIMAP SUPPORT SOFTWARE.
#                           (GIPSY PRIMITIVE)
#           CLOSE        CLOSE A FILE. (GIPSY PRIMITIVE)
#           PPOP         POP PROGRAM NAME FROM THE ERROR STACK
#                           (GIPSY PRIMITIVE)

```

```

# REMARKS

```

```

#           THE STARTING ADDRESS AND EXECUTION
#           TIME-LIMIT ARE RETRIEVED FROM
#           THE MINIMAP ADMINISTRATION COMMON,
#           AND ARE EXPECTED TO ALREADY
#           CONTAIN CONSISTENT VALUES.
#           IF THE TIME-LIMIT IS EXCEEDED, THE
#           MMTOST FLAG WILL BE SET BY THE MMINTR ROUTINE.
#

```

```

#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
#   INCLUDE GIPSY
#
#   SUBROUTINE MMEXEC( WAIT, IEV, * )
#
#
#   INCLUDE GIPCOM
#   INCLUDE TTCOM
#   INCLUDE MINIMAP
#
#   INTEGER WAIT, IEV, IDENT( .IDLENGTH ), PREV( 22 ), I,
#   INTEGER STATUS( 22 ), NZEROS, J, ITEM( 20 ), COUNT
#   LOGICAL TRACE, DEBUG, DTRACE
#   CHARACTER FFSYMB( .FILENAMELENGTH ), FDSYMB( .FDLENGTH )
#   CHARACTER INSTYP( 60 ), JUNK, ZERO, LABELI( 16 )
#   CHARACTER LABELO( 12 ), LABELN( 12 ), DIGITS( 5 )
#   CHARACTER LABELR( 13 )
#   PCHARACTER PNAME( .MMNLEN )
#   MAPINTEGER MSTAT( 22 )
#   MAPREAL MRSTAT( 20 )
#   REAL RPREV( 20 ), RSTAT( 20 )
#   INTEGER OSFNAM, OSINFD, OSMMAP, OSSIEV, OSGIEV, OSPACK
#   REAL AMINI, AMAXI
#
#   EQUIVALENCE ( TRACE, SFLAG( .A ) ), ( DTRACE, SFLAG( .D ) )
#
#   DATA ZERO / '0' /
#   DATA LABELI / EVAL( "      INTEGER " ) /
#   DATA LABELR / EVAL( "      REAL " ) /
#   DATA LABELO / EVAL( "      OLD = " ) /
#   DATA LABELN / EVAL( "      NEW = " ) /
#
#           PUSH PROGRAM NAME ONTO ERROR TRACE STACK.
#
#   CALL PPUSH( "MMEXEC" )
#   IF ( TRACE )
#       $(
#       CALL PUTF( FDRUNO, "AP trace: MMEXEC called%N" )
#       STARTTIMER
#       $)
#
#           THE MINIMAP MUST BE READY FOR EXECUTION PRIOR
#           TO THE CALL TO THIS ROUTINE.  VERIFY THE WAIT
#           OPTION.
#
#   IF ( - MMCHKD )
#       GOTO 9000
#   IF ( - MMACTV )

```

```

GOTO 9010
IF ( - MMREDY )
GOTO 9020
IF ( WAIT -= .WAIT & WAIT -= .NOWAIT & WAIT -= .NOWTARR )
GOTO 9030
IF ( MMEXCD )
GOTO 9040
#
#           FOR DEBUG MODE, PICK UP THE INITIAL SETTINGS
#           OF THE DEBUG VARIABLES, AND SAVE THEM FOR
#           LATER COMPARISON.
#
IF ( MMDEBUG == .DEBUGACTIVE )
$(
CALL MMRECV( MSTAT, 2, .MAPINTMODE,
             .MAPSCALARS, IEV, %9900 )
DO I = 1, 2
  IF ( .OK -= OSMMAP( .INTMODE, MSTAT( I ),
                    STATUS( I ) ) )
    GOTO 9800
#
#           IF TRACE ON AND THE MODULE IS IN ACTIVE DEBUG
#           MODE, OPEN THE SYMBOL FILE THAT INTERPRETS THE
#           INSTRUCTION TYPE CODES, IF IT EXISTS.
#
IF ( DTRACE )
$(
IF ( .OK == OSFNAM( .USRMSG, FFSYMB, MMNAME ) )
  IF ( .OK == OSINFD( FFSYMB, FDSYMB ) )
    $(
CALL RDKINL( FDSYMB, IDENT, .READONLY, IEV, %1000 )
MMDEBUG = .DEBUGOPENFILE
1000 CONTINUE
CALL PRESET( "MMEXEC" )
$)
#
#           RESET THE SETTINGS OF THE INTERNAL SCALARS,
#           EXCEPT THE DEBUGGING PARAMETERS.
#
CALL MMRECV( MSTAT( 3 ), 20, .MAPINTMODE,
             .MAPSCALARS + 2, IEV, %9900 )
CALL MMRECV( MRSTAT, 20, .MAPREALMODE,
             .MAPSCALARS + 22, IEV, %9900 )
$)
$)
#
#           THE REPEAT LOOP IS EXECUTED ONCE TO START THE
#           MINIMAP IF DEBUG MODE IS INACTIVE, AND SEVERAL
#           TIMES, OTHERWISE.
#
DEBUG = ( MMDEBUG == .DEBUGACTIVE ) !

```

```

                ( MMDEBUG == .DEBUGOPENFILE )
REPEAT
  $(
    IF ( DEBUG )
      MMILAT = 2
#
#           FLAG THE EXECUTION OF THIS ROUTINE AND CLEAR
#           THE SETTING OF MMTOST FOR A POSSIBLE DEVICE
#           TIME-OUT.  MMINTR MAY SET THIS FLAG.
#
    MMEXCD = .TRUE.
    MMTOST = .FALSE.
    IF ( .OK ~= OOSTIME( MMSECS, 1 ) )
      MMSECS = 0.0
    MMSECS = MMSECS + MMTIME
#
#           START THE MINIMAP USING THE CURRENT SETTING
#           OF THE STARTING ADDRESS.  THIS SHOULD BE THE
#           ONLY PLACE FOR A CALL TO STRTMM.
#
    IEV = .OK
    CALL STRTMM( MMSADR, IEV )
    IF ( .OK ~= OSSIEV( IEV, "STRTMM called" ) )
      GOTO 9050
    IF ( TRACE )
      CALL PUTF( FDRUNO, "AP trace: Minimap started%N" )
#
#           FOR DEBUG MODE, PICK UP THE DEBUGGING
#           PARAMETERS AND COMPARE THEM TO THE PREVIOUS
#           SETTINGS.  EXIT THE DEBUG LOOP IF THEY ARE
#           UNCHANGED.
#
    IF ( DEBUG )
      $(
        CALL MMWAIT( MMTIME, IEV, %9900 )
        IF ( DTRACE )
          IF ( MMILAT < 2 )
            CALL PUTF( FDRUNO,
              "AP debug: illegal address detected%N" )
        CALL MMRECV( MSTAT, 2, .MAPINTMODE,
          .MAPSCALARS, IEV, %9900 )
        DO I = 1, 2
          $(
            PREV( I ) = STATUS( I )
            IF ( .OK ~= OSMMAP( .INTMODE, MSTAT( I ),
              STATUS( I ) ) ) GOTO 9800
          $)
        DEBUG = ( STATUS( 1 ) ~= PREV( 1 ) !
          STATUS( 2 ) ~= PREV( 2 ) )
#
#           IF TRACE ON AND THE DEBUGGING PARAMETERS ARE

```



```

#           CHANGED, DISPLAY THE SEQUENCE NUMBER.  FOR THE
#           INSTRUCTION TYPE, DISPLAY TEXT IF THE SYMBOL
#           TABLE IS OPEN, OTHERWISE USE THE CODE.
#
IF ( DEBUG )
  $(
    IF ( DTRACE )
      $(
        IEV = .OK + 1
        IF ( STATUS( 1 ) == -12345 &
            MMDBUG == .DEBUGOPENFILE )
          $(
            CALL ICCVT( STATUS( 2 ), DIGITS, 5, NZEROS )
            NZEROS = NZEROS - 1
            IF ( NZEROS > 0 )
              DO I = 1, NZEROS
                DIGITS( I ) = ZERO
            CALL BTRSCH( FDSYMB, IDENT, DIGITS, 5, INSTYP,
                       60, JUNK, IEV, %2000 )
            2000 CONTINUE
          $(
        IF ( IEV == .OK )
          CALL PUTF( FDRUNO, "AP debug: at %60A%N",
                   INSTYP )
        ELSE
          CALL PUTF( FDRUNO,
                   "AP debug: step ( %I , %I )%N",
                   STATUS( 1 ), STATUS( 2 ) )
#
#           RETRIEVE THE OTHER SCALARS.
#
CALL MMRECV( MSTAT( 3 ), 20, .MAPINTMODE,
             .MAPSCALARS + 2, IEV, %9900 )
CALL MMRECV( MRSTAT, 20, .MAPREALMODE,
             .MAPSCALARS + 22, IEV, %9900 )
#
#           EXAMINE CHANGES IN MINIMAP-LOCAL INTEGERS.
#           LOAD ITEM LIST WITH SUBSCRIPTS OF MODIFIED
#           INTEGERS.
#
J = 1
DO I = 3, 22
  $(
    PREV( I ) = STATUS( I )
    IF ( .OK -= OSMMAP( .INTMODE, MSTAT( I ),
                      STATUS( I ) ) ) GOTO
9800
    IF ( PREV( I ) -= STATUS( I ) )
      $(
        ITEM( J ) = I
        J = J + 1

```

```

                                $)
                                $)
#
#   IF ANY INTEGERS HAVE BEEN MODIFIED, REPORT THE
#   OLD AND NEW VALUES.
#
IF ( J > 1 )
  $(
  ITEM( J ) = 0
  COUNT = ( J - 2 ) / 4
  DO I = 0, COUNT
    $(
    CALL PUTF( FDRUNO,
              "AP debug: integer(s) modified:%N      " )
    DO J = 1, 4
      IF ( ITEM( 4 * I + J ) /= 0 )
        CALL PUTF( FDRUNO, "%S%2I", LABELI,
                  ITEM( 4 * I + J ) - 2 )
      ELSE
        BREAK
    CALL PUTF( FDRUNO, "%N      " )
    DO J = 1, 4
      IF ( ITEM( 4 * I + J ) /= 0 )
        CALL PUTF( FDRUNO, "%S%6I", LABELO,
                  PREV( ITEM( 4 * I + J ) ) )
      ELSE
        BREAK
    CALL PUTF( FDRUNO, "%N      " )
    DO J = 1, 4
      IF ( ITEM( 4 * I + J ) /= 0 )
        CALL PUTF( FDRUNO, "%S%6I", LABELN,
                  STATUS( ITEM( 4 * I + J ) ) )
      ELSE
        BREAK
    CALL PUTEOL( FDRUNO )
    $)
  $)
#
#   EXAMINE CHANGES IN MINIMAP-LOCAL REALS, USING
#   THE SAME TECHNIQUE AS WITH INTEGERS.
#
J = 1
DO I = 1, 20
  $(
  RPREV( I ) = RSTAT( I )
  IF ( .OK /= OSMMAP( .REALMODE, MRSTAT( I ),
                    RSTAT( I ) ) ) GOTO 9800
  IF ( RPREV( I ) /= RSTAT( I ) )
  $(
  ITEM( J ) = I
  J = J + 1

```

```

    $)
    $)
#
#     IF ANY REALS HAVE BEEN MODIFIED, REPORT THE OLD
#     AND NEW VALUES.
#
IF ( J > 1 )
    $(
        ITEM( J ) = 0
        COUNT = ( J - 2 ) / 3
        DO I = 0, COUNT
    $(
        CALL PUTF( FDRUNO,
                  "AP debug: real(s) modified:%N      " )
        DO J = 1, 3
            IF ( ITEM( 3 * I + J ) /= 0 )
                CALL PUTF( FDRUNO, "%S%2I      ", LABELR,
                          ITEM( 3 * I + J ) )
            ELSE
                BREAK
        CALL PUTF( FDRUNO, "%N      " )
        DO J = 1, 3
            IF ( ITEM( 3 * I + J ) /= 0 )
                CALL PUTF( FDRUNO, "%S%12.5R", LABELO,
                          AMIN1( AMAX1( RPREV( ITEM(
                            3 * I + J ) ), - .LARGEREAL / 10. ),
                            .LARGEREAL / 10.0 ) )
            ELSE
                BREAK
        CALL PUTF( FDRUNO, "%N      " )
        DO J = 1, 3
            IF ( ITEM( 3 * I + J ) /= 0 )
                CALL PUTF( FDRUNO, "%S%12.5R", LABELN,
                          AMIN1( AMAX1( RSTAT( ITEM(
                            3 * I + J ) ), - .LARGEREAL / 10. ),
                            .LARGEREAL / 10.0 ) )
            ELSE
                BREAK
        CALL PUTEOL( FDRUNO )
    $)
    $)
$)
#
#     DEBUG MODE FORCES ALL ARRAYS TO BE TRANSFERRED
#     TO THE HOST MEMORY AT EACH BREAKPOINT, UNLESS
#     DISABLED BY THE .NOWTARR OPTION.
#
    IF ( MMACNT > 0 & WAIT /= .NOWTARR )
DO I = 1, MMACNT
    $(
        IF ( .OK /= OSPACK( PNAME, MMHNAME( 1, I ), 1,

```

```

                                .MMNLEN, -1 ) ) GOTO 9800
    CALL MMRCVA( PNAME, IEV, %9900 )
    $)
    CALL MMBRAK
    $)
#
#       DETERMINE WHETHER MMINTR TRAPPED ONE OR TWO
#       ILLEGAL ADDRESS EXCEPTIONS.
#
    IF ( MMILAT < 2 )
        GOTO 9060
    MMILAT = 0
    $)
    $)
    UNTIL ( ~ DEBUG )
#
#       IF THE WAIT OPTION IS SPECIFIED, WAIT HERE,
#       USING THE TIMER SETTING IN THE MINIMAP COMMON.
#
    IF ( WAIT == .WAIT )
        CALL MMWAIT( MMTIME, IEV, %9900 )
#
#       CLOSE THE MODULE SYMBOL TABLE.
#
    IF ( MMDEBUG == .DEBUGOPENFILE )
        $(
        CALL CLOSE( FDSYMB )
        MMDEBUG = .DEBUGACTIVE
        $)
#
#       DONE. MINIMAP HAS EITHER EXECUTED A STOP OR
#       PAUSE INSTRUCTION.
#
    CHECKTIMER
    CALL PPOP
    RETURN
#
#       ERROR CONDITIONS:
#
#       MMINIT ROUTINE HAS NOT BEEN EXECUTED YET.
#
    9000 CONTINUE
    IEV = -2023
    GOTO 9900
#
#       MINIMAP INITIALIZATION HAS FAILED.
#
    9010 CONTINUE
    IEV = -2024
    GOTO 9900
#

```

```

#           MINIMAP NOT READY FOR EXECUTION.
#
9020 CONTINUE
IEV = -2030
GOTO 9900
#
#           WAIT OPTION IS NOT .WAIT OR .NOWAIT.
#
9030 CONTINUE
IEV = -3031
GOTO 9900
#
#           MMEXEC HAS ALREADY BEEN CALLED.
#
9040 CONTINUE
IEV = -2044
GOTO 9900
#
#           CALL TO STRTMM FAILED.
#
9050 CONTINUE
IEV = -2036
GOTO 9900
#
#           MINIMAP DETECTED AN ILLEGAL ADDRESS.
#
9060 CONTINUE
IEV = -2045
MMILAT = 0
GOTO 9900
#
#           RETRIEVE ERROR FROM LOWER ROUTINE.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#
#           TAKE THE ALTERNATE RETURN.
#
9900 CONTINUE
IF ( MMDEBUG == .DEBUGOPENFILE )
    CALL CLOSE( FDSYMB )
CHECKTIMER
RETURN 1
#
#
END
#
#           MMBRAK IS THE HOST BREAKPOINT FOR HOST
#           MONITORING OF MINI-MAP TRACE POINTS.
#
SUBROUTINE MMBRAK

```

RETURN
END

A.5 MMWAIT Routine

```
#--MMWAIT  WAIT FOR MINIMAP FOR MINIMAP SUPPORT SOFTWARE
#
#  IDENTIFICATION
#
#      TITLE           MMWAIT
#      AUTHOR          GREGG R FABREGAS
#      VERSION         A.01
#      DATE            31-JUL-1984
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#
#  UPDATE
#
#      UPDATE          # 1
#      AUTHOR          GREGG R FABREGAS
#      DATE            07-OCT-1985
#      VERSION         A.01
#      PURPOSE         ADD CALLS TO STARTTIMER
#                     AND CHECKTIMER TO
#                     MONITOR ELAPSED CPU TIME.
#
#  PURPOSE
#
#      ONCE THE MINIMAP HAS STARTED
#      EXECUTION, THE STATE OF THE
#      MINIMAP IS CERTAIN ONLY WHEN
#      SYNCHRONIZATION IS RE-ESTABLISHED.
#      THIS ROUTINE GUARANTEES THAT
#      SYNCHRONIZATION AND RESPONDS TO
#      OCCURRENCES OF MINIMAP TIME-OUT.
#
#  ENTRY POINT
#
#      CALL MMWAIT( TIMER, IEV, ERRET )
#
#  ARGUMENT LISTING
#
#      TIMER  INTEGER          MAXIMUM TIME, IN SECONDS,
#                               THAT THE MINIMAP MAY EXECUTE
#                               BEFORE A TIME-OUT
#                               ERROR IS GENERATED.
#
#      IEV    INTEGER          INTEGER EVENT VARIABLE.
#                               THE FOLLOWING
#                               CODES ARE RETURNED WHEN DONE:
#                               .OK  IF THE CALL SUCCEEDS,
#                               -2023 IF THE MMINIT ROUTINE
#                               HAS NOT BEEN
```

```

#                                     CALLED YET,
#                                     -2024 IF THE MMINIT ROUTINE
#                                     HAS FAILED,
#                                     -2026 IF THE TIMER OPTION
#                                     IS OUT OF BOUNDS,
#                                     -2030 IF THE MINIMAP IS NOT
#                                     READY FOR EXECUTION,
#                                     -2037 IF THE WAIT FAILED,
#                                     -2041 IF UNABLE TO SET THE
#                                     TIME-OUT INTERVAL, AND
#                                     -2045 IF THE MINIMAP
#                                     TIMED-OUT.
#           ERRET      LABEL          LABEL IN CALLING ROUTINE FOR
#                                     ALTERNATE ERROR RETURN.
#
# INCLUDE FILES/COMMONS
#
#           GIPSY      INCLUDE      GIPSY GENERAL SYMBOL DEF'NS
#           GIPCOM     COMMON       GIPSY COMMAND LINE COMMON DATA
#           TTCOM      COMMON       FD'S FOR TERMINAL & RUNFILE IO
#           MINIMAP    COMMON       COMMON FOR THE ADMINISTRATION
#                                     OF THE MINIMAP
#                                     SUPPORT SOFTWARE.
#
# HARDWARE REQUIRED
#
#           A CSPI MINIMAP ARRAY PROCESSOR
#           MUST BE OPENED AND LOADED WITH A
#           GIPSY-COMPATIBLE LOAD MODULE.
#
# LANGUAGE FEATURES REQUIRED
#
#           ALTERNATE RETURNS ARE NEEDED.
#
# PROGRAM ENVIROMENT
#
#           THIS ROUTINE MAY BE EXECUTED
#           ANY NUMBER OF TIMES AFTER A
#           SUCCESSFUL CALL TO MMINIT AND
#           MMLOAD.  MMLOAD IS CALLED FROM
#           INSIDE MMINIT.  THIS ROUTINE
#           IS INTENDED TO BE CALLED AFTER A
#           CALL TO MMEXEC, WHICH REQUIRES
#           A CALL TO THIS ROUTINE BEFORE IT
#           MAY BE CALLED AGAIN.
#
# ALGORITHM
#
#           (1) SET THE TIME-OUT INTERVAL
#               ACCORDING TO THE INPUT ARGUMENT,
#           (2) WAIT FOR THE MINIMAP TO

```



```

#           TERMINATE PROCESSING IF THE MMEXEC
#           ROUTINE HAS BEEN CALLED BEFORE
#           THE PREVIOUS CALL TO THE
#           MMWAIT ROUTINE, AND
#           (3) CLEAR THE EXECUTION FLAG AND
#           CHECK THE TIME-OUT FLAG.
#
# DATA FORMAT
#
#           THE TIMER IS MEASURED IN SECONDS.
#           THE MAXIMUM TIME-OUT INTERVAL
#           IS 65535 SECONDS.
#
# LIMITATIONS
#
#           THE TIME-OUT STATUS IS SET
#           BY A CONDITION-HANDLER. IF THE
#           CONDITION-HANDLER IS NOT IMPLEMENTED,
#           THERE IS NO WAY TO TRAP,
#           AND ACT ON, THE TIME-OUT CONDITION.
#           ALSO, THE CONDITION-HANDLER
#           IS RESPONSIBLE FOR DISMISSING THE WAIT STATE.
#
# ROUTINES CALLED
#
#           PPUSH           PUSH PROGRAM NAME ONTO ERROR STACK
#                           (GIPSY PRIMITIVE)
#           PUTF           MACHINE INDEPENDENT FORMATTED OUTPUT
#                           (GIPSY PRIMITIVE)
#           STTOMM        HSL ROUTINE: SET TIME-OUT INTERVAL
#                           FOR MINIMAP. (GIPSY LIBRARY)
#           OSSIEV        PUT EVENT VARIABLE (IEV) INTO SAVE
#                           AREA (GIPSY OS KERNEL)
#           WAITMM        HSL ROUTINE: WAIT FOR MINIMAP.
#                           (GIPSY LIBRARY)
#           PPOP           POP PROGRAM NAME FROM THE ERROR STACK
#                           (GIPSY PRIMITIVE)
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
# INCLUDE GIPSY
#
# SUBROUTINE MMWAIT( TIMER, IEV, * )
#
#
# INCLUDE GIPCOM
# INCLUDE TTCOM
# INCLUDE MINIMAP
#

```

```

INTEGER TIMER, IEV
INTEGER OSSIEV
LOGICAL TRACE
#
EQUIVALENCE ( TRACE, SFLAG( .A ) )
#
#           PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
CALL PPUSH( "MMWAIT" )
IF ( TRACE )
    $(
    CALL PUTF( FDRUNO, "AP trace: MMWAIT called%N" )
    STARTTIMER
    $)
#
#           MINIMAP MUST BE READY FOR EXECUTION BEFORE
#           THIS ROUTINE CAN BE CALLED.  THE TIMER
#           INTERVAL MUST BE POSITIVE OR ZERO.
#
IF ( - MMCHKD )
    GOTO 9000
IF ( - MMACTV )
    GOTO 9010
IF ( - MMREDY )
    GOTO 9020
IF ( TIMER < 0 ! TIMER > 65535 )
    GOTO 9030
#
#           SET THE TIME-OUT INTERVAL.  IF ZERO, THE
#           INTERVAL WILL BE SET TO THE MAXIMUM ALLOWED.
#
IF ( TIMER == 0 )
    TIMER = 65535
MMSECS = MMSECS - MMTIME + TIMER - 1
IEV = .OK
IFDEF VAXVMS
    CALL STTOMM( TIMER, IEV )
    IF ( .OK /= OSSIEV( IEV, "STTOMM called" ) )
        IF ( IEV == 113 )
            GOTO 9030
        ELSE
            GOTO 9040
    IF ( TRACE )
        CALL PUTF( FDRUNO,
            "AP trace: set time-out interval to %I seconds%N",
            TIMER )
ENDIF
#
#           WAIT FOR THE MINIMAP TO FINISH.
#           IF THE TIMER EXPIRES BEFORE THE WAIT RETURNS,
#           THIS ROUTINE WILL BE INTERRUPTED.  THIS SHOULD

```

```

#           BE THE ONLY PLACE WHERE WAITMM IS CALLED.
#
#   IF ( MMEXCD )
#       $(
#           CALL WAITMM( IEV )
#           IF ( .OK -= OSSIEV( IEV, "WAITMM called" ) )
#               GOTO 9050
#           IF ( TRACE )
#               CALL PUTF( FDRUNO, "AP trace: wait succeeded%N" )
#       $)
#
#           CLEAR THE EXECUTION FLAG AND CHECK THE TIME-OUT
#           FLAG TO DETERMINE WHETHER A TIME-OUT HAS
#           OCCURRED.
#
#   MMEXCD = .FALSE.
#   IF ( MMTOST )
#       GOTO 9060
#
#           DONE.
#
#   CHECKTIMER
#   CALL PPOP
#   RETURN
#
#           ERROR CONDITIONS:
#
#           MMINIT ROUTINE HAS NOT BEEN EXECUTED YET.
#
#   9000 CONTINUE
#   IEV = -2023
#   GOTO 9900
#
#           MINIMAP INITIALIZATION HAS FAILED.
#
#   9010 CONTINUE
#   IEV = -2024
#   GOTO 9900
#
#           MINIMAP NOT READY FOR EXECUTION.
#
#   9020 CONTINUE
#   IEV = -2030
#   GOTO 9900
#
#           TIMER REQUEST OUT OF BOUNDS.
#
#   9030 CONTINUE
#   IEV = -2026
#   GOTO 9900
#

```

```
#          CALL TO STTOMM FAILED.
#
9040 CONTINUE
IEV = -2041
GOTO 9900
#
#          CALL TO WAITMM FAILED.
#
9050 CONTINUE
IEV = -2037
GOTO 9900
#
#          MINIMAP TIMED-OUT.
#
9060 CONTINUE
IEV = -2045
MMTOST = .FALSE.
#
#          TAKE THE ALTERNATE RETURN.
#
9900 CONTINUE
CHECKTIMER
RETURN 1
#
#
END
```

A.6 MMSKIP Routine

```
#--MMSKIP SKIP MINIMAP CONTROL FOR MINIMAP SUPPORT SOFTWARE
#
# IDENTIFICATION
#
# TITLE MMSKIP
# AUTHOR GREGG R FABREGAS
# VERSION A.01
# DATE 31-JUL-1984
# LANGUAGE RATFOR
# SYSTEM VAX 11/780
#
#
# UPDATE
#
# UPDATE # 1
# AUTHOR GREGG R FABREGAS
# DATE 07-OCT-1985
# VERSION A.01
# PURPOSE ADD CALLS TO STARTTIMER
# AND CHECKTIMER TO
# MONITOR ELAPSED CPU TIME.
#
# PURPOSE
#
# FOR MANY APPLICATIONS, THE ALGORITHM
# CAN BE IMPLEMENTED ON BOTH
# THE HOST AND THE MINIMAP. IN A
# TIME-SHARING ENVIRONMENT, IT IS
# ANTICIPATED THAT THE MINIMAP WILL
# OPERATE ON A FIRST-COME,
# FIRST-SERVED BASIS WITH OR WITHOUT
# THE BENEFIT OF ANY QUEUEING
# STRATEGY. AS A RESULT, THE USER
# MAY OR MAY NOT HAVE A MINIMAP
# AVAILABLE WHEN THE APPLICATION
# SOFTWARE IS USED. THIS ROUTINE
# IS USED TO BRANCH AROUND THE
# MINIMAP CODE IF THE MINIMAP IS
# NOT AVAILABLE.
#
# ENTRY POINT
#
# FLAG = MMSKIP( MEMORY )
#
# ARGUMENT LISTING
#
# FLAG LOGICAL SET IF THE MINIMAP IS NOT
# READY FOR EXECUTION.
```

```

#         MEMORY LOGICAL           INDICATES THE REQUIREMENT
#                                     FOR PROPER MINIMAP MEMORY
#                                     ALLOCATION.  FOR MOST
#                                     OPERATIONS, THIS SHOULD BE
#                                     .TRUE.
#
# INCLUDE FILES/COMMONS
#
#         GIPSY      INCLUDE      GIPSY GENERAL SYMBOL DEF'NS
#         GIPCOM     COMMON       GIPSY COMMAND LINE COMMON DATA
#         TTCOM      COMMON       FD'S FOR TERMINAL & RUNFILE IO
#         MINIMAP    COMMON       COMMON FOR THE ADMINISTRATION
#                                     OF THE MINIMAP
#                                     SUPPORT SOFTWARE.
#
# HARDWARE REQUIRED
#
#         NO HARDWARE IS REQUIRED.
#
# LANGUAGE FEATURES REQUIRED
#
#         ALTERNATE RETURNS ARE NEEDED.
#
# PROGRAM ENVIROMENT
#
#         THIS CALL CAN BE USED ANYWHERE,
#         SINCE THERE ARE NO ERROR
#         CONDITIONS TO REPORT.  HOWEVER,
#         THE CALL IS INTENDED TO BE USED
#         AFTER THE CALL TO MMALOC WHEN A
#         FIRM DECISION MUST BE MADE AS
#         TO WHETHER THE MINIMAP OR THE
#         HOST EXECUTES THE ALGORITHM.  THE
#         FORM OF THE CALL ALLOWS THE
#         PROGRAMMER TO PUT THE DECISION IN
#         A SIMPLE IF STATEMENT.
#
# ALGORITHM
#
#         THE BRANCH IS ALWAYS SET IF THE
#         MINIMAP IS NOT READY FOR
#         EXECUTION.  THE BRANCH WILL ALSO
#         BE SET IF THE USER REQUIRES
#         MINIMAP MEMORY (MEMORY ARGUMENT
#         SET TO .TRUE.) AND AN ERROR
#         OCCURRED IN ALLOCATION.
#
# ROUTINES CALLED
#
#         PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#                   (GIPSY PRIMITIVE)

```

```

#          PUTF          MACHINE INDEPENDENT FORMATTED OUTPUT
#                      (GIPSY PRIMITIVE)
#          PPOP          POP PROGRAM NAME FROM THE ERROR STACK
#                      (GIPSY PRIMITIVE)
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
#          INCLUDE GIPSY
#
#          LOGICAL FUNCTION MMSKIP( MEMORY )
#
#
#          INCLUDE GIPCOM
#          INCLUDE TTCOM
#          INCLUDE MINIMAP
#
#          LOGICAL MEMORY, TRACE, BRANCH
#
#          EQUIVALENCE ( TRACE, SFLAG( .A ) )
#
#          PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
#          CALL PPUSH( "MMSKIP" )
#          IF ( TRACE )
#              $(
#              CALL PUTF( FDRUNO, "AP trace: MMSKIP called%N" )
#              STARTTIMER
#              9900 CONTINUE
#              $)
#
#          IF MMINIT HAS NOT BEEN CALLED, THEN MMREDY IS
#          IN AN INCONSISTENT STATE. MMREDY WILL BE
#          CLEARED IN THIS CASE.
#
#          IF ( ~ MMCHKD )
#              $(
#              MMREDY = .FALSE.
#              IF ( TRACE )
#                  CALL PUTF( FDRUNO,
#                          "AP trace: no prior call to MMINIT%N" )
#              $)
#
#          "BRANCH" DETERMINES WHETHER TO BRANCH OR NOT.
#          THE BRANCH IS AUTOMATICALLY TAKEN IF THE
#          MINIMAP IS NOT READY.
#
#          BRANCH = ~ MMREDY
#          IF ( TRACE )
#              CALL PUTF( FDRUNO,

```

```

"AP trace: AP ready: %5B; AP memory allocated: %5B%N",
    MMREDY, MMALCD )
#
#     IF MEMORY IS NOT REQUESTED, THEN IGNORE THE
#     STATUS OF MEMORY ALLOCATION, OTHERWISE BRANCH
#     IF MEMORY IS NOT ALLOCATED.
#
IF ( MEMORY )
    BRANCH = BRANCH ! ( - MMALCD )
IF ( TRACE )
    CALL PUTF( FDRUNO, "AP trace: branch state: %5B%N",
        BRANCH )
#
#     RETURN ON SETTING OF MATCH VARIABLE.
#
CHECKTIMER
CALL PPOP
RETURN ( BRANCH )
#
#
END

```


B APPENDIX B: LISTINGS OF GROUP TWO ROUTINES

What follows is the RATFOR source code for the GETWPM, MMALOC, MMSNDA, MMRCVA, MMSNDS, MMRCVS, MMSEND and MMRECV routines. They are located in the GIPSY_PRIM directory in the GIPSY package.

B.1 GETWPM Routine

```
#--GETWPM  GETWP SUPPORT FOR THE MINIMAP SUPPORT SOFTWARE
#
# IDENTIFICATION
#
#     TITLE           GETWPM
#     AUTHOR          GREGG R FABREGAS
#     VERSION         A.01
#     DATE            31-JUL-1984
#     LANGUAGE        RATFOR
#     SYSTEM          VAX 11/780
#
#
# UPDATE
#
#     UPDATE          # 1
#     AUTHOR          GREGG R FABREGAS
#     DATE            22-JUN-1985
#     VERSION         A.01
#     PURPOSE         ADD FOURTH ARGUMENT TO
#                   CALL SO ARRAYS ARE
#                   NAMED INSTEAD OF
#                   NUMBERED BY GETWPM CALL.
#
#     UPDATE          # 2
#     AUTHOR          GREGG R FABREGAS
#     DATE            06-OCT-1985
#     VERSION         A.01
#     PURPOSE         ADD CALLS TO STARTTIMER
#                   AND CHECKTIMER TO
#                   MONITOR ELAPSED CPU TIME.
#
#     UPDATE          # 3
#     AUTHOR          GREGG R FABREGAS
#     DATE            06-MAR-1986
#     VERSION         A.01
#     PURPOSE         ALIGN STARTING ADDRESS OF ALLOCATION
#                   SO ELEMENTS ARE NOT SPLIT ACROSS
#                   MEMORY BOUNDARIES.
#
#     UPDATE          # 4
#     AUTHOR          GREGG R FABREGAS
#     DATE            30-MAR-1986
#     VERSION         A.01
#     PURPOSE         ADD TRACE MESSAGE ABOUT
#                   ARRAY SIZES AND LOCS.
#
# PURPOSE
#
```

```

# THIS ROUTINE IS MODELED AFTER
# THE GETWP ROUTINE, WHICH COMPUTES
# THE SUBSCRIPT IN A LARGE WORK ARRAY,
# TO BE USED AS THE STARTING
# POINT FOR A SMALLER ARRAY POSITIONED
# OVER A PORTION OF THE WORK
# ARRAY. THIS ROUTINE IS USED
# TO IMPLEMENT DYNAMIC MEMORY
# ALLOCATION ON THE MINIMAP IN A
# FORMAT IDENTICAL TO THE SCHEME
# USED IN GIPSY. IT DIFFERS FROM
# GETWP SINCE MEMORY IS ALLOCATED
# FOR BOTH THE MINIMAP AND GIPSY.
# IN GETWP, ONLY GIPSY MEMORY
# IS ALLOCATED. ALSO, A NAME IS
# GIVEN TO EACH ARRAY FOR
# REFERENCE BY THE MMSNDA AND
# MMRCVA ROUTINES AND BY THE MINIMAP
# DRIVER ROUTINE.
#
# ENTRY POINT
#
# PTR = GETWPM( NXT, MODE, LEN, NAME )
#
# ARGUMENT LISTING
#
# PTR      INTEGER      STARTING SUBSCRIPT IN
#                                WORK ARRAY FOR THE
#                                CURRENT PARTITION.
#
# NXT      INTEGER      MAINTAINS THE AMOUNT
#                                OF WORK ARRAY SPACE
#                                CONSUMED BY SMALLER
#                                ARRAY PARTITIONS.
#                                NXT IS INITIALIZED
#                                TO 1 PRIOR TO THE
#                                FIRST CALL TO GETWP
#                                OR GETWPM. AFTER
#                                ALL OF THE CALLS TO
#                                GETWP AND GETWPM,
#                                NXT CONTAINS THE TOTAL
#                                SPACE REQUIRED.
#
# MODE     INTEGER      DATA TYPE FOR THE NEW
#                                ARRAY PARTITION.
#
# LEN      INTEGER      NUMBER OF ELEMENTS IN
#                                THE NEW PARTITION.
#
# NAME     PCHAR ARRAY  ARRAY NAME, SIX-CHARACTER
#                                MAXIMUM LENGTH.
#
# INCLUDE FILES/COMMONS
#
# GIPSY    INCLUDE      GIPSY GENERAL SYMBOL DEF'NS

```

```
# GIPCOM COMMON GIPSY COMMAND LINE COMMON DATA
# TTCOM COMMON FD'S FOR TERMINAL & RUNFILE IO
# MINIMAP COMMON COMMON FOR THE ADMINISTRATION
# OF THE MINIMAP
# SUPPORT SOFTWARE.
```

```
# HARDWARE REQUIRED
```

```
# A CSPI MINIMAP ARRAY PROCESSOR MUST BE OPENED.
```

```
# PROGRAM ENVIROMENT
```

```
# THIS ROUTINE IS EXPECTED TO BE
# USED IN A GIPSY COMMAND DRIVER
# TO DETERMINE THE MEMORY REQUIREMENTS
# FOR THE CURRENT COMMAND.
# SPECIFICALLY, THIS STATEMENT MUST
# ONLY BE USED BETWEEN THE ASSIGNMENT,
# "NXT = 1", AND THE CALL TO MMALOC.
# MINIMAP ALLOCATION IS DISABLED
# IF THE MMINIT ROUTINE HAS NOT
# COMPLETED SUCCESSFULLY, OR IF AN
# ERROR HAS DEVELOPED BY A PRIOR
# CALL TO GETWPM OR MMALOC. IN
# EITHER CASE, NO ERROR IS REPORTED
# UNTIL MMALOC IS CALLED. AN
# ERROR CONDITION OCCURS IF MMALOC
# HAS ALREADY BEEN CALLED WITH A
# POSITIVE ARGUMENT PRIOR TO THE
# CALL TO THIS ROUTINE. ALLOCATION
# IS RE-ENABLED BY A CALL TO MMALOC
# WITH A ZERO ARGUMENT.
```

```
# ALGORITHM
```

```
# THERE ARE THREE VALUES STORED IN THE MMHARR ARRAY:
# (1) NXT IS COPIED,
# (2) LENGTH OF THE ARRAY IN MODE UNITS, AND
# (3) THE DATA MODE OF THE ARRAY.
# THERE ARE TWO VALUES STORED IN THE MMMARR ARRAY:
# (1) THE OFFSET IN MINIMAP MEMORY FOR THE
# STARTING ADDRESS, AND
# (2) THE LENGTH OF THE ARRAY IN MAP-INTEGERS.
```

```
# LIMITATIONS
```

```
# A LIMITED NUMBER OF SLOTS ARE AVAILABLE.
```

```
# ROUTINES CALLED
```

```
# PUTF MACHINE INDEPENDENT FORMATTED OUTPUT
```

```

#           (GIPSY PRIMITIVE)
#   ICEIL   CEILING OF A/B (GIPSY OS KERNEL)
#   GETWP   GET WORK ARRAY POINTER
#           (GIPSY PRIMITIVE)
#
#   REMARKS
#
#           SINCE THERE IS NO ERROR RETURN
#           OR ERROR REPORTING SCHEME IN
#           GETWP OR GETWPM, THERE IS NO
#           NEED FOR CALLS TO PPUSH OR PPOP.
#           HOWEVER, THE A-FLAG CAN BE USED
#           TO TRACE THE ACTIVITY OF THIS
#           ROUTINE.
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#   INCLUDE GIPSY
#
#   INTEGER FUNCTION GETWPM( NXT, MODE, LEN, NAME )
#
#
#   INCLUDE GIPCOM
#   INCLUDE TTCOM
#   INCLUDE MINIMAP
#
#   INTEGER NXT, MODE, LEN
#   PCHARACTER NAME( .MMNLEN )
#   INTEGER I
#   REAL RATIO
#   CHARACTER DOT
#   INTEGER MOD, ICEIL, GETWP
#   LOGICAL TRACE
#
#   DATA DOT / '.' /
#
#   EQUIVALENCE ( TRACE, SFLAG( .A ) )
#
#           SINCE THIS IS USED AS SUPPORT LIKE GETWP,
#           THERE IS NO PPUSH, PPOP, OR ALTERNATE RETURN.
#
#   IF ( TRACE )
#       $(
#       CALL PUTF( FDRUNO, "AP trace: GETWPM called%N" )
#       STARTTIMER
#       $)
#
#           ALL MINIMAP ACTIVITY IS BYPASSED IF MMINIT HAS
#           NOT EXECUTED CORRECTLY OR A PREVIOUS CALL TO

```

```

#           GETWPM HAS FAILED.
#
IF ( MMCHKD & MMACTV & MMAERR == 0 )
$(
#
#           THIS ROUTINE MAY ONLY BE CALLED AFTER A CALL TO
#           MMALOC WITH A ZERO ARGUMENT.  THIS IS FIRST
#           DONE BY MMINIT.
#
IF ( MMALCD )
    GOTO 9000
#
#           THERE ARE THREE REQUIREMENTS FOR INSERTION INTO
#           THE ARRAY TABLE:
#           (1) SPACE FOR THE ENTRY,
#           (2) A VALID DATA MODE, AND
#           (3) THE ARRAY LENGTH MAY NOT EXCEED A NON-ZERO
#           ARRAY LIMIT.
#
IF ( MMACNT >= .MMAMAX )
    GOTO 9010
IF ( MODE < .MODEMIN ! MODE > .MODEMAX )
    GOTO 9020
IF ( LEN > MMMAXP & MMMAXP > 0 )
    GOTO 9030
#
#           GO AHEAD AND INSERT THE ENTRY, AND UPDATE THE
#           MINIMAP MEMORY REQUIREMENT IN MMCSIZ.
#
MMACNT = MMACNT + 1
DO I = 1, .MMNLEN
    MMHNAM( I, MMACNT ) = .EOS
CALL OSUNPK( NAME, MMHNAM( 1, MMACNT ),
            1, .MMNLEN, DOT )
IF ( TRACE )
    CALL PUTF( FDRUNO,
              "AP trace: slot %I used for %6A%N",
              MMACNT,
              MMHNAM( 1, MMACNT ) )
MMHARR( .MMHNXT, MMACNT ) = NXT
MMHARR( .MMHMODE, MMACNT ) = MODE
MMHARR( .MMHLEN, MMACNT ) = LEN
#
#           ALIGN BEGINNING OF MINI-MAP ARRAY WITH NEW MODE
#
RATIO = ICEIL( MMMODE( MODE ), MMMODE( .MAPINTMODE ) )
IF ( RATIO > 1.0 )
    MMCSIZ = AINT( ( MMCSIZ + RATIO ) / RATIO ) * RATIO
MMMARR( MMACNT, .MMMWORK ) = MMCSIZ + 1.0
MMMARR( MMACNT, .MMMLLEN ) = ICEIL( MMMODE( MODE ) * LEN,
    MMMODE( .MAPINTMODE ) )

```

```

MMCSIZ = MMCSIZ + MMMARR( MMACNT, .MMMLEN )
IF ( TRACE )
    CALL PUTF( FDRUNO,
        "AP trace: Start at %R; Length = %R; Total = %R%N",
        MMMARR( MMACNT, .MMMWORK ),
        MMMARR( MMACNT, .MMMLEN ),
        MMCSIZ )
#
#     FINISHED WITH MINIMAP WORK.  NOW DO THE GIPSY
#     ALLOCATION.
#
CHECKTIMER
RETURN ( GETWP( NXT, MODE, LEN ) )
#
#     ERRORS TO BE REPORTED LATER:
#
9000 CONTINUE
MMAERR = -2028
IF ( TRACE )
    CALL PUTF( FDRUNO,
        "AP trace: GETWPM called after MMALOC called%N" )
GOTO 9900
#
9010 CONTINUE
MMAERR = -2029
IF ( TRACE )
    CALL PUTF( FDRUNO,
        "AP trace: exhausted pointer space in GETWPM%N" )
GOTO 9900
#
9020 CONTINUE
MMAERR = -2032
IF ( TRACE )
    CALL PUTF( FDRUNO,
        "AP trace: illegal data mode in GETWPM%N" )
GOTO 9900
#
9030 CONTINUE
MMAERR = -2042
IF ( TRACE )
    CALL PUTF( FDRUNO,
        "AP trace: array size exceeds maximum%N" )
$)
#
#     THIS RETURN IS TAKEN IF THE MINIMAP IS NOT
#     AVAILABLE OR AN ERROR OCCURS.
#
9900 CONTINUE
CHECKTIMER
RETURN ( GETWP( NXT, MODE, LEN ) )
#

```


END

B.2 MMALOC Routine

```
#--MMALOC  VERIFY MEMORY FOR THE MINIMAP SUPPORT SOFTWARE
#
#  IDENTIFICATION
#
#      TITLE           MMALOC
#      AUTHOR          GREGG R FABREGAS
#      VERSION         A.01
#      DATE            18-JUL-1984
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#
#  UPDATE
#
#      UPDATE          # 1
#      AUTHOR          GREGG R FABREGAS
#      DATE            06-OCT-1985
#      VERSION         A.01
#      PURPOSE         ADD CALLS TO STARTTIMER
#                     AND CHECKTIMER TO
#                     MONITOR ELAPSED CPU TIME.
#
#  PURPOSE
#
#      THIS ROUTINE VERIFIES THAT THE
#      REQUESTS FOR ARRAY SPACE WITHIN
#      THE MINIMAP MEMORY SPACE DOES
#      NOT EXCEED THE SIZE OF THE
#      MINIMAP MEMORY.  ADDITIONALLY,
#      THIS ROUTINE INITIALIZES
#      VARIABLES IN THE MINIMAP COMMON
#      THAT ARE USED FOR MEMORY
#      ALLOCATION.  VERIFICATION INCLUDES
#      SENDING THE ARRAY
#      INFORMATION TO THE MINIMAP SO
#      THAT GIPSY ARRAYS MAP ONTO
#      MINIMAP ARRAYS.
#
#  ENTRY POINT
#
#      IEV = MMALOC( NXT )
#
#  ARGUMENT LISTING
#
#      IEV      INTEGER      INTEGER EVENT VARIABLE.
#                                     RETURNS THE FOLLOWING
#                                     STATUS CODES WHEN DONE:
#                                     .OK   IF THE CALL EXECUTED
```

```

#
# CORRECTLY,
# -2023 IF THE MMINIT ROUTINE
# HAS NOT YET BEEN
# CALLED,
# -2024 IF THE MMINIT ROUTINE
# FAILED,
# -2027 IF MMALOC HAS ALREADY
# BEEN CALLED WITH AN
# ALLOCATION ARGUMENT,
# -2028 IF A CALL WAS MADE TO
# GETWPM AFTER MMALOC
# VERIFIED ALLOCATION,
# -2029 IF ARRAY SPACE WAS
# EXHAUSTED IN
# THE MINIMAP COMMON
# WHEN A CALL WAS
# MADE TO GETWPM,
# -2032 IF AN ILLEGAL DATA MODE
# WAS PASSED TO GETWPM,
# -2035 IF THE REQUEST FOR
# MEMORY EXCEEDS THE
# CAPACITY OF THE
# MINIMAP, AND
# -2042 IF THE REQUEST FOR
# MEMORY EXCEEDS THE
# MAXIMUM SET BY THE
# MODULE LOADED
# INTO THE MINIMAP.
#
#           NXT           INTEGER
#
#           IF POSITIVE, VERIFY THE
#           MEMORY ALLOCATION, OTHERWISE,
#           RESET THE STATUS
#           OF MEMORY ALLOCATION.
#
# INCLUDE FILES/COMMONS
#
#           GIPSY           INCLUDE           GIPSY GENERAL SYMBOL DEF'NS
#           GIPCOM          COMMON           GIPSY COMMAND LINE COMMON DATA
#           TTCOM           COMMON           FD'S FOR TERMINAL & RUNFILE IO
#           MINIMAP         COMMON           COMMON FOR THE ADMINISTRATION
#
#                                           OF THE MINIMAP
#                                           SUPPORT SOFTWARE.
#
# HARDWARE REQUIRED
#
#           A CSPI MINIMAP ARRAY PROCESSOR MUST BE OPENED.
#
# PROGRAM ENVIROMENT
#
#           THIS ROUTINE SHOULD BE CALLED
#           AFTER THE CALL TO MMINIT AND A
#           VARIED NUMBER OF CALLS TO THE

```

```

# GETWPM ROUTINE. IN GIPSY, THE
# CALL IS IDENTICAL TO THE CALL
# TO OSALOC, AND THIS CALL SHOULD
# BE PLACED AFTER THE LAST CALL
# TO OSALOC WITH A POSITIVE
# ARGUMENT, AS WELL AS ANY CALLS
# TO OSALOC WITH A ZERO ARGUMENT.
#
# ROUTINES CALLED
#
# PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#             (GIPSY PRIMITIVE)
# PUTF       MACHINE INDEPENDENT FORMATTED OUTPUT
#             (GIPSY PRIMITIVE)
# MMSSEND    SEND DATA TO MINIMAP FOR MINIMAP
#             SUPPORT SOFTWARE. (GIPSY PRIMITIVE)
# OSMMAP     MACHINE-DEPENDENT CODE FOR MINIMAP
#             SUPPORT SOFTWARE. (GIPSY OS KERNEL)
# MMEXEC     EXECUTE LOAD MODULES FOR MINIMAP
#             SUPPORT SOFTWARE. (GIPSY PRIMITIVE)
# OSSIEV     PUT EVENT VARIABLE (IEV) INTO SAVE
#             AREA (GIPSY OS KERNEL)
# PPOP       POP PROGRAM NAME FROM THE ERROR STACK
#             (GIPSY PRIMITIVE)
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
# INCLUDE GIPSY
#
# INTEGER FUNCTION MMALOC( NXT )
#
#
# INCLUDE GIPCOM
# INCLUDE TTCOM
# INCLUDE MINIMAP
#
# INTEGER NXT, COMPAR, IEV, SAVE
# LOGICAL TRACE
# INTEGER OSMMAP, OSSIEV
# REAL FLOAT
#
# EQUIVALENCE ( TRACE, SFLAG( .A ) )
#
#           PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
# CALL PPUSH( "MMALOC" )
# IF ( TRACE )
#     $(
#     CALL PUTF( FDRUNO, "AP trace: MMALOC called%N" )

```

```

STARTTIMER
$(
#
#           MMINIT MUST HAVE EXECUTED CORRECTLY BEFORE THIS
#           ROUTINE CAN EXECUTE.  ALSO, THERE MUST BE NO
#           PENDING ERRORS STORED IN MMAERR BY GETWPM.
#
IF ( ~ MMCHKD )
    GOTO 9000
IF ( ~ MMACTV )
    GOTO 9010
IF ( MMAERR < 0 )
    GOTO 9900
#
#           FOR A POSITIVE NXT VALUE, THIS MUST BE THE CALL
#           JUST AFTER THE LAST CALL TO OSALOC.  WATCH FOR
#           PRIOR CALLS TO MMALOC, AND CHECK IF THERE IS
#           SUFFICIENT MEMORY IN THE MINIMAP FOR THE
#           REQUEST.  SEND ARRAY PARAMETERS TO MINIMAP.
#           IF OK, SET MMALCD.
#
IF ( NXT > 0 )
    $(
        IF ( MMALCD )
            GOTO 9020
        IF ( MMCSIZ > MMMAXC )
            GOTO 9030
        CALL MMSEND( MMMARR, .MMAMAX * .MMMROWS,
            .REALMODE, 1, MMAERR, %9900 )
        SAVE = MMDBUG
        MMDBUG = .DEBUGDISABLED
        CALL MMEEXEC( .WAIT, MMAERR, %9900 )
        MMALCD = .TRUE.
        MMDBUG = SAVE
    $)
ELSE
#
#           FOR NEGATIVE OR ZERO NXT VALUES, RESET THE
#           MEMORY POINTERS AND CLEAR MMALCD.
#
    $(
        MMCSIZ = FLOAT( .MAPWORKBASE - 1 )
        IF ( .OK ~= OSMMAP( .MAPMEMSIZE, MMMAXC ) ) GOTO 9800
        MMMAXC = MMMAXC - .WINDOWBASE
        MMAENT = 0
        MMAERR = 0
        MMALCD = .FALSE.
    $)
#
#           FINISHED WITHOUT ANY ERRORS.
#

```

```

CALL OSSIEV( .OK, "MMALOC succeeded" )
CHECKTIMER
CALL PPOP
RETURN ( .OK )
#
#           ERROR CONDITIONS ARE SAVED IN COMMON SINCE
#           MEMORY PROBLEMS ARE NOT FATAL TO SOME
#           OPERATIONS:
#
#           MMINIT ROUTINE HAS NOT BEEN EXECUTED YET.
#
9000 CONTINUE
MMAERR = -2023
GOTO 9900
#
#           MINIMAP INITIALIZATION HAS FAILED.
#
9010 CONTINUE
MMAERR = -2024
GOTO 9900
#
#           MMALOC HAS ALREADY BEEN CALLED WITH A POSITIVE
#           ARGUMENT.
#
9020 CONTINUE
MMAERR = -2027
GOTO 9900
#
#           MINIMAP MEMORY LIMIT HAS BEEN EXCEEDED.
#
9030 CONTINUE
MMAERR = -2035
GOTO 9900
#
#           RETRIEVE ERROR FROM OSMMAP.
#
9800 CONTINUE
MMAERR = OSGIEV( MMAERR )
#
#           RETURN THE ERROR STORED IN COMMON.
#
9900 CONTINUE
IEV = OSSIEV( MMAERR, "MMALOC failed" )
CALL PRESET( "MMALOC" )
CHECKTIMER
CALL PPOP
RETURN ( IEV )
#
#
END

```

B.3 MMSNDA Routine

```
#--MMSNDA SEND ARRAY FOR MINIMAP SUPPORT SOFTWARE
#
# IDENTIFICATION
#
# TITLE MMSNDA
# AUTHOR GREGG R FABREGAS
# VERSION A.01
# DATE 31-JUL-1984
# LANGUAGE RATFOR
# SYSTEM VAX 11/780
#
#
# UPDATE
#
# UPDATE # 1
# AUTHOR GREGG R FABREGAS
# DATE 22-JUN-1985
# VERSION A.01
# PURPOSE CHANGE FIRST ARGUMENT
# FROM A NUMBER TO AN
# ARRAY NAME TO SIMPLIFY
# ROUTINE USAGE.
#
# UPDATE # 2
# AUTHOR GREGG R FABREGAS
# DATE 07-OCT-1985
# VERSION A.01
# PURPOSE ADD CALLS TO STARTTIMER
# AND CHECKTIMER TO
# MONITOR ELAPSED CPU TIME.
#
# PURPOSE
#
# THIS ROUTINE TRANSFERS AN ARRAY
# TO THE MINIMAP THAT HAS BEEN
# ALLOCATED USING THE GETWPM FACILITY.
# SINCE ALL OF THE
# NECESSARY INFORMATION HAS BEEN
# PROVIDED EARLIER, THE PROGRAMMER
# ONLY PASSES THE ARRAY NAME THAT
# WAS USED TO IDENTIFY THE
# ENTRY SLOT IN THE GETWPM ROUTINE.
# THIS ROUTINE TRANSFERS
# THE ENTIRE ARRAY. TO SEND ONLY A
# PORTION OF THE ARRAY, THE
# MMSSEND ROUTINE MUST BE USED.
#
# ENTRY POINT
```

```

#
#       CALL MMSNDA( NAME, IEV, ERRET )
#
# ARGUMENT LISTING
#
#       NAME      PCHAR ARRAY    SPECIFIES THE NAME OF
#
#                                     THE ARRAY AS DECLARED
#
#                                     BY THE GETWPM ROUTINE.
#
#       IEV       INTEGER        INTEGER EVENT VARIABLE.
#
#                                     THE FOLLOWING
#
#                                     CODES ARE RETURNED WHEN DONE:
#
#                                     .OK   IF THE CALL SUCCEEDS,
#
#                                     -2023 IF THERE WAS NO PRIOR
#
#                                     CALL TO THE
#
#                                     MMINIT ROUTINE,
#
#                                     -2024 IF THE CALL TO THE
#
#                                     MMINIT ROUTINE FAILED,
#
#                                     -2030 IF THE MINIMAP IS NOT
#
#                                     LOADED WITH A GIPSY-
#
#                                     COMPATIBLE MODULE,
#
#                                     -2033 IF THERE WAS SOME
#
#                                     FAILURE IN THE
#
#                                     ALLOCATION OF THE
#
#                                     ARRAYS, AND
#
#                                     -2034 IF THE NAME DOES
#
#                                     NOT MATCH ANY ENTRY
#
#                                     IN THE ARRAY TABLES.
#
#       ERRET     LABEL          LABEL IN CALLING ROUTINE
#
#                                     FOR ALTERNATE ERROR RETURN.
#
# INCLUDE FILES/COMMONS
#
#       GIPSY     INCLUDE        GIPSY GENERAL SYMBOL DEF'NS
#
#       GIPCOM    COMMON         GIPSY COMMAND LINE COMMON DATA
#
#       TTCOM    COMMON         FD'S FOR TERMINAL & RUNFILE IO
#
#       MINIMAP  COMMON         COMMON FOR THE ADMINISTRATION
#
#                                     OF THE MINIMAP
#
#                                     SUPPORT SOFTWARE.
#
#       MEMCOM   COMMON         MEMORY ALLOCATION COMMON DATA
#
#
# HARDWARE REQUIRED
#
#
#       A CSPI MINIMAP ARRAY PROCESSOR MUST BE
#
#       OPENED AND READY FOR EXECUTION.
#
# LANGUAGE FEATURES REQUIRED
#
#
#       ALTERNATE RETURNS ARE NEEDED.
#
# PROGRAM ENVIROMENT
#
#

```

```

#       THIS ROUTINE REQUIRES PRIOR
#       SUCCESSFUL CALLS TO MMINIT, MMLOAD,
#       AND MMALOC.  ANY NUMBER OF CALLS TO
#       THIS ROUTINE MAY FOLLOW THEREAFTER.
#
# ALGORITHM
#
#       (1) MATCH THE POINTER TO A TABLE ENTRY
#           FOR THE GETWPM ARRAYS,
#       (2) CALL MMSEND TO SEND THE ARRAY TO THE MINIMAP.
#
# LIMITATIONS
#
#       ARRAYS MUST BE PRE-DEFINED BY THE GETWPM ROUTINE.
#
# ROUTINES CALLED
#
#       PPOUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#                   (GIPSY PRIMITIVE)
#       PUTF        MACHINE INDEPENDENT FORMATTED OUTPUT
#                   (GIPSY PRIMITIVE)
#       MMSEND      SEND DATA TO MINIMAP FOR MINIMAP
#                   SUPPORT SOFTWARE. (GIPSY PRIMITIVE)
#       PPOP        POP PROGRAM NAME FROM THE ERROR STACK
#                   (GIPSY PRIMITIVE)
#
#
#*****
#
# INCLUDE GIPSY
#
# SUBROUTINE MMSNDA( NAME, IEV, * )
#
#
# INCLUDE GIPCOM
# INCLUDE TTCOM
# INCLUDE MINIMAP
# INCLUDE MEMCOM
#
# PCHARACTER NAME( .MMNLEN )
# INTEGER PTR, IEV, COUNT, MODE
# INTEGER INDEX, LOC
# CHARACTER CHNAME( .MMNLEN ), DOT
# LOGICAL TRACE
# INTEGER INT, STRCMP, STRLOC
#
# DATA DOT / '.' /
#
# EQUIVALENCE ( TRACE, SFLAG( .A ) )
#
#

```



```

#           PUSH ROUTINE ONTO ERROR TRACE STACK.
#
CALL PPUSH( "MMSNDA" )
IF ( TRACE )
    $(
        CALL PUTF( FDRUNO, "AP trace: MMSNDA called%N" )
        STARTTIMER
    $)
#
#           THIS ROUTINE REQUIRES A PRIOR CALL TO MMINIT,
#           THE MINIMAP TO BE PROPERLY LOADED, AND A PRIOR
#           CALL TO MMALOC WITH A POSITIVE ARGUMENT.
#
IF ( ~ MMCHKD )
    GOTO 9000
IF ( ~ MMACTV )
    GOTO 9010
IF ( ~ MMALCD )
    GOTO 9020
IF ( ~ MMREDY )
    GOTO 9030
#
#           THE NAME MUST BE PREVIOUSLY DECLARED BY
#           THE GETWPM ROUTINE.
#
CALL OSUNPK( NAME, CHNAME, 1, .MMNLEN, DOT )
LOC = STRLOC( DOT, 1, CHNAME, .MMNLEN )
IF ( LOC == 0 )
    LOC = .MMNLEN
FOR ( INDEX = 1; INDEX <= MMACNT; INDEX = INDEX + 1 )
    IF ( STRCMP( CHNAME, MMHNAM( 1, INDEX ), LOC ) == 0 )
        BREAK
IF ( TRACE )
    CALL PUTF( FDRUNO, "AP trace: %6A matched to slot %I%N",
                CHNAME, INDEX )
IF ( INDEX > MMACNT )
    GOTO 9040
#
#           RETRIEVE THE ARRAY ATTRIBUTES FROM THE MINIMAP
#           COMMON.
#
PTR = MMHARR( .MMHNXT, INDEX )
COUNT = MMHARR( .MMHLEN, INDEX )
MODE = MMHARR( .MMHMODE, INDEX )
#
#           SEND THE ARRAY TO THE MINIMAP.
#
CALL MMSEND( DYN( PTR ), COUNT, MODE,
             INT( MMMARR( INDEX, .MMMWORK ) ), IEV, %9900 )
#
#           FINISHED.

```

```

#
CHECKTIMER
CALL PPOP
RETURN
#
#           ERROR CONDITIONS:
#
#           MMINIT ROUTINE HAS NOT BEEN EXECUTED YET.
#
9000 CONTINUE
IEV = -2023
GOTO 9900
#
#           MINIMAP INITIALIZATION HAS FAILED.
#
9010 CONTINUE
IEV = -2024
GOTO 9900
#
#           EITHER A MISSING OR FAILED CALL TO MMALOC.
#
9020 CONTINUE
IEV = -2033
GOTO 9900
#
#           MINIMAP NOT READY FOR EXECUTION.
#
9030 CONTINUE
IEV = -2030
GOTO 9900
#
#           INDEX SLOT IS NOT USED IN MINIMAP COMMON.
#           PROBABLY, THE ARRAY WAS NOT ALLOCATED USING
#           GETWPM.
#
9040 CONTINUE
IEV = -2034
#
#           TAKE THE ALTERNATE ERROR RETURN.
#
9900 CONTINUE
CHECKTIMER
RETURN 1
#
#
END

```

B.4 MMRCVA Routine

```
#--MMRCVA  GET ARRAY FOR MINIMAP SUPPORT SOFTWARE
#
#  IDENTIFICATION
#
#      TITLE           MMRCVA
#      AUTHOR          GREGG R FABREGAS
#      VERSION         A.01
#      DATE            31-JUL-1984
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#
#  UPDATE
#
#      UPDATE          # 1
#      AUTHOR          GREGG R FABREGAS
#      DATE            22-JUN-1985
#      VERSION         A.01
#      PURPOSE         CHANGE FIRST ARGUMENT
#                     FROM A NUMBER TO AN
#                     ARRAY NAME TO SIMPLIFY
#                     ROUTINE USAGE.
#
#      UPDATE          # 2
#      AUTHOR          GREGG R FABREGAS
#      DATE            07-OCT-1985
#      VERSION         A.01
#      PURPOSE         ADD CALLS TO STARTTIMER
#                     AND CHECKTIMER TO
#                     MONITOR ELAPSED CPU TIME.
#
#  PURPOSE
#
#      THIS ROUTINE RETRIEVES AN ARRAY
#      FROM THE MINIMAP THAT HAS BEEN
#      ALLOCATED USING THE GETWPM FACILITY.
#      SINCE ALL OF THE
#      NECESSARY INFORMATION HAS BEEN
#      PROVIDED EARLIER, THE PROGRAMMER
#      ONLY PASSES THE ARRAY NAME THAT
#      WAS USED TO IDENTIFY THE
#      ENTRY SLOT IN THE GETWPM ROUTINE.
#      THIS ROUTINE TRANSFERS
#      THE ENTIRE ARRAY.  TO RETRIEVE
#      ONLY A PORTION OF THE ARRAY, THE
#      MMRECV ROUTINE MUST BE USED.
#
#  ENTRY POINT
```

```

#
# CALL MMRCVA( NAME, IEV, ERRET )
#
# ARGUMENT LISTING
#
# NAME PCHAR ARRAY SPECIFIES THE NAME OF
# THE ARRAY AS DECLARED
# BY THE GETWPM ROUTINE.
# IEV INTEGER INTEGER EVENT VARIABLE.
# THE FOLLOWING
# CODES ARE RETURNED WHEN DONE:
# .OK IF THE CALL SUCCEEDS,
# -2023 IF THERE WAS NO
# PRIOR CALL TO THE
# MMINIT ROUTINE,
# -2024 IF THE CALL TO THE
# MMINIT ROUTINE FAILED,
# -2030 IF THE MINIMAP IS NOT
# LOADED WITH A GIPSY-
# COMPATIBLE MODULE,
# -2033 IF THERE WAS SOME
# FAILURE IN THE
# ALLOCATION OF THE
# ARRAYS, AND
# -2034 IF THE NAME DOES
# NOT MATCH ANY
# ENTRY IN THE
# ARRAY TABLES.
# ERRET LABEL LABEL IN CALLING ROUTINE FOR
# ALTERNATE ERROR RETURN.
#
# INCLUDE FILES/COMMONS
#
# GIPSY INCLUDE GIPSY GENERAL SYMBOL DEF'NS
# GIPCOM COMMON GIPSY COMMAND LINE COMMON DATA
# TTCOM COMMON FD'S FOR TERMINAL & RUNFILE IO
# MINIMAP COMMON COMMON FOR THE ADMINISTRATION
# OF THE MINIMAP
# SUPPORT SOFTWARE.
# MEMCOM COMMON MEMORY ALLOCATION COMMON DATA
#
# HARDWARE REQUIRED
#
# A CSPI MINIMAP ARRAY PROCESSOR MUST BE OPENED
# AND READY FOR EXECUTION.
#
# LANGUAGE FEATURES REQUIRED
#
# ALTERNATE RETURNS ARE NEEDED.
#
# PROGRAM ENVIROMENT

```

```

#
#   THIS ROUTINE REQUIRES PRIOR
#   SUCCESSFUL CALLS TO MMINIT, MMLOAD,
#   AND MMALOC.  ANY NUMBER OF CALLS TO
#   THIS ROUTINE MAY FOLLOW THEREAFTER.
#
# ALGORITHM
#
#   (1) MATCH THE POINTER TO A TABLE
#       ENTRY FOR THE GETWPM ARRAYS,
#   (2) CALL MMRECV TO RETRIEVE THE
#       ARRAY FROM THE MINIMAP.
#
# LIMITATIONS
#
#   ARRAYS MUST BE PRE-DEFINED BY THE GETWPM ROUTINE.
#
# ROUTINES CALLED
#
#   PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#              (GIPSY PRIMITIVE)
#   PUTF      MACHINE INDEPENDENT FORMATTED OUTPUT
#              (GIPSY PRIMITIVE)
#   MMRECV    RECEIVE DATA FROM MINIMAP FOR MINIMAP
#              SUPPORT SOFTWARE. (GIPSY PRIMITIVE)
#   PPOP      POP PROGRAM NAME FROM THE ERROR STACK
#              (GIPSY PRIMITIVE)
#
#
#*****
#
#   INCLUDE GIPSY
#
#   SUBROUTINE MMRCVA( NAME, IEV, * )
#
#   INCLUDE GIPCOM
#   INCLUDE TCOM
#   INCLUDE MINIMAP
#   INCLUDE MEMCOM
#
#   PCHARACTER NAME( .MMNLEN )
#   INTEGER PTR, IEV, COUNT, MODE
#   INTEGER INDEX, LOC
#   CHARACTER CHNAME( .MMNLEN ), DOT
#   LOGICAL TRACE
#   INTEGER INT, STRCMP, STRLOC
#
#   DATA DOT / '.' /
#
#

```

```

EQUIVALENCE ( TRACE, SFLAG( .A ) )
#
#           PUSH ROUTINE ONTO ERROR TRACE STACK.
#
CALL PPUSH( "MMRCVA" )
IF ( TRACE )
    $(
    CALL PUTF( FDRUNO, "AP trace: MMRCVA called%N" )
    STARTTIMER
    $)
#
#           THIS ROUTINE REQUIRES A PRIOR CALL TO MMINIT,
#           THE MINIMAP TO BE PROPERLY LOADED, AND A PRIOR
#           CALL TO MMALOC WITH A POSITIVE ARGUMENT.
#
IF ( - MMCHKD )
    GOTO 9000
IF ( - MMACTV )
    GOTO 9010
IF ( - MMALCD )
    GOTO 9020
IF ( - MMREDY )
    GOTO 9030
#
#           THE NAME MUST BE PREVIOUSLY DECLARED BY
#           THE GETWPM ROUTINE.
#
CALL OSUNPK( NAME, CHNAME, 1, .MMNLEN, DOT )
LOC = STRLOC( DOT, 1, CHNAME, .MMNLEN )
IF ( LOC == 0 )
    LOC = .MMNLEN
FOR ( INDEX = 1; INDEX <= MMACNT; INDEX = INDEX + 1 )
    IF ( STRCMP( CHNAME, MMHNAM( 1, INDEX ), LOC ) == 0 )
        BREAK
IF ( TRACE )
    CALL PUTF( FDRUNO, "AP trace: %6A matched to slot %I%N",
              CHNAME, INDEX )
IF ( INDEX > MMACNT )
    GOTO 9040
#
#           RETRIEVE THE ARRAY ATTRIBUTES FROM THE MINIMAP
#           COMMON.
#
PTR = MMHARR( .MMHNXT, INDEX )
COUNT = MMHARR( .MMHLEN, INDEX )
MODE = MMHARR( .MMHMODE, INDEX )
#
#           RETRIEVE THE ARRAY FROM THE MINIMAP.
#
CALL MMRECV( DYN( PTR ), COUNT, MODE,
            INT( MMMARR( INDEX, .MMMWORK ) ), IEV, %9900 )

```

```

#
#           FINISHED.
#
CHECKTIMER
CALL PPOP
RETURN
#
#           ERROR CONDITIONS:
#
#           MMINIT ROUTINE HAS NOT BEEN EXECUTED YET.
#
9000 CONTINUE
IEV = -2023
GOTO 9900
#
#           MINIMAP INITIALIZATION HAS FAILED.
#
9010 CONTINUE
IEV = -2024
GOTO 9900
#
#           EITHER A MISSING OR FAILED CALL TO MMALOC.
#
9020 CONTINUE
IEV = -2033
GOTO 9900
#
#           MINIMAP NOT READY FOR EXECUTION.
#
9030 CONTINUE
IEV = -2030
GOTO 9900
#
#           INDEX SLOT IS NOT USED IN MINIMAP COMMON.
#           PROBABLY, THE ARRAY WAS NOT ALLOCATED USING
#           GETWPM.
#
9040 CONTINUE
IEV = -2034
#
#           TAKE THE ALTERNATE ERROR RETURN.
#
9900 CONTINUE
CHECKTIMER
RETURN 1
#
#
END

```

B.5 MMSNDS Routine

```
#--MMSNDS SEND SCALAR FOR MINIMAP SUPPORT SOFTWARE
#
# IDENTIFICATION
#
# TITLE MMSNDS
# AUTHOR GREGG R FABREGAS
# VERSION A.01
# DATE 11-SEP-1984
# LANGUAGE RATFOR
# SYSTEM VAX 11/780
#
#
# UPDATE
#
# UPDATE # 1
# AUTHOR GREGG R FABREGAS
# DATE 07-OCT-1985
# VERSION A.01
# PURPOSE ADD CALLS TO STARTTIMER
# AND CHECKTIMER TO
# MONITOR ELAPSED CPU TIME.
#
# UPDATE # 2
# AUTHOR GREGG R FABREGAS
# DATE 06-MAR-1986
# VERSION A.01
# PURPOSE CORRECT ADDR COMPUTATION FOR REALS.
#
# PURPOSE
#
# THIS ROUTINE COPIES THE SPECIFIED
# VALUE TO AN INTEGER OR REAL
# SCALAR IN THE MINIMAP. SCALARS
# ONE THROUGH TWENTY REFER TO THE
# INTEGER VARIABLES, I1 THROUGH I20,
# PROVIDED IN THE MINIMAP
# DRIVER ROUTINE. SCALARS -1
# THROUGH -20 REFER TO THE REAL
# VARIABLES, R1 THROUGH R20, ALSO
# PROVIDED IN THE MINIMAP DRIVER
# ROUTINE. THE VALUE IS CONVERTED
# FROM THE HOST DATA MODE TO THE
# CORRESPONDING MINIMAP DATA MODE.
#
# ENTRY POINT
#
# CALL MMSNDS( VALUE, SCALAR, IEV, ERRET )
#
#
```



```

# ARGUMENT LISTING
#
# VALUE INT OR REAL VALUE TO BE SENT
# TO THE MINIMAP.
# SCALAR INTEGER INDICATES WHICH SCALAR
# TO MODIFY.
# IEV INTEGER INTEGER EVENT VARIABLE.
# THE FOLLOWING
# CODES ARE RETURNED WHEN DONE:
# .OK IF THIS CALL SUCCEEDS,
# -2023 IF THERE WAS NO PRIOR
# CALL TO THE
# MMINIT ROUTINE,
# -2024 IF THE CALL TO THE
# MMINIT ROUTINE FAILED,
# -2030 IF THE MINIMAP IS
# NOT LOADED WITH
# A GIPSY-COMPATIBLE
# MODULE, AND
# -2031 IF THE SCALAR INDEX
# IS OUT OF BOUNDS.
# ERRET LABEL LABEL IN CALLING ROUTINE FOR
# ALTERNATE ERROR RETURN.
#
# INCLUDE FILES/COMMONS
#
# GIPSY INCLUDE GIPSY GENERAL SYMBOL DEF'NS
# GIPCOM COMMON GIPSY COMMAND LINE COMMON DATA
# TTCOM COMMON FD'S FOR TERMINAL & RUNFILE IO
# MINIMAP COMMON COMMPN FOR THE ADMINISTRATION
# OF THE MINIMAP
# SUPPORT SOFTWARE.
#
# HARDWARE REQUIRED
#
# A CSPI MINIMAP ARRAY PROCESSOR MUST BE OPENED
# AND READY FOR EXECUTION.
#
# LANGUAGE FEATURES REQUIRED
#
# ALTERNATE RETURNS ARE NEEDED.
#
# PROGRAM ENVIROMENT
#
# ANY NUMBER OF CALLS TO THIS ROUTINE
# MAY FOLLOW A SUCCESSFUL CALL
# TO THE MMINIT AND MMLOAD ROUTINES.
# NOTE THAT THE MMLOAD ROUTINE
# IS CALLED INSIDE MMINIT.
#
# DATA FORMAT

```

```

#
#       AN INTEGER IS CONVERTED TO A
#       MAP-INTEGGER BEFORE TRANSMISSION.
#       A REAL IS CONVERTED TO A MAP-REAL
#       BEFORE TRANSMISSION.
#
# LIMITATIONS
#
#       TWENTY INTEGER AND TWENTY REAL
#       SCALARS ARE AVAILABLE.
#
# ROUTINES CALLED
#
#       PPUSH          PUSH PROGRAM NAME ONTO ERROR STACK
#                     (GIPSY PRIMITIVE)
#       PUTF           MACHINE INDEPENDENT FORMATTED OUTPUT
#                     (GIPSY PRIMITIVE)
#       OSMMAP         MACHINE-DEPENDENT CODE FOR THE MINIMAP
#                     SUPPORT SOFTWARE. (GIPSY OS KERNEL)
#       MMSEND         SEND DATA TO MINIMAP FOR THE MINIMAP
#                     SUPPORT SOFTWARE. (GIPSY PRIMITIVE)
#       PPOP           POP PROGRAM NAME FROM THE ERROR STACK
#                     (GIPSY PRIMITIVE)
#       OSGIEV         GET EVENT VARIABLE (IEV) FROM OS SAVE
#                     AREA (GIPSY OS KERNEL)
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
# INCLUDE GIPSY
#
# SUBROUTINE MMSNDS( VALUE, SCALAR, IEV, * )
#
#
# INCLUDE GIPCOM
# INCLUDE TTCOM
# INCLUDE MINIMAP
#
# INTEGER VALUE, SCALAR, IEV, ADDR, MODE, MVALUE( 2 )
# LOGICAL TRACE
# INTEGER OSMMAP, OSGIEV
#
# EQUIVALENCE ( TRACE, SFLAG( .A ) )
#
#       PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
# CALL PPUSH( "MMSNDS" )
# IF ( TRACE )
#   $(
#     CALL PUTF( FDRUNO, "AP trace: MMSNDS called%N" )

```

```

STARTTIMER
$(
#
#       THIS ROUTINE REQUIRES A PRIOR SUCCESSFUL CALL
#       TO MMINIT, AND THE MINIMAP TO BE PROPERLY
#       LOADED.
#
IF ( ~ MMCHKD )
    GOTO 9000
IF ( ~ MMACTV )
    GOTO 9010
IF ( ~ MMREDY )
    GOTO 9020
#
#       THE SCALAR INDEX MUST BE IN RANGE.
#
IF ( SCALAR == 0 ! SCALAR > 20 )
    GOTO 9030
#
#       A POSITIVE INDEX REFERS TO AN INTEGER SCALAR.
#
IF ( SCALAR > 0 )
    $(
    MODE = .MAPINTMODE
    ADDR = .MAPIS + SCALAR - 1
    $)
#
#       A NEGATIVE INDEX REFERS TO A REAL SCALAR.
#
ELSE
    $(
    MODE = .MAPREALMODE
    ADDR = .MAPRS - 2 * SCALAR - 2
    $)
#
#       AUTOMATIC CONVERSION TO MINIMAP FORMAT.
#
IF ( .OK ~ = OSMMAP( MODE, VALUE, MVALUE ) ) GOTO 9800
#
#       SEND SCALAR TO MINIMAP MEMORY.
#
CALL MMSSEND( MVALUE, 1, MODE, ADDR, IEV, %9900 )
#
#       FINISHED.
#
CHECKTIMER
CALL PPOP
RETURN
#
#       ERROR CONDITIONS:
#

```

```
#           MMINIT ROUTINE HAS NOT BEEN EXECUTED YET.
#
9000 CONTINUE
IEV = -2023
GOTO 9900
#
#           MINIMAP INITIALIZATION HAS FAILED.
#
9010 CONTINUE
IEV = -2024
GOTO 9900
#
#           MINIMAP NOT READY FOR EXECUTION.
#
9020 CONTINUE
IEV = -2030
GOTO 9900
#
#           SCALAR INDEX OUT OF BOUNDS.
#
9030 CONTINUE
IEV = -2031
GOTO 9900
#
#           ERROR OCCURRED IN OSMMAP.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#
#           TAKE ALTERNATE ERROR RETURN.
#
9900 CONTINUE
CHECKTIMER
RETURN 1
#
#
END
```

B.6 MMRCVS Routine

```
#--MMRCVS  GET SCALAR FOR MINIMAP SUPPORT SOFTWARE
#
#  IDENTIFICATION
#
#      TITLE           MMRCVS
#      AUTHOR          GREGG R FABREGAS
#      VERSION         A.01
#      DATE            11-SEP-1984
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#
#  UPDATE
#
#      UPDATE          # 1
#      AUTHOR          GREGG R FABREGAS
#      DATE            07-OCT-1985
#      VERSION         A.01
#      PURPOSE         ADD CALLS TO STARTTIMER
#                     AND CHECKTIMER TO
#                     MONITOR ELAPSED CPU TIME.
#
#      UPDATE          # 2
#      AUTHOR          GREGG R FABREGAS
#      DATE            06-MAR-1986
#      VERSION         A.01
#      PURPOSE         CORRECT ADDR COMPUTATION FOR REALS.
#
#  PURPOSE
#
#      THIS ROUTINE COPIES THE SPECIFIED
#      VALUE FROM AN INTEGER OR REAL
#      SCALAR IN THE MINIMAP.  SCALARS ONE
#      THROUGH TWENTY REFER TO THE
#      INTEGER VARIABLES, I1 THROUGH I20,
#      PROVIDED IN THE MINIMAP
#      DRIVER ROUTINE.  SCALARS -1
#      THROUGH -20 REFER TO THE REAL
#      VARIABLES, R1 THROUGH R20, ALSO
#      PROVIDED IN THE MINIMAP DRIVER
#      ROUTINE.  THE VALUE IS CONVERTED
#      TO THE HOST DATA MODE FROM THE
#      CORRESPONDING MINIMAP DATA MODE.
#
#  ENTRY POINT
#
#      CALL MMRCVS( VALUE, SCALAR, IEV, ERRET )
#
```

```

# ARGUMENT LISTING
#
# VALUE INT OR REAL VALUE RETURNED FROM
# THE MINIMAP.
# SCALAR INTEGER INDICATES WHICH SCALAR
# TO BE COPIED.
# IEV INTEGER INTEGER EVENT VARIABLE.
# THE FOLLOWING
# CODES ARE RETURNED WHEN DONE:
# .OK IF THIS CALL SUCCEEDS,
# -2023 IF THERE WAS NO PRIOR
# CALL TO THE
# MMINIT ROUTINE,
# -2024 IF THE CALL TO THE
# MMINIT ROUTINE FAILED,
# -2030 IF THE MINIMAP IS
# NOT LOADED WITH
# A GIPSY-COMPATIBLE
# MODULE, AND
# -2031 IF THE SCALAR INDEX
# IS OUT OF BOUNDS.
# ERRET LABEL LABEL IN CALLING ROUTINE FOR
# ALTERNATE ERROR RETURN.
#
# INCLUDE FILES/COMMONS
#
# GIPSY INCLUDE GIPSY GENERAL SYMBOL DEF'NS
# GIPCOM COMMON GIPSY COMMAND LINE COMMON DATA
# TTCOM COMMON FD'S FOR TERMINAL & RUNFILE IO
# MINIMAP COMMON COMMPN FOR THE ADMINISTRATION
# OF THE MINIMAP
# SUPPORT SOFTWARE.
#
# HARDWARE REQUIRED
#
# A CSPI MINIMAP ARRAY PROCESSOR MUST BE OPENED
# AND READY FOR EXECUTION.
#
# LANGUAGE FEATURES REQUIRED
#
# ALTERNATE RETURNS ARE NEEDED.
#
# PROGRAM ENVIROMENT
#
# ANY NUMBER OF CALLS TO THIS ROUTINE
# MAY FOLLOW A SUCCESSFUL CALL
# TO THE MMINIT AND MMLOAD ROUTINES.
# NOTE THAT THE MMLOAD ROUTINE
# IS CALLED INSIDE MMINIT.
#
# DATA FORMAT

```

```

#
#       A MAP-INTEGERS IS CONVERTED TO
#       AN INTEGER AFTER TRANSMISSION.
#       A MAP-REAL IS CONVERTED TO A
#       REAL AFTER TRANSMISSION.
#
# LIMITATIONS
#
#       TWENTY INTEGER AND TWENTY REAL
#       SCALARS ARE AVAILABLE.
#
# ROUTINES CALLED
#
#       PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#                  (GIPSY PRIMITIVE)
#       PUTF      MACHINE INDEPENDENT FORMATTED OUTPUT
#                  (GIPSY PRIMITIVE)
#       OSMMAP    MACHINE-DEPENDENT CODE FOR THE MINIMAP
#                  SUPPORT SOFTWARE. (GIPSY OS KERNEL)
#       MMRECV    GET DATA FROM MINIMAP FOR THE MINIMAP
#                  SUPPORT SOFTWARE. (GIPSY PRIMITIVE)
#       PPOP      POP PROGRAM NAME FROM THE ERROR STACK
#                  (GIPSY PRIMITIVE)
#       OSGIEV    GET EVENT VARIABLE (IEV) FROM OS SAVE
#                  AREA (GIPSY OS KERNEL)
#
#
#*****
#
# INCLUDE GIPSY
#
# SUBROUTINE MMRCVS( VALUE, SCALAR, IEV, * )
#
# INCLUDE GIPCOM
# INCLUDE TTCOM
# INCLUDE MINIMAP
#
# INTEGER VALUE, SCALAR, IEV, ADDR, MODE, TOMODE, MVALUE( 2 )
# LOGICAL TRACE
# INTEGER OSMMAP, OSGIEV
#
# EQUIVALENCE ( TRACE, SFLAG( .A ) )
#
#       PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
# CALL PPUSH( "MMRCVS" )
# IF ( TRACE )
#   $(
#     CALL PUTF( FDRUNO, "AP trace: MMRCVS called%N" )

```

```

STARTTIMER
$)
#
#           THIS ROUTINE REQUIRES A PRIOR SUCCESSFUL CALL
#           TO MMINIT, AND THE MINIMAP TO BE PROPERLY
#           LOADED.
#
IF ( ~ MMCHKD )
  GOTO 9000
IF ( ~ MMACTV )
  GOTO 9010
IF ( ~ MMREDY )
  GOTO 9020
#
#           THE SCALAR INDEX MUST BE IN RANGE.
#
IF ( SCALAR == 0 ! SCALAR > 20 )
  GOTO 9030
#
#           A POSITIVE INDEX REFERS TO AN INTEGER SCALAR.
#
IF ( SCALAR > 0 )
  $(
  MODE = .MAPINTMODE
  ADDR = .MAPIS + SCALAR - 1
  TOMODE = .INTMODE
  $)
#
#           A NEGATIVE INDEX REFERS TO A REAL SCALAR.
#
ELSE
  $(
  MODE = .MAPREALMODE
  ADDR = .MAPRS - 2 * SCALAR - 2
  TOMODE = .REALMODE
  $)
#
#           RETRIEVE SCALAR FROM MINIMAP MEMORY.
#
CALL MMRECV( MVALUE, 1, MODE, ADDR, IEV, %9900 )
#
#           AUTOMATIC CONVERSION TO HOST FORMAT.
#
IF ( .OK ~= OSMMAP( TOMODE, MVALUE, VALUE ) ) GOTO 9800
#
#           FINISHED.
#
CHECKTIMER
CALL PPOP
RETURN
#

```



```

#           ERROR CONDITIONS:
#
#           MMINIT ROUTINE HAS NOT BEEN EXECUTED YET.
#
9000 CONTINUE
IEV = -2023
GOTO 9900
#
#           MINIMAP INITIALIZATION HAS FAILED.
#
9010 CONTINUE
IEV = -2024
GOTO 9900
#
#           MINIMAP NOT READY FOR EXECUTION.
#
9020 CONTINUE
IEV = -2030
GOTO 9900
#
#           SCALAR INDEX OUT OF BOUNDS.
#
9030 CONTINUE
IEV = -2031
GOTO 9900
#
#           ERROR OCCURRED IN OSMMAP.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#
#           TAKE ALTERNATE ERROR RETURN.
#
9900 CONTINUE
CHECKTIMER
RETURN 1
#
#
END

```

B.7 MMSEND Routine

```
#--MMSEND SEND DATA TO MINIMAP FOR MINIMAP SUPPORT SOFTWARE
#
# IDENTIFICATION
#
# TITLE MMSEND
# AUTHOR GREGG R FABREGAS
# VERSION A.01
# DATE 30-JUL-1984
# LANGUAGE RATFOR
# SYSTEM VAX 11/780
#
#
# UPDATE
#
# UPDATE # 1
# AUTHOR GREGG R FABREGAS
# DATE 07-OCT-1985
# VERSION A.01
# PURPOSE ADD CALLS TO STARTTIMER
# AND CHECKTIMER TO
# MONITOR ELAPSED CPU TIME.
#
# UPDATE # 2
# AUTHOR GREGG R FABREGAS
# DATE 14-OCT-1985
# VERSION A.01
# PURPOSE ADD CHECK FOR THE MMDMAE
# FLAG AND THE ABILITY
# TO SEND THE ARRAY USING
# A DMA FACILITY.
#
# UPDATE # 3
# AUTHOR GREGG R FABREGAS
# DATE 30-MAR-1986
# VERSION A.01
# PURPOSE FIX CALCULATION OF 'NPAGES'.
#
# PURPOSE
#
# MINIMAP MEMORY IS VIEWED AS A
# CONTIGUOUS ARRAY OF MAP-INTEGERS,
# POSSIBLY SUPERIMPOSED BY VALUES OF
# OTHER DATA TYPES. THIS
# ROUTINES STORES VALUES IN A GIVEN
# DATA TYPE INTO MINIMAP MEMORY
# STARTING AT THE SPECIFIED ADDRESS.
# NO INTERPRETATION IS DONE ON
# THE VALUE PASSED. THE MODE IS
```

```

#         USED TO DETERMINE HOW MANY BITS
#         ARE TO BE TRANSFERRED.
#
# ENTRY POINT
#
#         CALL MMSEND( VAL, COUNT, MODE, ADR, IEV, ERRET )
#
# ARGUMENT LISTING
#
#         VAL      VARIES      VALUE TO BE PASSED TO
#                                     THE MINIMAP.  THE DATA TYPE
#                                     IS SPECIFIED BY THE MODE.
#
#         COUNT    INTEGER     NUMBER OF VALUES TO
#                                     PASS TO THE MINIMAP.
#                                     COUNT MUST BE POSITIVE.
#
#         MODE     INTEGER     SPECIFIES THE DATA
#                                     TYPE OF THE VALUE.
#
#         ADR      INTEGER     DESTINATION ADDRESS
#                                     IN MINIMAP MEMORY.
#                                     THE FIRST ADDRESS (1)
#                                     CORRESPONDS TO THE
#                                     FIRST AVAILABLE MEMORY
#                                     ADDRESS, WHICH IS
#                                     DETERMINED BY THE
#                                     VALUE OF .WINDOWBASE.
#
#         IEV      INTEGER     INTEGER EVENT VARIABLE.
#                                     RETURNS ONE OF THE
#                                     FOLLOWING CODES WHEN DONE:
#                                     .OK   IF THE CALL SUCCEEDS,
#                                     -2003 IF COUNT IS
#                                     NOT POSITIVE,
#                                     -2022 IF UNABLE TO MODIFY
#                                     THE MAPPING REGISTERS,
#                                     -2023 IF THE MMINIT ROUTINE
#                                     HAS NOT BEEN
#                                     CALLED PRIOR TO
#                                     THIS ROUTINE,
#                                     -2024 IF THE MMINIT ROUTINE
#                                     HAD FAILED,
#                                     -2030 IF THE MINIMAP IS NOT
#                                     READY FOR EXECUTION,
#                                     -2031 IF ANY ADDRESSES
#                                     FOR DATA TRANSFER
#                                     FALL OUTSIDE THE
#                                     MEMORY LIMITS,AND
#                                     -2040 IF THE DATA MODE
#                                     IS UNDEFINED.
#
#         ERRET    LABEL       LABEL IN CALLING ROUTINE FOR
#                                     ALTERNATE ERROR RETURN.
#
# INCLUDE FILES/COMMONS

```

```

#
#       GIPSY      INCLUDE      GIPSY GENERAL SYMBOL DEF'NS
#       GIPCOM     COMMON      GIPSY COMMAND LINE COMMON DATA
#       TTCOM      COMMON      FD'S FOR TERMINAL & RUNFILE IO
#       MINIMAP    COMMON      COMMON FOR THE ADMINISTRATION
#                               OF THE MINIMAP
#                               SUPPORT SOFTWARE.
#       WINDOW     COMMON      COMMON FOR THE SHARED
#                               MEMORY WINDOW.
#
# HARDWARE REQUIRED
#
#       A CSPI MINIMAP ARRAY PROCESSOR MUST BE OPENED
#       AND READY FOR EXECUTION.
#
# LANGUAGE FEATURES REQUIRED
#
#       ALTERNATE RETURNS AND MEMORY-MAPPED I/O ARE NEEDED.
#
# PROGRAM ENVIROMENT
#
#       ANY NUMBER OF CALLS MAY BE
#       MADE TO THIS ROUTINE AFTER A
#       SUCCESSFUL CALL TO MMINIT AND
#       MMLOAD. A CALL TO MMLOAD IS MADE
#       WITHIN MMINIT.
#
# ALGORITHM
#
#       (1) COMPUTE THE STARTING AND ENDING
#           ADDRESSES USED IN THE ARRAY,
#       (2) ALIGN THE MEMORY WINDOW FOR
#           EACH PAGE TO BE MODIFIED,
#       (3) TRANSFER THE BULK OF THE ARRAY
#           IN MAP-INTEGERS INCREMENTS, AND
#       (4) TRANSFER THE REMAINING BITS.
#
# LIMITATIONS
#
#       VALUES ARE PASSED ONE MAP-INTEGERS AT A TIME.
#
# ROUTINES CALLED
#
#       PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#                   (GIPSY PRIMITIVE)
#       PUTF       MACHINE INDEPENDENT FORMATTED OUTPUT
#                   (GIPSY PRIMITIVE)
#       MOD        REMAINDER ( ARG1 MODULO ARG2 )
#                   (BUILTIN ROUTINE)
#       SETMMR     HSL ROUTINE: SET MEMORY MAPPING
#                   REGISTERS. (GIPSY LIBRARY)

```

```

#      OSSIEV      PUT EVENT VARIABLE (IEV) INTO SAVE
#                  AREA (GIPSY OS KERNEL)
#      APACK       PACK AN ARRAY INTO A BIT STRING
#                  (GIPSY OS KERNEL)
#      PPOP        POP PROGRAM NAME FROM THE ERROR STACK
#                  (GIPSY PRIMITIVE)
#      OSGIEV      GET EVENT VARIABLE (IEV) FROM OS SAVE
#                  AREA (GIPSY OS KERNEL)
#
# REMARKS
#
#      THIS ROUTINE IS DESIGNED SOLELY
#      FOR THE VAX-11/VMS OPERATING
#      SYSTEM.  THERE ARE A NUMBER OF
#      SYSTEM-DEPENDENT FEATURES, SUCH
#      AS MEMORY-MAPPED I/O AND INTEGERS
#      KNOWN TO USE FOUR BYTES OF
#      STORAGE.  THE ROUTINE WAS DESIGNED FOR
#      EXECUTION SPEED, RATHER THAN PORTABILITY.
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
#      INCLUDE GIPSY
#
#      SUBROUTINE MMSEND( VAL, COUNT, MODE, ADR, IEV, * )
#
#
#      INCLUDE GIPCOM
#      INCLUDE TTCOM
#      INCLUDE MINIMAP
#      INCLUDE WINDOW
#      DEFINE(NEWPAGE,CALL SETMMR( MMPHYS, IEV );
#              IF ( .OK -= OSSIEV( IEV, "SETMMR called" ) )
#                  GOTO 9060;
#      IF ( TRACE )
#          CALL PUTF( FDRUNO,
#                   "AP trace: set mapping register to %I%N",
#                   IPHYS );)
#
#      INTEGER COUNT, MODE, ADR, IEV, LEN
#      INTEGER MBITS, MSIZE, STPAGE, STOFFS, ENDPAG
#      INTEGER ENDOFS, MCOUNT, I, J, SIZE, NPAGES, IPHYS, START
#      MAPINTEGER VAL( * )
#      INTEGER OSSIEV, MOD, OSGIEV, ICEIL, APACK, INT
#      LOGICAL TRACE
#
#      EQUIVALENCE ( TRACE, SFLAG( .A ) ), ( IPHYS, MMPHYS )
#
#      PUSH ROUTINE NAME ONTO ERROR TRACE STACK.

```

```

#
CALL PPUSH( "MMSSEND" )
IF ( TRACE )
    $(
    CALL PUTF( FDRUNO, "AP trace: MMSSEND called%N" )
    STARTTIMER
    $)
#
#           THIS ROUTINE REQUIRES A PREVIOUSLY SUCCESSFUL
#           CALL TO MMINIT AND A SUCCESSFUL MODULE LOAD.
#
IF ( - MMCHKD )
    GOTO 9000
IF ( - MMACTV )
    GOTO 9010
IF ( - MMREDY )
    GOTO 9020
#
#           VERIFY THE MODE IS LEGAL, AND THE ADDRESS
#           SPACE TO RECEIVE THE DATA IS IN BOUNDS.
#
IF ( COUNT <= 0 )
    GOTO 9030
IF ( MODE < .MODEMIN ! MODE > .MODEMAX )
    GOTO 9040
IF ( ADR < 1 ! ADR > INT( MMMAXC ) )
    GOTO 9050
SIZE = MMMODE( MODE )
MSIZE = MMMODE( .MAPINTMODE )
LEN = ICEIL( SIZE * COUNT, MSIZE )
IF ( ADR + LEN - 1 > INT( MMMAXC ) )
    GOTO 9050
#
#           COMPUTE THE STARTING AND ENDING PAGE ADDRESSES
#           AND THE OFFSETS WITHIN EACH PAGE.
#
MBITS = MOD( SIZE * COUNT, MSIZE )
IF ( MBITS > 0 )
    LEN = LEN - 1
STPAGE = 2 * ( .WINDOWMINT * ( ( ADR - 1 ) / .WINDOWMINT )
    + .WINDOWBASE )
STOFFS = MOD( ADR - 1, .WINDOWMINT ) + 1
ENDPAG = 2 * ( .WINDOWMINT * ( ( ADR + LEN - 2 ) /
    .WINDOWMINT ) + .WINDOWBASE )
ENDDOFS = MOD( ADR + LEN - 2, .WINDOWMINT ) + 1
#
#           IF DMA TRANSFERS ARE ENABLED, USE IT FOR ALL
#           FULLY WRITTEN MAPINTEGERS.
#
IF ( MMDMAE & ( - MMEXCD ) )
    $(

```

```

START = 2 * ( ADR - 1 + .WINDOWBASE )
IF ( TRACE )
    $(
        CALL PUTF( FDRUNO,
            "AP trace: using DMA transfer mechanism%N" )
        CALL PUTF( FDRUNO,
            "AP trace: address = %I; length = %I bytes%N",
            START, LEN * 2 )
    $)
CALL WRITMM( VAL, START, LEN * 2 )
MCCOUNT = LEN

#
#     TRANSFER EXTRA BITS USING DMA READ AND WRITE.
#
IF ( MBITS > 0 )
    $(
        START = START + ( 2 * ( LEN + 1 ) )
        CALL READMM( I, START, 2 )
        MCCOUNT = MCCOUNT + 1
        IF ( .OK != APACK( VAL( MCCOUNT ), I, 1, MBITS,
            .UNSIGNED ) ) GOTO 9800
        CALL WRITMM( I, START, 2 )
    $)
ELSE
    $(
#
#     ALIGN THE MEMORY WINDOW TO THE STARTING PAGE.
#
IF ( STPAGE != IPHYS )
    $(
        IPHYS = STPAGE
        NEWPAGE
    $)

#
#     INITIALIZE DATA COUNT.
#
MCCOUNT = 0

#
#     IF THE MEMORY REFERENCES FIT IN ONE PAGE, THEN
#     ONLY ONE LOOP IS NEEDED TO GENERATE ALL OF THE
#     MEMORY REFERENCES.
#
IF ( STPAGE == ENDPAG )
    $(
        IF ( TRACE )
            CALL PUTF( FDRUNO,
                "AP trace: array in one page: %I to %I%N",
                STOFFS, ENDOFS )
        IF ( LEN > 0 )
            DO I = STOFFS, ENDOFS
    $)

```

```

        $(
        MCOUNT = MCOUNT + 1
        MMWNDW( I ) = VAL( MCOUNT )
        $)
    $)
ELSE
#
#     THE ARRAY SPANS MORE THAN ONE PAGE.
#
    $(
    NPAGES = ( ENDPAG - STPAGE ) /
              ( 2 * .WINDOWMINT ) + 1
    IF ( TRACE )
        $(
            CALL PUTF( FDRUNO, "AP trace: array spans %I ",
                      NPAGES )
            CALL PUTF( FDRUNO,
                      "page%?s: start at %I, end at %I%N",
                      NPAGES, STOFFS, ENDOFS )
            $)
#
#     (1) FINISH OFF THE FIRST PAGE.
#
    DO I = STOFFS, .WINDOWMINT
        $(
            MCOUNT = MCOUNT + 1
            MMWNDW( I ) = VAL( MCOUNT )
            $)
#
#     (2) REMAP FOR EACH INTERMEDIATE PAGE AND MOVE
#         THE ENTIRE PAGE.
#
    IF ( NPAGES > 2 )
        DO J = 1, NPAGES - 2
            $(
            IPHYS = IPHYS + ( 2 * .WINDOWMINT )
            NEWPAGE
            DO I = 1, .WINDOWMINT
                $(
                    MCOUNT = MCOUNT + 1
                    MMWNDW( I ) = VAL( MCOUNT )
                    $)
            $)
#
#     (3) REMAP FOR THE LAST PAGE AND FINISH OFF THE
#         ARRAY.
#
    IPHYS = IPHYS + ( 2 * .WINDOWMINT )
    NEWPAGE
    DO I = 1, ENDOFS
        $(

```



```

        MCOUNT = MCOUNT + 1
        MMWNDW( I ) = VAL( MCOUNT )
        $)
    $)
#
#     AFTER THE ARRAY IS TRANSFERRED IN FIXED-SIZE
#     BIT STREAMS, THERE MAY BE EXTRA TRAILING BITS.
#     REMAP IF NECESSARY, AND TRANSFER THESE BITS,
#     OVERLAYING ONLY A PART OF THE NEXT ADDRESS.
#
IF ( MBITS > 0 )
    $(
        IF ( TRACE )
            CALL PUTF( FDRUNO,
                "AP trace: extra bits left in array%N" )
        I = ENDOFS + 1
        IF ( ENDOFS == .WINDOWMINT )
            $(
                IPHYS = IPHYS + ( 2 * .WINDOWMINT )
                NEWPAGE
                I = 1
            $)
        MCOUNT = MCOUNT + 1
        IF ( .OK ~= APACK( VAL( MCOUNT ), MMWNDW( I ), 1,
            MBITS, .UNSIGNED ) )
            GOTO 9800
        $)
    $)
#
#     FINISHED.
#
IF ( TRACE )
    CALL PUTF( FDRUNO, "AP trace: wrote to %I locations%N",
        MCOUNT )
#
CHECKTIMER
CALL PPOP
RETURN
#
#     ERROR CONDITIONS:
#
#     MMINIT ROUTINE HAS NOT BEEN EXECUTED YET.
#
9000 CONTINUE
IEV = -2023
GOTO 9900
#
#     MINIMAP INITIALIZATION HAS FAILED.
#
9010 CONTINUE
IEV = -2024

```

```
GOTO 9900
#
#           MINIMAP NOT READY FOR EXECUTION.
#
9020 CONTINUE
IEV = -2030
GOTO 9900
#
#           COUNT IS NOT POSITIVE.
#
9030 CONTINUE
IEV = -2003
GOTO 9900
#
#           MMSSEND DETECTED AN ILLEGAL DATA MODE.
#
9040 CONTINUE
IEV = -2040
GOTO 9900
#
#           MINIMAP ADDRESS OUT OF BOUNDS.
#
9050 CONTINUE
IEV = -2031
GOTO 9900
#
#           CALL TO SETMMR FAILED.
#
9060 CONTINUE
IEV = -2022
GOTO 9900
#
#           RETRIEVE ERROR FROM LOWER GIPSY ROUTINE.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#
#           TAKE THE ALTERNATE ERROR RETURN.
#
9900 CONTINUE
CHECKTIMER
RETURN 1
#
#
END
```

B.8 MMRECV Routine

```
#--MMRECV  GET DATA FOR MINIMAP SUPPORT SOFTWARE
#
#  IDENTIFICATION
#
#      TITLE           MMRECV
#      AUTHOR          GREGG R FABREGAS
#      VERSION         A.01
#      DATE            30-JUL-1984
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#
#  UPDATE
#
#      UPDATE          # 1
#      AUTHOR          GREGG R FABREGAS
#      DATE            07-OCT-1985
#      VERSION         A.01
#      PURPOSE         ADD CALLS TO STARTTIMER
#                     AND CHECKTIMER TO
#                     MONITOR ELAPSED CPU TIME.
#
#      UPDATE          # 2
#      AUTHOR          GREGG R FABREGAS
#      DATE            14-OCT-1985
#      VERSION         A.01
#      PURPOSE         ADD CHECK FOR THE MMDMAE
#                     FLAG AND THE ABILITY
#                     TO RECEIVE THE ARRAY
#                     USING A DMA FACILITY.
#
#      UPDATE          # 3
#      AUTHOR          GREGG R FABREGAS
#      DATE            30-MAR-1986
#      VERSION         A.01
#      PURPOSE         FIX CALCULATION OF 'NPAGES'.
#
#  PURPOSE
#
#      MINIMAP MEMORY IS VIEWED AS A
#      CONTIGUOUS ARRAY OF MAP-INTEGERS,
#      POSSIBLY SUPERIMPOSED BY VALUES
#      OF OTHER DATA TYPES.  THIS
#      ROUTINES RETRIEVES VALUES IN A
#      GIVEN DATA TYPE FROM MINIMAP MEMORY
#      STARTING AT THE SPECIFIED ADDRESS.
#      NO INTERPRETATION IS DONE ON
#      THE VALUES PASSED.  THE MODE IS
```

```

#         USED TO DETERMINE HOW MANY BITS
#         ARE TO BE TRANSFERRED.
#
# ENTRY POINT
#
#         CALL MMRECV( VAL, COUNT, MODE, ADR, IEV, ERRET )
#
# ARGUMENT LISTING
#
#         VAL      VARIES      VALUE TO BE PASSED
#                                     TO THE MINIMAP.  THE
#                                     DATA TYPE IS SPECIFIED
#                                     BY THE MODE.
#
#         COUNT    INTEGER     NUMBER OF VALUES TO
#                                     PASS TO THE MINIMAP.
#                                     COUNT MUST BE POSITIVE.
#
#         MODE     INTEGER     SPECIFIES THE DATA
#                                     TYPE OF THE VALUE.
#
#         ADR      INTEGER     SOURCE ADDRESS IN
#                                     MINIMAP MEMORY.
#                                     THE FIRST ADDRESS (1)
#                                     CORRESPONDS TO THE
#                                     FIRST AVAILABLE MEMORY
#                                     ADDRESS, WHICH IS
#                                     DETERMINED BY THE
#                                     VALUE OF .WINDOWBASE.
#
#         IEV      INTEGER     INTEGER EVENT VARIABLE.
#                                     RETURNS ONE OF THE
#                                     FOLLOWING CODES WHEN DONE:
#                                     .OK   IF THE CALL SUCCEEDS,
#                                     -2003 IF COUNT IS
#                                     NOT POSITIVE,
#                                     -2022 IF UNABLE TO MODIFY THE
#                                     MAPPING REGISTERS,
#                                     -2023 IF THE MMINIT ROUTINE
#                                     HAS NOT BEEN CALLED
#                                     PRIOR TO THIS ROUTINE,
#                                     -2024 IF THE MMINIT ROUTINE
#                                     HAD FAILED,
#                                     -2030 IF THE MINIMAP IS NOT
#                                     READY FOR EXECUTION,
#                                     -2031 IF ANY ADDRESSES FOR
#                                     DATA TRANSFER
#                                     FALL OUTSIDE THE
#                                     MEMORY LIMITS, AND
#                                     -2040 IF THE DATA MODE
#                                     IS UNDEFINED.
#
#         ERRET    LABEL       LABEL IN CALLING ROUTINE FOR
#                                     ALTERNATE ERROR RETURN.
#
# INCLUDE FILES/COMMONS

```

```

#
#       GIPSY      INCLUDE      GIPSY GENERAL SYMBOL DEF'NS
#       GIPCOM     COMMON      GIPSY COMMAND LINE COMMON DATA
#       TTCOM      COMMON      FD'S FOR TERMINAL & RUNFILE IO
#       MINIMAP    COMMON      COMMON FOR THE ADMINISTRATION
#                               OF THE MINIMAP
#                               SUPPORT SOFTWARE.
#       WINDOW     COMMON      COMMON FOR THE SHARED
#                               MEMORY WINDOW.
#
# HARDWARE REQUIRED
#
#       A CSPI MINIMAP ARRAY PROCESSOR MUST BE OPENED
#       AND READY FOR EXECUTION.
#
# LANGUAGE FEATURES REQUIRED
#
#       ALTERNATE RETURNS AND MEMORY-MAPPED I/O ARE NEEDED.
#
# PROGRAM ENVIROMENT
#
#       ANY NUMBER OF CALLS MAY BE
#       MADE TO THIS ROUTINE AFTER A
#       SUCCESSFUL CALL TO MMINIT AND
#       MMLOAD.  A CALL TO MMLOAD IS MADE
#       WITHIN MMINIT.
#
# ALGORITHM
#
#       (1) COMPUTE THE STARTING AND ENDING
#           ADDRESSES USED IN THE ARRAY,
#       (2) ALIGN THE MEMORY WINDOW FOR
#           EACH PAGE TO BE MODIFIED,
#       (3) TRANSFER THE BULK OF THE ARRAY
#           IN MAP-INTEGER INCREMENTS, AND
#       (4) TRANSFER THE REMAINING BITS.
#
# LIMITATIONS
#
#       VALUES ARE PASSED ONE MAP-INTEGER AT A TIME.
#
# ROUTINES CALLED
#
#       PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#                 (GIPSY PRIMITIVE)
#       PUTF       MACHINE INDEPENDENT FORMATTED OUTPUT
#                 (GIPSY PRIMITIVE)
#       MOD        REMAINDER ( ARG1 MODULO ARG2 )
#                 (BUILTIN ROUTINE)
#       SETMMR     HSL ROUTINE: SET MEMORY MAPPING
#                 REGISTERS. (GIPSY LIBRARY)
#

```

```

#      OSSIEV      PUT EVENT VARIABLE (IEV) INTO SAVE
#                  AREA (GIPSY OS KERNEL)
#      APACK       PACK AN ARRAY INTO A BIT STRING
#                  (GIPSY OS KERNEL)
#      PPOP        POP PROGRAM NAME FROM THE ERROR STACK
#                  (GIPSY PRIMITIVE)
#      OSGIEV      GET EVENT VARIABLE (IEV) FROM OS SAVE
#                  AREA (GIPSY OS KERNEL)
#
# REMARKS
#
#      THIS ROUTINE IS DESIGNED SOLELY
#      FOR THE VAX-11/VMS OPERATING
#      SYSTEM. THERE ARE A NUMBER OF
#      SYSTEM-DEPENDENT FEATURES, SUCH
#      AS MEMORY-MAPPED I/O AND INTEGERS
#      KNOWN TO USE FOUR BYTES OF
#      STORAGE. THE ROUTINE WAS DESIGNED
#      FOR EXECUTION SPEED, RATHER
#      THAN PORTABILITY.
#
#
#*****
#
#      INCLUDE GIPSY
#
#      SUBROUTINE MMRECV( VAL, COUNT, MODE, ADR, IEV, * )
#
#      INCLUDE GIPCOM
#      INCLUDE TTCOM
#      INCLUDE MINIMAP
#      INCLUDE WINDOW
#      DEFINE(NEWPAGE,CALL SETMMR( IPHYS, IEV );
#            IF ( .OK ^= OSSIEV( IEV, "SETMMR called" ) )
#              GOTO 9060;
#            IF ( TRACE )
#              CALL PUTF( FDRUNO,
#                "AP trace: set mapping register to %I%N",
#                IPHYS );)
#
#      INTEGER COUNT, MODE, ADR, IEV, LEN
#      INTEGER MBITS, MSIZE, STPAGE, STOFFS, ENDPAG
#      INTEGER ENDOFS, MCOUNT, I, J, SIZE, NPAGES, IPHYS, START
#      MAPINTEGER VAL( * )
#      INTEGER OSSIEV, MOD, OSGIEV, ICEIL, APACK, INT
#      LOGICAL TRACE
#
#      EQUIVALENCE ( TRACE, SFLAG( .A ) ), ( IPHYS, MMPHYS )
#
#

```

```

#           PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
CALL PPUSH( "MMRECV" )
IF ( TRACE )
    $(
        CALL PUTF( FDRUNO, "AP trace: MMRECV called%N" )
        STARTTIMER
    $)
#
#           THIS ROUTINE REQUIRES A PREVIOUSLY SUCCESSFUL
#           CALL TO MMINIT AND A SUCCESSFUL MODULE LOAD.
#
IF ( - MMCHKD )
    GOTO 9000
IF ( - MMACTV )
    GOTO 9010
IF ( - MMREDY )
    GOTO 9020
#
#           VERIFY THE MODE IS LEGAL, AND THE ADDRESS
#           SPACE TO RETRIEVE THE DATA IS IN BOUNDS.
#
IF ( COUNT <= 0 )
    GOTO 9030
IF ( MODE < .MODEMIN ! MODE > .MODEMAX )
    GOTO 9040
IF ( ADR < 1 ! ADR > INT( MMMAXC ) )
    GOTO 9050
SIZE = MMMODE( MODE )
MSIZE = MMMODE( .MAPINTMODE )
LEN = ICEIL( SIZE * COUNT, MSIZE )
IF ( ADR + LEN - 1 > INT( MMMAXC ) )
    GOTO 9050
#
#           COMPUTE THE STARTING AND ENDING PAGE ADDRESSES
#           AND THE OFFSETS WITHIN EACH PAGE.
#
MBITS = MOD( SIZE * COUNT, MSIZE )
IF ( MBITS > 0 )
    LEN = LEN - 1
STPAGE = 2 * ( .WINDOWMINT * ( ( ADR - 1 ) / .WINDOWMINT )
    + .WINDOWBASE )
STOFFS = MOD( ADR - 1, .WINDOWMINT ) + 1
ENDPAG = 2 * ( .WINDOWMINT * ( ( ADR + LEN - 2 ) /
    .WINDOWMINT ) + .WINDOWBASE )
ENDOFFS = MOD( ADR + LEN - 2, .WINDOWMINT ) + 1
#
#           IF DMA TRANSFERS ARE ENABLED, USE IT FOR FULLY
#           READ MAPINTEGERS.
#
IF ( MMDMAE & ( - MMEXCD ) )

```

```

$(
START = 2 * ( ADR - 1 + .WINDOWBASE )
IF ( TRACE )
    $(
    CALL PUTF( FDRUNO,
              "AP trace: using DMA transfer mechanism%N" )
    CALL PUTF( FDRUNO,
              "AP trace: address = %I; length = %I bytes%N",
              START, LEN * 2 )
    $)
CALL READMM( VAL, START, LEN * 2 )
MCOUNT = LEN
#
#         READ NEXT MAPINTEGER IF EXTRA BITS MUST BE
#         EXTRACTED AND DEPOSITED INTO THE HOST ARRAY.
#
IF ( MBITS > 0 )
    $(
    START = START + ( 2 * ( LEN + 1 ) )
    CALL READMM( I, START, 2 )
    MCOUNT = MCOUNT + 1
    IF ( .OK != APACK( I, VAL( MCOUNT ), 1,
                     MBITS, .UNSIGNED ) )
        GOTO 9800
    $)
    $)
ELSE
    $(
#
#         ALIGN THE MEMORY WINDOW TO THE STARTING PAGE.
#
IF ( STPAGE != IPHYS )
    $(
    IPHYS = STPAGE
    NEWPAGE
    $)
#
#         INITIALIZE DATA COUNT.
#
MCOUNT = 0
#
#         IF THE MEMORY REFERENCES FIT IN ONE PAGE, THEN
#         ONLY ONE LOOP IS NEEDED TO GENERATE ALL OF THE
#         MEMORY REFERENCES.
#
IF ( STPAGE == ENDPAG )
    $(
    IF ( TRACE )
        CALL PUTF( FDRUNO,
                  "AP trace: array in one page: %I to %I%N",
                  STOFFS, ENDOFS )
    $)

```



```

IF ( LEN > 0 )
    DO I = STOFFS, ENDOFS
    $(
    MCOUNT = MCOUNT + 1
    VAL( MCOUNT ) = MMWNDW( I )
    $)
    $)
ELSE
#
#     THE ARRAY SPANS MORE THAN ONE PAGE.
#
$(
NPAGES = ( ENDPAG - STPAGE ) /
( 2 * .WINDOWMINT ) + 1
IF ( TRACE )
    $(
    CALL PUTF( FDRUNO, "AP trace: array spans %I ",
    NPAGES )
    CALL PUTF( FDRUNO,
    "page%?s: start at %I, end at %I%N", NPAGES,
    STOFFS, ENDOFS )
    $)
#
#     (1) FINISH OFF THE FIRST PAGE.
#
DO I = STOFFS, .WINDOWMINT
    $(
    MCOUNT = MCOUNT + 1
    VAL( MCOUNT ) = MMWNDW( I )
    $)
#
#     (2) REMAP FOR EACH INTERMEDIATE PAGE AND MOVE
#     THE ENTIRE PAGE.
#
IF ( NPAGES > 2 )
    DO J = 1, NPAGES - 2
    $(
    IPHYS = IPHYS + ( 2 * .WINDOWMINT )
    NEWPAGE
    DO I = 1, .WINDOWMINT
    $(
    MCOUNT = MCOUNT + 1
    VAL( MCOUNT ) = MMWNDW( I )
    $)
    $)
#
#     (3) REMAP FOR THE LAST PAGE AND FINISH OFF THE
#     ARRAY.
#
IPHYS = IPHYS + ( 2 * .WINDOWMINT )
NEWPAGE

```

```

DO I = 1, ENDOFS
  $(
    MCOUNT = MCOUNT + 1
    VAL( MCOUNT ) = MMWNDW( I )
  $)
$)

#
# AFTER THE ARRAY IS TRANSFERRED IN FIXED-SIZE
# BIT STREAMS, THERE MAY BE EXTRA TRAILING BITS.
# REMAP IF NECESSARY, AND TRANSFER THESE BITS,
# OVERLAYING ONLY A PART OF THE NEXT ADDRESS.
#

IF ( MBITS > 0 )
  $(
    IF ( TRACE )
      CALL PUTF( FDRUNO,
        "AP trace: extra bits left in array%N" )
    I = ENDOFS + 1
    IF ( ENDOFS == .WINDOWMINT )
      $(
        IPHYS = IPHYS + ( 2 * .WINDOWMINT )
        NEWPAGE
        I = 1
      $)
    MCOUNT = MCOUNT + 1
    IF ( .OK /= APACK( MMWNDW( I ), VAL( MCOUNT ),
      1, MBITS, .UNSIGNED ) )
      GOTO 9800
  $)
$)

#
# FINISHED.
#
IF ( TRACE )
  CALL PUTF( FDRUNO, "AP trace: read from %I locations%N",
    MCOUNT )

#
CHECKTIMER
CALL PPOP
RETURN

#
# ERROR CONDITIONS:
#
# MMINIT ROUTINE HAS NOT BEEN EXECUTED YET.
#
9000 CONTINUE
IEV = -2023
GOTO 9900

#
# MINIMAP INITIALIZATION HAS FAILED.
#

```

```
9010 CONTINUE
IEV = -2024
GOTO 9900
#
#           MINIMAP NOT READY FOR EXECUTION.
#
9020 CONTINUE
IEV = -2030
GOTO 9900
#
#           COUNT IS NOT POSITIVE.
#
9030 CONTINUE
IEV = -2003
GOTO 9900
#
#           MMRECV DETECTED AN ILLEGAL DATA MODE.
#
9040 CONTINUE
IEV = -2040
GOTO 9900
#
#           MINIMAP ADDRESS OUT OF BOUNDS.
#
9050 CONTINUE
IEV = -2031
GOTO 9900
#
#           CALL TO SETMMR FAILED.
#
9060 CONTINUE
IEV = -2022
GOTO 9900
#
#           RETRIEVE ERROR FROM LOWER GIPSY ROUTINE.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#
#           TAKE THE ALTERNATE ERROR RETURN.
#
9900 CONTINUE
CHECKTIMER
RETURN 1
#
#
END
```

C APPENDIX C: LISTINGS OF GROUP THREE ROUTINES

What follows is the RATFOR source code for the OSMMAP, MMINTR, and MMHSLD routines. OSMMAP is located in the GIPSY_OS directory, and MMINTR and MMHSLD are located in the GIPSY_PRIM directory in the GIPSY package.

C.1 OSMMAP Routine

```
#--OSMMAP          MACHINE-DEPENDENT CODE FOR MINIMAP SUPPORT
#
# IDENTIFICATION
#
#     TITLE          OSMMAP
#     AUTHOR         GREGG R FABREGAS
#     VERSION        A.01
#     DATE           28-JUL-1984
#     LANGUAGE       RATFOR
#     SYSTEM         VAX 11/780
#
#
# UPDATE
#
#     UPDATE         # 1
#     AUTHOR         GREGG R FABREGAS
#     DATE           13-OCT-1985
#     VERSION        A.01
#     PURPOSE        ADD .DMAENABLED OPERATION TO CHECK
#                   SETTING OF THE GIPSY$APDMA SYMBOL.
#
#     UPDATE         # 2
#     AUTHOR         GREGG R FABREGAS
#     DATE           23-JAN-1986
#     VERSION        A.01
#     PURPOSE        CHANGE MINI-MAP DEVICE REFERENCES
#                   FROM "MM:" TO "GIPSY_MINIMAP:".
#
# PURPOSE
#
#     AS FAR AS FEASIBILITY PERMITS,
#     ALL MACHINE-DEPENDENT CODE NEEDED
#     BY THE MINIMAP SUPPORT SOFTWARE
#     IS LOCALIZED IN THIS ROUTINE.
#     THIS REDUCES THE EXTENT OF
#     MODIFICATIONS NEEDED WHEN THE CODE IS
#     MOVED TO ANOTHER COMPUTER SYSTEM.
#     EACH OPERATION IS IDENTIFIED
#     BY A UNIQUE CODE. THIS IS THE
#     ONLY REQUIRED ARGUMENT FOR EVERY
#     CALL. EACH OPERATION REQUIRES
#     ITS OWN SPECIFIC OPERANDS.
#
# ENTRY POINT
#
#     IEV = OSMMAP( OP [, MODULE [, RETVAL [, COMPAR ]])
#     NOTE: ARGUMENTS IN SQUARE BRACKETS ([]) ARE
#           OPTIONAL.
```



```

#           .MAPMEMSIZE  TO RETURN
#           MINIMAP MEMORY
#           SIZE, IN
#           MAP-INTEGERS,
#           .ALLOCATE    TO ALLOCATE A
#           MINIMAP FOR
#           PRIVATE USE,
#           .DEALLOCATE  TO RETURN THE
#           MINIMAP TO THE
#           FREE DEVICE
#           POOL, AND
#           .DMAENABLED  TO TEST THE
#           SETTING, IF ANY,
#           OF THE DMA FLAG,
#           GIPSY$APDMA.
#
#           MODULE  VARIES      FIRST OPTIONAL ARGUMENT.
#                               CAN BE A NAME
#                               OF A LOAD MODULE,
#                               AN INPUT VALUE FOR
#                               CONVERSION, A
#                               TWO-ELEMENT MAP-INTEGERS
#                               ARRAY, OR MEMORY SIZE
#                               IN MAP-INTEGERS,
#                               EXPRESSED AS A REAL VALUE.
#           RETVAL  VARIES      SECOND OPTIONAL ARGUMENT.
#                               RETURNED VALUE FROM CONVERTED
#                               INPUT, OR THE SECOND
#                               TWO-ELEMENT ARRAY.
#           COMPAR  INTEGER     THIRD OPTIONAL ARGUMENT.
#                               RETURNS -1 IF THE FIRST ARRAY
#                               IS LESS THAN THE SECOND,
#                               1 IF GREATER, AND 0 IF EQUAL.
#
#           INCLUDE FILES/COMMONS
#
#           GIPSY    INCLUDE    GIPSY GENERAL SYMBOL DEF'NS
#
#           HARDWARE REQUIRED
#
#           A CSPI MINIMAP ARRAY PROCESSOR MUST BE OPENED.
#
#           LANGUAGE FEATURES REQUIRED
#
#           A FORTRAN OPEN IS USED TO SELECT A
#           SPECIFIC LOAD MODULE.
#
#           PROGRAM ENVIROMENT
#
#           THERE SHOULD BE NO NEED FOR THE
#           .LOADMODULE OPERATION OUTSIDE OF
#           THE MMLoad ROUTINE.  THE DATA

```

```

#      CONVERSIONS AND MAP-INTEGER ARRAY
#      OPERATIONS MAY BE EXECUTED ANYTIME.
#      MINIMAP MEMORY SIZE CAN BE
#      DETERMINED ON AN OPEN MINIMAP.
#
# LIMITATIONS
#
#      NO CHECK IS MADE ON THE STATE OF
#      THE MINIMAP FOR ANY OPERATION.
#
# ROUTINES CALLED
#
#      OSPUSH      PUSH PROGRAM NAME ON STACK
#                  (GIPSY OS KERNEL)
#      OSFNAM      CONSTRUCT SYSTEM DEPENDENT FILE NAME
#                  (GIPSY OS KERNEL)
#      OSPATH      MAKE SYSTEM DEPENDENT SEARCH FILE NAME
#                  (GIPSY OS KERNEL)
#      OSPCHR      PACK CHARACTER ARRAY INTO "CHAR" ARRAY
#                  (GIPSY OS KERNEL)
#      MIGLUN      ALLOCATE A LOGICAL UNIT NUMBER (LUN)
#                  (GIPSY OS KERNEL)
#      ERRFIX      CONVERT ERROR FROM SYSTEM TO OS STANDARD
#                  (GIPSY OS KERNEL)
#      LOADMM      HSL ROUTINE: LOAD MODULE INTO MINIMAP.
#                  (GIPSY LIBRARY)
#      MIPLUN      DEALLOCATE A LOGICAL UNIT NUMBER (LUN)
#                  (GIPSY OS KERNEL)
#      OSSIEV      PUT EVENT VARIABLE (IEV) INTO SAVE
#                  AREA (GIPSY OS KERNEL)
#      AUPACK      UNPACK A BIT STRING INTO AN ARRAY
#                  (GIPSY OS KERNEL)
#      APACK       PACK AN ARRAY INTO A BIT STRING
#                  (GIPSY OS KERNEL)
#      SIZEMM      HSL ROUTINE: RETURN SIZE OF MINIMAP
#                  MEMORY. (GIPSY LIBRARY)
#      OSPOP       POP ROUTINE NAME OFF STACK
#                  (GIPSY OS KERNEL)
#      OSGIEV      GET EVENT VARIABLE (IEV) FROM OS SAVE
#                  AREA (GIPSY OS KERNEL)
#
#
# *****
#
# INCLUDE GIPSY
#
# INTEGER FUNCTION OSMMAP( OP, MODULE, RETVAL, COMPAR )
#
#
# INTEGER OP, MODULE( 7 ), IEV, LUN, JEV

```



```

INTEGER I, RETVAL, COMPAR, COLON
CHARACTER PATH( 14 ), MODNAM( .FILENAMELENGTH )
CHARACTER NULLFD( .FDLENGTH )
PSCHARACTER PSMNAM * .FILENAMELENGTH, STRING * 256
REAL R
LOGICAL FOUND, SYS$ALLOC, SYS$DALLOC, FLAG
LOGICAL LIB$GET_SYMBOL, OTS$CVT_TI_L
INTEGER OSFNAM, OSPATH, OSPCHR, MIGLUN, OSSIEV
INTEGER IAND, JZEXT, OSGIEV, MOD, AUPACK, APACK
#
EQUIVALENCE ( I, R )
#
DATA PATH / EVAL( "GIPSY_MAPPATH" ) /
DATA COLON / ':' /
#
#           PUSH ROUTINE NAME ONTO KERNEL TRACE STACK.
#
CALL OSPUSH( "OSMMAP" )
#
#           EACH FUNCTION CODE IN OP HAS ITS OWN
#           OPERATION DEFINITION.
#
CASE ( OP )
  $(
#
#           LOADING A MODULE:
#
#           (1) CONSTRUCT A FILE SPECIFICATION TEMPLATE
#               FROM THE LOGICAL NAME IN "PATH" (OSPATH)
#               AND THE .USRMAP TEMPLATE (OSFNAM),
#           (2) FIND THE FIRST FILE THAT FITS THE TEMPLATE,
#           (3) OPEN THE FILE WITH AN ALLOCATED LUN,
#           (4) CALL LOADMM - NOTE: THIS SHOULD BE THE ONLY
#               PLACE WHERE LOADMM IS CALLED, AND
#           (5) CLOSE THE FILE AND FREE THE LUN.
#
#           CONDITION .LOADMODULE:
#           IF ( .OK ~= OSFNAM( .USRMAP, MODNAM, MODULE ) ) GOTO
9800
#           IF ( .OK ~= OSPATH( .SETPATH, PATH ) ) GOTO 9800
#           IF ( .OK ~= OSPATH( .SETROOT, MODNAM ) ) GOTO 9800
#           FOUND = .FALSE.
#           WHILE ( .OK == OSPATH( .GETPATH, MODNAM ) )
#             $(
#               FOR ( I = 1; MODNAM( I ) ~= .EOS; I = I + 1 ) ;
#               IF ( .OK ~= OSPCHR( PSMNAM, MODNAM, 1,
#                                   I - 1, -1 ) ) GOTO 9800
#               IF ( .OK ~= MIGLUN( NULLFD, LUN ) ) GOTO 9800
#               OPEN ( UNIT = LUN, FILE = PSMNAM,
#                     TYPE = 'OLD', READONLY,
#                     ACCESS = 'DIRECT', RECL = 32, IOSTAT = IEV )

```

```

CALL ERRFIX( IEV )
IF ( IEV == .OK )
  $(
    FOUND = .TRUE.
    CALL LOADMM( LUN, JEV )
    CLOSE ( UNIT = LUN )
  $)
CALL MIPLUN( NULLFD )
IF ( IEV == .OK )
  $(
    IF ( .OK == OSSIEV( JEV, "LOADMM called" ) )
      BREAK
    IF ( JEV == 101 )
      GOTO 9000
    GOTO 9010
  $)
  $)
IF ( - FOUND )
  GOTO 9020

#
# THE NEXT FOUR OPERATIONS ARE FOR DATA TYPE
# CONVERSION. THE OPERATION CODE DICTATES THE
# RETURNED DATA TYPE DESIRED:
#
# FOR .INTMODE, THE INPUT IN MODULE( 1 ) IS
# ASSUMED TO BE A MAP-INTEGGER. THE LOW-ORDER
# 16 BITS ARE SIGN-EXTENDED TO FILL AN INTEGER.
#
CONDITION .INTMODE:
  IF ( .OK ~= AUPACK( RETVAL, MODULE( 1 ), 1,
    16, .SIGNED ) ) GOTO 9800

#
# FOR .MAPINTMODE, THE INPUT IS ASSUMED TO BE AN
# INTEGER. EXTRACT THE LOW-ORDER 16 BITS AND
# PLACE THEM IN THE LOW-ORDER 16 BITS OF RETVAL.
#
CONDITION .MAPINTMODE:
  IF ( .OK ~= APACK( MODULE( 1 ), RETVAL, 1,
    16, .SIGNED ) ) GOTO 9800

#
# FOR .REALMODE AND .MAPREALMODE, COPY THE INPUT
# TO THE OUTPUT, SINCE THE MODES ARE IDENTICAL.
#
CONDITION .REALMODE, .MAPREALMODE:
  RETVAL = MODULE( 1 )

#
# FOR .CHARMODE, TAKE THE LOW-ORDER BYTE AND
# CONVERT IT TO A1 FORMAT.
#
CONDITION .CHARMODE:
  IF ( .OK ~= OSUNPK( MODULE( 1 ), RETVAL,

```

```

        1, 1, -1 ) ) GOTO 9800
#
#       FOR .ADDARRAY, THE INPUTS AND OUTPUT ARE
#       TWO-ELEMENT MAP-INTEGERS.  THE FIRST
#       INPUT ARRAY IS ADDED TO THE SECOND INPUT ARRAY
#       AND THE RESULTS IS RETURNED IN THE SECOND
#       ARRAY.
#
CONDITION .ADDARRAY:
    RETVAL = RETVAL + MODULE( 1 )
#
#       FOR .MULTARRAY, THE INPUTS AND OUTPUT ARE
#       TWO-ELEMENT MAP-INTEGERS.  THE FIRST
#       INPUT ARRAY IS MULTIPLIED WITH THE SECOND INPUT
#       ARRAY AND THE PRODUCT IS RETURNED IN THE SECOND
#       ARRAY.
#
CONDITION .MULTARRAY:
    RETVAL = RETVAL * MODULE( 1 )
#
#       FOR .ARRAYADDR, THE TWO-ELEMENT ARRAY MUST BE
#       CONVERTED TO TWO POSITIVE ELEMENTS USED AS
#       SUBSCRIPTS TO THE WORK ARRAY.  ALWAYS ADD ONE
#       TO THE HIGH-ORDER ELEMENT, AND ADD ONE AGAIN IF
#       THE LOW-ORDER ELEMENT IS NEGATIVE.  MAKE THE
#       LOW-ORDER ELEMENT POSITIVE.
#
CONDITION .ARRAYADDR:
    RETVAL = MODULE( 1 ) + 2 ** 16
    IF ( IAND( RETVAL, 2 ** 15 ) > 0 )
        RETVAL = RETVAL + 2 ** 15
#
#       FOR .COMPARE, RETURN THE SIGN OF THE DIFFERENCE
#       BETWEEN THE FIRST AND SECOND ARRAYS.
#
CONDITION .COMPARE:
    COMPAR = JISIGN( 1, MODULE( 1 ) - RETVAL )
#
#       FOR .MAPMEMSIZE, RETURN THE SIZE OF THE MINIMAP
#       IN MAP-INTEGERS.
#
CONDITION .MAPMEMSIZE:
    CALL SIZEMM( I )
    R = FLOAT( I / 2 )
    MODULE( 1 ) = I
#
#       ALLOCATE A MINIMAP AS A PRIVATE RESOURCE.  THE
#       NAME OF THE DEVICE IS RETURNED IN THE SECOND
#       ARGUMENT AS AN .EOS-TERMINATED STRING.
#
CONDITION .ALLOCATE:

```

```

IF ( ~ SYS$ALLOC( 'GIPSY_MINIMAP:', I, PSMNAM, ) )
  $(
    IEV = .NODEVIEV
    GOTO 9900
  $)
IF ( .OK ~= OSUCHR( PSMNAM, MODULE, 1,
                   6, COLON ) ) GOTO 9800
MODULE( 7 ) = .EOS

#
#       DEALLOCATE A MINIMAP AS A PRIVATE RESOURCE.
#       NO ARGUMENTS BESIDE THE OPERATION CODE IS USED.
#
CONDITION .DEALLOCATE:
  IF ( ~ SYS$DALLOC( 'GIPSY_MINIMAP:', ) )
    $(
      IEV = .NODEVIEV
      GOTO 9900
    $)

#
#       FOR .DMAENABLED, CHECK THE VALUE OF THE CLI
#       GIPSY SYMBOL, GIPSY$APDMA. IF THE VALUE
#       IS ZERO, RETURN FALSE. IF THE VALUE IS ONE,
#       RETURN TRUE, ELSE RETURN FALSE.
#
CONDITION .DMAENABLED:
  FLAG = .FALSE.
  IF ( LIB$GET_SYMBOL( 'GIPSY$APDMA', STRING ) )
    IF ( OTS$CVT_TI_L( STRING, I, %VAL( 4 ),
                     %VAL( 1 ) ) )
      FLAG = ( IAND( I, 1 ) == 1 )
  MODULE( 1 ) = JZEXT( FLAG )

#
#       ADD NEW OPERATIONS BEFORE THIS COMMENT.
#
OTHERWISE:
  GOTO 9030
$(

#
#       OPERATION COMPLETED - RETURN.
#
CALL OSSIEV( .OK, "OSMMAP succeeded" )
CALL OSPOP
RETURN ( .OK )

#
#       ERROR CONDITIONS:
#
9000 CONTINUE
IEV = OSSIEV( .MAPLOADNOREAD,
             "LOADMM failed while reading load module" )
GOTO 9900
#

```

```

9010 CONTINUE
IEV = OSSIEV( .MAPLOADFAILED, "LOADMM failed" )
GOTO 9900
#
9020 CONTINUE
IEV = OSSIEV( .NOTFOUNDIEV,
              "mini-MAP load module not found" )
GOTO 9900
#
9030 CONTINUE
IEV = OSSIEV( .ILLEGALPARAM,
              "option not LOADMODULE, [MAP]INTMODE, [MAP]REALMODE" )
GOTO 9900
#
#           ERROR OCCURRED IN LOWER ROUTINE - GET ERROR.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#
#           RETURN ERROR IEV.
#
9900 CONTINUE
IEV = OSSIEV( IEV, "OSMMAP failed" )
CALL OSPOP
RETURN ( IEV )
#
#
END

```

C.2 MMINTR Routine

#--MMINTR HANDLE SIGNALS GENERATED BY MINIMAP HSL LIBRARY

#

IDENTIFICATION

#

#	TITLE	MMINTR
#	AUTHOR	GREGG R FABREGAS
#	VERSION	A.01
#	DATE	21-JAN-1984
#	LANGUAGE	RATFOR
#	SYSTEM	VAX 11/780

#

#

UPDATE

#

#	UPDATE	# 1
#	AUTHOR	GREGG R FABREGAS
#	DATE	14-MAY-1984
#	VERSION	A.01
#	PURPOSE	REMOVE ALL REFERENCES TO HSL__CHKSUM SINCE THE SYMBOL IS NO LONGER USED IN THE HSL.

#

#

#	UPDATE	# 2
#	AUTHOR	GREGG R FABREGAS
#	DATE	30-JUL-1984
#	VERSION	A.01
#	PURPOSE	(1) ADD INCLUDE FOR MINIMAP COMMON, AND (2) ADD CHECK FOR DEVICE TIMEOUT WHEN THE MMEXCD FLAG IS SET, AND (3) ADD CHECK FOR ILLEGAL MEMORY INTERRUPTIONS FROM THE MINIMAP.

#

#

#	UPDATE	# 3
#	AUTHOR	GREGG R FABREGAS
#	DATE	07-OCT-1985
#	VERSION	A.01
#	PURPOSE	REMOVE DECLARATION OF OSTIME SINCE THE DECLARATION IS NOW MADE IN THE MINIMAP COMMON

#

#

#	UPDATE	# 4
#	AUTHOR	GREGG R FABREGAS
#	DATE	06-MAR-1986
#	VERSION	A.01


```

# INCLUDE FILES/COMMONS
#
#     GIPSY      INCLUDE      GIPSY GENERAL SYMBOL DEF'NS
#     GIPCOM     COMMON      GIPSY COMMAND LINE COMMON DATA
#     TTCOM     COMMON      FD'S FOR TERMINAL & RUNFILE IO
#
# PROGRAM ENVIROMENT
#
#     THIS ROUTINE IS ONLY EXPECTED
#     TO BE CALLED BY A CONDITION
#     HANDLER WHICH IS TRYING TO
#     DIFFERENTIATE BETWEEN INFORMATIONAL
#     MESSAGES GENERATED BY THE HOST
#     SUPPORT SOFTWARE AND OTHER CONDITIONS.
#
# ALGORITHM
#
#     THE ALTERNATE RETURN IS ONLY
#     TAKEN WHEN ONE OF THE SPECIFIED
#     INFORMATIONAL CONDITIONS IS
#     DETECTED. FAILURES IN DETECTION
#     OR CONDITIONS WHICH ARE NOT
#     INFORMATIONAL WILL PROMOTE A NORMAL RETURN.
#
# ROUTINES CALLED
#
#     PPU      PUSH PROGRAM NAME ONTO ERROR STACK
#             (GIPSY PRIMITIVE)
#     PUTF     MACHINE INDEPENDENT FORMATTED OUTPUT
#             (GIPSY PRIMITIVE)
#     %LOC     RETURNS THE ADDRESS OF THE ARGUMENT.
#             (VAX/VMS ROUTINE)
#     PPOP     POP PROGRAM NAME FROM THE ERROR STACK
#             (GIPSY PRIMITIVE)
#
# *****
#
# INCLUDE GIPSY
#
# SUBROUTINE MMINTR( ERR, IEV, * )
#
# INCLUDE GIPCOM
# INCLUDE TTCOM
# INCLUDE MINIMAP
# EXTERNAL HSL__NOREGMAP, HSL__REGMAP, HSL__CLOSED
# EXTERNAL HSL__ILGAM_ADR, SS$_CONTINUE, SS$_TIMEOUT
#
# INTEGER ERR( 4 ), IEV

```



```

LOGICAL TRACE
REAL RSECS
#
EQUIVALENCE ( TRACE, SFLAG( .A ) )
#
CALL PPUSH( "MMINTR" )
IF ( TRACE )
    CALL PUTF( FDRUNO, "AP trace: MMINTR called%N" )
#
CASE ( ERR( 2 ) )
    $(
    CONDITION %LOC( HSL__NOREGMAP ):
        IF ( TRACE )
            CALL PUTF( FDRUNO,
                "AP trace: control registers not mapped%N" )
#
    CONDITION %LOC( HSL__REGMAP ):
        IF ( TRACE )
            CALL PUTF( FDRUNO,
                "AP trace: control registers mapped%N" )
#
    CONDITION %LOC( HSL__CLOSED ):
        IF ( TRACE )
            CALL PUTF( FDRUNO, "AP trace: closed%N" )
#
    OTHERWISE:
        IF ( ERR( 2 ) == %LOC( SS$_TIMEOUT ) )
            IF ( MMEXCD )
                IF ( MMSECS > 0.0 )
                    IF ( .OK == OSTIME( RSECS, 1 ) )
                        IF ( RSECS >= MMSECS )
                            $(
                                MMTOST = .TRUE.
                                IF ( TRACE )
                                    CALL PUTF( FDRUNO,
                                        "AP trace: mini-MAP timed-out%N" )
                                BREAK
                            $)
            IF ( MMILAT > 0 &
                ERR( 2 ) == %LOC( HSL__ILGAM_ADR ) )
                $(
                    MMILAT = MMILAT - 1
                    IF ( TRACE )
                        $(
                            CALL PUTF( FDRUNO,
                                "AP trace: illegal address detected%N" )
                        $)
                IF ( MMILAT > 0 )
                    CALL PUTF( FDRUNO,
                        "AP trace: %I more allowed%N",
                        MMILAT )
                    $)

```

```
                BREAK
                $)
            CALL PPOP
            RETURN
        $)
#
# IEV = %LOC( SS$_CONTINUE )
# CALL PPOP
# RETURN 1
#
#
END
```

C.3 MMHSLD Routine

```
#--MMHSLD  HOST-SUPPORT  LIBRARY  DUMMY  ROUTINES  FOR  MINI-MAP
#
#  IDENTIFICATION
#
#      TITLE           MMHSLD
#      AUTHOR          GREGG R. FABREGAS
#      VERSION         A.01
#      DATE            22-AUG-1985
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#
#  UPDATE
#
#      UPDATE          # 1
#      AUTHOR          GREGG R FABREGAS
#      DATE            06-MAR-1986
#      VERSION         A.01
#      PURPOSE         ADD READMM AND WRITMM
#                     DUMMY ROUTINES.
#
#  PURPOSE
#
#      THESE ROUTINES ARE CALLED ONLY
#      IF THE MINI-MAP IS NOT SUPPORTED
#      BY GIPSY.  THE MINI-MAP SUPPORT
#      ROUTINES CALL THESE ROUTINE FOR
#      SYSTEM SUPPORT.  IF THE MINI-MAP
#      IS SUPPORTED, THE TRUE HSL
#      LIBRARY IS USED INSTEAD OF THESE ROUTINES.
#
#  ENTRY POINT
#
#      CALL CLOSMM( IEV )
#      CALL LOADMM( ARG1, IEV )
#      CALL OPENMM( ARG1, ARG2, ARG3, IEV )
#      CALL READMM( ARG1, ARG2, ARG3 )
#      CALL SETMMR( ARG1, IEV )
#      CALL SIZEMM( ARG1 )
#      CALL STRTMM( ARG1, IEV )
#      CALL STSGMM( ARG1 )
#      CALL STTOMM( ARG1, IEV )
#      CALL WAITMM( IEV )
#      CALL WRITMM( ARG1, ARG2, ARG3 )
#
#  ARGUMENT LISTING
#
#      ARG1    VARIES          OPERATION-DEPENDENT
```

```

#           ARG2      INTEGER      FIRST ARGUMENT
#           ARG3      INTEGER      OPERATION-DEPENDENT
#           IEV       INTEGER      SECOND ARGUMENT
#           IEV       INTEGER      OPERATION-DEPENDENT
#           IEV       INTEGER      THIRD ARGUMENT
#           IEV       INTEGER      INTEGER EVENT VARIABLE
#           IEV       INTEGER      ALWAYS RETURNS
#           IEV       INTEGER      THE ERROR, .NOTIMPLEMENTED
#
# INCLUDE FILES/COMMONS
#
#           GIPSY     INCLUDE      GIPSY GENERAL SYMBOL DEF'NS
#
# ROUTINES CALLED
#
#           OSSIEV    PUT EVENT VARIABLE (IEV) INTO SAVE
#           OSSIEV    AREA (GIPSY OS KERNEL)
#
#*****
#
# INCLUDE GIPSY
#
#--CLOSMM  FREES THE MINI-MAP FOR OTHER TASKS
#
# SUBROUTINE CLOSMM( IEV )
#
# INTEGER IEV, OSSIEV
#
# IEV = OSSIEV( .NOTIMPLEMENTED,
#              "CLOSMM/MINIMAP not implemented" )
#
# RETURN
#
# END
#
#--LOADMM  LOADS THE MINI-MAP WITH A PROGRAM GIVEN A LUN
#
# SUBROUTINE LOADMM( ARG1, IEV )
#
# INTEGER ARG1, IEV, OSSIEV
#
# IEV = OSSIEV( .NOTIMPLEMENTED,
#              "LOADMM/MINIMAP not implemented" )
#
# RETURN
#
# END

```

```

#
#--OPENMM  ASSIGNS MINI-MAP TO PRESENT TASK
#
SUBROUTINE OPENMM( ARG1, ARG2, ARG3, IEV )
#
#
INTEGER ARG1, ARG2, ARG3, IEV, OSSIEV
#
IEV = OSSIEV( .NOTIMPLEMENTED,
              "OPENMM/MINIMAP not implemented" )
RETURN
#
#
END
#
#--READMM  READ AN ARRAY FROM THE MINI-MAP USING DMA
#
SUBROUTINE READMM( ARG1, ARG2, ARG3 )
#
#
INTEGER ARG1, ARG2, ARG3
#
RETURN
#
#
END
#
#--SETMMR  SETS THE MINI-MAP MAPPING REGISTER
#
SUBROUTINE SETMMR( ARG1, IEV )
#
#
INTEGER ARG1, IEV, OSSIEV
#
IEV = OSSIEV( .NOTIMPLEMENTED,
              "SETMMR/MINIMAP not implemented" )
RETURN
#
#
END
#
#--SIZEMM  RETURN SIZE ON MINI-MAP MEMORY
#
SUBROUTINE SIZEMM( ARG1 )
#
#
INTEGER ARG1
#
ARG1 = 0
RETURN
#

```

```

#
  END
#
#--STRTMM  STARTS MINI-MAP IPU RUNNING
#
  SUBROUTINE STRTMM( ARG1, IEV )
#
#
  INTEGER ARG1, IEV, OSSIEV
#
  IEV = OSSIEV( .NOTIMPLEMENTED,
                "STRTMM/MINIMAP not implemented" )
  RETURN
#
#
  END
#
#--STSGMM  SET SIGNAL STATUS ENABLE/DISABLE FLAG
#
  SUBROUTINE STSGMM( ARG1 )
#
#
  LOGICAL ARG1
#
  RETURN
#
#
  END
#
#--STTOMM  SET TIME-OUT INTERVAL FOR MINI-MAP
#
  SUBROUTINE STTOMM( ARG1, IEV )
#
#
  INTEGER ARG1, IEV, OSSIEV
#
  IEV = OSSIEV( .NOTIMPLEMENTED,
                "STTOMM/MINIMAP not implemented" )
  RETURN
#
#
  END
#
#--WAITMM  WAITS UNTIL MINI-MAP COMPLETES ITS CURRENT TASK
#
  SUBROUTINE WAITMM( IEV )
#
#
  INTEGER IEV, OSSIEV
#
  IEV = OSSIEV( .NOTIMPLEMENTED,

```

```
                "WAITMM/MINIMAP not implemented" )
RETURN
#
#
END
#
#--WRITMM WRITE ARRAY TO MINI-MAP USING DMA
#
SUBROUTINE WRITMM( ARG1, ARG2, ARG3 )
#
#
INTEGER ARG1, ARG2, ARG3
#
RETURN
#
#
END
```

D APPENDIX D: LISTINGS OF HOST INCLUDE FILES

What follows is the RATFOR source code for the MAPDEF, MAPOPTS and WINDOW include files. They are located in the GIPSY_INCL directory in the GIPSY package.

D.1 MAPDEF Include File

```
#--MAPDEF GENERAL MINI-MAP SUPPORT DEFINITIONS
#
DEFINE(MAPINTEGER,INTEGER*2) # data type: MAP-integer
DEFINE(MAPREAL,REAL) # data type: MAP-real
DEFINE(MAPCOMPLEX,COMPLEX) # data type: MAP-complex
#
# OSMMAP options:
#
DEFINE(.LOADMODULE,-1) # load new module in mini-MAP
DEFINE(.ADDARRAY,-2) # add two-element arrays
DEFINE(.ARRAYADDR,-3) # convert array to work array
# address
DEFINE(.COMPARE,-4) # compare two-element arrays
DEFINE(.MAPMEMSIZE,-5) # return mini-MAP memory size,
# in MAP-integers
DEFINE(.MULTARRAY,-6) # multiply two-element arrays
DEFINE(.ALLOCATE,-7) # allocate mini-MAP
DEFINE(.DEALLOCATE,-8) # deallocate mini-MAP
DEFINE(.DMAENABLED,-9) # check external DMA-enable
# flag
#
# Memory definitions:
#
DEFINE(.WINDOWSIZE,2048) # size, in integers,
# of mini-MAP window
DEFINE(.WINDOWMINT,4096) # size, in MAP-integers,
# of mini-MAP window
DEFINE(.WINDOWBASE,2048) # base adr, in MAP-integers,
# of data area
DEFINE(.MAPBEGIN,0) # address to start mini-MAP
# program
DEFINE(.MAPCONTINUE,-1) # address to continue mini-MAP
# after pause
DEFINE(.MAPSCALARS,81) # starting address of mini-MAP
# scalars
DEFINE(.MAPIS,83) # starting address of mini-MAP
# integer scalars
DEFINE(.MAPRS,103) # starting address of mini-MAP
# real scalars
DEFINE(.MAPWORKBASE,143) # starting address of mini-MAP
# work array
#
# Debug status:
#
DEFINE(.DEBUGDISABLED,0) # disabled
DEFINE(.DEBUGENABLED,1) # enabled by module option
DEFINE(.DEBUGACTIVE,2) # enabled and active
DEFINE(.DEBUGOPENFILE,3) # enabled, active, and symbol
```

```
# translation table is open
#
# MMEEXEC wait option:
#
DEFINE(.NOWTARR,2) # no wait, no debug arrays
```

D.2 MAPOPTS Include File

```
!--MAPOPTS  OPTIONS FOR MINI-MAP MODULES
#
DEFINE(.OPTTIMER,1)           # set time-out interval
                              # for module execution
DEFINE(.OPTSIZE,2)           # max number of points
                              # to process per run.
DEFINE(.OPTDEBUG,3)          # enable module debugger.
#
# add new options before this comment.
#
DEFINE(.OPTMAX,10)           # maximum no._ of
options.
DEFINE(.OPTNAME,11)          # starting addr of name.
DEFINE(.OPTNAMELEN,6)        # length of name block.
DEFINE(.OPTEND,EVAL(.OPTMAX,+,6)) # length of option and
                              # name block.
DEFINE(.OPTUNDEF,UNDEFINE(.OPTENABLE))
                              # disable setting of
                              # module options.
DEFINE(.OPTENABLE,0)         # enable the setting of
                              # module options.
```

D.3 WINDOW Include File

```
#--WINDOW COMMON FOR THE SHARED MINIMAP MEMORY WINDOW
#
# IDENTIFICATION
#
# TITLE WINDOW
# AUTHOR GREGG R FABREGAS
# VERSION A.01
# DATE 31-JUL-1984
# LANGUAGE RATFOR
# SYSTEM VAX 11/780
#
#
# UPDATE
#
# UPDATE # 1
# AUTHOR GREGG R FABREGAS
# DATE 06-MAR-1986
# VERSION A.01
# PURPOSE CHANGE SIZE OF WINDOW COMMON FROM
# .WINDOWMINT TO 65536 FOR
# COMPATIBILITY WITH THE
# NEW HSL SOFTWARE.
#
# PURPOSE
#
# THE COMMON SPANS HOST ADDRESSES WHICH ARE REMAPPED,
# DURING A MEMORY REFERENCE, TO ADDRESSES WITHIN THE
# MINIMAP.
#
# ENTRY POINT
#
# NONE
#
# ROUTINES CALLED
#
# NONE
#
#
#*****
#
#
# MAPINTEGER MMWNDW( 65536 )
#
# COMMON / WINDOW / MMWNDW
```

E APPENDIX E: LISTINGS OF MINI-MAP INCLUDE FILES

What follows is the RATFOR source code for the MAPDRV, MAPCHAR, MAPID, MAPOPTS and MAPSUB include files. They are located in the GIPSY_INCL directory in the GIPSY package.

E.1 MAPDRV Include File

```

#--MAPDRV  RATFOR SUPPORT FOR MINI-MAP MODULE DRIVERS
#
#           DEFINITIONS FOR ALPHABET AND NUMERALS.
#
INCLUDE MAPCHAR
#
#           DEFINITIONS FOR PHASE 1: MODULE IDENTIFICATION
#           AND CONTEXT.
#
INCLUDE MAPID
#
#           DEFINITIONS FOR PHASE 2: RECEIVE AND FIXUP
#           ARRAY INFORMATION FROM HOST.
#
DEFINE(.ENBARRFIX,1)
DEFINE(GETARRAYS,IFDEF .ENBARRFIX GETASECT1
          UNDEFINE(.ENBARRFIX) ENDIF)
DEFINE(GETASECT1,DO ARRWK1 = 1\, 20;
          ARRLOC( ARRWK1 ) = ARRLOC( ARRWK1 )
          - 142.0;
          PAUSE;)
#
#           DEFINITIONS FOR DEBUG OPERATIONS.
#
DEFINE(DEBUGTYPE,)
DEFINE(DEBUGON,DEFINE(.ENBTRACE,1))
DEFINE(DEBUGOFF,UNDEFINE(.ENBTRACE))
DEFINE	TRACE(N),IFDEF .ENBTRACE ARRWK1 = N; ARRWK2 = 0;
          PAUSE; ENDIF)
DEFINE	TRACE2(M,N),IFDEF .ENBTRACE ARRWK1 = M; ARRWK2 = N;
          PAUSE; ENDIF)
DEFINE(DEBUG,IFDEF .ENBTRACE ARRWK1\, ARRWK2 ENDIF
          IFNDEF .ENBTRACE JUNK1\, JUNK2 ENDIF)
#
#           DEFINITIONS FOR EQUIVALENCING INTEGER, REAL,
#           AND COMPLEX ARRAYS WITH ONE OR TWO DIMENSIONS.
#
DEFINE(ARRIRC(N),INTEGER I N\, IAR N( 1\, 1 )\, IAR N L;
          INTEGER RAR N L\, CAR N L;
          REAL R N\, RAR N( 1\, 1 );
          REAL CAR N( 1\, 1 );
          PARAMETER( IAR N L = N, RAR N L = N );
          PARAMETER( CAR N L = N ); )
DEFINE(ARRAYLEN(X),ARRLEN(X L))
DEFINE(ARRIDENT(X),
  IF ( X >= 1.0 )
    $( ARRWK1 = INT( AMOD( X - 1.0\, 32767.0 ) + 1.0 );
      ARRWK2 = INT( ( ( X - 1.0 ) / 32767.0 ) + 1.0 ) $)

```

```

ELSE
  $( ARRWK1 = 0; ARRWK2 = 0 $))
DEFINE(IDESC1(N),ARRIDENT( ARRLOC( N ) )
  CALL IDENT( IAR N\, WORK( ARRWK1\, ARRWK2 )\, 1\,
    ARRLLEN( N ) );
  IFDEF .ENBTRACE PAUSE ENDIF )
DEFINE(RDESC1(N),ARRIDENT( ARRLOC( N ) )
  CALL IDENT( RAR N\, WORK( ARRWK1\, ARRWK2 )\, 1\,
    ARRLLEN( N ) / 2.0 );
  IFDEF .ENBTRACE PAUSE ENDIF )
DEFINE(CDESC1(N),ARRIDENT( ARRLOC( N ) )
  CALL IDENT( CAR N\, WORK( ARRWK1\, ARRWK2 )\, 2\,
    ARRLLEN( N ) / 4.0 );
  IFDEF .ENBTRACE PAUSE ENDIF )
DEFINE(IDESC2(N),ARRIDENT( ARRLOC( N ) )
  CALL IDENT( IAR N\, WORK( ARRWK1\, ARRWK2 )\,
    32767\, 32767\, 1\, 32767 );
  IFDEF .ENBTRACE PAUSE ENDIF )
DEFINE(RDESC2(N),ARRIDENT( ARRLOC( N ) )
  CALL IDENT( RAR N\, WORK( ARRWK1\, ARRWK2 )\,
    32767\, 32767\, 1\, 32767 );
  IFDEF .ENBTRACE PAUSE ENDIF )
DEFINE(CDESC2(N),ARRIDENT( ARRLOC( N ) )
  CALL IDENT( CAR N\, WORK( ARRWK1\, ARRWK2 )\,
    32767\, 32767\, 1\, 32767 );
  IFDEF .ENBTRACE PAUSE ENDIF )

```

```

#
#
#

```

```

GENERATE CODE FOR DRIVER ROUTINE ON MINIMAP.

```

```

SUBROUTINE DRIVER
ARRIRC(1)ARRIRC(2)ARRIRC(3)ARRIRC(4)ARRIRC(5)ARRIRC(6)
ARRIRC(7)ARRIRC(8)ARRIRC(9)ARRIRC(10)ARRIRC(11)
ARRIRC(12)ARRIRC(13)ARRIRC(14)ARRIRC(15)ARRIRC(16)
ARRIRC(17)ARRIRC(18)ARRIRC(19)ARRIRC(20)
INTEGER ARRWK1, ARRWK2, WORK, INT, JUNK1, JUNK2
REAL ARRLOC, ARRLLEN, AMOD
COMMON / WINDOW / ARRLOC( 20 ), ARRLLEN( 20 ), ARRWK1,
  ARRWK2, I1, I2, I3, I4, I5, I6, I7,
  I8, I9, I10, I11, I12, I13, I14,
  I15, I16, I17, I18, I19, I20, R1,
  R2, R3, R4, R5, R6, R7, R8, R9, R10,
  R11, R12, R13, R14, R15, R16, R17,
  R18, R19, R20, WORK( 32767, 513 )

```

E.2 MAPCHAR Include File

```
#--MAPCHAR  RATFOR SUPPORT FOR ALPHANUMERICS ON MINI-MAP
#
DEFINE(.A,65)
DEFINE(.B,66)
DEFINE(.C,67)
DEFINE(.D,68)
DEFINE(.E,69)
DEFINE(.F,70)
DEFINE(.G,71)
DEFINE(.H,72)
DEFINE(.I,73)
DEFINE(.J,74)
DEFINE(.K,75)
DEFINE(.L,76)
DEFINE(.M,77)
DEFINE(.N,78)
DEFINE(.O,79)
DEFINE(.P,80)
DEFINE(.Q,81)
DEFINE(.R,82)
DEFINE(.S,83)
DEFINE(.T,84)
DEFINE(.U,85)
DEFINE(.V,86)
DEFINE(.W,87)
DEFINE(.X,88)
DEFINE(.Y,89)
DEFINE(.Z,90)
DEFINE(.D0,48)
DEFINE(.D1,49)
DEFINE(.D2,50)
DEFINE(.D3,51)
DEFINE(.D4,52)
DEFINE(.D5,53)
DEFINE(.D6,54)
DEFINE(.D7,55)
DEFINE(.D8,56)
DEFINE(.D9,57)
```


E.3 MAPID Include File

```
#--MAPID  RATFOR SUPPORT FOR MINI-MAP MODULE IDENTIFICATION
#
DEFINE(OPTION(OPT,VALUE),IFDEF .OPTENABLE
      WORK( OPT\, 1 ) = VALUE; ENDIF)
DEFINE(NAME1(A),NAME6(A,0,0,0,0,0))
DEFINE(NAME2(B,C),NAME6(B,C,0,0,0,0))
DEFINE(NAME3(D,E,F),NAME6(D,E,F,0,0,0))
DEFINE(NAME4(G,H,I,J),NAME6(G,H,I,J,0,0))
DEFINE(NAME5(K,L,M,N,O),NAME6(K,L,M,N,O,0))
DEFINE(NAME6(P,Q,R,S,T,U),
      IFDEF .OPTENABLE
      .OPTUNDEF;
      IFDEF .ENBTRACE WORK(.OPTDEBUG,1)=0; ENDIF
      WORK(11,1)=P;WORK(12,1)=Q;WORK(13,1)=R;
      WORK(14,1)=S;WORK(15,1)=T;WORK(16,1)=U;PAUSE;
      ENDIF)
INCLUDE MAPOPTS
```

E.4 MAPOPTS Include File

```
#--MAPOPTS  OPTIONS FOR MINI-MAP MODULES
#
DEFINE(.OPTTIMER,1)           # set time-out interval
                              # for module execution
DEFINE(.OPTSIZE,2)           # max number of points
                              # to process per run.
DEFINE(.OPTDEBUG,3)         # enable module debugger.
#
# add new options before this comment.
#
DEFINE(.OPTMAX,10)           # maximum no._ of
options.
DEFINE(.OPTNAME,11)         # starting addr of name.
DEFINE(.OPTNAMELEN,6)       # length of name block.
DEFINE(.OPTEND,EVAL(.OPTMAX,+,6)) # length of option and
                              # name block.
DEFINE(.OPTUNDEF,UNDEFINE(.OPTENABLE))
                              # disable setting of
                              # module options.
DEFINE(.OPTENABLE,0)        # enable the setting of
                              # module options.
```

E.5 MAPSUB Include File

```
#--MAPSUB  RATFOR SUPPORT FOR MINI-MAP MODULE SUBROUTINES
#
DEFINE(DEBUGON,DEFINE(.ENBTRACE,1))
DEFINE(DEBUGOFF,UNDEFINE(.ENBTRACE))
DEFINE	TRACE(N),IFDEF .ENBTRACE
    DBGTR1 = N; DBGTR2 = 0; PAUSE;
    ENDIF)
DEFINE	TRACE2(M,N),IFDEF .ENBTRACE
    DBGTR1 = M; DBGTR2 = N; PAUSE;
    ENDIF)
DEFINE(DEBUG,DBGTR1\, DBGTR2)
```

F APPENDIX F: LISTINGS OF VAX/VMS PROCEDURES

What follows are VAX/VMS command procedures (.COM files) that execute at DCL level. These include the GIPSYM, GIPLOG, GENCMD, and GENMAP procedures. Also included are the linker option files, MMDUM.LNK and MMSYM.LNK, which provide special instructions to the linker called in the GENCMD procedure. The GENCMD and GENMAP procedures are located in the GIPSY_UTIL directory, and the others are located in the GIPSY_MGR directory in the GIPSY package.

F.1 GIPSYM Procedure

```

$!
$!XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
$! gipsym command symbol definitions for GIPSY
$!XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
$!
$!      This DCL command file should be executed by
$!      SYLOGIN.COM to define command symbols for GIPSY
$!      and related utilities.  This file is invoked by
$!      GIPSETUP.COM.
$!
$!
$!      chkup      ::= @GIPSY_UTIL:chkup
$!      chkupm    ::= @GIPSY_UTIL:chkupm
$!      cmddoc    ::= $GIPSY_UTIL:cmddoc
$!      delight   ::= write sys$output -
$!                  """"Use RELDSP in place of DELIGHT
$!      fixit     ::= $GIPSY_UTIL:fixit
$!      gen*cmd   ::= @GIPSY_UTIL:gencmd
$!      genm*ap   ::= @GIPSY_UTIL:genmap
$!      get*dsp   ::= @GIPSY_UTIL:getdsp
$!      getlight  ::= write sys$output -
$!                  """"Use GETDSP in place of GETLIGHT
$!      ghelp     ::= $GIPSY_EXE:help
$!      gip*sy    ::= $GIPSY_EXE:GIPSY
$!      header    ::= $GIPSY_UTIL:header
$!      hi*story  ::= @GIPSY_MGR:history
$!      mar       ::= MACRO
$!      mk*cmd    ::= @GIPSY_MGR:mkcmd
$!      mklib     ::= @GIPSY_MGR:mklib
$!      mmr*atfor ::= @GIPSY_UTIL:mmratfor
$!      newcmd    ::= @GIPSY_MGR:newcmd
$!      parse     ::= @GIPSY_UTIL:parse
$!      ratexe    ::= $GIPSY_UTIL:ratfor
$!      ratfil    ::= @GIPSY_UTIL:ratfil
$!      rat*for   ::= @GIPSY_UTIL:ratfor
$!      rel*dsp   ::= @GIPSY_UTIL:reldsp
$!      update    ::= @GIPSY_UTIL:update
$!      gv*erify  ::= @GIPSY_MGR:verify
$!      xlist     ::= @GIPSY_MGR:rtnxlist
$!
$! These global symbols modify the execution
$! of Gipsy support commands
$!
$! gipsy$apdma == 0 ! array processor may transmit
$!               ! data via DMA mechanism
$! gipsy$check == 1 ! check modify times of files
$! gipsy$force == 0 ! force compilation of all files
$! gipsy$history == 0 ! record modifications in
$!                   ! internal history file

```

```
$ gipsy$minimap == 0    ! GIPSY does not have CSPI
$      ! mini-MAP support
$ gipsy$share == 1 ! use GIPRDSHR & GIPWRSHR
$      ! shareable libraries
$ gipsy_dropbase == 4  ! Set default priority handler
$ gipsy_priority == 0  ! Set flag for priority handler
```

F.2 GIPLOG Procedure

```

$!
$!XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
$! giplog logical name definitions for GIPSY
$!XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
$!
$!   This DCL command file should be executed by
$!   SYLOGIN.COM to define logical names for GIPSY
$!   and related utilities.  This file is invoked
$!   by GIPSETUP.COM.
$!
$!
$! Gipsy prompt and file search paths
$!
$   define gipsy_prompt "G: "
$   define gipsy_cmdpath "GIPSY_DEFAULT:|GIPSY_EXE:"
$   define gipsy_docpath "GIPSY_DOC:"
$   define gipsy_mappath "GIPSY_DEFAULT:|GIPSY_MAPLOAD:"
$!
$! Gipsy/OS internal logical names
$!
$   define gipsy_doc gipsyroot:[doc]
$   define gipsy_etc gipsyroot:[etc]
$   define gipsy_exe gipsyroot:[exe]
$   define gipsy_tmp sys$login:
$   define gipsy_usr sys$login:
$   define gipsy_util gipsyroot:[util.exe]
$!
$! Define Physical Devices for GIPSY.
$!
$!   define gipsy_light1 tta0:
$!   define gipsy_light2 ttal:
$!   define gipsy_light3 tta2:
$!   define gipsy_light4  tta3:
$!   define gipsy_vicom   ttc7:
$!   define gipsy_hp      ttf0:
$!   define gipsy_tek     ttd3:
$!   define gipsy_ramtek  tta4:
$   define gipsy_print lpa0:
$!   define gipsy_minimap ama0:
$   define gipsy_raster  xaa0:
$!
$! Gipsy maintenance logical names
$!
$   define gipsy_code gipsyroot:[src]
$   define gipsy_data gipsyroot:[data]
$   define gipsy_eval gipsyroot:[eval]
$   define gipsy_exdsp gipsyroot:[exdsp]
$   define gipsy_incl gipsyroot:[include]

```

```
$ define gipsy_lib gipsyroot:[lib]
$ define gipsy_light gipsyroot:[light]
$ define gipsy_lnk gipsyroot:[link]
$ define gipsy_manual gipsyroot:[manual]
$ define gipsy_mgr gipsyroot:[mgr]
$ define gipsy_map gipsy_maplink,-
    gipsy_mapload,-
    gipsy_mapobj,-
    gipsy_mapsrc
$ define gipsy_maplink gipsyroot:[minimap.link]
$ define gipsy_mapload gipsyroot:[minimap.load]
$ define gipsy_mapobj gipsyroot:[minimap.obj]
$ define gipsy_mapsrc gipsyroot:[minimap.src]
$ define gipsy_obj gipsyroot:[object]
$ define gipsy_obs gipsyroot:[obsolete]
$ define gipsy_old gipsyroot:[old]
$ define gipsy_os gipsyroot:[os]
$ define gipsy_ostest gipsyroot:[ostest]
$ define gipsy_prim gipsyroot:[prim]
$ define gipsy_rat gipsyroot:[src]
$ define gipsy_run gipsyroot:[run]
$ define gipsy_src gipsyroot:[src]
$ define gipsy_stat gipsyroot:[stat]
$ define gipsy_tom gipsyroot:[tomove]
$ define gipsy_wk gipsyroot:[wk]
$ define remote 1: ""200=""
```


F.3 GENCMD Procedure

```

$!
$!XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
$! gencmd generate a Gipsy command
$!XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
$!
$! on error then goto DONE
$   chkopt = ""
$   check = gipsy$check
$   dirs = "-d GIPSY_SRC: -d GIPSY_OBJ:"
$   force = gipsy$force
$   opts = ""
$   share = gipsy$share
$   user = 0
$!
$! Generate per-process unique file name
$!
$   tmp = ""
$   tmp = "T" + f$getjpi("", "PID")
$!
$! Check the input parameters
$!
$   if pl .nes. "" then goto GOTARGS
$ASK:
$   if pl .eqs. "" then inquire/nopun pl "$_Command: "
$   if pl .eqs. "" then goto ASK
$   inquire/nopun p2 "$_Files:      "
$   inquire/nopun p3 "$_Libraries:  "
$GOTARGS:
$!   on control_y then goto DONE
$!
$! Check for options
$!
$   parse "INIT" -
     "/CHECK/DEBUG/DIRECTORY=/FORCE/SHARE/USER/MAP/XREF/" -
     'pl'
$OPTLOOP:
$   parse
$   if opt$opt .eqs. "" then goto OPTOUT
$   if opt$opt .eqs. "CHECK" then check = opt$flg
$   if opt$opt .eqs. "DEBUG" .and. opt$flg then -
     opts = opts + "/DEBUG"
$   if opt$opt .eqs. "DIRECTORY" then -
     dirs = dirs + " -D" + opt$val
$   if opt$opt .eqs. "FORCE" then force = opt$flg
$   if opt$opt .eqs. "SHARE" then share = opt$flg
$   if opt$opt .eqs. "USER" then user = opt$flg
$   if opt$opt .eqs. "USER" .and. opt$flg then dirs = ""
$   if opt$opt .eqs. "MAP" .and. opt$flg then -

```

```

        opts = opts + "/MAP"
$       if opt$opt .eqs. "XREF" .and. opt$flg then -
        opts = opts + "/CROSS"
$       if opt$more then goto OPTLOOP
$OPTOUT:
$       pl = opt$fil
$       if pl .eqs. "" then exit %x380E4
$!
$! Extract the command and driver names
$!
$       command = f$parse(pl,pl,,"NAME")
$       if p2 .eqs. "" then p2 = command
$       driver = p2 + ","
$       n = f$locate(", ",driver)
$       driver = f$extract(0,n,driver)
$       driver = f$parse(driver,driver,,"NAME")
$       if .not. check then dirs = "-n"
$       if force then chkopt = "-f"
$!
$! Set up options for calling chkup.
$!
$       files := 'p2' 'p3'
$       ret = "RET1"
$       goto COMP
$ RET1:
$       compfile = linkargs
$       if p4 .eqs. "" then goto SKIPONE
$       files := 'p4' 'p5'
$       if f$locate ( "_LIB:", p4 ) .ne. -
        f$length ( p4 ) then goto SKIPLIB1
$       ret = "RET2"
$       goto COMP
$ RET2:
$       compfile = compfile + "," + linkargs
$       if p6 .eqs. "" then goto SKIPONE
$       files := 'p6' 'p7'
$       if f$locate ( "_LIB:", p6 ) .ne. -
        f$length ( p6 ) then goto SKIPLIB2
$       ret = "RET3"
$       goto COMP
$ RET3:
$       compfile = compfile + "," + linkargs
$       if p8 .eqs. "" then goto SKIPONE
$       files := 'p8'
$       if f$locate ( "_LIB:", p8 ) .ne. -
        f$length ( p8 ) then goto SKIPLIB3
$       ret = "RET4"
$       goto comp
$ RET4:
$       compfile = compfile + "," + linkargs
$       goto SKIPONE

```

```

$ SKIPLIB1:
$     LIBS = files
$     goto FIXLIB
$ SKIPLIB2:
$     LIBS = files
$     goto FIXLIB
$ SKIPLIB3:
$     LIBS = files
$ FIXLIB:
$!
$! First check to see if there are multiple libraries
$!
$     libloc = f$locate ( ",", LIBS )
$     if libloc .eq. f$length ( LIBS ) then goto SINGLE
$!
$! There are multiple libraries, therefore append "/LIB"
$! after each library name that is found. Search
$! for "," to locate libraries.
$!
$     loc = 0
$     skip = 0
$ LOOP1:
$     skip = loc + skip
$     string = f$extract ( skip, f$length ( LIBS ), LIBS )
$     loc = f$locate ( ",", string )
$     if loc .eq. f$length ( string ) then goto SINGLE
$     LIBS = f$extract ( 0, skip, LIBS ) -
$           + f$extract ( 0, loc, string ) -
$           + "/LIB" -
$           + f$extract( loc, f$length( string ), string )
$     loc = loc + 5
$     goto loop1
$ SINGLE:
$     LIBS = LIBS + "/LIB"
$ LIBFIXED:
$     compfile = compfile + "," + LIBS
$     goto SKIPONE
$COMP:
$!
$! Compile out of date source files and
$! return link args in LINKARGS
$!
$     assign/user 'tmp'.for sys$output:
$     chkup 'chkopt' 'dirs' 'pl' 'files'
$     if $status .ne. 1 then goto DONE
$     @'tmp'.for
$     if $status .ne. 1 then goto DONE
$     goto 'ret'
$!
$! Now Generate the driver for the command
$! and then link it.!
```

```

$!
$SKIPONE:
$      write sys$output "Gen ""'command'""
$      if user then goto USERCMD
$!
$! Create the user command main on the fly -- relies
$! on the source USRMAIN.RAT
$!
$      open/write main 'tmp'.for
$      write main "      PROGRAM ''command'$MAIN"
$      write main "      INTEGER ARGS ( 128 )"
$      write main "      EXTERNAL ''driver', GPINTR"
$      write main "      CALL LIB$ESTABLISH ( GPINTR )"
$      write main "      CALL OSARGS ( ARGS )"
$      write main "      CALL GPMAIN ( ' ''command'.', " -
+ " ''driver' , ARGS )"
$      write main "      STOP"
$      write main "      END"
$      close main
$      fortran 'tmp'
$      mm_support = "GIPSY_MGR:mmdum.lnk/options"
$      if gipsy$minimap then -
+ "GIPSY_MGR:mmsym.lnk/options"
$      if .not. share then goto NOSHARECMD
$!
$! Link the user command
$!
$      link'opts'/exe='pl'.exe 'tmp','compfile',-
+ sys$input:/option,'mm_support'
!
! GIPSY readable shareable image
!
GIPSY_SHARE:GIPRDSHR.EXE/SHAREABLE
!
! global section match parameters
!
GSMATCH=LEQUAL,0,0
!
! all commons are process local except ACCCOM
! which is shareable and writeable
!
PSECT=ACCCOM,SHR,WRT
PSECT=BUFCOM,NOSHR
PSECT=CERROR,NOSHR
PSECT=CHOICE,NOSHR
PSECT=CHRCOM,NOSHR,NOWRT
PSECT=CIOBUF,NOSHR
PSECT=CLUNS,NOSHR
PSECT=CMDCOM,NOSHR
PSECT=COMENT,NOSHR

```

```

PSECT=CSENF,NOSHR
PSECT=DSCCOM,NOSHR
PSECT=ERROR,NOSHR
PSECT=FILCOM,NOSHR,NOWRT
PSECT=GIPCOM,NOSHR
PSECT=HSL_STATE,NOSHR,LCL
PSECT=HSTCOM,NOSHR
PSECT=INTFLG,NOSHR
PSECT=IO,NOSHR
PSECT=MEMCOM,NOSHR
PSECT=MMDEV,NOSHR
PSECT=OSCCOM,NOSHR
PSECT=OSERR,NOSHR
PSECT=PTHCOM,NOSHR
PSECT=RIO,NOSHR
PSECT=RUNCOM,NOSHR
PSECT=TBLASC,NOSHR,NOWRT
PSECT=TTCOM,NOSHR
PSECT=UNIQUE,NOSHR
PSECT=WINDOW,NOSHR
PSECT=WINDOW$,NOSHR
$!
$!
$! Command was successfully generated
$!
$      goto DONE
$NOSHARECMD:
$      link'opts'/exe='pl'.exe 'tmp','compfile',-
      'mm_support',gipsy_lib:accblk.ojt,-
      blockd.ojt,giplib/lib,osrplib/lib
$      goto DONE
$!
$! Generate user command
$!
$USERCMD:
$      link'opts'/exe='pl'.exe 'compfile'
$!
$! Clean up and exit
$!
$DONE:
$      if f$search("''tmp'.map") .nes. "" then -
      renam/nolog 'tmp'.map 'command'
$      if f$search("''tmp'.*") .nes. "" then -
      delete 'tmp'.*;*
$      exit 1
$!
$!*****
$!
$! gencmd <CMD>[/option...] <FILES> <LIBS>
$!
$!          <CMD>          name of command to generate

```

```

$!      <FILES>          list of files comprising
$!      <COMMAND> separated by commas
$!      <LIBS>          list of libraries comprising
$!      <COMMAND> separated by commas
$!      /[NO]CHECK      [don't] check file modify times
$!      /DEBUG          link command with debugger
$!      /DIR=...        optional list of dirs to search
$!      /[NO]FORCE      [don't] force compilation
$!                        of all files
$!      /[NO]SHARE      [don't] link with Gipsy
$!                        shareable libraries
$!      /USER           generate user command
$!                        (rather than Gipsy)
$!      /MAP            link command with map generated
$!      /XREF           link command with symbol
$!                        cross-referencing
$!
$!      If the global symbol gipsy$check .eq. 1 then
$!      file modify time checks are done.
$!
$!      If the global symbol gipsy$force .eq. 1 then
$!      all files are recompiled.
$!
$!      If the global symbol gipsy$share .eq. 1 then
$!      the shareable libraries GIPSY_SHARE:GIPRDSHR.EXE
$!      and GIPSY_SHARE:GIPWRSHR.EXE are used.
$!
$!      Local option values override the gipsy$* global
$!      symbol values.
$!
$!*****
$!

```

F.4 GENMAP Procedure

```

$!*****
$! genmap generate a Gipsy-compatible mini-MAP module
$!*****
$!
$   if .not. gipsy$minimap then goto NOSUPPORT
$   on error then goto DONE
$   chkopt = ""
$   check = gipsy$check
$   dirs = "-d GIPSY_MMR: -d GIPSY_MMO:"
$   force = gipsy$force
$   opts = ""
$   user = 0
$   save = 0
$!
$! Generate per-process unique file name
$!
$   tmp = ""
$   tmp = "T" + f$getjpi("", "PID")
$!
$! Check the input parameters
$!
$   if pl .nes. "" then goto GOTARGS
$ASK:
$   if pl .eqs. "" then inquire/nopun pl "$_Module:      "
$   if pl .eqs. "" then goto ASK
$   inquire/nopun p2 "$_Files:          "
$   inquire/nopun p3 "$_Libraries:     "
$GOTARGS:
$   on control_y then goto DONE
$!
$! Check for options
$!
$   parse "INIT" -
$     "/CHECK/DIRECTORY=/FORCE/USER/MAP/SAVE/" -
$     'pl'
$OPTLOOP:
$   parse
$   if opt$opt .eqs. "" then goto OPTOUT
$   if opt$opt .eqs. "CHECK" then check = opt$flg
$   if opt$opt .eqs. "DIRECTORY" then -
$     dirs = dirs + " -D" + opt$val
$   if opt$opt .eqs. "FORCE" then force = opt$flg
$   if opt$opt .eqs. "USER" then user = opt$flg
$   if opt$opt .eqs. "USER" .and. opt$flg then dirs = ""
$   if opt$opt .eqs. "MAP" .and. opt$flg then -
$     opts = opts + "/MAP"
$   if opt$opt .eqs. "SAVE" then save = opt$flg
$   if opt$more then goto OPTLOOP

```

```

$OPTOUT:
$   pl = opt$fil
$   if pl .eqs. "" then exit %x380E4
$!
$! Extract the module and driver names
$!
$   module = f$parse(pl,pl,,"NAME")
$   if p2 .eqs. "" then p2 = module
$   driver = p2 + ","
$   n = f$locate(",","driver")
$   driver = f$extract(0,n,driver)
$   driver = f$parse(driver,driver,,"NAME")
$   if .not. check then dirs = "-n"
$   if force then chkopt = "-f"
$!
$! Compile out of date source files and
$! return link args in LINKARGS
$!
$   assign/user 'tmp'.for sys$output:
$   chkupm 'chkopt' 'dirs' 'p1' 'p2' 'p3' 'p4' -
       'p5' 'p6' 'p7' 'p8'
$   if $status .ne. 1 then goto DONE
$   @'tmp'.for
$!   if $status .ne. 1 then goto DONE
$   write sys$output "Gen ""'module'""
$   if user then goto USERCMD
$!
$! Create the include file
$!
$   open/write incfile 'module'.INC
$   write incfile "ORDER WINDOW=C33619226/D$1000"
$FILELOOP:
$   if linkargs .eqs. "" then goto FILEDONE
$   sep = f$locate(",","linkargs")
$   file = f$extract(0,sep,linkargs)
$   suffix = file + "()"
$   prefix = "ORDER"
$   lib = f$locate("/LIB",suffix)
$   len = f$length(suffix)
$   if lib .lt. len then prefix = "FILES"
$   if lib .lt. len then -
       suffix = f$extract(0,lib,file) + "()"
$   write incfile ""'prefix' ''suffix'"
$   linkargs = linkargs - file - ","
$   goto FILELOOP
$FILEDONE:
$   if .not. user then goto INCDONE
$   write sys$output "Enter MLINK instructions, " -
       + "and finish with END"
$LINELOOP:
$   inquire line

```



```

$      if line .nes. "END" then write incfile line
$      if line .nes. "END" then goto LINELOOP
$INCDONE:
$      write incfile "FILES RTLIB:MMRTL"
$      write incfile "FILES MCLDIR:MCLIB"
$      write incfile "FILES SSLDIR:MMSSL"
$      write incfile "END"
$      close incfile
$!
$! Link the user module
$!
$      mlink 'module' /I
$      if .not. save then delete/nolog 'module'.inc;*
$!
$! Module was successfully generated
$!
$      goto DONE
$!
$! Clean up and exit
$!
$DONE:
$      if f$search("'*tmp'.*") .nes. "" then -
$          delete 'tmp'.*;*
$      exit 1
$NOSUPPORT:
$      write sys$output "genmap: mini-MAP not supported"
$      exit 1
$!
$!*****
$!
$! genmap <MODULE>[/option...] <FILES> <LIBS>
$!
$!      <MODULE> name of module to be generated
$!      <FILES> list of files comprising <MODULE>
$!              separated by commas
$!      <LIBS> list of libraries comprising <MODULE>
$!              separated by commas
$!      /[NO]CHECK [don't] check file modify times
$!      /DIR=... optional list of directories to search
$!      /[NO]FORCE [don't] force compilation of all files
$!      /USER generate user module (rather than Gipsy)
$!      /MAP link module with map generated
$!
$!      If the global symbol gipsy$check .eq. 1 then
$!      file modify time checks are done.
$!
$!      If the global symbol gipsy$force .eq. 1 then
$!      all files are recompiled.
$!
$!      Local option values override the gipsy$* global
$!      symbol values.

```

\$!
\$!XX
\$!

F.5 MMDUM Linker Options File For GENCMD

```
!  
! CSPI mini-MAP symbols for external reference by MMINTR  
!  
SYMBOL=HSL__NOREGMAP,%X092C8033  
SYMBOL=HSL__REGMAP,%X092C803B  
SYMBOL=HSL__CLOSED,%X092C8043  
SYMBOL=HSL__ILGAM_ADR,%X092C801C  
!  
! CSPI mini-MAP Host Support Library dummy routines  
!  
GIPSY_LIB:GIPLIB.OLB/INCLUDE=-  
      (CLOSMM,LOADMM,OPENMM,READMM,SETMMR,SIZEMM,-  
      STRTMM,STSGMM,STTOMM,WAITMM,WRITMM)
```

F.6 MMSYM Linker Options File For GENCMD

```
!  
! CSPI mini-MAP support if mini-MAP is available  
!  
MMHSL/LIB/INCLUDE=-  
    (CLOSMM,LOADMM,OPENMM,READMM,SETMMR,SIZEMM,-  
    STRTMM,STSGMM,STTOMM,WAITMM,WRITMM)
```

G APPENDIX G: INCLMM COMMAND

What follows are the files associated with the GIPSY command, INCLMM. This includes the following files:

1. GIPSY_DOC:INCLMM.DOC - on-line documentation,
2. GIPSY_RUN:INCLMM.RUN - a run-file to test the command,
3. GIPSY_LNK:INCLMM.LNK - a link file which instructs the system on how to build the command,
4. GIPSY_SRC:DINCMM.RAT - the command driver,
5. GIPSY_SRC:IMMARR.RAT - searches input file for GETWPM calls,
6. GIPSY_SRC:RATLIN.RAT - scans an input file for instruction clauses,

7. GIPSY_SRC:LINFIL.RAT - fills input buffer until terminator, and

8. GIPSY_INCL:RATTAB. - tables for parsing RATFOR clauses.

G.1 INCLMM Documentation

*INCLMM Build include files for mini-MAP command drivers.

VERSION: A.01 DATE: 03-08-86 CONTACT: G. Fabregas

ACTION: This command reads GIPSY command drivers to build an include file that equivalences host arrays with mini-MAP arrays. The include file has, by default, the name "X" + command name + ".". For example, the EDGE command would be used to create the XEDGE. include file. The include file is included in the mini-MAP counterpart to the GIPSY command driver so that arrays referenced in the host address the same arrays referenced in the mini-MAP. Host arrays are accessible by the mini-MAP if they are dynamically allocated using the GETWPM primitive. The fourth argument in the call to GETWPM is the name of the array. This name is referenced by the mini-MAP driver without any other declarations beside those written to the include file. The GETWPM routine dynamically allocates memory in both the host and in the mini-MAP. This command writes four types of RATFOR instructions or macros to the include file:

1. The module identifier - a NAME macro which loads memory with the name of the command when executed;
2. The array allocator - a GETARRAYS macro which reads array information from the host and modifies the information for local use;
3. The array descriptor - a DESC macro which dynamically declares an array with the proper data type and dimensionality; and
4. The array definition - a DEFINE statement which equivalences an array with the proper name.

SOURCE: GIPSY command driver written in RATFOR.

DESTINATION: command name (not a file);
optional include file name.

FLAGS: NONE

QUESTIONS: (1) Driver routine, if not specified on the command line;
(2) Command name, if not specified on the command line.

COMMAND STRING EXAMPLE:

G: INCLMM GIPSY_SRC:DEGE.RAT > EDGE
(Scanning input file)

For IIN, Enter number of array dimensions
(D = 1, 1 to 2) -- 1
For IOU, Enter number of array dimensions
(D = 1, 1 to 2) -- 1

G:

Since no include was specified, the output is written to the XEDGE. include file. Two arrays, IIN and IOU, were found in the fourth argument positions of two different assignment statements making reference to the GETWPM function. The array dimensions are specified as "1" if the subscripts can only extend from one to 32767, and "2" if the subscripts are bounded only by memory. The XEDGE. include file contains the following:

```
NAME4(.E,.D,.G,.E)
GETARRAYS
IDESC1( 1 )
DEFINE(IIN,IAR1)
IDESC1( 2 )
DEFINE(IOU,IAR2)
```

This means that the module may only be used if the command name is "EDGE", or the module is explicitly loaded using the MMLOAD primitive. The "IIN" array named in the GETWPM statement in GIPSY_SRC:DEGE.RAT has space reserved in the first array slot, IAR1, and is referenced by the mini-MAP driver routine as "IIN". The "IOU" array has space reserved in the second array slot, IAR2, and is identified as "IOU" in the mini-MAP driver routine.

COMMENTS: The INCLMM command is provided as a convenience. The instructions in the include file can be coded directly into the mini-MAP command driver. However, if the order of the GETWPM calls are changed, then the array descriptor and definition statements require modification. This command should be executed after the driver has been altered, so the include file will reflect those changes without requiring any modification to the mini-MAP command driver. Of course, both the host driver and mini-MAP driver will subsequently require compilation.

G.2 INCLMM Run-file

```
$! TESTING THE COMMAND INCLMM
$! INCLMM: BUILD INCLUDE FILES FOR MINI-MAP COMMAND DRIVERS.
$!
$      INCLMM
GIPSY_SRC:DABSIM.RAT
ABSIMG
1
$!
$! THE OUTPUT SHOULD MATCH THE FILE IN GIPSY_INCL:
$!
$      SYS DIFFERENCES XABSIMG. GIPSY_INCL:
$!
$! DELETE THE INCLUDE FILE.
$!
$      DELETE XABSIMG.
$!
$! DONE
$!
$ EXIT
```

. G.3 INCLMM Link File

‡ GENCMD INCLMM DINCMM, IMMARR, RATLIN, LINFIL

G.4 DINCMM Routine

```
#--DINCMM          DRIVER FOR THE 'INCLMM' GIPSY COMMAND.
#
# IDENTIFICATION
#
#     TITLE          DINCMM
#     AUTHOR         GREGG R. FABREGAS
#     VERSION        A.01
#     DATE           14-JUL-1985
#     LANGUAGE       RATFOR
#     SYSTEM         VAX 11/780
#
#
# PURPOSE
#
#     THIS COMMAND BUILDS AN INCLUDE FILE
#     FOR INCLUSION IN THE DRIVER
#     ROUTINE EXECUTED ON THE CSPI MINIMAP
#     ARRAY PROCESSOR.  THE
#     INTENT IS TO SIMPLIFY THE PROGRAMMER'S
#     TASK BY PUTTING MOST OF
#     THE CODE HANDLING SYNCHRONIZATION
#     AND GIPSY-INTERFACING, AS WELL
#     AS EQUIVALENCE INFORMATION SO
#     THAT THE ARRAYS DEFINES ON THE
#     HOST MATCH REFERENCES TO THE SAME
#     ARRAYS BY THE MINIMAP, IN AN
#     EXTERNAL FILE.  THIS AMELIORATES
#     THE NEED TO DIRECTLY MODIFY THE
#     MINIMAP CODE WHENEVER THE
#     PROGRAMMER MODIFIES THE HOST CODE.
#
# ENTRY POINT
#
#     CALL DINCMM( WORK, * )
#
# ARGUMENT LISTING
#
#     WORK      INT ARRAY      WORK SPACE FOR ALLOCATED
#                                     ARRAYS.
#     ERRET     LABEL          LABEL IN CALLING ROUTINE FOR
#                                     ALTERNATE ERROR RETURN.
#
# INCLUDE FILES/COMMONS
#
#     GIPSY     INCLUDE        GIPSY GENERAL SYMBOL DEF'NS
#     GIPCOM    COMMON         GIPSY COMMAND LINE COMMON DATA
#     ERROR     COMMON         ERROR TRACE STACK COMMON DATA
#     TTCOM     COMMON         FD'S FOR TERMINAL & RUNFILE IO
```

```

#          MAPOPTS   INCLUDE      MINIMAP MODULE OPTIONS.
#
#  ROUTINES CALLED
#
#          PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#                    (GIPSY PRIMITIVE)
#          OSGTNM     GET NAME FROM FILE DESCRIPTOR
#                    (GIPSY OS KERNEL)
#          PUTF       MACHINE INDEPENDENT FORMATTED OUTPUT
#                    (GIPSY PRIMITIVE)
#          GETFC      READ FIXED LENGTH STRING FOR GETF
#                    (FORMATTED INPUT) (GIPSY PRIMITIVE)
#          OSINFD     INITIALIZE FILE DESCRIPTOR GIVEN GENERIC
#                    NAME (GIPSY OS KERNEL)
#          STRCPY     COPY A .EOS TERMINATED STRING TO ANOTHER
#                    (GIPSY OS KERNEL)
#          OSFNAM     CONSTRUCT SYSTEM DEPENDENT FILE NAME
#                    (GIPSY OS KERNEL)
#          OSPINF     PUT PARAMETER VALUES INTO FILE
#                    DESCRIPTOR (GIPSY OS KERNEL)
#          OSOPNS     OPEN FILE FOR SEQUENTIAL I/O
#                    (GIPSY OS KERNEL)
#          STRLEN     RETURN LENGTH OF .EOS-TERMINATED
#                    UNPACKED CHARACTER STRING.
#                    (GIPSY OS KERNEL)
#          CFOLD      CONVERT CHARACTER ARRAY TO UPPER CASE
#                    (GIPSY PRIMITIVE)
#          MEMBER     DETERMINE IF CHAR IS MEMBER OF A SET
#                    (GIPSY PRIMITIVE)
#          GETWP      GET WORK ARRAY POINTER.
#                    (GIPSY PRIMITIVE)
#          OSALOC     CHECK AND EXTEND USER WORK SPACE
#                    (GIPSY OS KERNEL)
#          IMMARR     SCAN INPUT FILE FOR GETWPM ARRAY
#                    INFORMATION. (GIPSY ROUTINE)
#          CLOSE      CLOSE A FILE. (GIPSY PRIMITIVE)
#          PPOP       POP PROGRAM NAME FROM THE ERROR STACK
#                    (GIPSY PRIMITIVE)
#          OSGIEV     GET EVENT VARIABLE (IEV) FROM OS SAVE
#                    AREA (GIPSY OS KERNEL)
#
#
# *****
#
#          INCLUDE GIPSY
#
#          SUBROUTINE DINCOMM( WORK, * )
#
#
#          INCLUDE GIPCOM

```

```

INCLUDE ERROR
INCLUDE TTCOM
INCLUDE MAPOPTS
DEFINE(.CMDMAX,EVAL(.OPTNAMELEN,+,1))
DEFINE(.INCLN,EVAL(.CMDMAX,+,1))
DEFINE(.LINELEN,200)
#
INTEGER WORK( .ARB )
INTEGER CMDLEN, I, LINE, NXT
CHARACTER FN( .FILENAMELENGTH ), CMDNAM( .CMDMAX )
CHARACTER INCNAM( .INCLN )
CHARACTER DIGITS( 11 ), LETTER( 27 ), DOT
INTEGER STRLEN, OSGTNM, MEMBER, PUTF, OSGIEV, OSINFD
INTEGER OSOPNS, GETWP, OSALOC, GETFC, OSFNAM, OSPINF
#
DATA INCNAM( 1 ) / 'X' /
DATA LETTER / EVAL( "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ) /
DATA DIGITS / EVAL( "1234567890" ) /
DATA DOT / '.' /
#
#           PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
CALL PPUSH( "DINCM" )
#
#           VERIFY THAT THE INPUT SOURCE FILE IS SPECIFIED,
#           AND PROMPT FOR IT IF NOT SPECIFIED.
#
IF ( .OK ~= OSGTNM( FD11, FN ) )
  $(
    IF ( .OK ~= PUTF( FDRUNO, "Driver routine: %C%F",
                    .BELLCHR1 ) ) GOTO 9800
    IF ( .OK ~= GETFC( FDRUNI, FN, .FILENAMELENGTH,
                    .FALSE. ) ) GOTO 9800
    IF ( .OK ~= OSINFD( FN, FD11 ) ) GOTO 9800
  $)
#
#           VERIFY THAT THE COMMAND NAME IS SPECIFIED.  IF
#           IT IS, COPY IT TO CMDNAM, TRUNCATING TO .CMDMAX
#           CHARACTERS IF NECESSARY.  OTHERWISE, PROMPT THE
#           USER FOR THE COMMAND NAME.
#
IF ( .OK == OSGTNM( FD01, FN ) )
  $(
    FN( .CMDMAX ) = .EOS
    CALL STRCPY( FN, CMDNAM, 1 )
  $)
ELSE
  $(
    IF ( .OK ~= PUTF( FDRUNO, "Command name: %C%F",
                    .BELLCHR1 ) ) GOTO 9800
    IF ( .OK ~= GETFC( FDRUNI, CMDNAM, .CMDMAX, .FALSE. ) )

```

```

        GOTO 9800
    $)
#
#       IF THE USER DID NOT SPECIFY AN OPTIONAL NAME
#       FOR THE OUTPUT INCLUDE FILE, CONSTRUCT THE
#       GENERIC INCLUDE FILE NAME FROM THE COMMAND
#       NAME, AND BUILD THE FULL FILE SPECIFICATION.
#
IF ( .OK -= OSGTNM( FD02, FN ) )
    $(
    CALL STRCPY( CMDNAM, INCNAM, 2 )
    IF ( .OK -= OSFNAM( .USRINCL, FN, INCNAM ) ) GOTO 9800
    IF ( .OK -= OSINFD( FN, FD02 ) ) GOTO 9800
    $)
#
#       OPEN THE OUTPUT INCLUDE FILE.
#
IF ( .OK -= OSPINF( FD02, .MODE, .SYSMODE ) ) GOTO 9800
IF ( .OK -= OSPINF( FD02, .LREC, .LINELEN ) ) GOTO 9800
IF ( .OK -= OSOPNS( FD02, .OUTPUT ) ) GOTO 9800
#
#       THE FIRST LINE OF OUTPUT IS THE STAGE ONE
#       SYNCHRONIZATION: MODULE IDENTIFICATION. THE
#       COMMAND NAME IS CONVERTED TO UPPER CASE, AND
#       TRANSLATED TO A MAP-DRIVER NAME MACRO. THIS
#       MACRO WILL BE EXPANDED BY RATFOR WHEN THE
#       MAP-DRIVER ROUTINE IS COMPILED.
#
CMDLEN = STRLEN( CMDNAM )
IF ( CMDLEN == 0 ) GOTO 9000
CALL CFOLD( CMDNAM, CMDLEN )
IF ( .OK -= PUTF( FD02, " NAME%I\(%C", CMDLEN, DOT ) )
    GOTO 9800
DO I = 1, CMDLEN
    $(
    IF ( MEMBER( CMDNAM( I ), LETTER, 1, 26 ) == 0 )
        $(
        IF ( MEMBER( CMDNAM( I ), DIGITS, 1, 10 ) == 0 )
            GOTO 9000
        ELSE
            IF ( .OK -= PUTF( FD02, "D" ) ) GOTO 9800
        $)
    IF ( .OK -= PUTF( FD02, "%C", CMDNAM( I ) ) ) GOTO 9800
    IF ( CMDNAM( I + 1 ) == .EOS )
        $(
        IF ( .OK -= PUTF( FD02, "\)%N" ) ) GOTO 9800
        $)
    ELSE
        IF ( .OK -= PUTF( FD02, "\,%C", DOT ) ) GOTO 9800
    $)
#

```

```

#           THE SECOND LINE OF OUTPUT IS THE STAGE TWO
#           SYNCHRONIZATION: ARRAY ALLOCATION.  THE
#           RATFOR SYMBOL, GETARRAYS, IS EXPANDED BY THE
#           RATFOR PREPROCESSOR.
#
IF ( .OK == PUTF( FD02, " GETARRAYS%N" ) ) GOTO 9800
#
#           ALLOCATE A LINE OF MEMORY FOR THE SCANNING
#           OF THE SOURCE CODE.
#
NXT = 1
LINE = GETWP( NXT, .INTMODE, .LINELEN )
IF ( .OK == OSALOC( NXT ) ) GOTO 9800
#
#           SCAN THE INPUT SOURCE CODE FOR ALLOCATION OF
#           MINIMAP MEMORY, AND ESTABLISH EQUIVALENCES
#           BETWEEN HOST ARRAY NAMES AND MINIMAP ARRAY
#           NAMES.
#
IF ( .OK == PUTF( FDRUNO, "(Scanning input file)%N" ) )
GOTO 9800
CALL IMMARR( FD11, FD02, WORK( LINE ),
             .LINELEN, IEV, %9900 )
#
#           FINISHED.  CLOSE THE OUTPUT INCLUDE FILE.
#
CALL CLOSE( FD02 )
CALL PPOP
RETURN
#
#           ERROR CONDITIONS:
#
#           COMMAND NAME IS NULL.
#
9000 CONTINUE
IEV = -5054
GOTO 9900
#
#           ERROR OCCURRED IN LOWER ROUTINE.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#
#           CLOSE THE OUTPUT INCLUDE FILE AND TAKE THE
#           ALTERNATE RETURN.
#
9900 CONTINUE
CALL CLOSE( FD02 )
RETURN 1
#
#

```

END

G.5 IMMARR Routine

```
#--IMMARR  ANALYZE MINIMAP ARRAYS FOR 'INCLMM' GIPSY COMMAND
#
#  IDENTIFICATION
#
#      TITLE           IMMARR
#      AUTHOR          GREGG R FABREGAS
#      VERSION         A.01
#      DATE            07-MAR-1986
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#
#  PURPOSE
#
#      THIS ROUTINE WRITES OUT THE
#      LAST SECTION OF AN INCLUDE FILE USED
#      IN A MINIMAP DRIVER ROUTINE TO
#      SIMPLIFY MODULE AND ARRAY
#      IDENTIFICATION.  THE LAST SECTION
#      EQUIVALENCES ARRAYS SPECIFIED
#      IN CALLS TO GETWPM IN HOST COMMAND
#      DRIVERS TO ARRAYS WITH THE
#      SAME NAME IN THE MINIMAP DRIVER ROUTINE.
#
#  ENTRY POINT
#
#      CALL IMMARR( FDI, FDO, LINE, LENGTH, IEV, ERRET )
#
#  ARGUMENT LISTING
#
#      FDI      CHAR ARR      FILE DESCRIPTOR OF HOST
#                                     COMMAND DRIVER.
#      FDO      CHAR ARR      FILE DESCRIPTOR OF OUTPUT
#                                     INCLUDE FILE.
#      LINE     CHAR ARR      BUFFER TO RECEIVE INPUT
#                                     FROM DRIVER.
#      LENGTH   INTEGER       MAXIMUM NUMBER OF CHARACTERS
#                                     IN "LINE".
#      IEV      INTEGER       INTEGER EVENT VARIABLE - SET
#                                     ON RETURN.
#      ERRET    LABEL         LABEL IN CALLING ROUTINE FOR
#                                     ERROR RETURN
#
#  INCLUDE FILES/COMMONS
#
#      TTCOM    COMMON        FD'S FOR TERMINAL & RUNFILE IO
#      RATTAB   INCLUDE       SYMBOLS AND DECLARATIONS FOR
#                                     RATLIN ROUTINE.
```

```

#
# LANGUAGE FEATURES REQUIRED
#
# ALTERNATE RETURNS.
#
# PROGRAM ENVIROMENT
#
# "FDI" AND "FDO" MUST BE OPEN.
# "LENGTH" MUST BE POSITIVE.
#
# ALGORITHM
#
# (1) OPEN INPUT FILE;
# (2) FIND FIRST SIMPLE-ASSIGNMENT STATEMENT
# IN INPUT FILE;
# (3) FOR EACH ASSIGNMENT STATEMENT WITH A
# CALL TO GETWPM:
# (A) IDENTIFY AND VALIDATE MODE FIELD,
# (B) EXTRACT NAME IN FOURTH ARGUMENT,
# (C) PROMPT USER FOR DIMENSIONALITY OF
# ARRAY ADDRESS SPACE, &
# (D) WRITE OUT ARRAY SPECIFICATION TO
# OUTPUT FILE;
# (4) CLOSE INPUT FILE ONCE "OSALOC" IS
# FOUND IN AN IF STATEMENT.
#
# DATA FORMAT
#
# (1) THE INPUT FILE IS ASSUMED TO BE RATFOR CODE;
# (2) THE CALLS TO GETWP OR GETWPM MUST BE SIMPLE
# ASSIGNMENT STATEMENTS TO SCALARS; AND
# (3) THE CALL TO OSALOC MUST BE IN AN IF STATEMENT.
#
# LIMITATIONS
#
# THE LINE BUFFER MUST BE LARGE ENOUGH TO CONTAIN AN
# ENTIRE INSTRUCTION CLAUSE, E.G., "IF ( CONDITION )".
#
# ROUTINES CALLED
#
# PPUSH PUSH PROGRAM NAME ONTO ERROR STACK
# (GIPSY PRIMITIVE)
# OSPINF PUT PARAMETER VALUES INTO FILE
# DESCRIPTOR (GIPSY OS KERNEL)
# OSOPNS OPEN FILE FOR SEQUENTIAL I/O
# (GIPSY OS KERNEL)
# RATLIN READ RATFOR CLAUSE OR LINE.
# (GIPSY ROUTINE)
# STRLOC LOCATE SUBSTRING (GIPSY OS KERNEL)
# PUTF MACHINE INDEPENDENT FORMATTED OUTPUT
# (GIPSY PRIMITIVE)

```

```

#           RNGETI      GET AN INTEGER WITHIN A RANGE (WITH
#                       PROMPT).  (GIPSY PRIMITIVE)
#           CLOSE      CLOSE A FILE.  (GIPSY PRIMITIVE)
#           PPOP        POP PROGRAM NAME FROM THE ERROR STACK
#                       (GIPSY PRIMITIVE)
#           OSGIEV      GET EVENT VARIABLE (IEV) FROM OS SAVE
#                       AREA  (GIPSY OS KERNEL)
#
#   REMARKS
#
#           NOTHING IS OUTPUT IF NO GETWPM STATEMENTS ARE FOUND.
#
#*****
#
#   INCLUDE GIPSY
#
#   SUBROUTINE IMMARR( FDI, FDO, LINE, LENGTH, IEV, * )
#
#   INCLUDE TTCOM
#   INCLUDE RATTAB
#
#   INTEGER LENGTH, IEV
#   CHARACTER FDI( .FDLENGTH ), FDO( .FDLENGTH )
#   INTEGER TYPE, MORE( 2 ), COUNT, DIM, START, FINISH
#   CHARACTER GETWPM( 7 ), PREFIX, COMMA, QUOTE, APOST
#   CHARACTER LINE( LENGTH ), DELIM, DOT, I, R
#   CHARACTER OSALOC( 7 ), INTMOD( 8 ), REALMD( 9 )
#   INTEGER STRLOC, OSPINF, OSOPNS, PUTF
#
#   DATA GETWPM / EVAL( "GETWPM" ) /
#   DATA OSALOC / EVAL( "OSALOC" ) /
#   DATA INTMOD / EVAL( "INTMODE" ) /
#   DATA REALMD / EVAL( "REALMODE" ) /
#   DATA COMMA / ', ' /
#   DATA I / 'I' /
#   DATA R / 'R' /
#   DATA QUOTE / '\ "' /
#   DATA APOST / '\ '' /
#   DATA DOT / '.' /
#
#           PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
#   CALL PPUSH( "IMMARR" )
#
#           OPEN INPUT SOURCE FILE.
#
#   IF ( .OK == OSPINF( FDI, .MODE, .SYSMODE ) ) GOTO 9800
#   IF ( .OK == OSOPNS( FDI, .INPUT ) ) GOTO 9800

```

```

#
#         ADVANCE PAST FIRST ASSIGNMENT STATEMENT.
#
MORE( 1 ) = 0
REPEAT
    REPEAT
        CALL RATLIN( FDI, LINE, LENGTH, TYPE,
                     MORE, IEV, %9900 )
    UNTIL ( MORE( 1 ) == 0 )
UNTIL ( TYPE == .RATASSIGN )
#
#         LOOK FOR ASSIGNMENT STATEMENTS.  BREAK OUT WHEN
#         CALL TO OSALOC MADE.
#
COUNT = 0
REPEAT
    $(
    CALL RATLIN( FDI, LINE, LENGTH, TYPE, MORE, IEV, %9900 )
    IF ( MORE( 1 ) != 0 ) GOTO 9000
    IF ( TYPE != .RATASSIGN )
        $(
        IF ( TYPE == .RATIF )
            IF ( STRLOC( OSALOC, 0, LINE, 0 ) > 0 )
                BREAK
        $)
#
#         LOOK FOR CALLS TO GETWPM IN THE ASSIGNMENT
#         STATEMENTS.  ADVANCE PAST THE FIRST COMMA TO
#         THE SECOND ARGUMENT.  SKIP THE LEADING SPACES.
#         LOCATE THE COMMA TERMINATING THE SECOND
#         ARGUMENT.  THE SECOND ARGUMENT IS THE DATA
#         MODE, AND THE INTEGER AND REAL MODES ARE
#         PROCESSED.  OUTPUT "IDESC" OR "RDESC" RATFOR
#         MACRO CALLS ( SEE MAPDRV INCLUDE FILE ).
#
ELSE
    $(
    START = STRLOC( GETWPM, 0, LINE, 0 )
    IF ( START > 0 )
        $(
        COUNT = COUNT + 1
        START = STRLOC( COMMA, 1, LINE( START + 6 ), 0 )
                    + START + 6
        WHILE ( STRLOC( LINE( START ), 1,
                      RATSPC, 0 ) > 0 )
        START = START + 1
        FINISH = STRLOC( COMMA, 1, LINE( START ), 0 )
        IF ( STRLOC( INTMOD, 0, LINE( START ),
                    FINISH ) > 0 )
        PREFIX = I
        ELSE IF ( STRLOC( REALMD, 0, LINE( START ),

```

```

                                FINISH ) > 0 )
        PREFIX = R
    ELSE
        GOTO 9100
    IF ( .OK -= PUTF( FDO, " %CDESC", PREFIX ) )
        GOTO 9800
#
#   ADVANCE TO LEADING QUOTE OF FOURTH ARGUMENT,
#   WHICH IS THE NAME OF THE ARRAY.  EXTRACT THE
#   STRING NAME FOR DIMENSION PROMPTING AND
#   OUTPUT OF A RATFOR DEFINITION TO EQUATE THE
#   ARRAY NAME WITH AN ARRAY SLOT.
#
    FINISH = START + FINISH
    START = STRLOC( QUOTE, 1, LINE( FINISH ), 0 )
        + FINISH
    IF ( START == 0 )
        $(
            START = STRLOC( APOST, 1, LINE( FINISH ),
                0 ) + FINISH
            IF ( START == 0 ) GOTO 9200
        $)
    DELIM = LINE( START - 1 )
    FINISH = STRLOC( DELIM, 1, LINE( START ), 0 ) - 1
    IF ( LINE( START + FINISH - 1 ) == DOT )
        FINISH = FINISH - 1
    LINE( START + FINISH ) = .EOS
    IF ( .OK -= PUTF( FDRUNO, "For %S, ",
        LINE( START ) ) ) GOTO 9800
    CALL RNGETI( "Enter number of array dimensions",
        1, 2, 1, DIM, IEV, %9900 )
    IF ( .OK -= PUTF( FDO, "%I( %I )%N", DIM,
        COUNT ) ) GOTO 9800
    IF ( .OK -= PUTF( FDO, " DEFINE(%S,%CAR%i)%N",
        LINE( START ),
        PREFIX, COUNT ) ) GOTO 9800
        $)
    $)
#
#   FINISHED WITH THE SOURCE CODE.
#
    CALL CLOSE( FDI )
    CALL PPOP
    RETURN
#
#   ERROR CONDITIONS:
#
#   INPUT LINE BUFFER OVERFLOW.
#
    9000 CONTINUE

```

```
IEV = -5054
GOTO 9900
#
#           ONLY INTEGER AND REAL MODES ARE SUPPORTED.
#
9100 CONTINUE
IEV = -5062
GOTO 9900
#
#           THE FOURTH ARGUMENT IS IMPROPERLY DELIMITED.
#
9200 CONTINUE
IEV = -5064
GOTO 9900
#
#           RETRIEVE ERROR FROM LOWER ROUTINE.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#
#           CLOSE THE SOURCE FILE AND TAKE THE ALTERNATE
#           RETURN.
#
9900 CONTINUE
CALL CLOSE( FDI )
RETURN 1
#
#
END
```

G.6 RATLIN Routine

```
##--RATLIN  SCAN RATFOR SOURCE FILE FOR RATFOR LINES/CLAUSES
#
#  IDENTIFICATION
#
#      TITLE           RATLIN
#      AUTHOR          GREGG R FABREGAS
#      VERSION         A.01
#      DATE            08-MAR-1986
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#
#  PURPOSE
#
#      THIS ROUTINE SCANS THE INPUT FILE,
#      AND RETURNS THE TEXT AND A
#      STATEMENT CLASSIFICATION.  NOT ALL
#      LANGUAGE FEATURES ARE
#      IDENTIFIED, BUT MANY FEATURES IN A
#      CLEANLY CODED PROGRAM ARE
#      ACCURATELY ISOLATED AND CLASSIFIED.
#      MOST FORTRAN-SPECIFIC
#      FEATURES ARE NOT IDENTIFIED.
#
#  ENTRY POINT
#
#      CALL RATLIN( FD, LINE, LENGTH, TYPE,
#                  MORE, IEV, ERRET )
#
#  ARGUMENT LISTING
#
#      FD      CHAR ARR      FILE DESCRIPTOR FOR RATFOR
#                                     SOURCE CODE.
#      LINE    CHAR ARR      BUFFER RETURNED WITH CLAUSE
#                                     OR LINE.
#      LENGTH  INTEGER       MAXIMUM LENGTH OF
#                                     .EOS-TERMINATED STRING
#                                     RETURNED IN LINE.
#      TYPE    INTEGER       RETURNED TEXT CLASSIFICATION.
#                                     SEE THE RATTAB INCLUDE FILE
#                                     FOR POSSIBLE VALUES.
#      MORE    INT ARR       TWO ELEMENT ARRAY USED
#                                     TO STORED THE
#                                     STATE OF THE ROUTINE
#                                     BETWEEN SUCCESSIVE
#                                     CALLS.  THE FIRST ELEMENT
#                                     IS NON-ZERO IF
#                                     THE ROUTINE RETURNS
```

```

#           BEFORE A COMPLETED
#           CLAUSE OR INSTRUCTION
#           IS RETURNED.  THE
#           SECOND ELEMENT IS USED
#           INTERNALLY FOR
#           OTHER CONTEXTUAL INFORMATION.
#           IEV      INTEGER      INTEGER EVENT VARIABLE.
#           ERRET   LABEL        LABEL IN CALLING ROUTINE FOR
#                               ALTERNATE ERROR RETURNS.
#
# INCLUDE FILES/COMMONS
#
#           GIPSY   INCLUDE      GIPSY GENERAL SYMBOL DEF'NS
#           RATTAB  INCLUDE      RATFOR ANALYSIS DECLARATIONS
#                               AND DEFINITIONS.
#
# LANGUAGE FEATURES REQUIRED
#
#           ALTERNATE RETURNS.
#
# PROGRAM ENVIROMENT
#
#           INITIALLY, MORE( 1 ) SHOULD BE
#           SET TO ZERO.  IT MAY BE READ
#           AFTER THE ROUTINE RETURNS TO
#           DETERMINE IF THE BUFFER WAS
#           EXHAUSTED BEFORE THE ENTIRE CLAUSE
#           COULD BE CONSTRUCTED.  THE
#           VALUE OF MORE( 1 ) MUST NOT BE
#           DISTURBED AFTER CALLING THIS ROUTINE.
#
# DATA FORMAT
#
#           THE FOLLOWING TABLE IDENTIFIES
#           THE POSSIBLE STATEMENT TYPES AND
#           THEIR RESPECTIVE STATEMENT PROTOTYPES:
#           .RATCASE CASE ( ... )
#           .RATDEFINE DEFINE ( ... )
#           .RATFOR   FOR ( ... )
#           .RATIF   IF ( ... )
#           .RATUNTIL UNTIL ( ... )
#           .RATWHILE WHILE ( ... )
#           .RATBREAK BREAK
#           .RATELSE ELSE
#           .RATNEXT NEXT
#           .RATREPEAT REPEAT
#           .RATCOND CONDITION ... <EOL>
#           .RATDO   DO ... <EOL>
#           .RATINCL INCLUDE ... <EOL>
#           .RATOTHER OTHERWISE ... <EOL>
#           .RATDONE END ... <EOL>

```



```

#       .RATASSIGN <UNSUBSCRIPTED IDENTIFIER> = ... <EOL>
#       .RATCOMM # ... .EOL
#       .RATBEGIN $(, {, OR [
#       .RATEND $), }, OR ]
#       .RATLABEL <SEQUENCE OF DIGITS>
#       .RATUNK <NOT MATCHABLE TO ANY OF THE ABOVE>
#       <EOL> SHOULD BE INTERPRETED
#       AS A SEMI-COLON (;) OR AN
#       END-OF-LINE CHARACTER (.EOL) NOT IMMEDIATELY
#       PRECEDED BY A LINE-CONTINUATION CHARACTER.
#

```

```

# LIMITATIONS
#

```

```

#       THIS ROUTINE DOES NOT PRESUME
#       TO ROBUSTLY DETECT ALL OF THE
#       ABOVE CLASSES WITHOUT ERROR,
#       IN ALL PROGRAMS. HOWEVER, THIS
#       ROUTINE SHOULD RETURN ACCURATE
#       RESULTS FOR MOST PROGRAMS.
#

```

```

# ROUTINES CALLED
#

```

```

#       PPUSH           PUSH PROGRAM NAME ONTO ERROR STACK
#                       (GIPSY PRIMITIVE)
#       LINFIL          FILL LINE BUFFER. (GIPSY ROUTINE)
#       GETINX          RETURN COMMAND INDEX GIVEN COMMAND
#                       STRING (GIPSY PRIMITIVE)
#       MEMBER          DETERMINE IF CHAR IS MEMBER OF A SET
#                       (GIPSY PRIMITIVE)
#       GGETC           GET A SINGLE CHARACTER
#                       (GIPSY PRIMITIVE)
#       GPTBK           PLACE A CHARACTER BACK IN INPUT STREAM
#                       (GIPSY PRIMITIVE)
#       PPOP            POP PROGRAM NAME FROM THE ERROR STACK
#                       (GIPSY PRIMITIVE)
#       OSGIEV          GET EVENT VARIABLE (IEV) FROM OS SAVE
#                       AREA (GIPSY OS KERNEL)
#

```

```

# REMARKS
#

```

```

#       THE CALLING ROUTINE SHOULD INCLUDE
#       THE RATTAB COMMON TO PROPERLY
#       INTERPRET THE RESULTS RETURNED BY THIS ROUTINE.
#

```

```

#*****
#

```

```

# INCLUDE GIPSY
#

```

```

# SUBROUTINE RATLIN( FD, LINE, LENGTH, TYPE, MORE, IEV, * )

```

```

#
#
INCLUDE RATTAB
DEFINE(CONTEXT,MORE( 1 ))
DEFINE(SAVE,MORE( 2 ))
DEFINE(PAREN,MORE( 2 ))
DEFINE(CONT,MORE( 2 ))
#
INTEGER LENGTH, TYPE, MORE( 2 ), IEV
CHARACTER FD( .FDLENGTH ), LINE( LENGTH )
INTEGER COUNT, FULL
CHARACTER CH
LOGICAL MUNCH, CHKEOL
INTEGER GETINX, MEMBER, GGETC, GPTBK, OSGIEV
#
#           PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
CALL PPUSH( "RATLIN" )
#
#           CLEAR THE LINE BUFFER, AND RETURN TO THE
#           PREVIOUS BREAKPOINT IF THE BUFFER WAS FULL.
#
COUNT = 0
FULL = LENGTH - 1
GOTO ( 1000, 2000, 3000, 3100, 3200, 3300, 3400, 4000,
      5000, 5100, 5200, 6000, 6100, 7000 ), CONTEXT
#
#           CLEAR FLAGS TO MUNCH REST OF CLAUSE (MUNCH),
#           AND TO EAT THE NEXT CHARACTER IF IT IS AN
#           END-OF-LINE CHARACTER (CHKEOL).
#
MUNCH = .FALSE.
CHKEOL = .FALSE.
#
#           SCAN OVER LEADING TABS OR SPACES AND NOTE
#           THE CHARACTER POSITION.
#
CONTEXT = 1
1000 CALL LINFIL( FD, LINE, FULL, COUNT,
                RATSPC, %8000, IEV, %9900 )
SAVE = COUNT
#
#           SCAN OVER A SEQUENCE OF LETTERS.
#
CONTEXT = 2
2000 CALL LINFIL( FD, LINE, FULL, COUNT,
                RATLET, %8000, IEV, %9900 )
#
#           IF A LETTER SEQUENCE IS FOUND, TRY TO MATCH IT
#           TO A KNOWN KEYWORD.
#

```

```

IF ( SAVE < COUNT )
  $(
    LINE( COUNT + 1 ) = .EOS
    TYPE = GETINX( RATKEY, RATKTP, LINE( SAVE + 1 ),
      .RATKMAX, .RATKLEN, 0 )
    CASE ( TYPE )
      $(
#
#         FOR CASE, DEFINE, FOR, IF, UNTIL, AND WHILE,
#         FIND A LEFT PARENTHESIS AND ITS CORRESPONDING
#         RIGHT PARENTHESIS.  THROW AWAY THE END-OF-LINE
#         CHARACTER IF IT IMMEDIATELY FOLLOWS.
#
#         CONDITION .RATCASE, .RATDEFINE, .RATFOR,
#                   .RATIF, .RATUNTIL, .RATWHILE:
#           CONTEXT = 3
3000      CALL LINFIL( FD, LINE, FULL, COUNT,
#                   RATSPC, %8000, IEV, %9900 )
#           PAREN = 0
3100      CONTEXT = 5
#           REPEAT
#             $(
3200      CALL LINFIL( FD, LINE, FULL, COUNT,
#                   RATNUL, %8000, IEV, %9900 )
#             IF ( LINE( COUNT ) == RATLFP )
#               PAREN = PAREN + 1
#             ELSE
#               IF ( LINE( COUNT ) == RATRTP )
#                 PAREN = PAREN - 1
#             $(
#           UNTIL ( PAREN == 0 )
#           CHKEOL = .TRUE.
#
#         FOR BREAK, ELSE, NEXT, AND REPEAT, CHECK AND
#         THROW AWAY THE END-OF-LINE CHARACTER.
#
#         CONDITION .RATBREAK, .RATELSE, .RATNEXT, .RATREPEAT:
#           CHKEOL = .TRUE.
#
#         FOR CONDITION, DO, INCLUDE, OTHERWISE, AND
#         END, SCAN TO THE END OF THE CLAUSE.
#
#         CONDITION .RATCOND, .RATDO, .RATINCL,
#                   .RATOTHER, .RATDONE:
#           MUNCH = .TRUE.
#
#         IF NOT IDENTIFIED, CHECK IF THE NEXT
#         NON-BLANK CHARACTER IS AN EQUAL SIGN (=), AND
#         IF SO, CHANGE THE TYPE TO .RATASSIGN, AND IN
#         EITHER CASE, SCAN TO THE END OF THE CLAUSE.
#

```

```

        CONDITION .RATUNK:
            CONTEXT = 6
3300     CALL LINFIL( FD, LINE, FULL, COUNT,
                RATSPC, %8000, IEV, %9900 )
            CONTEXT = 7
3400     CALL LINFIL( FD, LINE, FULL, COUNT,
                RATNUL, %8000, IEV, %9900 )
            IF ( LINE( COUNT ) == RATASN )
                TYPE = .RATASSIGN
            IF ( LINE( COUNT ) /= .EOL )
                MUNCH = .TRUE.
        $)
    $)
ELSE
#
#     IF DIGITS FOLLOW THE LEADING SPACES, RETURN
#     THEM AS A LABEL.
#
    $(
        CONTEXT = 8
        4000 CALL LINFIL( FD, LINE, FULL, COUNT,
            RATDIG, %8000, IEV, %9900 )
#
#     IF NO DIGITS, CHECK FOR ALTERNATE CLAUSE
#     FORMATS.
#
        IF ( SAVE >= COUNT )
            $(
                CONTEXT = 9
                5000 CALL LINFIL( FD, LINE, FULL, COUNT,
                    RATNUL, %8000, IEV, %9900 )
                CONTEXT = 10
                IF ( LINE( COUNT ) == RATDOL )
                    5100 CALL LINFIL( FD, LINE, FULL, COUNT,
                        RATNUL, %8000, IEV, %9900 )
                LINE( COUNT + 1 ) = .EOS
                TYPE = GETINX( RATALT, RATATP, LINE( SAVE + 1 ),
                    .RATAMAX, .RATALEN, 0)
                CASE ( TYPE )
                    $(
#
#     FOR COMMENTS, SCAN TO THE .EOL CHARACTER.
#
                        CONDITION .RATCOMM:
                            CONTEXT = 11
                            REPEAT
                    5200     CALL LINFIL( FD, LINE, FULL, COUNT,
                                RATNUL, %8000, IEV,
                                %9900 )
                                UNTIL ( LINE( COUNT ) == .EOL )
#

```

```

#           IF A BLOCK BEGIN OR BLOCK END IS FOUND, ADD
#           THE END-OF-LINE CHARACTER IF IT IS NEXT.
#
#           CONDITION .RATBEGIN, .RATEND:
#               CHKEOL = .TRUE.
#
#           FOR ANYTHING ELSE, SCAN TO THE END OF THE
#           CLAUSE.
#
#           CONDITION .RATUNK:
#               MUNCH = .TRUE.
#           $)
#       $)
ELSE
    TYPE = .RATLABEL
#)
#
#           IF ENABLED, SCAN THE INPUT FILE FOR ANY
#           END-OF-LINE CONDITION:
#           (1) .EOL NOT PRECEDED BY A CONTINUATION FLAG,
#           OR (2), A SEMI-COLON.
#           A BACKSLASH CANCELS THE INTERPRETATION OF THE
#           NEXT CHARACTER.
#
IF ( MUNCH )
    $(
    CONT = 0
    REPEAT
        $(
        CONTEXT = 12
6000 CALL LINFIL( FD, LINE, FULL, COUNT,
                RATNUL, %8000, IEV, %9900 )
        IF ( LINE( COUNT ) == .EOL )
            $(
            IF ( CONT == 0 )
                BREAK
            $)
        CONT = 0
        IF ( MEMBER( LINE( COUNT ), RATCHR,
                1, .RATCLEN ) > 0 )
            CONT = 1
        ELSE IF ( LINE ( COUNT ) == RATBSL )
            $(
            CONTEXT = 13
6100 CALL LINFIL( FD, LINE, FULL, COUNT,
                RATNUL, %8000, IEV, %9900 )
            $)
        ELSE IF ( LINE( COUNT ) == RATSEM )
            BREAK
        $)
    $)

```

```

#
#           IF ENABLED, APPEND THE NEXT CHARACTER TO THE
#           LINE BUFFER IF IT IS A .EOL OR SEMI-COLON.
#
IF ( CHKEOL )
  $(
    IF ( .OK -= GGETC( FD, CH ) ) GOTO 9800
    IF ( .OK -= GPTBK( FD, CH ) ) GOTO 9800
    CONTEXT = 14
    IF ( CH == .EOL ! CH == RATSEM )
      7000 CALL LINFIL( FD, LINE, FULL, COUNT,
                    RATNUL, %8000, IEV, %9900 )
  $)
#
#           THE CLAUSE IS COMPLETED OR THE BUFFER IS FULL,
#           SO TACK ON THE .EOS
#           CHARACTER AND TAKE THE NORMAL RETURN.
#
CONTEXT = 0
8000 CONTINUE
LINE( COUNT + 1 ) = .EOS
CALL PPOP
RETURN
#
#           RETRIEVE ERROR IEV FROM A LOWER ROUTINE.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#
#           TAKE THE ALTERNATE ERROR RETURN.
#
9900 CONTINUE
RETURN 1
#
#
END

```

G.7 LINFIL Routine

```
#--LINFIL  FILL A LINE BUFFER UNTIL SET MEMBER FOUND OR FULL
#
#  IDENTIFICATION
#
#      TITLE           LINFIL
#      AUTHOR          GREGG R FABREGAS
#      VERSION         A.01
#      DATE            08-MAR-1986
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#
#  PURPOSE
#
#      THIS ROUTINE READS CHARACTERS
#      FROM AN INPUT FILE INTO A LINE
#      BUFFER.  THE ROUTINE RETURNS
#      IF ANY ONE OF THE FOLLOWING FOUR
#      CONDITIONS IS SATISFIED:
#          (1) A CHARACTER CAN NOT BE READ
#              FROM THE INPUT FILE,
#          (2) A CHARACTER CAN NOT BE PUT
#              BACK INTO THE INPUT BUFFER,
#          (3) THE LINE BUFFER IS FILLED, OR
#          (4) A MATCH IS MADE BETWEEN A READ CHARACTER
#              AND ANY CHARACTER IN THE TERMINATOR SET.
#
#  ENTRY POINT
#
#      CALL LINFIL( FD, LINE, LENGTH, COUNT,
#                  SET, ALTRET, IEV, ERRET )
#
#  ARGUMENT LISTING
#
#      FD      CHAR ARR      INPUT FILE DESCRIPTOR.
#      LINE    CHAR ARR      LINE BUFFER TO BE FILLED.
#      LENGTH  INTEGER       MAXIMUM NUMBER OF CHARACTERS
#                              THAT CAN BE
#                              STORED IN THE LINE BUFFER.
#      COUNT   INTEGER       NUMBER OF CHARACTERS ALREADY
#                              IN BUFFER, AND NUMBER OF
#                              CHARACTERS RETURNED.
#      SET     CHAR ARR      .EOS-TERMINATED STRING WITH
#                              TERMINATOR CHARACTERS TO
#                              CAUSE PREMATURE RETURN.
#      ALTRET  LABEL         LABEL IN CALLING ROUTINE TO
#                              RECEIVE CONTROL WHEN THE
#                              LINE BUFFER IS FULL.
```

```

#       IEV       INTEGER       INTEGER EVENT VARIABLE.
#       ERRET     LABEL         LABEL IN CALLING ROUTINE
#                                     FOR ERROR PROCESSING.
#
# INCLUDE FILES/COMMONS
#
#       GIPSY     INCLUDE       GIPSY GENERAL SYMBOL DEF'NS
#
# LANGUAGE FEATURES REQUIRED
#
#       ALTERNATE RETURNS.
#
# PROGRAM ENVIROMENT
#
#       THE COUNT ARGUMENT IS NOT CLEARED
#       IN THIS ROUTINE. IF THE
#       CALLING ROUTINE IS EXPECTING A
#       FRESH BUFFER, SET COUNT TO ZERO.
#
# DATA FORMAT
#
#       THE LINE BUFFER WILL NOT BE
#       .EOS-TERMINATED UNLESS THE .EOS
#       CHARACTER CAME FROM THE INPUT FILE.
#       END-OF-LINE CHARACTERS ARE
#       IGNORED UNLESS THE CHARACTERS
#       ARE IN THE TERMINATOR SET. ANY
#       VALID CHARACTERS MAY BE IN THE
#       TERMINATOR SET, EXCEPT THE .EOS CHARACTER.
#
# LIMITATIONS
#
#       THE LENGTH ARGUMENT MUST BE
#       GREATER THAN ZERO. NO CHARACTERS
#       ARE RETURNED IF COUNT IS NOT LESS
#       THAN THE LINE LENGTH.
#
# ROUTINES CALLED
#
#       PPUSH     PUSH PROGRAM NAME ONTO ERROR STACK
#                (GIPSY PRIMITIVE)
#       STRLEN    FIND LENGTH OF .EOS-TERMINATED STRING.
#                (GIPSY OS KERNEL)
#       GGETC     GET A SINGLE CHARACTER
#                (GIPSY PRIMITIVE)
#       MEMBER    DETERMINE IF CHAR IS MEMBER OF A SET
#                (GIPSY PRIMITIVE)
#       GPTBK     PLACE A CHARACTER BACK IN INPUT STREAM
#                (GIPSY PRIMITIVE)
#       PPOP      POP PROGRAM NAME FROM THE ERROR STACK
#                (GIPSY PRIMITIVE)

```



```

#           OSGIEV           GET EVENT VARIABLE (IEV) FROM OS SAVE
#                               AREA   (GIPSY OS KERNEL)
#
#  REMARKS
#
#           THE ERROR RETURN IS TAKEN IF THE
#           FIRST CHARACTER CAN NOT BE READ
#           OR THE LAST CHARACTER READ CAN
#           NOT BE PUT BACK INTO THE INPUT
#           BUFFER.  THE ALTERNATE RETURN IS
#           TAKEN IF THE LINE BUFFER IS
#           FULL ( COUNT >= LENGTH ).  ALL
#           OTHER CONDITIONS, NOTABLY READ
#           FAILURES AFTER THE FIRST CHARACTER,
#           USE THE NORMAL RETURN.
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
#  INCLUDE GIPSY
#
#  SUBROUTINE LINFIL( FD, LINE, LENGTH, COUNT,
#                   SET, *, IEV, * )
#
#
#  INTEGER LENGTH, COUNT, IEV
#  CHARACTER FD( .FDLENGTH ), LINE( LENGTH ), SET( .ARB )
#  INTEGER SETSIZ
#  CHARACTER CH
#  LOGICAL FULL, PASS
#  INTEGER GGETC, MEMBER, STRLEN, GPTBK, OSGIEV
#
#           PUSH ROUTINE NAME ONTO ERROR TRACE STACK.
#
#  CALL PPUSH( "LINFIL" )
#
#           ASSUME THE BUFFER IS NOT FULL.  AUTOMATICALLY
#           EXECUTE THE WHILE LOOP AT LEAST ONCE IF THE
#           TERMINATOR SET IS EMPTY.
#
#  FULL = .FALSE.
#  SETSIZ = STRLEN( SET )
#  PASS = ( SETSIZ == 0 )
#
#           PICK UP CHARACTERS UNTIL THE LINE BUFFER IS
#           FULL OR A MATCH IS FOUND IN THE TERMINATOR SET.
#           RETURN THE LAST CHARACTER TO THE INPUT BUFFER.
#
#  IF ( .OK == GGETC( FD, CH ) ) GOTO 9800
#  WHILE ( PASS ! MEMBER( CH, SET, 1, SETSIZ ) > 0 )

```

```

$(
  IF ( COUNT < LENGTH )
    $(
      COUNT = COUNT + 1
      LINE( COUNT ) = CH
    $)
  ELSE
    $(
      FULL = .TRUE.
      BREAK
    $)
  IF ( .OK -= GGETC( FD, CH ) ) GOTO 1000
  PASS = .FALSE.
  $)
  IF ( .OK -= GPTBK( FD, CH ) ) GOTO 9800
#
#       AN END-OF-FILE CONDITION IS NOT FATAL THE
#       FIRST TIME AROUND.  TAKE THE NORMAL RETURN IF
#       THE LINE BUFFER IS NOT FULL.
#
1000 CONTINUE
CALL PPOP
IF ( FULL )
  RETURN 1
RETURN
#
#       RETRIEVE THE ERROR IEV FROM THE LOWER ROUTINE
#       AND TAKE THE ERROR RETURN.
#
9800 CONTINUE
IEV = OSGIEV( IEV )
RETURN 2
#
#
END

```

G.8 RATTAB Include File

```
#--RATTAB INCLUDE FILE FOR THE RATLIN FILE SCANNER ROUTINE
#
# IDENTIFICATION
#
# TITLE RATTAB
# AUTHOR GREGG R FABREGAS
# VERSION A.01
# DATE 08-MAR-1986
# LANGUAGE RATFOR
# SYSTEM VAX 11/780
#
#
# PURPOSE
#
# THIS INCLUDE FILE CONTAINS SYMBOLS
# AND DECLARATIONS USED BY THE
# RATLIN ROUTINE TO IDENTIFY
# STATEMENT CLAUSES IN A RATFOR SOURCE
# FILE. ALL DECLARATIONS BEGIN
# WITH "RAT" AND ALL RATFOR
# DEFINITIONS BEGIN WITH ".RAT".
#
# ENTRY POINT
#
# NONE
#
# ROUTINES CALLED
#
# NONE
#
#
#*****
#
#
# DEFINE(.RATCLEN,11)
# CHARACTER RATSPC( 3 ), RATLET( 27 ), RATNUL( 1 )
# CHARACTER RATDIG( 11 ), RATASN, RATDOL, RATBSL
# CHARACTER RATCHR( .RATCLEN ), RATLFP, RATRTP, RATSEM
# DATA RATSPC / EVAL( " " ) /
# DATA RATLET / EVAL( "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ) /
# DATA RATNUL / EVAL( "" ) /
# DATA RATDIG / EVAL( "1234567890" ) /
# DATA RATCHR / EVAL( "+-=*,|&(!_" ) /
# DATA RATLFP / '(' /
# DATA RATRTP / ')' /
# DATA RATASN / '=' /
# DATA RATDOL / '$' /
# DATA RATBSL / '\\'
```

```

DATA RATSEM / ';' /
#
#           THE FOLLOWING LIST DEFINES ALL OF THE VALID
#           TYPES RETURNED BY THE RATLIN ROUTINE.
#
DEFINE(.RATUNK,0)
DEFINE(.RATCASE,1)
DEFINE(.RATDEFINE,2)
DEFINE(.RATFOR,3)
DEFINE(.RATIF,4)
DEFINE(.RATUNTIL,5)
DEFINE(.RATWHILE,6)
DEFINE(.RATBREAK,7)
DEFINE(.RATELSE,8)
DEFINE(.RATNEXT,9)
DEFINE(.RATREPEAT,10)
DEFINE(.RATCOND,11)
DEFINE(.RATDO,12)
DEFINE(.RATINCL,13)
DEFINE(.RATOTHER,14)
DEFINE(.RATASSIGN,15)
DEFINE(.RATCOMM,16)
DEFINE(.RATBEGIN,17)
DEFINE(.RATEND,18)
DEFINE(.RATLABEL,19)
DEFINE(.RATDONE,20)
#
#           THIS SECTION DECLARES ALL OF THE ALPHABETIC
#           TOKENS RECOGNIZED BY RATFOR.
#
DEFINE(.RATKMAX,16)
DEFINE(.RATKLEN,10)
CHARACTER RATKEY( .RATKLEN, .RATKMAX )
INTEGER RATKTP( .RATKMAX )
DATA RATKEY / EVAL( "x", .RATKLEN ),
    EVAL( "CASE", .RATKLEN ),
    EVAL( "DEFINE", .RATKLEN ),
    EVAL( "FOR", .RATKLEN ),
    EVAL( "IF", .RATKLEN ),
    EVAL( "UNTIL", .RATKLEN ),
    EVAL( "WHILE", .RATKLEN ),
    EVAL( "BREAK", .RATKLEN ),
    EVAL( "ELSE", .RATKLEN ),
    EVAL( "NEXT", .RATKLEN ),
    EVAL( "REPEAT", .RATKLEN ),
    EVAL( "CONDITION", .RATKLEN ),
    EVAL( "DO", .RATKLEN ),
    EVAL( "INCLUDE", .RATKLEN ),
    EVAL( "OTHERWISE", .RATKLEN ),
    EVAL( "END", .RATKLEN ) /
DATA RATKTP / .RATUNK, .RATCASE, .RATDEFINE,

```

```

        .RATFOR, .RATIF, .RATUNTIL,
        .RATWHILE, .RATBREAK, .RATELSE,
        .RATNEXT, .RATREPEAT, .RATCOND,
        .RATDO, .RATINCL, .RATOTHER, .RATDONE /
#
#           THIS SECTION DECLARES ALL OF THE
#           NON-ALPHANUMERIC TOKENS RECOGNIZED BY RATFOR.
#
DEFINE(.RATAMAX,8)
DEFINE(.RATALEN,3)
CHARACTER RATALT( .RATALEN, .RATAMAX )
INTEGER RATATP( .RATAMAX )
DATA RATALT / EVAL( "×", .RATALEN ),
              EVAL( "#", .RATALEN ),
              EVAL( "$(", .RATALEN ),
              EVAL( "$)", .RATALEN ),
              EVAL( "{", .RATALEN ),
              EVAL( "}", .RATALEN ),
              EVAL( "[", .RATALEN ),
              EVAL( "]", .RATALEN ) /
DATA RATATP / .RATUNK, .RATCOMM, .RATBEGIN,
              .RATEND, .RATBEGIN,
              .RATEND, .RATBEGIN, .RATEND /

```

H APPENDIX H: MKRAN COMMAND

What follows are the files associated with the GIPSY command, MKRAN. This includes the following files:

1. GIPSY_DOC:MKRAN.DOC - on-line documentation,
2. GIPSY_RUN:MKRAN.RUN - a run-file to test the command,
3. GIPSY_LNK:MKRAN.LNK - a link file which instructs the system on how to build the command,
4. GIPSY_SRC:DMKRAN.RAT - the command driver,
5. GIPSY_SRC:MKRNC.RAT - the main work routine.

H.1 MKRAN Documentation

*MKRAN Convert seq. list file to random msg file w/B-tree

VERSION: A.01 DATE: 07-25-83 CONTACT: G. Fabregas

ACTION: This command takes a sequential list file, notably the lists of routines and include files on GIPSY_ETC, and writes a SIF file containing a B-tree and a text section. The B-tree is made by packing the first field of LENGTH characters into records with the same width as the second field, containing text. The list file is assumed to have names in alphabetical order in the first field. The resulting message file allows a constant access time for the retrieval of the text corresponding to the name used as a key. These message files are particularly important in the creation of the heading on each source file, by the HEADER command.

SOURCE: Disk, sequential file with a key and text field on each line

DESTINATION: Disk, SIF file in line format.

FLAGS: NONE

QUESTIONS: (1) Key length

The key length is the number of characters, starting from the left, that are allocated for the key value. Note that trailing spaces are part of that value.

(2) Tree depth

This fixes the size of the tree, that is, the maximum number of keys the tree may contain. To determine the best depth, divide the number of characters used in the text field (80 character line assumed) by the length of the key field, and discard the fraction. This is called the BASE. Choose the smallest DEPTH such that BASE**DEPTH is greater than or equal to the number of entries in the list file.

COMMAND STRING EXAMPLE:

```
G: MKRAN ROUTINE.LST > ROUTINE.MSG
ENTER KEY LENGTH ( D = 10, 1 TO 40 ) -- 10
ENTER TREE DEPTH ( D = 4, 1 TO 10 ) -- 4
G:
```

This example will generate a SIF with enough space for

2401, or 7×4 , entries. The base, seven, is determined as follows: $\text{base} = \text{int}((80 - \text{length}) / \text{length}) = 70 / 10 = 7$. The resulting SIF will have an image size of 2801 lines x 70 characters per line.

ALGORITHM: The tree contains BASE slots in each node containing keys. No space is used for pointers. For each level of the tree, such that the root node is level 1, a key is inserted in a node on that level if the count, starting at zero, is evenly divisible by $\text{BASE} \times (\text{DEPTH} - \text{LEVEL})$. This puts all the keys in a given level, in all the levels below. The text from each entry is stored after the tree.

H.2 MKRAN Run-file

```

$!
$!      TEST THE COMMAND 'MKRAN'
$!
$ MKRAN GIPSY_DATA:MKRAN.DAT > MKRAN.MSG
  4
  1
$!
$!      MKRAN.MSG IS A SIF FILE WITH THE
$!      DIGITS '1' THROUGH '9' ON THE
$!      FIRST LINE AND NINE LINES OF TEXT
$!      BEGINNING WITH THE WORD,
$!      'TEXT', AND FOLLOWED BY THE NAME
$!      OF THE DIGIT.  THE DIGIT
$!      NAMES SHOULD BE IN ORDER.
$!
$ EXSIF MKRAN.MSG
  FORM (10(1X,70A1))
  BLK 1 10
  DONE
$!
$!      CLEAN UP AND EXIT
$!
$ DELETE MKRAN.MSG
$ EXIT
```

H.3 MKRAN Link File

⌘ GENCMD MKRAN DMKRAN,MKRNC

H.4 DMKRAM Routine

```
#--DMKRAM CONVERT SEQUENTIAL LIST FILE TO SIF FILE W/BTREE
#
# IDENTIFICATION
#
# TITLE DMKRAM
# AUTHOR GREGG R FABREGAS
# VERSION A.01
# DATE 21-JUL-1983
# LANGUAGE RATFOR
# SYSTEM VAX 11/780
#
#
# PURPOSE
#
# THIS IS THE DRIVER FOR THE 'MKRAM'
# COMMAND. IT IS PRIMARILY
# USED TO CONVERT THE SEQUENTIAL
# LISTS OF INCLUDE FILES AND GIPSY
# SOURCE ROUTINES INTO A B+ TREE,
# IN SIF FORMAT FOR FASTER ACCESS.
# THIS ROUTINE ONLY ALLOCATES THE
# NECESSARY SPACE AFTER GETTING OR
# COMPUTING NEEDED PARAMETERS.
#
# ENTRY POINT
#
# CALL DMKRAM( WORK, ERRET )
#
# ARGUMENT LISTING
#
# WORK INT ARR WORK SPACE FOR ARRAYS
# ERRET LABEL ALTERNATE ERROR RETURN
#
# INCLUDE FILES/COMMONS
#
# MACA1 INCLUDE GIPSY GENERAL SYMBOL DEF'NS
# GIPCOM COMMON GIPSY COMMAND LINE COMMON DATA
# ERROR COMMON ERROR TRACE STACK COMMON DATA
#
# LANGUAGE FEATURES REQUIRED
#
# ALTERNATE RETURNS
#
# ROUTINES CALLED
#
# PPU SH PUSH PROGRAM NAME ONTO ERROR STACK
# (GIPSY ROUTINE)
# RNGETI PROMPT FOR AN INTEGER WITHIN A RANGE
```

```

#           (GIPSY ROUTINE)
#   GETWP   GET WORK ARRAY POINTER   (GIPSY ROUTINE)
#   OSALOC  CHECK AND EXTEND USER WORK SPACE
#           (SYSTEM ROUTINE)
#   MKRNC   NUMBER CRUNCHER FOR THE 'MKRAN' CMD
#           (GIPSY ROUTINE)
#   PPOP    POP PROGRAM NAME FROM ERROR STACK
#           (GIPSY ROUTINE)
#
#
#*****
#
#   SUBROUTINE DMKLAN( WORK, * )
#
#   IMPLICIT INTEGER ( A - Z )
#
#   INCLUDE MACAL
#   INCLUDE GIPCOM
#   INCLUDE ERROR
#
#   INTEGER WORK( .ARB )
#
#           PUT PROGRAM NAME ON ERROR STACK
#
#   CALL PPUSH( "DMKLAN" )
#
#           GET TREE PARAMETERS
#
#   CALL RNGETI( "ENTER KEY LENGTH", 1, 40, 10,
#               LENGTH, IEV, %9900 )
#   CALL RNGETI( "ENTER TREE DEPTH", 1, 10, 4,
#               DEPTH, IEV, %9900 )
#
#           COMPUTE REMAINING PARAMETERS
#
#   WIDTH = 80
#   MAXLEN = WIDTH - LENGTH
#   BASE = MAXLEN / LENGTH
#   NLEVS = DEPTH + 1
#
#           COMPUTE NEEDED WORK SPACE
#
#   NXT = 1
#   TREE = GETWP( NXT, .INTMODE, DEPTH * WIDTH )
#   TEXT = GETWP( NXT, .INTMODE, MAXLEN )
#   BUFFER = GETWP( NXT, .SYSMODE, WIDTH )
#   LINE = GETWP( NXT, .INTMODE, WIDTH )
#   ENTRY = GETWP( NXT, .INTMODE, DEPTH )
#   LEVEL = GETWP( NXT, .INTMODE, NLEVS )

```

```

LAST = GETWP( NXT, .INTMODE, NLEVS )
#
#       ALLOCATE WORK SPACE
#
IF ( .OK -= OSALOC( NXT ) ) GOTO 9000
#
#       CALL ROUTINE TO DO THE WORK
#
CALL MKRNC( FDI1, FDI1, WORK( TREE ), WIDTH,
           DEPTH, WORK( TEXT ), MAXLEN,
           WORK( BUFFER ), WORK( LINE ),
           WORK( ENTRY ), WORK( LEVEL ),
           WORK( LAST ), NLEVS, BASE, LENGTH, IEV, %9900 )
#
#       FINISHED - POP NAME OFF STACK
#
CALL PPOP
RETURN
#
#       INSUFFICIENT WORK SPACE
#
9000 CONTINUE
IEV = -5010
GOTO 9900
#
#       TAKE THE ALTERNATE RETURN
#
9900 CONTINUE
RETURN 1
#
#
END

```

H.5 MKRNC Routine

```
#--MKRNC          NUMBER CRUNCHER FOR THE 'MKRAN' COMMAND
#
# IDENTIFICATION
#
#     TITLE          MKRNC
#     AUTHOR         GREGG R FABREGAS
#     VERSION        A.01
#     DATE           21-JUL-1983
#     LANGUAGE       RATFOR
#     SYSTEM         VAX 11/780
#
#
# PURPOSE
#
#     THIS ROUTINE DOES THE CONVERSION
#     OF A SEQUENTIAL FILE TO A
#     RANDOM FILE.  THE SEQUENTIAL FILE
#     CONTAINS TWO TYPES OF LINES:
#     A COMMENT LINE BEGINNING WITH '#',
#     AND INFORMATION LINES WITH
#     TWO FIELDS.  THE FIRST FIELD
#     CONTAINS A FIXED-LENGTH CHARACTER
#     KEY, WITH SPACES PADDED ON THE
#     RIGHT.  THE SECOND FIELD IS
#     INTERPRETED AS CHARACTER TEXT.
#     THE KEY IS PACKED INTO A 'B+'
#     TREE IN THE FIRST DATA RECORDS.
#     SOME KEYS WILL BE SITUATED ON
#     MORE THAN ONE LEVEL OF THE TREE,
#     BUT AT THE LEVEL ABOVE THE LEAF
#     NODES, THE LIST OF KEYS IS BOTH
#     COMPLETE AND UNIQUE.  THE TEXT
#     COMPRISES THE LEAF NODES.  THERE
#     ARE NO POINTERS IN THIS TREE AS
#     IT'S SIZE IS FIXED WHEN IT IS
#     CREATED.  HOWEVER, THERE IS UNUSED
#     SPACE IN THE INTERNAL NODES IF
#     THE DATA DOES NOT FILL UP THE TREE.
#
# ENTRY POINT
#
#     CALL MKRNC( FDI, FDO, TREE, WIDTH, DEPTH,
#               TEXT, MAXLEN, BUFFER, LINE,
#               ENTRY, LEVEL, LAST, NLEVS,
#               BASE, LENGTH, IEV, ERRET )
#
# ARGUMENT LISTING
#
```

```

#       FDI       CHR ARR       INPUT FILE DESCRIPTOR
#       FDO       CHR ARR       OUTPUT FILE DESCRIPTOR
#       TREE      INT MAT       WORK SPACE FOR INTERNAL
#                                     NODES; ONE LINE
#                                     FOR EACH INTERNAL LEVEL
#       WIDTH     INTEGER       INPUT FILE LINE WIDTH,
#                                     IN CHARACTERS
#       DEPTH     INTEGER       NUMBER OF LEVELS FOR
#                                     INTERNAL NODES
#       TEXT      CHR ARR       TEXT EXTRACTED FROM SECOND
#                                     FIELD OF INPUT LINE
#       MAXLEN    INTEGER       MAXIMUM LENGTH OF TEXT,
#                                     IN CHARACTERS
#       BUFFER    PCH ARR       PACKED CHARACTER ARRAY TO
#                                     RECEIVE AN INPUT LINE FROM
#                                     SYSTEM
#       LINE      CHR ARR       UNPACKED CHARACTER ARRAY
#                                     TO RECEIVE UNPACKED DATA
#                                     FROM THE BUFFER ARRAY
#       ENTRY     INT ARR       MAINTAINS COUNT OF THE NUMBER
#                                     OF KEYS STORED IN EACH NODE
#                                     LEVEL IN THE 'TREE' ARRAY
#       LEVEL     INT ARR       MAINTAINS RELATIVE RECORD
#                                     NUMBERS FOR THE NEXT OUTPUT
#                                     TO EACH NODE LEVEL
#       LAST      INT ARR       CONTAINS THE HIGHEST RECORD
#                                     NUMBER TO BE USED FOR EACH
#                                     LEVEL OF THE 'B+' TREE
#       NLEVS     INTEGER       TOTAL NUMBER OF LEVELS IN
#                                     THE 'B+' TREE
#       BASE      INTEGER       MAXIMUM NUMBER OF KEYS STORED
#                                     IN AN INTERNAL NODE
#       LENGTH    INTEGER       LENGTH OF KEY FIELD,
#                                     IN CHARACTERS
#       IEV       INTEGER       INTEGER EVENT VARIABLE
#       ERRET     LABEL        ALTERNATE ERROR RETURN

```

```

# INCLUDE FILES/COMMONS

```

```

#       MACAL     INCLUDE      GIPSY GENERAL SYMBOL DEF'NS

```

```

# ALGORITHM

```

```

#       NODES ARE COMPOSED OF 'BASE' SLOTS
#       FOR KEYS. SINCE THE TREE IS
#       A FIXED STRUCTURE AT CREATION,
#       THERE IS NO NEED FOR POINTERS.
#       INSTEAD, THE LEVELS OF THE TREE
#       AND THE NODES WITHIN EACH LEVEL
#       CAN BE LOCATED WITH THE FOLLOWING FORMULAE:
#           LEVEL( 1 ) = 1

```

```

#           LEVEL( 2 ) = 2
#           LEVEL( 3 ) = LEVEL( 2 ) + BASE
#           LEVEL( 4 ) = LEVEL( 3 ) + BASE ** 2
#           ...
#           LEVEL( I ) = LEVEL( I - 1 ) + BASE ** ( I - 2 )
#           KEYLOC = [ 0, BASE - 1 ]
#           <KEY LOCATION WITHIN NODE>
#           NODE( 1 ON LEVEL 1 ) = 1
#           NODE( 1 ON LEVEL 2 ) = 2
#           NODE( I ON LEVEL J ) = LEVEL( J ) +
#           ( KEYLOC( J - 1 ) * BASE ** ( I - 2 ) )
#
# FOR EACH LEVEL, A SEARCH IS MADE
# FOR THE LAST KEY IN A GIVEN
# NODE, DETERMINED FROM THE
# PREVIOUS LEVEL, WHICH IS EARLIER IN
# ALPHABETICAL ORDER OR EQUAL TO
# THE SEARCH KEY. THIS YIELDS THE
# KEYLOC FOR THAT LEVEL. FROM THIS
# THE NODE TO SEARCH ON THE NEXT
# LEVEL CAN BE DETERMINED. THE
# LEAF NODES CONTAIN NO KEYS, BUT
# INSTEAD CONTAIN THE TEXT
# ASSOCIATED WITH ITS PARENT KEY.
#
# DATA FORMAT
#
# AN INPUT LINE HAS UP TO EIGHTY
# CHARACTERS. THE FIRST 'LENGTH'
# CHARACTERS CONTAIN THE KEY; THE
# REST IS TEXT. THE OUTPUT LINE
# LENGTH IS ALWAYS EQUAL TO THE
# MAXIMUM LENGTH OF THE TEXT, OR
# ( 80 - 'LENGTH' ). A LOGICAL
# OUTPUT RECORD CONTAINS EITHER KEYS
# OR TEXT, BUT NOT BOTH.
#
# LIMITATIONS
#
# THE MAXIMUM LINE WIDTH IS EIGHTY
# CHARACTERS. THE MAXIMUM NUMBER
# OF KEYS IS FIXED AT 'BASE' ** 'DEPTH',
# WHERE 'BASE' IS COMPUTED
# FROM KEY LENGTH, AND 'DEPTH'
# IS PROVIDED BY THE USER. IF THE
# TREE IS NOT FILLED UP, THE
# REMAINING SPACE IS LEFT EMPTY.
#
# ROUTINES CALLED
#
# PPUSH    PUSH PROGRAM NAME ONTO ERROR STACK
#          (GIPSY ROUTINE)

```



```

#      OSPINF  PUT PARAMETER VALUES INTO FD
#      (SYSTEM ROUTINE)
#      OSOPNS  SEQUENTIAL FILE OPEN  (SYSTEM ROUTINE)
#      OPNDSR  OPEN A TEMP FILE (HISTORY STUFF)
#      (GIPSY ROUTINE)
#      DSCNAM  WRITE DESCRIPTOR RECORD -- NAME RECORD
#      (GIPSY ROUTINE)
#      PDSCPS  PUT DESCRIPTOR RECORD -- PACKED STRING
#      (GIPSY ROUTINE)
#      PDSCI   PUT DESCRIPTOR RECORD -- INT
#      (GIPSY ROUTINE)
#      COPYYS  COPY DESCRIPTOR RECORDS FROM TEMP FILE
#      (GIPSY ROUTINE)
#      OSINFO  GET SYSTEM INFORMATION  (SYSTEM ROUTINE)
#      OSRDS   SEQUENTIAL FILE READ  (SYSTEM ROUTINE)
#      AUPACK  UNPACK A BIT STRING INTO AN ARRAY
#      (SYSTEM ROUTINE)
#      RWRITE  RANDOM FILE WRITE  (GIPSY PRIMITIVE)
#      OSWTR   RANDOM FILE WRITE  (SYSTEM ROUTINE)
#      PPOP    POP PROGRAM NAME FROM ERROR STACK
#      (GIPSY ROUTINE)
#      OSGIEV  GET IEV FROM OS SAVE AREA (SYSTEM ROUTINE)
#
#
#*****
#
#
#      SUBROUTINE MKRNC( FDI, FDO, TREE, WIDTH, DEPTH, TEXT,
#                        MAXLEN, BUFFER, LINE, ENTRY, LEVEL,
#                        LAST, NLEVS, BASE, LENGTH, IEV, * )
#
#
#      IMPLICIT INTEGER ( A - Z )
#
#      INCLUDE MACA1
#
#      CHARACTER FDI( .FDLENGTH ), FDO( .FDLENGTH )
#      CHARACTER LINE( WIDTH ), TEXT( MAXLEN )
#      INTEGER JDENT( .IDLLENGTH )
#      INTEGER TREE( WIDTH, DEPTH ), ENTRY( DEPTH )
#      INTEGER LEVEL( NLEVS ), LAST( NLEVS )
#      PCHARACTER BUFFER( WIDTH )
#
#      PUSH PROGRAM NAME ON TRACE STACK
#
#      CALL PPUSH( "MKRNC" )
#
#      OPEN SEQUENTIAL FILE FOR INPUT
#
#      IF ( .OK == OSPINF( FDI, .MODE, .SYSMODE ) ) GOTO 9800
#      IF ( .OK == OSOPNS( FDI, .INPUT ) ) GOTO 9800

```

```

#
#           CREATE HISTORY FOR OUTPUT SIF
#
CALL OPNDSR( IEV, %9900 )
CALL DSCNAM( 'MKRNC ', IEV, %9900 )
CALL PDSCPS( "MESSAGE FILE FROM SEQUENTIAL LIST FILE",
            IEV, %9900 )
CALL PDSCI( "INPUT LINE WIDTH", WIDTH, IEV, %9900 )
CALL PDSCI( "KEY LENGTH", LENGTH, IEV, %9900 )
CALL PDSCI( "TREE DEPTH", DEPTH, IEV, %9900 )
#
#           FILL IN IDENTIFICATION ARRAY FOR OUTPUT SIF
#
DO I = 1, .IDLLENGTH
    JDENT( I ) = 0
#
JDENT( .IDUSR1 ) = LENGTH
JDENT( .IDUSR2 ) = DEPTH
JDENT( .IDNBITS ) = 8
JDENT( .IDNPPL ) = MAXLEN
SUM = 1
LEVEL( 1 ) = 1
LAST( 1 ) = 1
DO I = 1, DEPTH
    $(
        LEVEL( I + 1 ) = SUM + 1
        SUM = SUM + BASE**I
        LAST( I + 1 ) = SUM
    $)
JDENT( .IDNLINS ) = SUM
JDENT( .IDNCOLS ) = MAXLEN
JDENT( .IDNROWS ) = 1
JDENT( .IDMAX ) = 255
JDENT( .IDNBDS ) = 1
#
#           CREATE OUTPUT SIF
#
CALL COPYDS( FDO, JDENT, IEV, %9900 )
#
#           INITIALIZE COUNTERS, POINTERS, AND ARRAYS
#
COUNT = 0
DO I = 1, DEPTH
    $(
        DO J = 1, WIDTH
            TREE( J, I ) = 0
        ENTRY( I ) = 0
    $)
IF ( .OK == OSINFO( .SYSMODE, NBITS ) ) GOTO 9800
#
#           CYCLE THROUGH INPUT LINES

```

```

#
REPEAT
  $(
    IEV = OSRDS( FDI, BUFFER, SIZE, WIDTH )
    IF ( IEV == .EOFIEV ) BREAK
    IF ( IEV /= .OK ) GOTO 9900
    IF ( .OK /= AUPACK( LINE, BUFFER, SIZE,
                      NBITS, .UNSIGNED ) ) GOTO 9800
#
#       IGNORE LINE BEGINNING WITH '#'
#
IF ( LINE( 1 ) == ICHAR( '#' ) ) NEXT
#
#       CHECK IF KEY IS TO BE INSERTED INTO NODE(S)
#
DO I = DEPTH - 1, 0, -1
  IF ( MOD( COUNT, BASE**I ) == 0 )
    $(
      J = DEPTH - I
      IF ( LEVEL( J ) > LAST( J ) ) GOTO 9000
      DO K = 1, LENGTH
        TREE( ENTRY( J ) * LENGTH + K, J ) =
          LINE( K )
      ENTRY( J ) = ENTRY( J ) + 1
#
#       WRITE OUT FULL NODE AND ADVANCE TO NEXT NODE
#
IF ( ENTRY( J ) >= BASE )
  $(
    ENTRY( J ) = 0
    CALL RWRITE( FDO, TREE( 1, J ), 1,
                LEVEL( J ), JDENT, .WAIT,
                IEV, %9900 )
    DO K = 1, WIDTH
      TREE( K, J ) = 0
      LEVEL( J ) = LEVEL( J ) + 1
    $)
  $)
#
#       STORE USEABLE TEXT IN LEAF NODE
#
DO I = LENGTH + 1, WIDTH
  IF ( I > SIZE )
    TEXT( I - LENGTH ) = 0
  ELSE
    TEXT( I - LENGTH ) = LINE( I )
CALL RWRITE( FDO, TEXT( 1 ), 1,
            LEVEL( NLEVS ), JDENT, .WAIT,
            IEV, %9900 )
COUNT = COUNT + 1
LEVEL( NLEVS ) = LEVEL( NLEVS ) + 1

```

```

    $)
#
#           WRITE OUT PARTIALLY FILLED INTERNAL NODES
#
DO I = 1, DEPTH
  IF ( ENTRY( I ) > 0 )
    $(
      CALL RWRITE( FDO, TREE( 1, I ), 1, LEVEL( I ),
                  JDENT, .WAIT, IEV, %9900 )
      DO J = 1, WIDTH
        TREE( J, I ) = 0
      LEVEL( I ) = LEVEL( I ) + 1
    $)
#
#           BLANK OUT UNUSED NODES
#
DO I = 2, NLEVS
  IF ( LEVEL( I ) <= LAST( I ) )
    DO J = LEVEL( I ), LAST( I )
      CALL RWRITE( FDO, TREE( 1, 1 ), 1, J,
                  JDENT, .WAIT, IEV, %9900 )
#
#           FINISHED - POP PROGRAM NAME OFF TRACE STACK
#
CALL CLOSE( FDI )
CALL CLOSE( FDO )
CALL PPOP
RETURN
#
#           TABLE OVERFLOW IN B-TREE
#
9000 CONTINUE
IEV = -5054
GOTO 9900
#
#           RETRIEVE IEV FROM BELOW
#
9800 CONTINUE
IEV = OSGIEV( IEV )
#
#           TAKE ALTERNATE ERROR RETURN
#
9900 CONTINUE
CALL CLOSE( FDI )
CALL CLOSE( FDO )
RETURN 1
#
#
END

```

I APPENDIX I: ABSIMG COMMAND

What follows are the files associated with the GIPSY command, ABSIMG. This includes the following files:

1. GIPSY_DOC:ABSIMG.DOC - on-line documentation,
2. GIPSY_RUN:ABSIMG.RUN - a run-file to test the command,
3. GIPSY_LNK:ABSIMG.LNK - a link file which instructs the system on how to build the command,
4. GIPSY_SRC:DABSIM.RAT - the command driver,
5. GIPSY_SRC:ABSIMG.RAT - work routine for the host,
6. GIPSY_SRC:ABSIMG.MLK - link file for the mini-MAP module,
7. GIPSY_SRC:ABSIMG.MMR - MAP-driver for the ABSIMG command,

8. GIPSY_SRC:ABSWRK.MMR - work routine for the mini-MAP,

9. GIPSY_SRC:XABSIMG. - include file for the MAP-driver,

I.1 ABSIMG Documentation

*ABSIMG Absolute value of an image

VERSION: A.01 DATE: 01-27-80 AUTHOR: OSCAR ZUNIGA

ACTION: This command operates on a standard image file with at least one numeric band and creates another standard image file in which every pixel is the absolute value of the corresponding pixel in the input image.

SOURCE: Disk, input file name.

DESTINATION: Disk, output file name.

FLAGS: None.

QUESTIONS: None.

COMMAND STRING EXAMPLE:

ABSIMG INPUT.NUM > OUTPUT.NUM

The output image OUTPUT.NUM will be the absolute value of the input image INPUT.NUM, pixel by pixel.

I.2 ABSIMG Run-file

```

$ ! TESTING THE COMMAND ABSIMG
$ ! ABSIMG: TAKES THE ABSOLUTE VALUE OF AN IMAGE
$ !
$ ! MAKE A RAMP
$ !
$ SURFAC ABI1.SIF (S)
    100 100 50 50 1 255 1 1 1 1 1
$ !
$ EXSIF ABI1.SIF
    BLK 97 100
    DONE
$ !
$ ! MULTIPLY IMAGE BY -1
$ !
$ ARITHM ABI1.SIF > ABI2.SIF (MB)
    -1 N
$ !
$ EXSIF ABI2.SIF
    BLK 97 100
    DONE
$ !
$ ! TAKE THE ABSOLUTE VALUE OF THE IMAGE USING ABSIMG
$ !
$ ABSIMG ABI2.SIF > ABI3.SIF
$ !
$ EXSIF ABI3.SIF
    BLK 97 100
    DONE
$ !
$ ! COMPARE THE RAMP IMAGE TO THE ABSIMG IMAGE
$ ! THEY SHOULD BE IDENTICAL
$ !
$ RMS ABI1.SIF,ABI3.SIF > TT
$ !
$ ! DELETE THE TEMPORARY IMAGES
$ !
$ DELETE
    ABI1.SIF
    ABI2.SIF
    ABI3.SIF
$ !
$ !
$ EXIT
```


I.3 ABSIMG Host Link File

⌘ GENCMD ABSIMG DABSIM,ABSIMG,ABSLIN

I.4 DABSIM Routine

```
#--DABSIM          ABSOLUTE VALUE OF AN IMAGE DRIVER
#
# IDENTIFICATION
#
#     TITLE          DABSIM
#     AUTHOR         OSCAR A. ZUNIGA
#     VERSION        A.01
#     DATE           JAN 16, 1980
#     LANGUAGE       RATFOR
#     SYSTEM         IBM-370
#
# UPDATE
#
#     UPDATE         # 1
#     AUTHOR         GREGG R FABREGAS
#     DATE           02-SEP-1983
#     VERSION        A.01
#     PURPOSE        TO CHECK FOR AND REMOVE
#                   CTRLT AND IGNORT CALL
#
#     UPDATE         # 2
#     AUTHOR         GREGG R FABREGAS
#     DATE           02-FEB-1984
#     VERSION        A.01
#     PURPOSE        (1) USE NXT/GETWP MECHANISM TO
#                   COMPUTE SPACE,
#                   (2) CHANGE ROUTINE NAME FROM
#                   "ABSIDV" TO "DABSIM",
#                   (3) CHANGE ERROR CODE -2012 TO
#                   -5051, AND
#                   (4) REMOVE EXTRANEIOUS CALLS TO
#                   CLOSE.
#
#     UPDATE         # 3
#     AUTHOR         GREGG R FABREGAS
#     DATE           22-JUN-1985
#     VERSION        A.01
#     PURPOSE        ADD "IOLINE" ARGUMENT TO CALL TO
#                   GETWPM SO ARRAY NAME CAN BE USED
#                   INSTEAD OF INDEX.
#
# PURPOSE
#
#     THIS IS THE DRIVER FOR THE COMMAND THAT COMPUTES THE
#     ABSOLUTE VALUE OF AN IMAGE
#
# ENTRY POINT
#
```

```

#       DABSIM ( WORK, ALTRET )
#
# ARGUMENT LISTING
#
#       WORK      INT           WORK ARRAY TO HOLD INPUT AND
#                               OUTPUT BUFFERS
#       ALTRET    INT           ALTERNATE ERROR RETURN
#
# INCLUDE FILES/COMMONS
#
#       MACA1     INCLUDE       MACRO FILE FOR TOKEN DEFINITION
#       GIPCOM    INCLUDE       INCLUDE FILE FOR GIPSY
#       ERROR     INCLUDE       INCLUDE FILE FOR COMMON ERROR
#
# ROUTINES CALLED
#
#       PPUSH     PUSHES PROGRAM NAME INTO ERROR STACK
#                (PRIMITIVE)
#       PPOP      POPS PROGRAM NAME FROM ERROR STACK
#                (PRIMITIVE)
#       RDKINL    INITIALIZES AND ACCESSES AN SIF FILE
#                (PRIMITIVE)
#       CLOSE     CLOSSES AN SIF FILE   (PRIMITIVE)
#       OSALOC    ALLOCATE DIM WORDS FOR DYNAMIC ARRAY
#                (PRIMITIVE)
#       ABSIMG    ABSOLUTE VALUE OF AN IMAGE   (GIPSY)
#
# *****
#
#       SUBROUTINE DABSIM ( WORK, *)
#
#       INCLUDE MACA1
#
#       IMPLICIT INTEGER (A-Z)
#
#       INCLUDE GIPCOM
#       INCLUDE ERROR
#
#       INTEGER WORK ( .ARB )
#       INTEGER IDENT ( .IDLENGTH )
#
#       EQUIVALENCE (NBITS, IDENT(.IDNBITS))
#       EQUIVALENCE (NPPL, IDENT(.IDNPPL))
#       EQUIVALENCE (NLINS, IDENT(.IDNLINS))
#       EQUIVALENCE (NCOLS, IDENT(.IDNCOLS))
#       EQUIVALENCE (NROWS, IDENT(.IDNROWS))
#       EQUIVALENCE (NTBND, IDENT(.IDNBND))
#       EQUIVALENCE (NSBND, IDENT(.IDNSBND))
#       EQUIVALENCE (MODE, IDENT(.IDMODE))

```

```

#
#
CALL PPUSH ("DABSIM")
CALL MMINIT( 0 )
#
#
#           SET UP INPUT FILES
#
CALL RDKINL ( FDI1, IDENT, .READONLY, IEV, %9999 )
CALL CLOSE ( FDI1 )
#
#
#           CHECK THE INPUT FILE
#
IF ( MODE /= .INTMODE & MODE /= 0 ) GO TO 9000
IF ( NPPL .NE. NCOLS .OR. NROWS .NE. 1 ) GO TO 9010
IF ( NTBND - NSBND .LE. 0 ) GO TO 9020
#
#
#           CHECK FOR AVAILABLE SPACE
#
NXT = 1
IOLINE = GETWPM( NXT, .INTMODE, NPPL, "IOLINE" )
IF ( .OK /= OSALOC( NXT ) ) GO TO 9030
CALL MMALOC( NXT )
#
#
#           GET THE BAND TO USE
#
CALL WHBAND ( FDI1, IDENT, -1, BND, IEV, %9999 )
BNDO = 1
#
#
#           CALL THE NUMBER CRUNCHER
#
CALL ABSIMG ( FDI1, FDOI, BND, BNDO, WORK( IOLINE ),
             NPPL, IEV, %9998 )
#
CALL MMCLOS( 0 )
CALL PPOP
RETURN
#
#
#           ABNORMAL CONDITIONS
#
9000  CONTINUE
#
#           NOT AN INTEGER FILE
#
IEV = -5051
GO TO 9998
#
9010  CONTINUE
#
#           NOT IN LINE FORMAT
#

```

```
        IEV = -5001
        GO TO 9998
#
# 9020  CONTINUE
#
#           NO NUMERIC BANDS
#
#           IEV = -5018
#           GO TO 9998
#
# 9030  CONTINUE
#
#           NOT ENOUGH WORK SPACE
#
#           IEV = -5010
#
# 9998  CONTINUE
#
#           ERROR IN SUBPROGRAM
#
#           ABNORMAL RETURN
#
# 9999  CALL MMCLOS( 0 )
#       RETURN 1
#       END
```

I.5 ABSIMG Routine

```
#--ABSIMG          ABSOLUTE VALUE OF AN IMAGE
#
# IDENTIFICATION
#
#     TITLE          ABSIMG
#     AUTHOR         OSCAR A. ZUNIGA
#     VERSION        A.01
#     DATE           JAN 10, 1980
#     LANGUAGE       RATFOR
#     SYSTEM         IBM-370
#
# UPDATE
#
#     UPDATE         # 1
#     AUTHOR         GREGG R FABREGAS
#     DATE           21-AUG-1984
#     VERSION        A.01
#     PURPOSE        (1) CHANGE WORK ARRAY TO IOLINE,
#                   (2) USE .READONLY ACCESS ON INPUT
#                   FILE,
#                   (3) REMOVE UNUSED EQUIVALENCES,
#                   (4) REMOVE ASSIGNMENT TO
#                   JDENT( .IDLENGTH ),
#                   (5) REPLACE ' WITH " WHERE
#                   APPLICABLE,
#                   (6) REMOVE UNUSED ERROR CONDITION
#                   PROCESSING AT LABEL 9010, AND
#                   (7) REMOVE LABEL 9999.
#
#     UPDATE         # 2
#     AUTHOR         GREGG R FABREGAS
#     DATE           22-JUN-1985
#     VERSION        A.01
#     PURPOSE        CHANGE INDEX OF 1 IN MMSNDA AND
#                   MMRCVA CALLS TO "IOLINE" SO ARRAY
#                   IS NAMED, NOT NUMBERED.
#
# PURPOSE
#
#     THIS ROUTINE PERFORMS A ABSOLUTE VALUE OPERATION ON
#     AN SIF IMAGE
#
# ENTRY POINT
#
#     ABSIMG ( FDI, FDO, BND, BNDO, IOLINE, NPPL,
#             IEV, ERRET )
#
# ARGUMENT LISTING
```

```

#
#       FDI       INT           INPUT FILE DESCRIPTOR
#       FDO       INT           OUTPUT FILE DESCRIPTOR
#       BND       INT           IMAGE BAND TO PROCESS
#       BNDO      INT           OUTPUT BAND
#       IOLINE    INT ARRAY     WORK ARRAY TO HOLD INPUT
#                               AND OUTPUT LINE.
#       NPPL      INT           SIZE OF WORK ARRAY
#       IEV       INT           INTEGER EVENT VARIABLE
#       ERRET     INT           ALTERNATE ERROR RETURN
#
# INCLUDE FILES/COMMONS
#
#       MACA1     INCLUDE       MACRO FILE FOR TOKEN DEFINITION
#
# ROUTINES CALLED
#
#       PPUSH     PUSHES PROGRAM NAME INTO ERROR STACK
#                (PRIMITIVE)
#       PPOP      POPS PROGRAM NAME FROM ERROR STACK
#                (PRIMITIVE)
#       RDKINL    INITIALIZES AND ACCESSES AN SIF FILE
#                (PRIMITIVE)
#       CLOSE     CLOSSES AN SIF FILE (PRIMITIVE)
#       CPYIDR    GET DESCRIPTOR RECORDS FROM INPUT FILE
#                (PRIMITIVE)
#       DSCNAM    WRITE THE NAME DESCRIPTOR RECORD
#                (PRIMITIVE)
#       PDSCI     WRITE INTEGER DESCRIPTOR RECORD
#                (PRIMITIVE)
#       COPYDS    COPY DESCRIPTOR RECORDS TO OUTPUT FILE
#                (PRIMITIVE)
#       RREAD     READS FROM AN SIF FILE (PRIMITIVE)
#       RWRITE    WRITES TO AN SIF FILE (PRIMITIVE)
#       ABSLIN    ABSOLUTE VALUE OF A LINE (GIPSY)
#
#
#*****
#
#       SUBROUTINE ABSIMG ( FDI, FDO, BND, BNDO, IOLINE,
#                           NPPL, IEV, * )
#
#       INCLUDE MACA1
#       IMPLICIT INTEGER ( A - Z )
#       CHARACTER FDI ( .FDLENGTH )
#       CHARACTER FDO ( .FDLENGTH )
#
#       INTEGER IDENT ( .IDLLENGTH ), JDENT ( .IDLLENGTH )
#       INTEGER IOLINE( NPPL )
#

```

```

EQUIVALENCE ( NBITS, IDENT(.IDNBITS) )
EQUIVALENCE ( NLIN, IDENT(.IDNLINS) )
#
CALL PPUSH ( "ABSIMG" )
#
#           OPEN INPUT FILE
#
CALL RDKINL ( FDI, IDENT, .READONLY, IEV, %9998 )
CALL CLOSE ( FDI )
#
DO I = 1, .IDLENGTH
    JDENT ( I ) = 0
#
JDENT ( .IDNBITS ) = NBITS
JDENT ( .IDNPPL ) = NPPL
JDENT ( .IDNLINS ) = NLIN
JDENT ( .IDNCOLS ) = NPPL
JDENT ( .IDNROWS ) = 1
JDENT ( .IDNBND ) = 1
#
#           COPY DESCRIPTOR RECORDS FROM INPUT IMAGES
#           ROUTINE NAME AND PARAMETERS TO TEMPORARY
#           SEQUENTIAL
#
CALL CPYIDR ( FDI, IDENT, .OPNTMP, IEV, %9998 )
#
CALL DSCNAM ( "ABSIMG", IEV, %9998 )
#
CALL PDSCI ( "INPUT IMAGE BAND NUMBER",
            BND, IEV, %9998 )
#
#           OPEN OUPUT IMAGE AND COPY TO IT THE
#           DESCRIPTOR RECORDS FROM TEMPORARY FILE
#
CALL COPYDS ( FDO, JDENT, IEV, %9998 )
#
#           ACTUAL NUMBER CRUNCHING
#
DO LN = 1, NLIN
    $(
    CALL RREAD ( FDI, IOLINE, BND, LN,
                IDENT, .WAIT, IEV, %9998 )
    IF ( MMSKIP( .TRUE. ) )
        CALL ABSLIN ( IOLINE, NPPL )
    ELSE
        $(
        CALL MMSNDA( "IOLINE", IEV, %9998 )
        CALL MMEXEC( .WAIT, IEV, %9998 )
        CALL MMRCVA( "IOLINE", IEV, %9998 )
        $)
    )

```



```

          CALL RWRITE ( FDO, IOLINE, BNDO, LN,
                        JDENT, .WAIT, IEV, %9998 )
          $)
#
#
          CALL CLOSE ( FDI )
          CALL CLOSE ( FDO )
#
          CALL PPOP
          RETURN
#
9998      CONTINUE
#
#
#
#
#
#
          CALL CLOSE ( FDI )
          CALL CLOSE ( FDO )
#
          RETURN 1
#
          END

```

I.6 ABSIMG Mini-MAP Link File

§ GENMAP ABSIMG ABSIMG,ABSWRK

I.7 ABSIMG MAP-driver

```
INCLUDE MAPDRV
INCLUDE XABSIMG
REPEAT
  $(
    TRACE( 1 )
    I1 = INT( ARRAYLEN( IOLINE ) )
    TRACE( 2 )
    CALL ABSWRK( IOLINE, I1, I2, I3, I4, I5, DEBUG )
    PAUSE
  $)
END
```

I.8 ABSWRK MAP-subroutine

```
INCLUDE MAPSUB
SUBROUTINE ABSWRK( IOLINE, LEN, INDEX, PTLOW,
                  PTHIGH, IDXH, DEBUG )
INTEGER LEN, IOLINE( LEN ), INDEX
INTEGER PTLOW, PTHIGH, IDXH, DEBUG
TRACE( 3 )
DO INDEX = 1, LEN, 2
  $(
    TRACE( 4 )
    IDXH = INDEX + 1
    TRACE( 5 )
    PTLOW = IOLINE( INDEX )
    TRACE( 6 )
    PTHIGH = IOLINE( IDXH )
    TRACE( 7 )
    IF ( PTHIGH < 0 )
      $(
        TRACE( 8 )
        IF ( PTLOW == 0 )
          $(
            TRACE( 9 )
            IOLINE( IDXH ) = -PTHIGH
            $)
          ELSE
            $(
              TRACE( 10 )
              IOLINE( INDEX ) = -PTLOW
              TRACE( 11 )
              IOLINE( IDXH ) = -PTHIGH - 1
              $)
            $)
      $)
  $)
RETURN
END
```

I.9 XABSIMG Include File

```
NAME6(.A,.B,.S,.I,.M,.G)
GETARRAYS
IDESC1( 1 )
DEFINE(IOLINE,IARI)
```

J APPENDIX J: EDGE COMMAND

What follows are the files associated with the GIPSY command, EDGE. This includes the following files:

1. GIPSY_DOC:EDGE.DOC - on-line documentation,
2. GIPSY_RUN:EDGE.RUN - a run-file to test the command,
3. GIPSY_LNK:EDGE.LNK - a link file which instructs the system on how to build the command,
4. GIPSY_SRC:DEGE.RAT - the command driver,
5. GIPSY_SRC:EDGENC.RAT - work routine for the host,
6. GIPSY_SRC:EDGE.MLK - link file for the mini-MAP module,
7. GIPSY_SRC:EDGE.MMR - MAP-driver for the EDGE command,

8. GIPSY_SRC:EDGENC.MMR - work routine for the mini-MAP,

9. GIPSY_SRC:XEDGE. - include file for the MAP-driver,

J.1 EDGE Documentation

*EDGE Generate an edge from a cubic facet image

VERSION: A.01 DATE: 03-15-81

AUTHOR: R.M.Haralick,T.Laffey

ACTION: This command takes the 11 bands generated from the command CNCFCT or FACET2 and processes bands 2-11 generating a binary edge image. Edges are placed at all pixels having a sufficiently close zero crossing of the second directional derivative taken in the line (L-flag) or sector direction of the gradient. An edge pixel has a value 0 and a nonedge pixel has value 1.

If the "A" flag is set but not "0" flag,, edge strength, contrast, direction, confidence interval, and curvature of contour will be calculated and output as follows:

BAND 1: Edge strength
BAND 2: Edge contrast
BAND 3: Edge direction (-180 to 180 degrees)
BAND 4: Confidence interval of direction
 (0 - 90 degrees)
BAND 5: Curvature of contour
 (1000*LOG(1+ABS(curvature)))
BAND 6: Binary edge image (0,1)

If the "A" flag is set with the "0" flag, only band 1,3,4 and 6 are output.

Edge direction is defined as the clockwise direction from the column axis in which as you walk along the edge, the higher valued area is to the right.

If the "A" flag is not set, only the the binary edge image will be output.

SOURCE: Disk, input file (output from 'CNCFCT or FACET2')

DESTINATION: Disk, symbolic image

(SIF image if "A" flag set)

FLAGS: (A) If this flag is set the user will be prompted for the confidence limit he desires on the edge direction. The zero crossing threshold is based on contrast and scale of the cubic. Edge strength, edge contrast,direction, confidence

interval on the edge direction and curvature of contour will be calculated and output as explained above.

- (B) If this flag is set, the least square estimates of the best fitting polynomial are modified to be a Bayesian estimate.
- (F) If this flag is set an F-test after the gradient magnitude being non zero will be performed. The user will be asked for the significance probability of the F-test.
- (L) If this flag is set, the zero crossing is looked for only on the gradient direction line. If this flag is not set, then the zero crossing is looked for in a sector of about ± 15 degrees around the gradient direction line.
- (O) If this flag is set, the zero crossing is computed exactly is given in the 1984 PAMI article by R.M.Haralick.
- (Q) If this flag is set, the edge must only have high enough gradient magnitude and low enough isotropic second derivative.

NOTE: If the user specifies the "A" or "F" flag, the "E" flag must be specified in the command 'CNCFACT or FACET2', in order to make the residual error band be calculated on the sum of squared error rather than the log of the sum of squared error.

- QUESTIONS:
- (1) Enter radius within which second directional derivative must take a zero in the direction of the gradient.
 - (2) Enter edge contrast value
(If "F" flag not specified)
 - (3) Enter probability for F-Test
(If "F" flag specified).
 - (4) Enter probability for confidence limit on the edge direction (If "A" flag specified).

COMMAND STRING EXAMPLE:

Example 1

EDGE IN.NUM > OUT.SYM

Enter radius for 2nd derivative

(0.0 to 2.0, D = .85) --- 1.0

Enter threshold for edge contrast
(0.0 to 1000.0, D = 10.0) ---

In this example, the input image IN.NUM will be processed and its edges will be output to OUT.SYM. The output file will have one band with edges marked by 1 and others marked by a 0. The edge pixels will be those with a zero crossing of the second directional derivative within a radius of 1 and a edge contrast above 10.

Example 2

EDGE IN.NUM > OUT.SYM (F)
Enter radius for 2nd derivative
(0.0 to 2.0, D = .85) --- 1.0
Enter significance probability for F-Test
(0.0 - 1.0, D = 0.05) --- .05

In this example, the input image IN.NUM will be processed and its edges will be output to OUT.SYM. The output file will have one band with edges marked by 1 and others marked by a 0. The edge pixels will be those with a zero crossing of the second directional derivative within a radius of 1 and also they must have passed the F-Test.

Example 3

EDGE IN.NUM > OUT.SYM (A)
Enter radius for 2nd derivative
(0.0 to 2.0, D = .85) ---
Enter threshold for edge contrast
(0.0 to 1000.0, D = 10.0) ---
Enter confidence limit for edge direction
(0.0 to 1.0, D = 0.95) ---

In this example the input file IN.NUM will be processed and its edge strengths, contrast, directions, confidence intervals, curvature of contour and binary edge pixels will be output to OUT.SYM. The output file will contain 6 bands with the last band being a symbolic band. Furthermore, one can assume with a 95% certainty that the direction of the edge pixels will be within their calculated confidence interval.

COMMENTS: (1) The angle for the edge is encoded as follows:
Angles are measures positive from the column axis clockwise. In pixels which do not have a meaningful angle, the value of 360 is stored. All angles are specified in degrees. The

vector encoded is such that the brighter region is to its right.

- (2) For more details on the calculations done, please see the following paper:

Haralick, R.M., "Digital Step Edges from Zero Crossing of Second Directional Derivative", IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume PAMI-6, No. 1, January 1984 p58-68.

- (3) If the "F" flag is specified the F-Test used goes as follows:

$$\frac{(\text{gradient})^2}{2}$$

$$\frac{(\text{error})^2}{(\text{degrees of freedom})}$$

This value is then compared to the value of the F-Distribution at a user specified probability level.

- (4) a. Raising the edge contrast threshold will reduce the number of edges detected.
b. Raising the probability for the F-Test will reduce the number of edges detected.
c. Raising the confidence limit on the edge direction will widen the confidence interval.
d. The output edge strength will be determined as follows:

$$\begin{aligned} \text{Edge Strength} &= \text{Gradient Magnitude, or} \\ \text{Edge Strength} &= \text{F-Test} \times 100 \\ &(\text{if "F" flag set}) \end{aligned}$$

- (5) The curvature of contour(cc) passing each pixel is defined as follows:

$$cc = \frac{(-f_c f_r) + (f_{rr} f_{rc}) - (f_{rc} f_{cc}) + (f_r)^2}{\text{SQRT}((f_r^2 + f_c^2)^3)}$$

Where f stands for fitting function, f_x stands for derivative of f respective to x and f_{xy} stands for derivative of f respective to x then to y (x(y) is equal to r or c). The sign of cc is determined as follows: the direction of contour coincides with edge direction, cc is positive (negative) if angle increases (decreases) in the clockwise direction. The actual output equals to

$$\text{OUTPUT} = \text{SIGN}(1000 \times \text{ALOG10}(1 + \text{ABS}(CC))), CC$$

(6) Edge contrast is proportional to the height difference between two extrema of the cubic curve which is an intersection of the cubic fitting surface and a vertical plane oriented the gradient direction.

J.2 EDGE Run-file

```
$ ! TESTING THE COMMAND EDGE
$ ! EDGE: GENERATES AN EDGE MAP FROM OUTPUT OF 'CNTFCT'
$ ! CREATE A BAR IMAGE
$ !
$     MKBAR EDGEB17.IMG (A)
      17,17
      17
      V
      L
$ !
$ ! SUBTRACT 8 FROM THIS IMAGE
$ !
$     ARITHM EDGEB17.SUB < EDGEB17.IMG (S)
      8
      N
$ !
$ ! RAISE THIS IMAGE TO A POWER
$ !
$     IMGPCR EDGEB17.PWR < EDGEB17.SUB
      3
      1.
$ !
$     ARITHM EDGEB17M.SUB < EDGEB17.SUB (M)
      7.
      N
$
$ ! SUBTRACT THIS FROM EDGEB17M.SUB
$ !
$     ARITHM EDGEB17.ART < EDGEB17M.SUB,EDGEB17.PWR (S)
      N
$ !
$ ! THE IMAGE HAS THE FORM  $-(C-8)**3+7.*(C-8)$ 
$ ! THE FIRST DERIVATIVE IS  $-3*(C-8)**2 +7.$ 
$ ! THE SECOND DERIVATIVE IS  $-6*(C-8)$ 
$ ! AT C=8 THE SECOND DERIVATIVE HAS A ZERO CROSSING
$ ! AND FIRST DERIVATIVE HAS A MAXIMUM
$ !
$ ! RUN THE FACET COMMAND ON IT
$ !
$     CNCFCT EDGEB17.CUB < EDGEB17.ART
      5,5
      16
$
$ !
$ ! NOW RUN THE EDGE COMMAND ON THIS IMAGE
$ !
$     EDGE EDGEB17.EDG < EDGEB17.CUB
      .85
```

```

5.
$ !
$   EDGE EDGEB17.EDO < EDGEB17.CUB (0)
    .85
    5.
$ !
$ ! COMPARE THE NEW VERSION WITH THE OLD VERSION
$ !
$   RMS TT < EDGEB17.EDG,EDGEB17.EDO
$ !
$ ! COMPARE THE NEW VERSION WITH THE RESULT WITH A FLAG
$ !
$   EDGE EDGEB17.EDA < EDGEB17.CUB (A)
    .85
    5.
    .8
$ !
$   SBIMG EDGEB17.EDS < EDGEB17.EDA
    1
    17
    1
    17
    1
    6
$   EXSIF EDGEB17.EDS
    MID 19 0
    Y
    DONE
$ !
$   RMS TT < EDGEB17.EDG,EDGEB17.EDS
$ !
$ ! COMPARE THE NEW VERSION WITH THE RESULT WITH F FLAG
$ !
$   EDGE EDGEB17.EDF < EDGEB17.CUB (F)
    .85
    5.
    .05
$ !
$   RMS TT < EDGEB17.EDG,EDGEB17.EDF
$ !
$ ! CHOP THE LEFT AND RIGHT BOUNDARIES OF THE IMAGE
$ !
$   SBIMG EDGEB13.EDG < EDGEB17.EDG
    1
    17
    3
    15
$ !
$ ! AND INVERT FOR LATER COMPARISON
$ !
$   INVIMG EDGEB13.INV < EDGEB13.EDG

```

```

$ !
$ ! NOW CREATE A CHECKERBOARD IMAGE
$ !
$      MKCHK EDGEB13.CHK
      17,13
      17,7
      0
      100
      0

$ !
$ ! AND COMPARE TO A CORRECT EDGE
$ !
$      ROVRT EDGEB13.RBT < EDGEB13.CHK
      1
$      TRSLD EDGEB13.TRS < EDGEB13.RBT (I)
      10

$ !
$ ! THERE SHOULD BE A DIFFERENCE OF 0
$ !
$      RMS TT < EDGEB13.TRS,EDGEB13.INV
$ !
$ ! DELETE ALL THESE FILES AND EXIT
$      DELETE
      EDGEB17.IMG
      EDGEB17.SUB
      EDGEB17.PWR
      EDGEB17.CUB
      EDGEB17.EDG
      EDGEB13.EDG
      EDGEB13.INV
      EDGEB13.TRS
      EDGEB13.CHK
      EDGEB17.ART
      EDGEB17M.SUB
      EDGEB13.RBT
      EDGEB17.EDO
      EDGEB17.EDF
      EDGEB17.EDA
      EDGEB17.EDS

$      EXIT

```

J.3 EDGE Host Link File

```
‡ GENCMD EDGE DEDGE,EDGENC,EDGENO,TEDGEN,CLCEXP -  
GIPSY_LIB:STATLIB
```


J.4 DEDGE Routine

```
#--DEEDGE          DRIVER FOR THE GIPSY 'EDGE' COMMAND
#
# IDENTIFICATION
#
# TITLE            DEDGE
# AUTHOR           ROBERT M. HARALICK
# VERSION          B.01
# DATE             APRIL 23, 1984
# LANGUAGE         RATFOR
# SYSTEM          VAX 11/780
#
#
# UPDATE
#
# UPDATE          # 1
# AUTHOR          ROBERT M. HARALICK
# DATE            17-JUN-1984
# VERSION         B.01
# PURPOSE         TO ADD THE L FLAG OPTION WHICH ONLY
#                 LOOKS FOR ZERO CROSSINGS ON THE
#                 GRADIENT DIRECTION LINE.
#
# UPDATE          # 2
# AUTHOR          ROBERT M. HARALICK
# DATE            26-AUG-1984
# VERSION         B.01
# PURPOSE         TO PUT IN THE ORIGINAL PAMI
#                 ALGORITHM.
#
# UPDATE          # 3
# AUTHOR          SUI WEI
# DATE            15-FEB-1985
# VERSION         B.01
# PURPOSE         IF A FLAG, OUTPUTS TWO MORE BANDS:
#                 EDGE CONTRAST AND CURVATURE OF
#                 CONTOUR.
#                 IF F FLAG, USE FPROB AS A
#                 SIGNIFICANCE PROBABILITY
#
# UPDATE          # 4
# AUTHOR          SUI WEI
# DATE            15-MAR-1985
# VERSION         B.01
# PURPOSE         (1) CHANGE RADIUS DEFAULT TO 0.7
#                 AND CONTRAST TO 12
#                 (2) CHANGE TEDGEN ACCORDING TO
#                 EDGENC.
```

```

#      UPDATE          # 5
#      AUTHOR          MICHAEL R. WADE
#      DATE            22-FEB-1986
#      VERSION         B.01
#      PURPOSE         TO FIX ERROR WITH DEFAULT VALUE
#                     FOR THRESHOLD FOR EDGE CONSTRAINT
#                     BEING LARGER THAN MAX
#                     PIXEL VALUE IN IMAGE FILE.
#                     IF MXGREY IS LESS THAN DEFAULT,
#                     12.0, THEN USE MXGREY AS THE
#                     DEFAULT VALUE, AS WELL AS THE
#                     HIGH VALUE.
#
#      UPDATE          # 6
#      AUTHOR          GREGG R FABREGAS
#      DATE            06-MAR-1986
#      VERSION         B.01
#      PURPOSE         (1) ADD CALL TO MMINIT FOR MINIMAP
#                     CHECK,
#                     (2) CHANGE GETWP TO GETWPM FOR
#                     IBUFF AND OBUFF POINTERS, AND
#                     IIN AND IOUT ARRAYS,
#                     (3) ADD CALLS TO MMCLOS AT ROUTINE
#                     EXITS, AND
#                     (4) ADD CALL TO MMALOC AFTER
#                     GETWP(M) CALLS.
#
#      PURPOSE
#
#      THE PURPOSE OF THIS COMMAND
#      IS TO TAKE THE ELEVEN BAND OUTPUT
#      OF THE COMMAND "CNCFACT" OR FACET2
#      AND DETECT THE EDGES ON IT.
#      EDGES ARE DETERMINED BY HAVING
#      A GRADIENT HIGHER THAN A
#      SPECIFIED THRESHOLD AND A
#      NEGATIVELY SLOPED ZERO CROSSING
#      WITH A SPECIED RADIUS OF THE
#      SECOND DIRECTIONAL DERIVATIVE.
#
#      ENTRY POINT
#
#      DEDGE( WORK, ERRET )
#
#      ARGUMENT LISTING
#
#      WORK      INT.ARR      WORKING STORAGE ARRAY
#      ERRET     INT          ALTERNATE ERROR RETURN
#
#      INCLUDE FILES/COMMONS
#

```



```

EQUIVALENCE (NTBND, IDENT(.IDNBND))
EQUIVALENCE (MODE, IDENT(.IDMODE))
EQUIVALENCE (MNGREY, IDENT(.IDMIN))
EQUIVALENCE (MXGREY, IDENT(.IDMAX))
EQUIVALENCE (AFLAG, UFLAG(.A))
EQUIVALENCE (BFLAG, UFLAG(.B))
EQUIVALENCE (FFLAG, UFLAG(.F))
EQUIVALENCE (LFLAG, UFLAG(.L))
EQUIVALENCE (PFLAG, UFLAG(.P))
EQUIVALENCE (QFLAG, UFLAG(.Q))
EQUIVALENCE (OFLAG, UFLAG(.O))

#
#           PUSH PROGRAM NAME ONTO ERROR
#           STACK
#
CALL PPUSH ("DEDGE")

#
#           SET UP INPUT FILE
#
CALL RDKINL( FD11, IDENT, .READONLY, IEV, %9999 )
CALL CLOSE(FD11)

#
#           CHECK THE INPUT FILE
#
IF( MODE == .INTMODE ) GO TO 9000
IF (NPPL == NCOLS ! NROWS == 1) GO TO 9010
IF (NTBND == 11) GO TO 9020

#
#           CHECK FOR MINIMAP AVAILABILITY
#
CALL MMINIT( 0 )

#
#           SET UP BUFFERS
#
NEXTT = 1
IBUFF = GETWPM(NEXTT, .INTMODE, 10*NPPL, "IIN")
OBUFF = GETWPM(NEXTT, .INTMODE, NPPL, "IOUT")
ABUFF = GETWP (NEXTT, .INTMODE, NPPL)
CBUFF = GETWP (NEXTT, .INTMODE, NPPL)
SBUFF = GETWP (NEXTT, .INTMODE, NPPL)
TBUFF = GETWP (NEXTT, .INTMODE, NPPL)
NBUFF = GETWP (NEXTT, .INTMODE, NPPL)

#
#           NOW ALLOCATE THAT SPACE
#
IF ( .OK == OSALOC (NEXTT) )
  GO TO 9030
IF ( -AFLAG & -BFLAG & -FFLAG &
    -LFLAG & -OFLAG & -QFLAG )
  CALL MMALOC( NEXTT )
#

```

```

#                                     GET THE USER PARAMETERS
#
IF(-QFLAG)
    CALL RNGETR ( 'Enter radius for 2nd derivative.',
                 0.0, 2., .7, RADIUS, IEV, %9998 )
#
#                                     IF THE F-FLAG IS NOT SPECIFIED, THE
#                                     EDGE THRESHOLD WILL BE FOR THE EDGE
#                                     CONTRAST, OTHERWISE IT WILL BE FOR THE
#                                     F-TEST
#
# IF MXGREY IS LESS THAN THE DEFAULT VALUE FOR EDGE
# CONTRAST THEN USE MXGREY AS THE DEFAULT VALUE.
#
    IF ( -OFLAG )
    {
        IF ( MXGREY < 12 )
            DEFLT = FLOAT ( MXGREY )
        ELSE
            DEFLT = 12.0
    }

CALL RNGETR ( 'Threshold for edge contrast.',
             0.0, FLOAT( MXGREY ), DEFLT, EDGTHR, IEV, %9998 )
IF ( -FFLAG & OFLAG )
    CALL RNGETR( 'Threshold for gradient.',0.0,
                FLOAT( MXGREY ), 5.0, EDGTHR, IEV, %9998 )
IF( FFLAG )
    CALL RNGETR(
        'Significance probability ' //
        'for gradient F-Test.',
        0.0, 1.0, 0.05, FPROB, IEV, %9999 )
IF( QFLAG )
    CALL RNGETR ( 'Enter 2nd derivative threshold.',
                 0.0, FLOAT(MXGREY), .5, D2TH, IEV, %9998 )
#
#                                     IF A-FLAG IS SET, GET CONFIDENCE
#                                     LIMIT FROM USER
#
IF ( AFLAG )
    CALL RNGETR (
        'Enter confidence limit for edge direction.',
        0.0, .9999, 0.95, PROB, IEV, %9999 )
IF(-PFLAG)
    $(
#
#                                     ALLOW USER COMMENTS
#
    CALL COMTIN( %9998 )
#
#                                     CALL THE NUMBER CRUNCHER

```

```

#
      IF( -OFLAG )
      CALL EDGENC (FDI1, FD01, WORK(IBUFF),
                  WORK(OBUFF), WORK(ABUFF),
                  WORK(CBUFF), WORK(SBUFF),
                  WORK(TBUFF), WORK(NBUFF),NPPL,
                  EDGTHR,D2TH, RADIUS, PROB, FPROB,
                  AFLAG, BFLAG, FFLAG, LFLAG,QFLAG,
                  IEV, %9998)
      IF( OFLAG)
      CALL EDGENO (FDI1, FD01, WORK(IBUFF),
                  WORK(OBUFF), WORK(ABUFF),
                  WORK(CBUFF), WORK(SBUFF),
                  NPPL, EDGTHR,D2TH, RADIUS,
                  PROB,FPROB, AFLAG,BFLAG,
                  FFLAG, LFLAG,QFLAG, IEV, %9998)
      $)
ELSE
  $ (
4000  CONTINUE
      CALL RNGETI("Enter row",0,NLINS,0,ROW,IEV,%9000)
      IF(ROW == 0) GO TO 5000
      CALL RNGETI("Enter column",1,NPPL,
                  NPPL/2,COL,IEV,%9000)
      CALL TEDGEN(FDI1,ROW,COL,WORK(IBUFF),WORK(OBUFF),
                  WORK(ABUFF),WORK(CBUFF),
                  WORK(SBUFF),WORK(TBUFF),
                  WORK(NBUFF),NPPL,EDGTHR,D2TH,RADIUS,PROB,FPROB,
                  AFLAG,BFLAG, FFLAG, LFLAG,QFLAG,IEV,%9998)
      GO TO 4000
      $)
5000  CONTINUE
#
#
#           NORMAL RETURN
#
      CALL MMCLOS( 0 )
      CALL PPOP
      RETURN
#
#
#           --- ABNORMAL CONDITIONS ---
#
9000  CONTINUE
#
#
#           NOT AN INTEGER FILE
#
      IEV = -2012
      GO TO 9998
#
9010  CONTINUE
#

```

```

#
#
#           IEV = -5001
#           GO TO 9998
#
# 9020      CONTINUE
#
#
#           NO NUMERIC BANDS
#
#           IEV = -5018
#           GO TO 9998
#
# 9030      CONTINUE
#
#
#           NOT ENOUGH WORK SPACE
#
#           IEV = -5010
#           GO TO 9999
#
# 9998      CONTINUE
#
#
#           ERROR IN SUBPROGRAM
#
#           CALL CLOSE ( FDI1 )
#           CALL CLOSE ( FD01 )
#
#
#           ABNORMAL RETURN
#
# 9999      CONTINUE
#           CALL MMCLOS( 0 )
#           RETURN 1
#           END

```

J.5 EDGENC Routine

```
#--EDGENC          NUMBER CRUNCHER FOR THE EDGE COMMAND
#
# IDENTIFICATION
#
#     TITLE          EDGENC
#     AUTHOR         ROBERT M. HARALICK
#     VERSION        C.01
#     DATE           FEBURARY 8, 1985
#     LANGUAGE       RATFOR
#     SYSTEM         VAX 11-750
#
#
#
# UPDATE
#
#     UPDATE         # 1
#     AUTHOR         SUI WEI
#     DATE           15-MAR-1985
#     VERSION        C.01
#     PURPOSE        (1)CHANGE THE RANGE OF SCALE
#                   (XLOW,XHIGH)
#                   (2)SOME TRIVIAL MISTAKES.
#
#     UPDATE         # 2
#     AUTHOR         GREGG R FABREGAS
#     DATE           07-MAR-1986
#     VERSION        C.01
#     PURPOSE        ADD MINIMAP SUPPORT.
#
# PURPOSE
#
#     THIS ROUTINE IS THE NUMBER CRUNCHER FOR THE GIPSY
#     'EDGE'
#     COMMAND. IT ATTEMPTS TO DETECT EDGE PIXEL BY THOSE
#     HAVING A SUFFICIENT CONTRAST THRESHOLD AND A ZERO
#     CROSSING OF THE SECOND DERIVATIVE
#     WITHIN A CERTAIN RADIUS.
#     IF THE "A" FLAG IS SPECIFIED,
#     EDGE DIRECTIONS AND CONFIDENCE
#     INTERVALS ARE CALCULATED. ALSO
#     EDGE STRENGTH, CONTRAST AND
#     CURVATURE OF CONTOUR ARE CALCULATED.
#
# ENTRY POINT
#
#     EDGENC ( FDI, FDO, IIN, IOUT, ABUFF,
#             CBUFF, SBUFF, TBUFF, OBUFF,
#             NPPL, EDGTHR, D2TH, RADIUS,
```



```

#           PROB, FPROB,AFLAG, BFLAG,
#           FFLAG, LFLAG, QFLAG, IEV, ERRET )
#
# ARGUMENT LISTING
#
#           FDI      CHR-ARR      INPUT FILE DESCRIPTOR
#           FDO      CHR-ARR      OUTPUT FILE DESCRIPTOR
#           IIN      INT-ARR      INPUT BUFFER ARRAY
#           IOUT     INT-ARR      OUTPUT BUFFER ARRAY
#           ABUFF    INT-ARR      OUTPUT BUFFER
#           CBUFF    INT-ARR      OUTPUT BUFFER
#           SBUFF    INT-ARR      OUTPUT BUFFER
#           TBUFF    INT-ARR      OUTPUT BUFFER
#           OBUFF    INT-ARR      OUTPUT BUFFER
#           NPPL     INT          NPPL OF INPUT FILE
#           EDGTHR   REAL         GRADIENT THRESHOLD
#                               IF F FLAG IS SPECIFIED,
#                               EDGTHR IS SIGNIFICANCE
#                               TEST PROBABILITY
#           D2TH     REAL         SECOND DERIVATIVE THRESHOLD
#           RADIUS   REAL         RADIUS WITHIN WHICH 2ND
#                               DERIVATIVE MUST BE 0.
#           PROB     REAL         PROBABILITY OF EDGE
#                               DIRECTION LYING WITHIN
#                               ITS CONFIDENCE INTERVAL
#           FPROB    REAL         SIGNIFICANCE PROBABILITY
#                               FOR GRADIENT F TEST
#           AFLAG    LOGICAL      INDICATES THAT EDGE
#                               DIRECTION AND CONFIDENCE
#                               INTERVALS ARE TO BE CALCULATED.
#           BFLAG    LOGICAL      INDICATES THAT A BAYESIAN
#                               ESTIMATE IS COMPUTED FOR
#                               COEFFICIENTS RATHER THAN THE
#                               USUAL ONES
#           FFLAG    LOGICAL      F-TEST WILL BE USED RATHER THAN
#                               GRADIENT MAGNITUDE
#           LFLAG    LOGICAL      ZERO-CROSSING ONLY LOOKED AT
#                               ON THE LINE IN THE GRADIENT
#                               DIRECTION AND NOT IN A SECTOR
#                               BASED ON THAT LINE
#           QFLAG    LOGICAL      INDICATES THAT A QUICK EDGE
#                               TEST IS TO BE DONE
#           IEV      INT          INTEGER EVENT VARIABLE
#           ERRET    INT          ALTERNATE ERROR RETURN
#
# INCLUDE FILES/COMMONS
#
#           MACA1     INCLUDE      GIPSY TOKEN DEFINITIONS INCLUDE
#                               FILE FOR A1 CHARACTERS.
#
# ROUTINES CALLED

```

```

#
#   PPUSH   PLACES THE PROGRAM NAME ONTO THE ERROR
#           STACK.   (GIPSY ROUTINE)
#   CPYIDR  ROUTINE TO OPEN THE INPUT FILE AND
#           COPY THE DESCRIPTOR RECORDS.
#           (GIPSY ROUTINE)
#   DSCNAM  WRITES SUBROUTINE NAME INTO THE
#           DESCRIPTOR RECORDS.   (GIPSY ROUTINE)
#   PDSCR   WRITES A REAL NUMBER TO THE
#           DESCRIPTOR RECORDS.   (GIPSY ROUTINE)
#   COPYDS  COPY THE DESCRIPTOR RECORDS FROM TEMP
#           FILE TO SIF & OPEN OUTPUT FILE.
#           (GIPSY ROUTINE)
#   GDSCAX  GET VALUES FROM THE DESCRIPTOR RECORD
#           (GIPSY ROUTINE)
#   RREAD   READ A RECORD FROM A SIF.   (GIPSY ROUTINE)
#   RWRITE  WRITE A RECORD TO A SIF.   (GIPSY ROUTINE)
#   CLCEXP  FUNCTION TO CALCULATE EXPONENT WHICH
#           MINIMIZES CONFIDENCE LIMIT (GIPSY ROUTINE)
#   FISINV  INVERSE OF THE F-DISTRIBUTION
#           (GIPSY ROUTINE)
#   CHIINV  INVERSE OF THE CHI-SQUARE DISTRIBUTION
#           (GIPSY ROUTINE)
#   ATAN2   ARC TANGENT   (SYSTEM ROUTINE)
#   SQRT    SQUARE ROOT   (SYSTEM ROUTINE)
#   ASIN    ARC SIN       (SYSTEM ROUTINE)
#   ABS     ABSOLUTE VALUE (SYSTEM ROUTINE)
#   CLOSE   ROUTINE TO CLOSE A FILE.   (GIPSY ROUTINE)
#   PPOP    REMOVES THE PROGRAM NAME FROM THE
#           ERROR STACK.   (GIPSY ROUTINE)

```

```

#
#   COMMENTS ON ALGORITHM
#

```

```

#           TRANSLATING THE POLYNOMIAL
#            $Y=C0+C1*X+C2*X**2+C3*X**3$ 
#           SO THAT THE INFLECTION POINT
#           IS AT THE ORIGIN PRODUCES
#           THE POLYNOMIAL

```

$$Y=-(D/3.*C3)*X+C3*X**3$$

```

#           WHERE  $D=-3*C1*C3-C2*C2$ 
#

```

```

#           IN ORDER FOR RELATIVE EXTREMA TO EXIST, C3 MUST BE OF
#           DIFFERENT SIGN FROM  $-(D/3.*C3)$ . IN THIS CASE, THERE
#           IS AN EDGE AT THE INFLECTION POINT. IF THEY ARE OF
#           DIFFERENT SIGNS, THEN C3 IS OF DIFFERENT SIGN FROM D.
#

```

```

#           SINCE WE ALWAYS CHOOSE THE GRADIENT DIRECTION,  $C1>0$ .
#           SINCE WE INSIST ON NEGATIVELY SLOPED ZERO CROSSING OF
#           SECOND DERIVATIVE,  $C3 < 0$ . THIS GUARANTEES THAT  $D>0$ .
#

```

```

#      HENCE, RELATIVE EXTREMA EXIST.
#
#      THE TRANSLATED POLYNOMIAL CAN BE REWRITTEN
#
#      Y=C(-A*X+A**3*X**3)
#
#      WHERE C=(D**1.5)/SQRT(27)*C3*C3
#      AND   A=SQRT(3)*C3/SQRT(D)
#
#      DISTANCE BETWEEN RELATIVE EXTREMA= 2./((SQRT(3)*A)
#      USING ODD NEIGHBORHOOD SIZES FROM 5 TO 23, THE AVG.
#      VALUE OF C FOR A PERFECT STEP EDGE OF CONTRAST 2 IS
#      CHAT=3.16257. FROM THIS FACT, WE CAN COMPUTE THE
#      CONTRAST ACROSS THE IDEAL STEP EDGE USING C.
#
#      CONTRAST=2*C/3.16257 = .6324*C.
#
#      THE RELATIVE ERROR MINIMIZING AHAT IN THE FIT A=AHAT/N
#      IS AHAT=1.793157. THUS FROM A AND AHAT, WE CAN COMPUTE
#      A SCALE FACTOR APPROPRIATE FOR GOOD STEP EDGES.
#
#      SCALE=A/(AHAT/N) = SQRT(3)*C3*N/1.793157*SQRT(D)
#
#      = .9659*C3*N/SQRT(D)
#
#      FOR IDEAL STEP EDGES, SCALE WILL BE VERY CLOSE TO 1.
#
#      IF WE REQUIRE THAT THE RELATIVE EXTREMA DISTANCE
#      BE LESS XHIGH=1.60897 TIMES THE NEIGHBORHOOD SIZE
#      AND LESS THAN XLOW=.585407 TIMES THE NEIGHBORHOOD
#      SIZE, THEN SCALE MUST SATISFY
#
#      .4 < SCALE < 1.1
#
#
#      THE CONTRAST AND SCALE VARIABLES IN THE PROGRAM
#      ARE THE SQUARES OF THOSE DEFINED ABOVE.
#
#      THE CURVATURE OF CONTOUR(CC ) PASSING EACH PIXEL IS
#      DEFINED AS FOLLOWS:
#
#      | -FC FR || FRR FRC | | -FC |
#      | FCR FCC | | FR |
#      CC = -----
#      SQRT((FR*FR + FC*FC)**3)
#
#      WHERE F STANDS FOR FITTING FUNCTION,
#      FX STANDS FOR DERIVATIVE
#      OF F RESPECTIVE TO X AND FXY STANDS FOR DERIVATIVE OF
#      F RESPECTIVE TO X THEN TO Y (X(Y) IS EQUAL TO R OR C).
#      THE SIGN OF CC IS DETERMINED AS FOLLOWS: THE DIRECTION
#      OF CONTOUR COINCIDES WITH EDGE DIRECTION,
#      CC IS POSITIVE (NEGATIVE) IF ANGLE INCREASES

```

```

# (DECREASES) IN THE CLOCKWISE DIRECTION.
# THE ACTUAL OUTPUT EQUALS TO
#     CONTOU = SIGN(1000*ALOG10(1+ABS(CC))), CC )
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
#     INCLUDE MACA1
#     DEFINE(.EDGE,0)
#     DEFINE (.NOEDGE,1)
#
#     SUBROUTINE EDGENC ( FDI, FDO, IIN, IOUT,
#                       ABUFF, CBUFF, SBUFF,TBUFF,
#                       OBUFF,NPPL, EDGTHR, D2TH,
#                       RADIUS, PROB, FPROB,
#                       AFLAG, BFLAG, FFLAG,
#                       LFLAG, QFLAG, IEV, *)
#
#     IMPLICIT INTEGER ( A - Z )
#
#     INCLUDE DSCAX
#     CHARACTER FDI ( .FDLENGTH )
#     CHARACTER FDO ( .FDLENGTH )
#
#     INTEGER IDENT( .IDLENGTH ), JDENT( .IDLENGTH )
#     INTEGER IIN(NPPL,10), IOUT(NPPL)
#     INTEGER ABUFF(NPPL), CBUFF(NPPL)
#     INTEGER SBUFF(NPPL),TBUFF(NPPL)
#     INTEGER NUMDR,A,OBUFF(NPPL)
#
#     REAL GRAD,GRADTH2, EDGTHR, RADIUS
#     REAL ANG, FCTDSC(13), FLOAT
#     REAL R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,D1V,D2V,D2TH
#     REAL F4,F5,F6,F7,F8,F9,F10,F11,F12,F13
#     REAL SN,CS,CS2,XINFLE,D,CONST,SCALE,GRADNT,DENOM
#     REAL R,C,C1,C2,C3,TT1,TT2,TT3,T1
#     REAL T2,T3,T4,T5,T6,T7,T8,T9,T10,S0
#     REAL SSQ2,SSQ3,SSQ4,SSQ5,SSQ6,SSQ7,SSQ8,SSQ9,SSQ10
#     REAL THETA, PI, PROB,FPROB, CRAD, HYPOT,EPS,CONTHR
#     REAL FISINV, CHIINV, SQRT, ALOG
#     REAL ATAN2, ASIN, ABS, CLCEXP
#     REAL P, Q, LAMDA, ANGSIN, QLN2
#     REAL PCHINV, EDGSTR, TEMP1, TEMP2
#     REAL DFREAL, SIZE,XLOW,XHIGH
#     REAL SNN,CSN,CST,SNT,GS
#     REAL SN2,SNC,CS2N,CNTOUR,FR,FC,FRR,FCC,FRC,FRC2
#
#     LOGICAL AFLAG, BFLAG, FFLAG, LFLAG, QFLAG
#
#     EQUIVALENCE ( NLIN, IDENT(.IDNLINS) )

```

```

#
DATA EPS / .001 /
DATA ANG / 0.26 / # ABOUT 15 DEGREES
DATA PI / 3.14159 /
DATA TT1,TT2,TT3/1.,.05,.0025/
DATA XLOW,XHIGH/.16,1.21/

#
#           PUSH PROGRAM NAME ONTO ERROR
#           STACK
#

CALL PPUSH ( "EDGENC" )
NUMDR = 1
CST=COS(ANG)
SNT=SIN(ANG)

#
#           OPEN INPUT FILES
#
#           COPY DESCRIPTOR RECORDS FROM INPUT
#           IMAGE, ROUTINE NAME, AND PARAMETERS
#           TO TEMPORARY SEQUENTIAL FILE.
#

CALL CPYIDR ( FDI, IDENT, .OPNTMP, IEV, %9998 )

#
CALL DSCNAM ( 'EDGENC', IEV, %9998 )

#
IF ( -FFLAG )
    CALL PDSCR ( 'Edge contrast threshold.',
                EDGTHR, IEV, %9998 )
ELSE
    CALL PDSCR ( 'Probability for F-Test.',
                EDGTHR, IEV, %9998 )
IF(QFLAG)
    CALL PDSCR('Second derivative threshold.',
                D2TH,IEV,%9998)

#
CALL PDSCR ( 'Radius.', RADIUS, IEV, %9998 )

#
IF ( AFLAG )
    $(
    CALL PDSCR ( 'Confidence Probability.',
                PROB, IEV, %9998 )
    CALL PDSCPS ( 'Band 1: Edge Strength.',
                IEV, %9998 )
    CALL PDSCPS ( 'Band 2: Edge contrast.',
                IEV, %9998 )
    CALL PDSCPS ( 'Band 3: Edge Direction.',
                IEV, %9998 )
    CALL PDSCPS ( 'Band 4: Confidence Interval.',
                IEV, %9998 )
    CALL PDSCPS ( 'Band 5: curvature of contour.',
                IEV, %9998 )
    )

```

```

        CALL PDSCPS ( 'Band 6: Binary Edge Image.',
                     IEV, %9998 )
        $)
    ELSE
        CALL PDSCPS ( 'Band 1: Binary Edge Image.',
                     IEV, %9998 )

#
#           OPEN OUTPUT IMAGE AND COPY TO IT THE
#           DESCRIPTOR RECORDS FROM TEMPORARY FILE
#

    DO I = 1, .IDLLENGTH
        JDENT( I ) = 0

#

        JDENT( .IDNPPL ) = NPPL
        JDENT( .IDNLINS ) = NLIN
        JDENT( .IDNCOLS ) = NPPL
        JDENT( .IDNROWS ) = 1
        JDENT( .IDNBNDS ) = 1
        JDENT( .IDNSBND ) = 1
        JDENT( .IDMODE ) = 0
        JDENT( .IDMIN ) = 0
        JDENT( .IDMAX ) = 1
        JDENT( .IDNBITS ) = 1

#

        IF ( AFLAG )
            $(
                JDENT( .IDNBNDS ) = 6
                MAXNUM = 99999
                JDENT( .IDMAX ) = MAXNUM
                MINNUM = -MAXNUM
                JDENT( .IDMIN ) = MINNUM
                JDENT( .IDNBITS ) = 0
            $)

#

        CALL COPYDS ( FDO, JDENT, IEV, %9998 )

#
#           IDENTIFY OUTPUT BAND NUMBERS
#

        EDGBND = 1

#

        IF ( AFLAG )
            $(
                STRBND = 1
                CONBND = 2
                DIRBND = 3
                CNFBND = 4
                CURBND = 5
                EDGBND = 6
            $)

#
#           PULL INFO FROM DESCRIPTOR RECORDS

```

```

#
CALL GDSCAX ( FDI, .CAXSLOPEFAC, FCTDSC,
              IDENT, FCTLNE, NUMDR, IEV, %9998)
#
      IF ( FCTLNE /= 13 )
        GO TO 9000
#
DO I = 4,13
  $(
    IF (FCTDSC(I) <= 0.)
      GO TO 9000
    FCTDSC(I) = 1. / FCTDSC(I)
  $)
#
F4=FCTDSC(4)
F5=FCTDSC(5)
F6=FCTDSC(6)
F7=FCTDSC(7)
F8=FCTDSC(8)
F9=FCTDSC(9)
F10=FCTDSC(10)
F11=FCTDSC(11)
F12=FCTDSC(12)
F13=FCTDSC(13)
#
#           CALCULATE DEGREES OF FREEDOM BASED
#           ON WINDOW SIZE USED
#
ROWSIZ = FCTDSC(1)
COLSIZ = FCTDSC(2)
DF      = ROWSIZ * COLSIZ - FCTDSC(3)
DFREAL = FLOAT( DF )
SIZE=ROWSIZ*COLSIZ
#
#           CALCULATE P AND Q FOR CONFIDENCE
#           INTERVAL
#
IF ( AFLAG )
  $(
    LAMDA = CLCEXP ( DF, PROB )
    P     = PROB**LAMDA
    Q     = PROB**(1.-LAMDA)
    QLN2  = -2. * ALOG ( 1.-Q )
    PCHINV = CHIINV ( P, DF, IEV )
  $)
#
#           ACTUAL NUMBER CRUNCHING
#
CONTHR = EDGTHR * EDGTHR
IF ( FFLAG)
GRADTH2 = FISINV ( 1.-FPROB, 2., DFREAL, IEV )

```



```

DO J=1,2
  CALL RREAD(FDI,IIN(1,J),J+1,I,
            IDENT,.WAIT,IEV,%9999)
DO K=1,NPPL
  $(
  R2=FLOAT(IIN(K,1))*F4
  R3=FLOAT(IIN(K,2))*F5
  GRAD=R2*R2+R3*R3
  S0=S0+GRAD
  N=N+1
  $)
  $)
S0=1+.5*S0/FLOAT(N)
S0=S0*DFREAL/FLOAT(ROWSIZ*COLSIZ)
T2=S0*TT1*SSQ2
T3=S0*TT1*SSQ3
T4=S0*TT2*SSQ4
T5=S0*TT2*SSQ5
T6=S0*TT2*SSQ6
T7=S0*TT3*SSQ7
T8=S0*TT3*SSQ8
T9=S0*TT3*SSQ9
T10=S0*TT3*SSQ10
$)
#
# IF USING THE MINIMAP, SEND NECESSARY SCALARS
#
IF ( - MMSKIP( .TRUE. ) )
  $(
  CALL MMSNDS( NPPL, 1, IEV, %9999 )
  CALL MMSNDS( RADIUS, -1, IEV, %9999 )
  CALL MMSNDS( SIZE, -2, IEV, %9999 )
  CALL MMSNDS( CONTHR, -3, IEV, %9999 )
  DO I = 4, 13
    CALL MMSNDS( FCTDSC( I ), - I, IEV, %9999 )
  $)
#
# GO THROUGH ALL ROWS IN IMAGE
#
DO I = 1,NLIN
  $(
#
# READ CUBIC FACET LINE
#
DO J = 1,10
  CALL RREAD( FDI, IIN(1,J), J+1, I, IDENT,
            .WAIT, IEV, %9999 )
#
# IF NOT USING MINIMAP, THEN
# FOR EACH PIXEL IN THE LINE:
#

```

```

IF ( MMSKIP( .TRUE. ) )
  DO K = 1, NPPL
    $(
#
#
#
      COMPUTE PROPERLY SCALED COEFFICIENTS

      R2 = FLOAT( IIN(K,1) ) * F4
      R3 = FLOAT( IIN(K,2) ) * F5
      R4 = FLOAT( IIN(K,3) ) * F6
      R5 = FLOAT( IIN(K,4) ) * F7
      R6 = FLOAT( IIN(K,5) ) * F8
      R7 = FLOAT( IIN(K,6) ) * F9
      R8 = FLOAT( IIN(K,7) ) * F10
      R9 = FLOAT( IIN(K,8) ) * F11
      R10 = FLOAT( IIN(K,9) ) * F12
      R11 = FLOAT( IIN(K,10) ) * F13

#
#
#
      IF BAYES FLAG, THEN DETERMINE
      BAYESIAN ESTIMATE

      IF(BFLAG)
        $(
          R2=R2/(1.+R11/T2)
          R3=R3/(1.+R11/T3)
          R4=R4/(1.+R11/T4)
          R5=R5/(1.+R11/T5)
          R6=R6/(1.+R11/T6)
          R7=R7/(1.+R11/T7)
          R8=R8/(1.+R11/T8)
          R9=R9/(1.+R11/T9)
          R10=R10/(1.+R11/T10)
        $)

#
      IF ISOTROPIC SECOND DERIVATIVE FLAG
      IF(QFLAG)
        $(
          D1V=R2*R2+R3*R3
          D2V=R4*R4+.5*R5*R5+R6*R6
          IF((D1V<CONTHR)!(D2V/D1V>D2TH)) A=.EDGE
          ELSE
          A=.NOEDGE
        $)
      ELSE
        $(
#
#
#
#
          IF THE "F" FLAG IS SPECIFIED,
          WE WILL BE USING THE F-TEST
          (CALCULATE THE EDGE STRENGTH)

          IF (AFLAG) CONST = 0.
          GRADNT = R2*R2 + R3*R3
          GS=SQRT(GRADNT)
#

```

```

IF ( FFLAG )
  $(
    DENOM = R11 / DFREAL
    IF ( DENOM == 0.0 ) DENOM = EPS
    EDGSTR =100.* GRADNT / ( 2. * DENOM )
  $)
ELSE EDGSTR = GRADNT

#
# INITIALIZE PIXEL AS A NOEDGE
# A=.NOEDGE
#
# CHECK TO MAKE SURE EDGE STRENGTH
# IS LARGE ENOUGH
#
IF ( GRADNT > GRADTH2 )
  $(
    SN=R2/GS
    CS=R3/GS
    CS2=CS*CS
  $)

#
# COMPUTE THE CUBIC IN THE DIRECTION
# OF THE GRADIENT. IT HAS THE FORM
# C3*X**3 + C2*X**2 + C1*X
# C1=R2*SN+R3*CS
# C2= (R4*SN+R5*CS)*SN+R6*CS2
# C3= (R7*SN+R8*CS)*SN*SN+
# (R9*SN+R10*CS)*CS2
#
# THE SECOND DERIVATIVE HAS THE FORM
# 6.*C3+2.*C2
# IF THE SLOPE IS TO BE NEGATIVE
# SO THAT THE GRADIENT IS A MAX RATHER
# THAN A MIN, THEN B MUST BE NONPOSITIVE.
# IF(C3<0.)
# $(
#
# FIND SMALLEST AND LARGEST VALUES
# OF SECOND LINE DIRECTIONAL
# DERIVATIVES IN THE THE DIRECTION
# THETA ALONG -RADIUS TO +RADIUS
#
# XINFLE=-C2/(3.*C3)
#
# TEST TO SEE IF SECOND LINE DIRECTIONAL
# DERIVATIVE HAS A ZERO CROSSING IN THE
# INTERVAL -RADIUS TO +RADIUS
#
# IF(ABS(XINFLE)<RADIUS)
# $(
#
#

```

```
          COMPUTE CONTRAST AND SCALE
#
#
D=(-3.*C1*C3+C2*C2)
#
#
D IS GUARANTEED TO BE POSITIVE
#
#
SINCE C3 IS NEGATIVE AND C1 IS POSITIVE
#
#
CONST=(.0148121/C3)*(D/C3)**3
SCALE=.93301*C3*C3*SIZE/D
IF( (CONST>CONTHR)&
    (SCALE>XLOW)&
    (SCALE < XHIGH) )
    A=.EDGE
#
#
$)
#
$)
#
$)
#
#
IF WE ARE DOING THE EDGE DETECTION IN
#
#
THE SECTOR AND NOT ALONG A LINE, THEN
#
#
THERE IS MORE WORK
IF( (A==.NOEDGE)&-LFLAG )
  $(
    DO II=1,2
    $(
#
#
#
#
CHECK TO MAKE SURE EDGE STRENGTH
IS LARGE ENOUGH
#
#
IF ( GRADNT > GRADTH2 )
  $(
    IF(II==1)
      $(
        SNN=SN*CST+SNT*CS
        CSN=CS*CST-SN*SNT
      $)
    ELSE
      $(
        SNN=SN*CST-SNT*CS
        CSN=CS*CST+SN*SNT
      $)
    CS2N=CSN*CSN
#
#
C1=R2*SNN+R3*CSN
C2= (R4*SNN+R5*CSN)*SNN+R6*CS2N
C3= (R7*SNN+R8*CSN)*SNN+SNN+
     (R9*SNN+R10*CSN)*CS2N
#
#
```

```

#           THE SECOND DERIVATIVE HAS THE FORM
#           6.*C3+2.*C2
#           IF THE SLOPE IS TO BE NEGATIVE
#           SO THAT THE GRADIENT IS A MAX RATHER
#           THAN A MIN, THEN B MUST BE NONPOSITIVE.
#           IF(C3<0.)
#             $(
#
#           FIND SMALLEST AND LARGEST VALUES
#           OF SECOND LINE DIRECTIONAL
#           DERIVATIVES IN THE THE DIRECTION
#           THETA ALONG -RADIUS TO +RADIUS
#
#           XINFLE=-C2/(3.*C3)
#
#           TEST TO SEE IF SECOND LINE DIRECTIONAL
#           DERIVATIVE HAS A ZERO CROSSING IN THE
#           INTERVAL -RADIUS TO +RADIUS
#
#           IF(ABS(XINFLE)<RADIUS)
#             $(
#
#                 COMPUTE CONTRAST AND SCALE
#
#                 D=(-3.*C1*C3+C2*C2)
#
#                 D IS GUARANTEED TO BE POSITIVE
#                 SINCE C3 IS NEGATIVE AND C1 IS POSITIVE
#
#                 CONST=(.0148121/C3)*(D/C3)**3
#                 SCALE=.93301*C3*C3*SIZE/D
#                 IF( (CONST>CONTHR)&
#                     (SCALE>XLOW)&
#                     (SCALE<XHIGH) )
#                   $(
#                     A=.EDGE
#                     BREAK
#                   $)
#                 $)
#             $)
#         $)
#     $)
#
#           IOUT(K)=A
#
#           IF "A" FLAG SPECIFIED, CALCULATE
#           STRENGTH, CONTRAST, DIRECTION AND

```

```

# CURVATURE CONTOUR
#
IF ( AFLAG )
$(
SBUFF(K) = EDGSTR
TBUFF(K)=MAXNUM
OBUFF(K)=SQRT(CONST)
#
IF( IOUT(K)=.NOEDGE)
$(
ABUFF(K)=360
FRC2 = 0.5*GRADNT*GS
IF(FRC2 > EPS)
$(
CNTOUR=(R2*R3*R5-R4*R3*R3-
R2*R2*R6)/FRC2
TBUFF(K)=SIGN(1000*ALOG10(1+
ABS(CNTOUR)),CNTOUR)
IF(TBUFF(K) < MINNUM)
TBUFF(K)=MINNUM
IF(TBUFF(K) >= MAXNUM)
TBUFF(K) = MAXNUM -1
$)
$)
ELSE
$(
#
# PUT EDGE DIRECTION IN ABUFF
#
EDGDIR = ATAN2( R3,R2 ) * 180. / PI
ABUFF(K) = -EDGDIR
#
# COMPUTE PARTIAL DERIVATIVE AT THE
# POINT (SN*XINFLE,CS*XINFLE). FR IS
# FIRST PARTIAL BY ROW, FC IS FIRST
# PARTIAL BY COL.,FRR IS HALF SECOND
# PARTIAL BY ROW, FCC IS HALF SECOND
# PARTIAL BY COL.,FRC SECOND PARTIAL
# BY ROW_COL.
SN2 = SN*SN
SNC = SN*CS
FR = R2+(2*R4*SN+R5*CS)*XINFLE +
(3*R7*SN2+2*R8*SNC+R9*CS2)*
XINFLE*XINFLE
FC = R3+(R5*SN+2*R6*CS)*XINFLE +
(R8*SN2+2*R9*SNC+3*R10*CS2)*
XINFLE*XINFLE
FRC2 = FR*FR +FC*FC
FRC2 = SQRT(FRC2)*FRC2
FRR = R4+(3*R7*SN+R8*CS)*XINFLE
FCC = R6+(R9*SN+3*R10*CS)*XINFLE

```

```

FRC = R5+2*(R8*SN+R9*CS)*XINFLE
CNTOUR=2*(FR*FC*FRC-FRR*FC*FC-
FR*FR*FCC)/FRC2
TBUFF(K)=SIGN(1000*ALOG10(1+
ABS(CNTOUR)),CNTOUR)
IF(TBUFF(K) < MINNUM)
TBUFF(K)=MINNUM
IF(TBUFF(K) >= MAXNUM)
TBUFF(K) = MAXNUM -1

```

```
#
```

```
    $)
```

```
#
```

```
#
```

```

CALCULATE CONFIDENCE OF EDGE DIRECTION
IF ITS AN EDGE PIXEL

```

```
#
```

```
CBUFF(K) = 90
```

```
#
```

```
IF ( IOUT(K) == .EDGE )
```

```
  $(
```

```
    CRAD = R11 * QLN2
```

```
    HYPOT = GRADNT * PCHINV
```

```
    IF ( CRAD > HYPOT )
```

```
      ANGSIN = 1.0
```

```
    ELSE
```

```
      ANGSIN = SQRT ( CRAD / HYPOT )
```

```
#
```

```
    THETA = ASIN ( ANGSIN )
```

```
    THETA = ABS ( THETA * 180. / PI )
```

```
    CBUFF(K) = THETA
```

```
  $)
```

```
#
```

```
    $)
```

```
#
```

```
  $)
```

```
#
```

```
#
```

```
PROCESSING USING THE MINIMAP
```

```
#
```

```
ELSE
```

```
  $(
```

```
    CALL MMSNDA( "IIN", IEV, %9999 )
```

```
    CALL MMEEXEC( .WAIT, IEV, %9999 )
```

```
    CALL MMRCVA( "IOUT", IEV, %9999 )
```

```
  $)
```

```
#
```

```
#
```

```
OUTPUT THE EDGE PIXELS
```

```
#
```

```

CALL RWRITE( FDO, IOUT, EDGBND, I, JDENT,
.WAIT, IEV, %9999 )

```

```
#
```

```
#
```

```

IF THE "A" FLAG IS SPECIFIED,
OUTPUT THE EDGE DIRECTIONS,

```

```
#
```


#

CONFIDENCE INTERVAL, AND EDGE
STRENGTH

```
IF ( AFLAG )
$(
CALL RWRITE ( FDO, SBUFF, STRBND, I, JDENT,
              .WAIT, IEV, %9999 )
CALL RWRITE ( FDO, OBUFF, CONBND, I, JDENT,
              .WAIT, IEV, %9999 )
CALL RWRITE ( FDO, ABUFF, DIRBND, I, JDENT,
              .WAIT, IEV, %9999 )
CALL RWRITE ( FDO, CBUFF, CNFBND, I, JDENT,
              .WAIT, IEV, %9999 )
CALL RWRITE ( FDO, TBUFF, CURBND, I, JDENT,
              .WAIT, IEV, %9999 )
$)
$)
```


#

CLOSE FILES AND RETURN

```
CALL CLOSE ( FDI )
CALL CLOSE ( FDO )
CALL PPOP
RETURN
```


#

--- ABNORMAL CONDITIONS ---

9000 CONTINUE

#

BAD VALUE IN DESCRIPTOR RECORD

```
IEV = -5037
GO TO 9998
```

#

9998 CONTINUE

#

READ OR WRITE ERROR

```
CALL CLOSE ( FDI )
CALL CLOSE ( FDO )
```

#

9999 CONTINUE
RETURN 1

#

END

J.6 EDGE Mini-MAP Link File

\$ GENMAP EDGEMM EDGE,EDGENC

J.7 EDGE MAP-driver

```
INCLUDE MAPDRV
DEFINE(NPPL,I1)
DEFINE(RADIUS,R1)
DEFINE(SIZE,R2)
DEFINE(GRADTH2,R3)
DEFINE(F4,R4)
DEFINE(F5,R5)
DEFINE(F6,R6)
DEFINE(F7,R7)
DEFINE(F8,R8)
DEFINE(F9,R9)
DEFINE(F10,R10)
DEFINE(F11,R11)
DEFINE(F12,R12)
DEFINE(F13,R13)
REAL EPS, XLOW, XHIGH, ANG, CST, SNT
REAL COS, SIN
DATA EPS / 0.001 /
DATA XLOW / 0.342702 /
DATA XHIGH / 2.59168 /
DATA ANG / 0.26 /
INCLUDE XEDGE
CST = COS( ANG )
SNT = SIN( ANG )
REPEAT
  $(
    CALL EDGENC( IIN, IOUT, NPPL * 2, RADIUS,
                SIZE, GRADTH2, F4, F5, F6, F7,
                F8, F9, F10, F11, F12, F13,
                EPS, XLOW, XHIGH, CST, SNT,
                NPPL * 20 )

    PAUSE
  $)
END
```

J.8 EDGENC MAP-subroutine

```
INCLUDE MAPSUB
SUBROUTINE EDGENC( IIN, IOU, NPPL2, RADIUS, SIZE,
                  GRDTH2, F4, F5, F6, F7, F8,
                  F9, F10, F11, F12, F13, EPS,
                  XLOW, XHIGH, CST, SNT, NP210 )

DEFINE(.EDGE,0)
DEFINE(.NOEDGE,1)
INTEGER NP210, NPPL2, IIN( NP210 ), IOU( NPPL2 ), K, A, II
REAL RADIUS, SIZE, GRDTH2, F4, F5, F6, F7
REAL F8, F9, F10, F11, F12, F13, EPS
REAL XLOW, XHIGH, CST, SNT, R2, R3, R4
REAL R5, R6, R7, R8, R9, R10, R11, GRADNT
REAL GS, EDGSTR, SN, CS, CS2, C1, C2, C3
REAL XINFLE, D, CONST, SCALE, SNN, CSN
REAL FLOAT, SQRT, ABS

#
#           FOR EACH PIXEL IN THE LINE
#
      DO K = 1, NPPL2, 2
        $(
#
#           COMPUTE PROPERLY SCALED COEFFICIENTS
#
          R2 = FLOAT( IIN(K) ) * F4
          R3 = FLOAT( IIN(K+NPPL2) ) * F5
          R4 = FLOAT( IIN(K+2*NPPL2) ) * F6
          R5 = FLOAT( IIN(K+3*NPPL2) ) * F7
          R6 = FLOAT( IIN(K+4*NPPL2) ) * F8
          R7 = FLOAT( IIN(K+5*NPPL2) ) * F9
          R8 = FLOAT( IIN(K+6*NPPL2) ) * F10
          R9 = FLOAT( IIN(K+7*NPPL2) ) * F11
          R10 = FLOAT( IIN(K+8*NPPL2) ) * F12
          R11 = FLOAT( IIN(K+9*NPPL2) ) * F13
          GRADNT = R2*R2 + R3*R3
          GS=SQRT(GRADNT)
          EDGSTR = GRADNT
          A=.NOEDGE

#
#           CHECK TO MAKE SURE EDGE STRENGTH
#           IS LARGE ENOUGH
#
          IF ( GS > EPS )
            $(
              SN=R2/GS
              CS=R3/GS
              CS2=CS*CS

#
              C1=R2*SN+R3*CS
```

```

C2= (R4*SN+R5*CS)*SN+R6*CS2
C3= (R7*SN+R8*CS)*SN*SN+
      (R9*SN+R10*CS)*CS2

```

```

#
#
#
#
#
#

```

```

      THE SECOND DERIVATIVE HAS THE FORM
                6.*C3+2.*C2
      IF THE SLOPE IS TO BE NEGATIVE
      SO THAT THE GRADIENT IS A MAX RATHER
      THAN A MIN, THEN B MUST BE NONPOSITIVE.
      IF(C3<0.)
        $(

```

```

#
#
#
#
#
#

```

```

      FIND SMALLEST AND LARGEST VALUES
      OF SECOND LINE DIRECTIONAL
      DERIVATIVES IN THE THE DIRECTION
      THETA ALONG -RADIUS TO +RADIUS

```

```

      XINFLE=-C2/(3.*C3)

```

```

#
#
#
#
#

```

```

      TEST TO SEE IF SECOND LINE DIRECTIONAL
      DERIVATIVE HAS A ZERO CROSSING IN THE
      INTERVAL -RADIUS TO +RADIUS

```

```

      IF(ABS(XINFLE)<RADIUS)
        $(

```

```

#
#
#

```

```

      COMPUTE CONTRAST AND SCALE

```

```

      D=(-3.*C1*C3+C2*C2)

```

```

#
#
#
#
#

```

```

      D IS GUARANTEED TO BE POSITIVE
      SINCE C3 IS NEGATIVE AND C1 IS POSITIVE

```

```

      CONST=(.0148121/C3)*
            ((D/C3)*(D/C3)*(D/C3))

```

```

#

```

```

      SCALE=.93301*C3*C3*SIZE/D
      IF(CONST>GRDTH2) A=.EDGE
      IF( (CONST>GRDTH2)&
          (SCALE>XLOW)&
          (SCALE < XHIGH) )
        A=.EDGE

```

```

#

```

```

      $)

```

```

#

```

```

      $)

```

```

      $)

```

```

      IF( A==.NOEDGE )

```

```

      $(
        DO II=1,2
          $(

```


#

CHECK TO MAKE SURE EDGE STRENGTH IS LARGE ENOUGH

IF (GRADNT > EPS)

\S (
 IF(II==1)
 \S (
 SNN=SN*CS+TNT*CS
 CSN=CS*CS-TNT*SN
 \S)
 ELSE
 \S (
 SNN=SN*CS-TNT*CS
 CSN=CS*CS+TNT*SN
 \S)
 CS2=CSN*CSN

C1=R2*SNN+R3*CSN
C2= (R4*SNN+R5*CSN)*SNN+R6*CS2
C3= (R7*SNN+R8*CSN)*SNN*SNN+
 (R9*SNN+R10*CSN)*CS2

THE SECOND DERIVATIVE HAS THE FORM
 6.*C3+2.*C2
IF THE SLOPE IS TO BE NEGATIVE
SO THAT THE GRADIENT IS A MAX RATHER
THAN A MIN, THEN B MUST BE NONPOSITIVE.
IF(C3<0.)
 \S (

FIND SMALLEST AND LARGEST VALUES
OF SECOND LINE DIRECTIONAL
DERIVATIVES IN THE THE DIRECTION
THETA ALONG -RADIUS TO +RADIUS

$$XINFLE=-C2/(3.*C3)$$

TEST TO SEE IF SECOND LINE DIRECTIONAL
DERIVATIVE HAS A ZERO CROSSING IN THE
INTERVAL -RADIUS TO +RADIUS

IF(ABS(XINFLE)<RADIUS)
 \S (

 COMPUTE CONTRAST AND SCALE

$$D=(-3.*C1*C3+C2*C2)$$

D IS GUARANTEED TO BE POSITIVE

```

#           SINCE C3 IS NEGATIVE AND C1 IS POSITIVE
#
#           CONST=(.0148121/C3)*
#               ((D/C3)*(D/C3)*(D/C3))
#           SCALE=.93301*C3*C3*SIZE/D
#           IF( (CONST>GRDTH2)&
#               (SCALE>XLOW)&
#               (SCALE<XHIGH) )
#           IF(CONST>GRDTH2)
#               $(
#                   A=.EDGE
#                   BREAK
#                   $)
#           $)
#           $)
#           $)
#           $)
#           IOUT(K)=A
#           IOUT(K+1)=0
#           $)
RETURN
END

```

J.9 XEDGE Include File

```
NAME4(.E,.D,.G,.E)
GETARRAYS
IDESC1( 1 )
DEFINE(IIN,IAR1)
IDESC1( 2 )
DEFINE(IOUT,IAR2)
```

K APPENDIX K: RIDVAL COMMAND

What follows are the files associated with the GIPSY command, RIDVAL. This includes the following files:

1. GIPSY_DOC:RIDVAL.DOC - on-line documentation,
2. GIPSY_RUN:RIDVAL.RUN - a run-file to test the command,
3. GIPSY_LNK:RIDVAL.LNK - a link file which instructs the system on how to build the command,
4. GIPSY_SRC:RIDDV.RAT - the command driver,
5. GIPSY_SRC:RIDNC.RAT - work routine for the host,
6. GIPSY_SRC:RIDVAL.MLK - link file for the mini-MAP module,
7. GIPSY_SRC:RIDVAL.MMR - MAP-driver for the RIDVAL command,

8. GIPSY_SRC:RIDNC.MMR - work routine for the mini-MAP,

9. GIPSY_MAPSRC:HDIZER.MMR - MAP-subroutine called by RIDNC

K.1 RIDVAL Documentation

*RIDVAL Ridge and Valley finder.

VERSION: A.02 DATE: 09-27-82
AUTHOR: R.M.HARALICK & SUI WEI

ACTION: This command takes the central neighborhood cubic facet parameters for each pixel's central neighborhood and uses them to define a line in the direction of maximum magnitude of second directional derivative. There are two such directions which are 90 degrees apart. If in one of these directions a zero crossing of the first directional derivative occurs sufficiently close to the pixel's center a ridge or valley is declared depending on the sign of the second directional derivative. Pixels are marked as follows:

- (0) None
- (1) Ridge
- (2) Valley

To avoid misclassifications due to noise or fitting artifacts additional requirements for a pixel to be labeled as a ridge or valley are that: the gradient angle change taken in the direction extremizing second directional derivative must exceed 15 degrees; the curvature must exceed 1; and if the direction of smaller curvature is the one having a zero-crossing then the ratio of smaller to larger curvature be greater than .25. Any of these default values can be changed by setting the appropriate flag (A,C,F, respectively).

The T flag can be used to obtain additional topographic labels of saddle (3), peak (4), pit(5), flat (6), slope (7) convex hill (8), concave hill (9), saddle hill (10), edge (11). The H flag can be used to combine all the hillside categories into one, hill (12).

The M flag will generate five numeric bands: one for direction of gradient for hillside or edge pixels, one for direction extremizing curvature for ridges and valleys, one for edge contrast, one for curvature across the ridge or valley, and one for the curvature of contour. It outputs as follows:

- BAND 1: Edge direction (-180 to 180 degrees)
- BAND 2: Ridge (valley) direction

(valleys [-180,0>, and ridges [0,180>)
 BAND 3: Edge contrast
 BAND 4: Curvature across the ridge or valley.
 (Multiplied by 100)
 BAND 5: Curvature of contour
 (1000×LOG(1+ABS(curvature)))
 BAND 6: Symbolic image of ridge valley
 and edge labels (0,1,2,11)

The edge direction is defined as the clockwise direction from the column axis in which as you walk along the edge, the higher valued area is to the right.

The ridge or valley direction is defined as the clockwise direction from the column axis. Ridge orientation angles are positive. Valley orientation angles are negative. For example, a ridge oriented parallel to the column axis is zero degrees whereas a valley oriented parallel to the column axis is -180 degrees; a ridge oriented vertical to the column axis is 90 degrees whereas a valley in the same orientation is -90 degrees.

SOURCE: Disk, input file name.

DESTINATION: Disk, output file name or
 if P flag set then terminal

FLAGS: A flag Gradient angle change threshold.
 C flag Curvature threshold.
 F flag Smaller to larger curvature ratio threshold.
 G flag Gradient treshold
 (for hillside or edge category).
 H flag Combine all hill categories
 to one hillside class (12).
 M flag Output gradient direction for hillside pixels
 in band 1 and extremal curvature direction
 for ridges and valleys in band 2. The
 convention is: hillsides [-180,180], valleys
 [-180,0>, and ridges [0,180>. Edge constrast
 is in Band 3, curvature of ridge or valley is
 in Band 4 and curvature of contour is in Band
 5.
 P flag Examines everything about a selected pixel
 and output to the terminal.
 T flag Produces additional topographic labels of
 saddle(3), peak(4), pit(5), flat (6), slope
 (7) convex hill (8), concave hill (9), saddle
 hill (10), and edge (11).

QUESTIONS: (1) Enter event radius (d=.85, .1 to 2)

The reply to this question should be one half the total length of the line on which the first derivative zero crossing is sought.

- (2) Enter depth threshold (D=.2, .05 to .90)
The reply to this questions should be the relativeheight of ridge or the depth of valley below which such ridge and valleys are considered to be too small to be significant.
- (3) If A flag is set, then a question asking the user to define a new gradient angle change threshold is asked.
Enter angle threshold (D=30., 10 to 180)
- (4) If C flag is set, then a question asking the user to define a new curvature threshold is asked.
Enter curvature threshold (D=1., 0 to 100)
- (5) If F flag is set, then a question asking the user to define a new threshold for the ratio of smaller to larger curvature is asked. If the ratio is smaller than the threshold then the smaller curvature is not used.
Enter curvature ratio (D=.25, 0 to 1)
- (6) If T flag is set, then a question asking the user to define a gradient threshold is asked.
Enter gradient threshold (D=5., 1 to 20)
- (7) If P flag is set, the user is asked to enter in a pixel position. A summary of all the various parameters of the surface fit and directional derivatives is given for that pixel position to the output device (terminal).

COMMAND STRING EXAMPLE:

RIDVAL BAR.RDV < BAR.CUB

Enter event radius (D=.85, .1 to 2.) 1.0
Enter depth threshold (D=.1, 0 to 100) 4.

The image containing the coefficients of the cubic fit is BAR.CUB and the created ridge valley image is BAR.RDV. If a zero-crossing of the first directional derivative taken in the direction of the maximum magnitude of second directional derivative exists within a distance 1 of the pixel's center, and the depth is greater than 4., then a ridge (1) or valley (2) is declared at that pixel's position. Non ridge - valley pixels are marked with a zero.

COMMENTS: (1) Let f stands for the fitted cubic function:

$$f = k1 + k2*r + k3*c + k4*r*c + k6*c**2 + k7*r**3 + k8*r**2*c + k9*r*c**2 + k10*c**3$$

(2) The curvature of contour: cc passing through a pixel is given by:

$$cc = \frac{\begin{vmatrix} -f_c & f_r & | & f_{rr} & f_{rc} & | & -f_c \\ & & & f_{rc} & f_{cc} & | & f_r \end{vmatrix}}{\text{SQRT}((f_r*f_r + f_c*f_c)**3)}$$

Where f_x stands for the partial derivative of f respective to x and f_{xy} stands for derivative of f respective to x then to y.

The sign of cc for the ridge curve is positive and the sign of cc for the valley curve is negative

The sign of cc for edge is determined as follows:

cc is positive (negative) if following the direction of the edge the high intensity value area is to the right.

$$\text{OUTPUT} = \text{SIGN}(1000*\text{ALOG10}(1+\text{ABS}(\text{CC})), \text{CC})$$

(3) The curvature cv across a ridge or valley passing through a pixel is given by:

$$cv = \frac{hpp}{\text{SQRT}((1+hp)**3)} = hpp$$

where h is the intersection of the fitted function of f and a vertical plane in the direction maximum magnitude of the second directional derivative (SN,CS).

$$\text{Let } r = p*SN \quad \text{and} \quad c = p*CS$$

$$\text{then } h = A*p**3 + B*p**2 + C*p + k1$$

where

$$A = k7*SN**3 + k8*SN**2*CS + k9*CS**2*SN + K10*CS**3$$

$$B = k4*SN**2 + k5*SN*CS + k6*CS**2$$

$$C = k2*SN + K3*CS$$

and

$$hpp = 6*A*p + 2B$$

p is the distance along the horizontal axis of the vertical plane. hp stands derivative of h relative to p and hpp stands second derivative of h relative to p. The numerator equals 1 because the derivative of h respective to p is 0 at ridge or valley pixels.

- (4) The gradient angle change gac taken in the direction extremizing second directional derivative is given by:

$$gac = ABS((CE*frr*SN + CE*frc*CS - SE*frc*SN - SE*fcc*CS) / grad)$$

where $grad$ is the gradient magnitude of the pixel,

$$grad = SQRT(fr**2+fc**2)$$

CE and SE is the gradient direction.

$$CE = fc/grad$$

$$SE = fr/grad$$

A simple explanation is following:

The gradient angle ag is equal to

$$ag = ATAN2(fr,fc)$$

The change of ag in the direction (SN,CS) is the partial derivative of ag relative to p :

$$\begin{aligned} (ag)_p &= (fc**2/(fr**2+fc**2)) * \\ &\quad (((fr)_p*fc - fr*(fc)_p) / fc**2) \\ &= (1/(fr**2+fc**2)) * \\ &\quad (frr*fc*SN + frc*fc*CS - \\ &\quad fr*frc*SN - frr*fcc*CS) \\ &= 1/SQRT(fr**2+fc**2) * \\ &\quad (fc/SQRT(fr**2+fc**2)*frr*SN + \\ &\quad fc/SQRT(fr**2+fc**2)*frc*CS - \\ &\quad fr/SQRT(fr**2+fc**2)*frr*SN - \\ &\quad fr/SQRT(fr**2+fc**2)*frc*CS) \end{aligned}$$

$$gac = ABS((ap)_p)$$

REFERENCES:

1. R.M.Haralick, 'Ridges and Valleys on Digital Images', Computer Vision, Graphics, and Image Processing 22, 1983, page 28-38.
2. R.M.Haralick, et, 'The Topographic Primal Sketch', The International Journal of Robotics Research, Vol. 2, No.1, Spring 1983, page 50-72.
3. R.M.Haralick, 'Digital Step Edges From Zero Crossing Second Directional Derivative', Pattern Analysis & Machine Intelligence. IEEE Trans.
4. R.M.Haralick, 'Second Directional Derivative Zero-Crossing Detection Using the Cubic Facet Model' Proceedings of the 4TH Scandinavian Conference Trondheim, June, 1985

K.2 RIDVAL Run-file

```
$ !
$ ! RIDVAL.RUN
$ !
$ ! MAKE A PARABOLA FACING UP
$ !
$ MKBAR BAR.IMG
50
V
L
$ ARITHM BAR.SUB < BAR.IMG (S)
25
N
$ IMGPWR BAR.PWR < BAR.SUB
2
1
$ !
$ ! GET THE CUBIC FACET PARAMETERS
$ !
$ CNCFCT BAR.CUB < BAR.PWR
5
5
17
$ !
$ ! MARK THE VALLEYS
$ !
$ RIDVAL BAR.RDV < BAR.CUB (C)
1.
1.5
$ !
$ ! TO TEST THE RESULT WE WILL MAKE ALL VALLEY PIXELS
$ ! TAKE A 1 AND THEN COMPARE
$ !
$ TRSLD BAR.TRS < BAR.PWR
0
$ !
$ !
$ ! TO COMPARE WE WILL MAKE ALL VALLEY PIXELS TAKE
$ ! THE VALUE 2
$ !
$ EXSIF
OPEN BAR.TRS
MID 18 0
Y
DONE
$ ARITHM BAR2.TRS < BAR.TRS (M)
2
N
```

```

$      EXSIF
      OPEN BAR2.TRS
      MID 18 1
      Y
      DONE
$      RMS TT < BAR2.TRS, BAR.RDV
$ !
$ !      NOW TEST FOR THE RIDGES
$ !
$ !
$ !      INVERT THE PARABOLA
$ !
$      INVIMG BARM.IMG < BAR.PWR
$      ARITHM BARINV.IMG < BARM.IMG (A)
      1000
      N
$ !
$ !      GET THE CUBIC FACET PARAMETERS
$ !
$      CNCFCT BARINV.CUB < BARINV.IMG
      5
      5
      17
$ !
$ !      MARK THE RIDGES
$ !
$      RIDVAL BARINV.RDV < BARINV.CUB (C)
      1.
      1.5
$ !
$      RMS TT < BARINV.RDV, BAR.TRS
$ !
$      DELETE
      BAR.IMG
      BAR.SUB
      BAR.PWR
      BAR.CUB
      BAR.RDV
      BAR.TRS
      BARM.IMG
      BARINV.IMG
      BARINV.CUB
      BARINV.RDV
      BAR2.TRS

$ !
$      EXIT

```


K.3 RIDVAL Host Link File

⌘ GENCMD RIDVAL RIDDV,RIDNC,RIDVAL,HDIZER,TRIDVL

K.4 RIDDV Routine

```
#--RIDDV          DRIVER FOR THE RIDGE VALLEY COMMAND
#
# IDENTIFICATION
#
#       TITLE          RIDDV
#       AUTHOR         ROBERT M HARALICK
#       VERSION        C.01
#       DATE           OCTOBER 28, 1983
#       LANGUAGE       RATFOR
#       SYSTEM         VAX-11-780
#
# UPDATE
#
#       UPDATE         # 1
#       AUTHOR         ROBERT M. HARALICK
#       DATE           23-JAN-1984
#       VERSION        C.01
#       PURPOSE        TO CHANGE THE DEFAULT PARAMETER
#                       ON CURVATURE
#
#       UPDATE         # 2
#       AUTHOR         ROBERT M. HARALICK
#       DATE           05-JUN-1984
#       VERSION        C.01
#       PURPOSE        CORRECT THE QUESTION ON CURVATURE
#                       CHANGE THE CURVATURE DEFAULT
#
#       UPDATE         # 3
#       AUTHOR         ROBERT M. HARALICK
#       DATE           06-JUL-1984
#       VERSION        C.01
#       PURPOSE        CHANGE DEFAULT FOR CURVATURE
#
# PURPOSE
#
#       THE PURPOSE OF THIS COMMAND IS TO TAKE THE
#       ELEVEN BAND OUTPUT OF THE COMMAND CNCFACT AND
#       DETECT THE RIDGES AND VALLEYS ON IT.
#
# ENTRY POINT
#
#       RIDDV( WORK, ERRET )
#
# ARGUMENT LISTING
#
#       WORK          INT.ARR          WORKING STORAGE ARRAY
#       ERRET         INT              ALTERNATE ERROR RETURN
#
```

```

# INCLUDE FILES/COMMONS
#
#       MACA1      INCLUDE      GIPSY TOKEN DEFINITIONS INCLUDE
#                               FILE FOR A1 CHARACTERS.
#       GIPCOM     COMMON       GIPSY COMMAND LINE INFORMATION
#       ERROR      COMMON       GIPSY ERROR STACK INFORMATION
#
# ROUTINES CALLED
#
#       RDKINL    OPEN A SIF AND FILL IN THE IDENT
#                   BLOCK.          (GIPSY ROUTINE)
#       CLOSE     ROUTINE TO CLOSE A FILE.  (GIPSY ROUTINE)
#       OSALOC    ROUTINE TO CHECK FOR EXCEEDED ARRAY
#                   SIZE.          (GIPSY ROUTINE)
#       RIDNC     THIS IS THE NUMBER CRUNCHER ROUTINE
#                   FOR THE COMMAND RIDVAL  (GIPSY ROUTINE)
#       PPOP      REMOVES THE PROGRAM NAME FROM THE
#                   ERROR STACK.          (GIPSY ROUTINE)
#       PPUSE     PLACES THE PROGRAM NAME ONTO THE ERROR
#                   STACK.  (GIPSY ROUTINE)
#
# REMARKS
#
#
# *****
#
#
#       INCLUDE MACA1
#       SUBROUTINE RIDDV ( WORK, *)
#
#
#       IMPLICIT INTEGER (A-Z)
#
#       INCLUDE GIPCOM
#       INCLUDE ERROR
#
#       INTEGER WORK ( .ARB )
#       INTEGER IDENT ( .IDLENGTH )
#       REAL GRADTH,FRATIO,RADIUS,CRVTHR,ANGTHR,DEPTHR
#       REAL DGRAD,GMAX,DMAX,DRADIUS,DCURVE
#       REAL DANG,DRV,DDEP,DFRAT
#       LOGICAL TFLAG,FFLAG,PFLAG,CFLAG
#       LOGICAL RFLAG,GFLAG,AFLAG,DFLAG,MFLAG
#
#       EQUIVALENCE (NPPL, IDENT(.IDNPPL)),
#                   (NLINS, IDENT(.IDNLINS))
#       EQUIVALENCE (NCOLS, IDENT(.IDNCOLS)),
#                   (NROWS, IDENT(.IDNROWS))
#       EQUIVALENCE (NTBND, IDENT(.IDNBND))

```

```
EQUIVALENCE (MODE, IDENT(.IDMODE))
EQUIVALENCE (AFLAG,UFLAG(1))
EQUIVALENCE (CFLAG,UFLAG(3))
EQUIVALENCE (FFLAG,UFLAG(6))
```

```
#
```

```
COMMON /RDV/MFLAG,TFLAG,PFLAG,HFLAG,ROW,COL
```

```
#
```

```
DATA DGRAD/5./
DATA DCURVE/2.1/
DATA DANG/15./
DATA DFRAT/.25/
DATA DRADIUS/.85/
DATA ROW/-1/
```

```
#
```

```
CALL PPUSH ("RIDDV ")
CALL MMINIT( 0 )
```

```
#
```

```
HFLAG = UFLAG(8)
MFLAG = UFLAG(13)
PFLAG = UFLAG(16)
TFLAG = UFLAG(20)
IF(HFLAG) TFLAG=.TRUE.
RFLAG = .TRUE.
DFLAG = .TRUE.
GFLAG = .TRUE.
GRADTH = DGRAD
RADIUS = DRADIUS
CRVTHR = DCURVE
ANGTHR = DANG
FRATIO = DFRAT
```

```
#
```

```
#
```

```
#
```

```
SET UP INPUT FILE
```

```
CALL RDKINL( FDI1, IDENT, .READONLY, IEV, %9998 )
CALL CLOSE(FDI1)
DDEP=.2
DMAX=.8
DEPTHR = DDEP
GMAX=FLOAT(IDENT(.IDMAX)-IDENT(.IDMIN) )
```

```
#
```

```
#
```

```
#
```

```
CHECK THE INPUT FILE
```

```
IF( MODE /= 0 & MODE /= .INTMODE ) GO TO 9000
IF (NPPL /= NCOLS ! NROWS /= 1) GO TO 9010
IF (NTBND /= 11) GO TO 9020
```

```
#
```

```
#
```

```
#
```

```
SET UP BUFFERS
```

```
NXT = 1
IBUFF = GETWPM(NXT,MODE,NPPL*9,"IIN")
OBUFF = GETWPM(NXT,MODE,NPPL,"IOUT")
```


RETURN 1
END

K.5 RIDNC Routine

```
#--RIDNC          NUMBER CRUNCHER FOR THE RIDGE VALLEY COMMAND
#
# IDENTIFICATION
#
#     TITLE          RIDNC
#     AUTHOR         ROBERT M HARALICK
#     VERSION        C.01
#     DATE           OCTOBER 28, 1983
#     LANGUAGE       RATFOR
#     SYSTEM         VAX-11-780
#
# UPDATE
#
#     UPDATE         # 1
#     AUTHOR         OSCAR A ZUNIGA
#     DATE           19-APR-1984
#     VERSION        C.01
#     PURPOSE        TO ELIMINATE CHANGE OF
#                   SIGN IN RIDGE(VALLEY)
#                   ANGLE, AND TO ROUND IT
#                   BEFORE WRITING IT TO
#                   THE OUTPUT FILE
#
# PURPOSE
#
#     THIS ROUTINE IS THE NUMBER CRUNCHER
#     FOR THE COMMAND RIDVAL
#
# ENTRY POINT
#
#     RIDNC ( FDI, FDO, IIN, IOUT, NPPL,
#             ANGLE, CBAND, RADIUS, CRVTHR,
#             ANGTHR,DEPTH,FRATIO,GRADTH,IEV, ERRET)
#
# ARGUMENT LISTING
#
#     FDI      CHR.ARR      INPUT FILE DESCRIPTOR
#     FDO      CHR.ARR      OUTPUT FILE DESCRIPTOR
#     IIN      INT.ARR      INPUT BUFFER ARRAY
#     IOUT     INT.ARR      OUTPUT BUFFER ARRAY
#     NPPL     INT          NPPL OF INPUT FILE
#     ANGLE    INT.ARR      ANGLE OUTPUT BUFFER ARRAY
#     CBAND    INTEGER      CURVATURE DIRECTION OUTPUT
#                   BUFFER ARRAY
#     RADIUS   REAL         DISTANCE OF RIDGE OR VALLEY
#                   TO CENTER OF PIXEL
#     CRVTHR   REAL         CURVATURE THRESHOLD WHICH MUST
#                   BE EXCEEDED IN ORDER FOR
```



```

# CURVATURE TO BE CONSIDERED
# NON-ZERO.
# ANGTHR REAL GRADIENT ANGLE DIFFERENCE
# THRESHOLD
# DEPTH REAL DEPTH THRESHOLD
# FRATIO REAL RATIO THAT SMALLER TO LARGER
# CURVATURE MUST EXCEED IN ORDER
# THAT SMALLER CURVATURE COUNT
# GRADTH REAL GRADIENT THRESHOLD
# IEV INT INTEGER EVENT VARIABLE
# ERRET INT ALTERNATE ERROR RETURN
#
# INCLUDE FILES/COMMONS
#
# MACA1 INCLUDE GIPSY TOKEN DEFINITIONS INCLUDE
# FILE FOR A1 CHARACTERS.
#
# ROUTINES CALLED
#
# PPU SH PLACES THE PROGRAM NAME ONTO THE ERROR
# STACK. (GIPSY ROUTINE)
# CPYIDR ROUTINE TO OPEN THE INPUT FILE AND
# COPY THE DESCRIPTOR RECORDS.
# (GIPSY ROUTINE)
# DSCNAM WRITES SUBROUTINE NAME INTO THE
# DESCRIPTOR RECORDS. (GIPSY ROUTINE)
# PDSCR WRITES A REAL NUMBER TO THE
# DESCRIPTOR RECORDS. (GIPSY ROUTINE)
# COPYDS COPY THE DESCRIPTOR RECORDS FROM TEMP
# FILE TO SIF & OPEN OUTPUT FILE.
# (GIPSY ROUTINE)
# GDSCAX GETS VALUES FROM THE DESCRIPTOR RECORDS
# (GIPSY ROUTINE)
# RREAD READ A RECORD FROM A SIF. (GIPSY ROUTINE)
# RWRITE WRITE A RECORD TO A SIF. (GIPSY ROUTINE)
# CLOSE ROUTINE TO CLOSE A FILE. (GIPSY ROUTINE)
# PPOP REMOVES THE PROGRAM NAME FROM THE
# ERROR STACK. (GIPSY ROUTINE)
#
#
#*****
#
# INCLUDE MACA1
# SUBROUTINE RIDNC ( FDI, FDO, IIN, IOUT,
# NPPL,ANGLE, CBAND,
# RADIUS,CRVTHR,ANGTHR,DEPTH,FRATIO,
# GRADTH,IEV,*)
#
# IMPLICIT INTEGER ( A - Z )
# INCLUDE DSCAX

```

```

CHARACTER FDI ( .FDLENGTH )
CHARACTER FDO ( .FDLENGTH )
#
INTEGER IDENT ( .IDLENGTH ), JDENT ( .IDLENGTH )
INTEGER IIN(NPPL,9),IOUT(NPPL)
INTEGER NUMDR,ANGLE(NPPL),CBAND(NPPL)
LOGICAL INITX
REAL SIZE
REAL CRVTHR,FRATIO,ANGTHR,THR,RADIUS
REAL FLOAT,FCTDSC(12),SCALE(12)
REAL GRADTH,GTHETA,RVTHET,R2
REAL R3,R4,R5,R6,R7,R8,R9,R10
LOGICAL HFLAG,MFLAG,TFLAG,PFLAG
#
EQUIVALENCE ( NLIN, IDENT(.IDNLINS) )
EQUIVALENCE ( MODE, IDENT(.IDMODE) )
#
COMMON /RDV/MFLAG,TFLAG,PFLAG,HFLAG,ROW,COL
#
CALL PPUSH ( "RIDNC " )
NUMDR = 1
#
IF(PFLAG)
#
$(
CALL RDKINL(FDI,IDENT,.READONLY,IEV,%9998)
#
CALL GDSCAX(FDI,.CAXSLOPEFAC,FCTDSC,
            IDENT,FCTLNE,NUMDR,IEV,%9998)
#
IF(FCTLNE =13) GO TO 9000
#
DO I=4,12
$(
IF(FCTDSC(I)<=0.) GO TO 9000
SCALE(I)=1./FCTDSC(I)
$)
SIZE=(FCTDSC(1)-1.)*.5
#
#
#
ACTUAL NUMBER CRUNCHING
DO J=1,9
CALL RREAD( FDI, IIN(1,J), J+1, ROW, IDENT,
            .WAIT, IEV, %9998 )
#
R2 = FLOAT( IIN(COL,1) ) * SCALE(4)
R3 = FLOAT( IIN(COL,2) ) * SCALE(5)
R4 = FLOAT( IIN(COL,3) ) * SCALE(6)
R5 = FLOAT( IIN(COL,4) ) * SCALE(7)
R6 = FLOAT( IIN(COL,5) ) * SCALE(8)
R7 = FLOAT( IIN(COL,6) ) * SCALE(9)

```



```

          $)
#
#           OPEN OUTPUT IMAGE AND COPY TO IT THE
#           DESCRIPTOR RECORDS FROM TEMPORARY FILE
#
IBAND=1
IF(MFLAG) IBAND=3
DO I = 1, .IDLENGTH
    JDENT ( I ) = 0
#
JDENT ( .IDNPPL ) = NPPL
JDENT ( .IDNLINS ) = NLIN
JDENT ( .IDNCOLS ) = NPPL
JDENT ( .IDNROWS ) = 1
JDENT ( .IDNBND ) = IBAND
JDENT ( .IDNSBND ) = 1
JDENT ( .IDMODE ) = 0
JDENT ( .IDMIN ) = 0
IF(MFLAG) JDENT(.IDMIN)=-180
JDENT ( .IDMAX ) = 2
IF(TFLAG) JDENT(.IDMAX)=11
IF(HFLAG) JDENT(.IDMAX)=12
IF(MFLAG) JDENT(.IDMAX)=360
#
#
CALL COPYDS ( FDO, JDENT, IEV, %9998 )
#
CALL GDSCAX(FDI, .CAXSLOPEFAC, FCTDSC,
            IDENT, FCTLNE, NUMDR, IEV, %9998)
#
IF(FCTLNE =13) GO TO 9000
#
DO I=4,12
  $(
  IF(FCTDSC(I)<=0.) GO TO 9000
  SCALE(I)=1./FCTDSC(I)
  $)
  SIZE=(FCTDSC(1)-1.)*.5
  IF ( - MMSKIP( .TRUE. ) )
    $(
    CALL MMSNDS( NPPL, 1, IEV, %9998 )
    CALL MMSNDS( RADIUS, -1, IEV, %9998 )
    CALL MMSNDS( CRVTHR, -2, IEV, %9998 )
    CALL MMSNDS( ANGTHR, -3, IEV, %9998 )
    DO I = 4, 12
      CALL MMSNDS( SCALE( I ), - I, IEV, %9998 )
    CALL MMSNDS( DEPTH, -13, IEV, %9998 )
    CALL MMSNDS( FRATIO, -14, IEV, %9998 )
    CALL MMSNDS( GRADTH, -15, IEV, %9998 )
    CALL MMSNDS( GTHETA, -16, IEV, %9998 )
    CALL MMSNDS( RVTHET, -17, IEV, %9998 )

```

```
CALL MMSNDS( SIZE, -18, IEV, %9998 )
$)
```

```
#
#
#
```

```
ACTUAL NUMBER CRUNCHING
```

```
DO I = 1,NLIN
  $(
    DO J = 1,9
      CALL RREAD( FDI, IIN(1,J), J+1, I, IDENT,
        .WAIT, IEV, %9998 )
```

```
#
```

```
IF ( MMSKIP( .TRUE. ) )
  DO K = 1,NPPL
    $(
      R2 = FLOAT( IIN(K,1) ) * SCALE(4)
      R3 = FLOAT( IIN(K,2) ) * SCALE(5)
      R4 = FLOAT( IIN(K,3) ) * SCALE(6)
      R5 = FLOAT( IIN(K,4) ) * SCALE(7)
      R6 = FLOAT( IIN(K,5) ) * SCALE(8)
      R7 = FLOAT( IIN(K,6) ) * SCALE(9)
      R8 = FLOAT( IIN(K,7) ) * SCALE(10)
      R9 = FLOAT( IIN(K,8) ) * SCALE(11)
      R10 = FLOAT( IIN(K,9) ) * SCALE(12)
      IOUT(K) = RIDVAL( R2,R3,R4,R5,R6,R7,
        R8,R9,R10,RADIUS,
        CRVTHR,ANGTHR,DEPTH,
        FRATIO,GRADTH,GTHETA,
        RVTHET,SIZE)
```

```
IF (MFLAG)
  $(
```

```
#
#
#
#
#
```

```
IF A COMPLETE TOPOGRAPHIC LABELING
THEN OUTPUT GRADIENT ANGLE ONLY IF
A HILLSIDE
```

```
IF( TFLAG)
  $(
    IF((IOUT(K) >= 7)&(IOUT(K) <= 12))
      ANGLE(K)=GTHETA
    ELSE
      ANGLE(K)=360
    $)
  ELSE
    ANGLE(K)=GTHETA
```

```
#
#
#
#
```

```
OUTPUT CURVATURE ONLY IF
A RIDGE, VALLEY OR SADDLE
```

```
IF( (IOUT(K) >=1) & (IOUT(K) <= 3) )
  $(
    CBAND(K)=RVTHET
```

```

                                $)
                                ELSE CBAND(K)=360
#
                                $)
                                IF(HFLAG&(IOUT(K)>6)&(IOUT(K)<11) )
                                IOUT(K)=12
                                $)
ELSE
$(
CALL MMSNDA( "IIN", IEV, %9998 )
CALL MMEXEC( .WAIT, IEV, %9998 )
CALL MMRCVA( "IOUT", IEV, %9998 )
$)
CALL RWRITE( FDO, IOUT, IBAND, I,
                                JDENT, .WAIT, IEV, %9998)
IF(MFLAG)
$(
CALL RWRITE(FDO, ANGLE, 1, I,
                                JDENT, .WAIT, IEV, %9998)
CALL RWRITE(FDO, CBAND, 2, I,
                                JDENT, .WAIT, IEV, %9998)
$)
$)
CALL CLOSE (FDO)
$)
#
#
CALL CLOSE ( FDI )
#
CALL PPOP
RETURN
#
#
                                ABNORMAL CONDITIONS
#
9000 CONTINUE
#
#
                                BAD VALUE IN DESCRIPTOR RECORD
#
IEV=-5037
9998 CONTINUE
#
#
                                READ OR WRITE ERROR
#
CALL CLOSE ( FDI )
CALL CLOSE ( FDO )
#
RETURN 1
#
END

```

K.6 HDIZER Routine

```
#--HDIZER ZERO CROSSING OF FIRST DIRECTIONAL DERIVATIVE
#
# IDENTIFICATION
#
# TITLE HDIZER
# AUTHOR ROBERT M HARALICK
# VERSION A.01
# DATE 22-MAR-1983
# LANGUAGE RATFOR
# SYSTEM VAX 11/780
#
#
# UPDATE
#
# UPDATE # 1
# AUTHOR ROBERT M .HARALICK
# DATE 10-MAY-1983
# VERSION A.01
# PURPOSE TO ADD THE COMMON ROOT
#
# UPDATE # 2
# AUTHOR ROBERT M. HARALICK
# DATE 05-JUN-1984
# VERSION A.01
# PURPOSE ADD IN THE INFLECTION POINT TO PIXEL
# CENTER AS A CRITERION FOR A GOOD
# RIDGE OR VALLEY.
#
# UPDATE # 3
# AUTHOR ROBERT M. HARALICK
# DATE 06-JUL-1984
# VERSION A.01
# PURPOSE TO ELIMINATE MONOTONIC DATA
# FROM CAUSING FALSE EXTREMA.
#
# UPDATE # 4
# AUTHOR SUI WEI
# DATE 15-AUG-1985
# VERSION A.01
# PURPOSE USE DISTANCE BETWEEN ROOTS
# AS ADDITIONAL CONSTRAIN IN
# RIDGE AND VELLE DETECTION
#
# PURPOSE
#
# GIVEN A CUBIC POLYNOMIAL EXPRESSION
# FOR THE FIRST DIRECTIONAL
# DERIVATIVE, THIS FUNCTION
```

```

#         SUBPROGRAM DETERMINES IF THERE IS
#         A ZERO-CROSSING OF THE FIRST
#         DIRECTIONAL DERIVATIVE WITHIN A
#         GIVEN RADIUS.  THE FUNCTION
#         SUBPROGRAM HDIZER RETURNS A 0 IF
#         THERE IS NONE.  IT RETURNS A 1
#         IF THERE IS A RIDGE AND A 2 IF
#         THERE IS A VALLEY AND A 6 IF A
#         FLAT. RIDGES AND VALLES ARE
#         ONLY DECLARED IF THE INFLECTION
#         POINT IS MORE THAN 1.0 AWAY
#         FROM THE PIXEL CENTER AND IF
#         THE DEPTH RATIO IS GREAT ENOUGH.
#
# ENTRY POINT
#
#         HDIZER(TA1,TA2,TA3,RADIUS,DEPTHR,SIZE)
#
# ARGUMENT LISTING
#
#         TA1      REAL          COEFFICIENT OF X
#         TA2      REAL          COEFFICIENT OF X**2
#         TA3      REAL          COEFFICIENT OF X**3
#         RADIUS   REAL          RADIUS WITHIN WHICH THE
#                               ZERO IS SOUGHT
#         DEPTH   REAL          RELATIVE EXTREMA DEPTH
#                               THRESHOLD
#         SIZE     REAL          RIGHT END POINT OF
#                               NEIGHBORHOOD
#
# ROUTINES CALLED
#
# COMMENT
#
#         THE DEPTH OF A VALLEY IS THE
#         HEIGHT OF WATER THAT COULD FILL
#         THE VALLEY UP TO THW SMALLER OF
#         THE HEIGHT OF THE ENDPOINT OR
#         THE INFLECTION POINT.  THE RANGE
#         IS THE HIGHEST POINT MINUS THE
#         THE LOWEST POINT ON THE DOMAIN
#         OF THE NEIGHBORHOOD INTERVAL.
#         THE DEPTH RATIO IS THE DEPTH/RANGE.
#
#
#*****
#
#
#         FUNCTION HDIZER(TA1,TA2,TA3,RADIUS,DEPTHR,SIZE)
#
#         REAL TA1,TA2,TA3,ROOTL,ROOTS

```



```

ELSE
#
#           DISCRIMINANT IS NEGATIVE; THERE
#           ARE NO REAL ROOTS
#
#           $(
#           NUMRT = 0
#           ROOTL = -9999.
#           ROOTS = -9999.
#           GO TO 1
#           $)
#
#           $)
#
ELSE
#
#           POLYNOMIAL IS NOT QUADRATIC
#
#           $(
#           IF(ABS(A1) > 0.)
#
#           POLYNOMIAL IS FIRST ORDER
#
#           $(
#           NUMRT = 1
#           ROOTL = 9999.
#           ROOTS = -A0/A1
#           $)
#           ELSE
#
#           POLYNOMIAL IS ZERO ORDER
#
#           $(
#           NUMRT = 0
#           ROOTL = -9999.
#           ROOTS = -9999.
#           GO TO 1
#           $)
#           $)
#
#           1
#           CONTINUE
#
#           IF(NUMRT==0) GO TO 10
#
#           BOUND THE ROOTS TO WITHIN THE
#           NEIGHBORHOOD
#
#           IF(ROOTL<--SIZE)ROOTL=-SIZE
#           IF(ROOTL> SIZE)ROOTL= SIZE
#           IF(ROOTS<--SIZE)ROOTS=-SIZE
#           IF(ROOTS> SIZE)ROOTS= SIZE
#
#

```

```

#           IS THE ZERO-CROSSING CLOSE ENOUGH
#
IF(ABS(ROOTS)<RADIUS)
$(
IF(NUMRT == 1)
$(
DEPTHA=999999.
DISEXM=999999.
IF(TA2 > 0. ) HD1ZER=VALLEY
ELSE          HD1ZER=RIDGE
GO TO 20
$)
IF(NUMRT==2)
$(
#
#           THE DISTANCE BETWEEN ROOTS MUST
#           BE GREATER THAN 2/SQRT(3) TO ASSURE
#           THAT MONOTONICALLY INCREASING SAMPLE
#           VALUES SEPARATED BY UNIT INTERVALS
#           DO NOT PRODUCE A POLYNOMIAL WITH
#           RELATIVE EXTREMA.
#           WE USE 1.5 WHICH IS GREATER THAN
#           2/SQRT(3) TO HAVE A MORE CONSERVATIVE
#           TEST.
#
IF(DISEXM < EXMTHR)
$(
HD1ZER=0
GO TO 20
$)
#
#           THERE IS A VALLEY IF THE 2ND DERIVATIVE
#           IS POSITIVE, OTHERWISE IT IS A RIDGE
#
IF ( A2*ROOTS+TA2 > 0. ) HD1ZER=VALLEY
ELSE          HD1ZER= RIDGE
#
#           IS DEPTH GREAT ENOUGH
#
IF(ABS(TA3)>EPS)
$(
#
#           DETERMINE THE INFLECTION POINT
#
XINFLE=-TA2/A2
#
#           FOR A GOOD RIDGE OR VALLEY, THE
#           INFLECTION POINT MUST BE FAR AWAY
#           FROM THE PIXEL CENTER.
#
DIST=ABS(XINFLE)

```

```

IF(DIST< DISTHR)
  $(
    HD1ZER=0
    GO TO 20
  $)

#
#   BOUND THE INFLECTION TO WITHIN
#   THE NEIGHBORHOOD
#

IF(XINFLE<-SIZE)XINFLE=-SIZE
IF(XINFLE>SIZE )XINFLE=SIZE

#
#   DETERMINE THE VALUES AT THE
#   LEFT AND RIGHT ENDPOINTS OF NEIGHBORHOOD
#

VLE=VALUE(-SIZE)
VRE=VALUE(SIZE)
VI=VALUE(XINFLE)
VS =VALUE(ROOTS)
VL =VALUE(ROOTL)

#
#   COMPUTE THE RANGE IN THE NEIGHBORHOOD
#

HIGH=AMAX1(VLE,VRE,VS,VL)
LOW =AMIN1(VLE,VRE,VS,VL)
RANGE=EPS+HIGH-LOW

#
#   COMPUTE THE DEPTH RATIO
#
#
#   IF ROOTS < ROOTL, THEN
#   -SIZE < ROOTS < XINFLE
#
#   IF ROOTS > ROOTL, THEN
#   XINFLE < ROOTS < SIZE
#

IF(ROOTS<ROOTL)
  IF(HD1ZER==VALLEY)
    DEPTHA=(AMIN1(VLE,VI)-VS)/RANGE
  ELSE
    DEPTHA=(VS-AMAX1(VLE,VI))/RANGE
#
ELSE
  IF(HD1ZER==VALLEY)
    DEPTHA=(AMIN1(VRE,VI)-VS)/RANGE
  ELSE
    DEPTHA=(VS-AMAX1(VRE,VI))/RANGE
#

IF( DEPTHA < DEPTHR) HD1ZER=FLAT
$(
ELSE DEPTHA = 999999.

```

```
        $)
        $)
        GO TO 20
#
10      CONTINUE
#
        ROOTS=-SIZE
        ROOTL=SIZE
20      CONTINUE
#
#      CALL PPOP
        RETURN
        END
```

K.7 RIDVAL Mini-MAP Link File

§ GENMAP RIDVAL RIDVAL,RIDNC,HD1ZER

K.8 RIDVAL MAP-driver

```
INCLUDE MAPDRV
DEFINE(IIN,IAR1)
DEFINE(IOUT,IAR2)
DEFINE(NPPL,I1)
DEFINE(RADIUS,R1)
DEFINE(CRVTHR,R2)
DEFINE(ANGTHR,R3)
DEFINE(SCALE(X),R X)
DEFINE(DEPTHR,R13)
DEFINE(FRATIO,R14)
DEFINE(GRADTH,R15)
DEFINE(GTHETA,R16)
DEFINE(RVTHET,R17)
DEFINE(SIZE,R18)
NAME6(.R,.I,.D,.V,.A,.L)
GETARRAYS
IDESC1(1)
IDESC1(2)
REPEAT
  $(
    CALL RIDNC( IIN, IOUT, NPPL * 18, NPPL * 2,
              SCALE( 4 ), SCALE( 5 ), SCALE( 6 ),
              SCALE( 7 ), SCALE( 8 ), SCALE( 9 ),
              SCALE( 10 ), SCALE( 11 ), SCALE( 12 ),
              RADIUS, CRVTHR, ANGTHR, DEPTHR,
              FRATIO, GRADTH, GTHETA, RVTHET, SIZE )
    PAUSE
  $)
END
```

K.9 RIDNC MAP-subroutine

```

INCLUDE MAPSUB
SUBROUTINE RIDNC( IIN, IOUT, NPPL18, NPPL2, R4, R5,
                 R6, R7, R8, R9, R10, R11, R12,
                 RADIUS, CRVTHR, ANGTHR, DEPTH,
                 FRATIO, GRADTH, GTHETA,
                 RVTHET, SIZE )
INTEGER NPPL18, NPPL2, IIN( NPPL18 )
INTEGER IOUT( NPPL2 ), I, CLASSA
INTEGER CLASSB, CLASS, RIDGE, VALLEY, FLAT
REAL R4, R5, R6, R7, R8, R9, R10, R11
REAL R12, RADIUS, CRVTHR, ANGTHR, DEPTH
REAL FRATIO, GRADTH, GTHETA, RVTHET, SIZE
REAL K2, K3, K4, K5, K6, K7, K8, K9, K10
REAL EPS, TH, T, CSA, SNA, CSB, SNB
REAL SRATIO, GRAD, SEE, CEE, RADDEG, XINFLE, X
REAL ROOTL, ROOTS, DEP, DEPTHA, ROOTAS
REAL ROOTAL, CURVEA, CRVEAM, R0, C0, SE, CE
REAL GRADA, GRDCHA, THETA, DEPTHB, ROOTBS
REAL ROOTBL, CURVEB, CRVEBM, GRADB
REAL GRDCHB, ANGLEA, ANGLEB
INTEGER HDIZER
REAL FLOAT, ABS, SQRT, ATAN2, AMIN1, AMAX1
COMMON / ROOT / ROOTL, ROOTS, DEP
DATA EPS / 1.0E-3 /
DATA RADDEG / 57.29577951 /
DATA RIDGE / 1 /
DATA VALLEY / 2 /
DATA FLAT / 6 /
#
DO I = 1, NPPL2, 2
  $(
    K2 = FLOAT( IIN( I ) ) * R4
    K3 = FLOAT( IIN( I + NPPL2 ) ) * R5
    K4 = FLOAT( IIN( I + NPPL2 * 2 ) ) * R6
    K5 = FLOAT( IIN( I + NPPL2 * 3 ) ) * R7
    K6 = FLOAT( IIN( I + NPPL2 * 4 ) ) * R8
    K7 = FLOAT( IIN( I + NPPL2 * 5 ) ) * R9
    K8 = FLOAT( IIN( I + NPPL2 * 6 ) ) * R10
    K9 = FLOAT( IIN( I + NPPL2 * 7 ) ) * R11
    K10 = FLOAT( IIN( I + NPPL2 * 8 ) ) * R12
#
#           DETERMINE FIRST ANGLE WHICH
#           EXTREMIZES SECOND DIRECTIONAL DERIVATIVE
#
    IF(ABS(K5) > EPS)
      $(
        TH=(K6-K4)/K5
        T=1./(ABS(TH)+SQRT(1+TH*TH))

```



```

        IF(TH < 0.) T=-T
        $)
ELSE T=0.
#
CSA=1./SQRT(1+T*T)
SNA=T*CSA
#
CSB=-SNA
SNB=CSA
#
SRATIO = AMIN1(1.,1.5*FRATIO)
GRAD = SQRT(K2*K2+K3*K3)
IF(GRAD < GRADTH) GTHETA=360.
ELSE
    $(
    SEE=K2/GRAD
    CEE=K3/GRAD
    GTHETA=RADDEG*ATAN2(K2,K3)
    $)
#
XINFLE = -999.
ANGLEA = 0.0
ANGLEB = 0.0
IOUT( I ) = 0
IOUT( I + 1 ) = 0
CLASSA = 0
CLASSB = 0
RVTHET=360.
#
#
#
TA1=K2*SNA+K3*CSA
TA2=(K4*SNA+K5*CSA)*SNA+K6*CSA*CSA
TA3=(K7*SNA+K8*CSA)*SNA*SNA +
    (K9*SNA+K10*CSA)*CSA*CSA
#
#           DETERMINE THE LOCATIONS OF THE
#           ZERO-CROSSINGS
#
CLASS=HD1ZER(TA1,TA2,TA3,RADIUS,DEPTHR,SIZE)
IF(CLASS==FLAT)CLASS=0
DEPTHA=DEP
ROOTAS=ROOTS
ROOTAL=ROOTL
IF(CLASS==0)X=0
ELSE           X=ROOTAS
#
CRVEAM = ABS(CURVEA)
#
#           IS THE CURVATURE HIGH ENOUGH
#

```

```

IF(CRVEAM < CRVTHR)GO TO 3000
IF(CLASS==0)GO TO 3000

#
#           CALCULATE THE RATE OF CHANGE OF
#           GRADIENT ANGLE DIRECTION TAKEN
#           IN THE DIRECTION OF (SNA,CSA)
#

R0=ROOTAS*SNA
C0=ROOTAS*CSA

#

SE=K2+(2.0*K4+2.0*K8*C0+3.0*K7*R0)*R0+
    (K5+K9*C0)*C0
CE=K3+(2.0*K6+2.0*K9*R0+3.0*K10*C0)*C0+
    (K5+K8*R0)*R0
GRADA=SQRT(SE*SE+CE*CE)

#

IF(GRADA > GRADTH)
    $(
    SE=SE/GRADA
    CE=CE/GRADA
    $)
ELSE
    $(
    SE=-CSA
    CE=SNA
    $)

#
#

GRDCHA=ABS(-CE*(2.0*K4+6.0*K7*R0+2.0*K8*C0)*
    SNA-CE*(K5+2.0*K8*R0+2.0*
    K9*C0)*CSA+SE*(K5+2.0*K8*R0+2.0*K9*C0)*
    SNA+SE*(2.0*K6+2.0*
    K9*R0+6.0*K10*C0)*CSA)
IF(GRDCHA>=1000.*GRAD) ANGLEA=1000.
ELSE          ANGLEA=GRDCHA/GRAD

#
#
#           CHECK THE DIFFERENCE IN ANGLES
#           JUST ON EITHER SIDE OF THE RIDGE
#           VALLEY CANDIDATE
#

IF(ANGLEA > ANGTHR)
    $(
    CLASSA=CLASS
    IF((CLASSA==RIDGE&SNA<0.)!
        (CLASSA==VALLEY&SNA>=0.))
        $(
        IF ( SNA == 0. ) SNA = EPS
        SNA=-SNA
        CSA=-CSA
        $)
    $)

```

```

        THETAA=RADDEG*ATAN2(SNA,CSA)
        RVTHET=THETAA
        $)
3000  CONTINUE
#
#
        TB1=K2*SNB+K3*CSB
        TB2=(K4*SNB+K5*CSB)*SNB+K6*CSB*CSB
        TB3=(K7*SNB+K8*CSB)*SNB*SNB +
            (K9*SNB+K10*CSB)*CSB*CSB
#
#           DETERMINE THE LOCATIONS OF THE
#           ZERO-CROSSINGS
#
        CLASS=HD1ZER(TB1,TB2,TB3,RADIUS,DEPTHR,SIZE)
        IF(CLASS==FLAT)CLASS=0
        DEPTHB=DEP
        ROOTBS=ROOTS
        ROOTBL=ROOTL
        IF(CLASS==0!CLASS==FLAT)X=0
        ELSE           X=ROOTBS
#
        CURVEB=6.0*TB3*X+2.0*TB2
        CRVEBM = ABS(CURVEB)
#
#           IS THE CURVATURE HIGH ENOUGH
#
        IF(CRVEBM < CRVTHR)GO TO 4000
        IF(CLASS==0)GO TO 4000
#
        R0=ROOTBS*SNB
        C0=ROOTBS*CSB
#
        SE=K2+(2.0*K4+2.0*K8*C0+3.0*K7*R0)*R0+
            (K5+K9*C0)*C0
        CE=K3+(2.0*K6+2.0*K9*R0+3.0*K10*C0)*C0+
            (K5+K8*R0)*R0
        GRADB=SQRT(SE*SE+CE*CE)
#
        IF(GRADB > GRADTH)
            $(
                SE=SE/GRADB
                CE=CE/GRADB
            $)
        ELSE
            $(
                SE=-CSB
                CE=SNB
            $)
#
        GRDCHB=ABS(-CE*(2.0*K4+6.0*K7*R0+2.0*K8*C0)*

```

```

        SNB-CE*(K5+2.0*K8*R0+2.0*
        K9*C0)*CSB+
            SE*(K5+2.0*K8*R0+2.0*K9*C0)*
        SNB+SE*(2.0*K6+2.0*
        K9*R0+6.0*K10*C0)*CSB)
    IF(GRDCHB>=1000*GRAD) ANGLEB=1000.
    ELSE                    ANGLEB=GRDCHB/GRAD

#
#
#           IS THE GRADIENT ANGLE DIFFERENCE
#           GREAT ENOUGH
#

    IF(ANGLEB > ANGTHR)
        $(
        CLASSB=CLASS
        IF((CLASSB==RIDGE & SNB < 0.) !
            (CLASSB==VALLEY & SNB >= 0.))
            $(
            IF ( SNB == 0. ) SNB = EPS
            SNB=-SNB
            CSB=-CSB
            $)
        IF((CLASSA==0)!(CLASSA--=0&CRVEBM>CRVEAM))
            RVTHET=ATAN2(SNB,CSB)*RADDEG
        $)
4000    CONTINUE
    IF ( CLASSA == 0 & CLASSB -= 0 )
        $(
#
#           TEST FOR RADIAL SYMMETRY
#
        IF( (ABS(TA3) < EPS) & (ABS(TB3) < EPS)
            & (ABS(TB2-TA2) < EPS) )
#
#           IT IS RADIALY SYMMETRIC
#
            IOUT( I ) = 0
#
#           IT IS NOT RADIALY SYMMETRIC
#
        ELSE
            IF (ABS(TB2)/(ABS(TA2)+EPS) > FRATIO)
                IOUT( I ) = CLASSB
            ELSE
#
#           IOUT( I ) = 0
#
                IOUT( I ) = 0
#
#
        GO TO 1000
        $)
    IF ( CLASSA -= 0 & CLASSB == 0 )
        $(

```



```

        AMN=AMIN1(ABS(CURVEB),ABS(CURVEA))
        AMX=AMAX1(ABS(CURVEB),ABS(CURVEA))
        IF ( ABS(CURVEB) > ABS(CURVEA) )
            IOUT( I )=CLASSB
        ELSE
            IOUT( I ) = CLASSA
        $)
1000    CONTINUE
        $)
#
#
#
        RETURN
        END

```

NORMAL RETURN

K.10 HDIZER MAP-subroutine

```

INTEGER FUNCTION HDIZER(TA1,TA2,TA3,
                        RADIUS,DEPTH,SIZE)
#
REAL TA1,TA2,TA3,ROOTL,ROOTS
REAL AMIN1,AMAX1,ABS,DIST,DISTHR
REAL SIZE,EPS,VLE,VRE,VI,VS
REAL VL,HIGH,LOW,RANGE,XINFLE,X
REAL VALUE,RADIUS,DEPTH,DEPTH
INTEGER RIDGE,VALLEY,FLAT
COMMON /ROOT/ ROOTL,ROOTS,DEPTH

#
DEFINE(VALUE(X),(TA1+(TA2+TA3*(X))*(X))*(X))
DATA RIDGE,VALLEY,FLAT/1,2,6/
DATA EPS/1.E-3/
DATA DISTHR /1.0/

#
#
CALL PPUSH("HDIZER")
HDIZER=0
DEPTH=-999999.

#
#
#           DETERMINE THE LOCATIONS OF THE
#           ZERO-CROSSINGS OF FIRST DERIVATIVE
#
A2=3.*TA3
A1=2.*TA2
A0=TA1

#
#
#           IS THE POLYNOMIAL QUADRATIC?
#
IF(ABS(A2)>0.)

#
#           IS THE DISCRIMINANT POSITIVE OR
#           NEGATIVE?
#
$(
D=A1*A1-4.*A2*A0
IF(D >= 0.)

#
#           DISCRIMINANT IS POSITIVE; THERE
#           ARE TWO ROOTS
#
$(
NUMRT = 2
IF(A1 > 0.)
ROOTL = (-A1 - SQRT(D))/(2.*A2)
ELSE
ROOTL = (-A1 + SQRT(D))/(2.*A2)

```

```

        IF(ABS(ROOTL) > 0.)
            ROOTS = A0/(A2*ROOTL)
        ELSE
            ROOTS = 0.
        $)
    ELSE
        #
        #         DISCRIMINANT IS NEGATIVE; THERE
        #         ARE NO REAL ROOTS
        #
        $(
            NUMRT = 0
            ROOTL = -9999.
            ROOTS = -9999.
            GO TO 1
        $)
    $)
#
ELSE
#
#         POLYNOMIAL IS NOT QUADRATIC
#
$(
    IF(ABS(A1) > 0.)
#
#         POLYNOMIAL IS FIRST ORDER
#
        $(
            NUMRT = 1
            ROOTL = 9999.
            ROOTS = -A0/A1
        $)
    ELSE
#
#         POLYNOMIAL IS ZERO ORDER
#
        $(
            NUMRT = 0
            ROOTL = -9999.
            ROOTS = -9999.
            GO TO 1
        $)
    $)
#
1    CONTINUE
#
    IF(NUMRT==0) GO TO 10
#
#         BOUND THE ROOTS TO WITHIN THE
#         NEIGHBORHOOD
#

```



```

#           INFLECTION POINT MUST BE FAR AWAY
#           FROM THE PIXEL CENTER.
#

DIST=ABS(XINFLE)
IF(DIST< DISTHR)
  $(
    HDIZER=0
    GO TO 20
  $)

#           BOUND THE INFLECTION TO WITHIN
#           THE NEIGHBORHOOD
#

IF(XINFLE<-SIZE)XINFLE=-SIZE
IF(XINFLE>SIZE )XINFLE=SIZE

#           DETERMINE THE VALUES AT THE
#           LEFT AND RIGHT ENDPOINTS OF NEIGHBORHOOD
#

VLE=VALUE(-SIZE)
VRE=VALUE(SIZE)
VI=VALUE(XINFLE)
VS =VALUE(ROOTS)
VL =VALUE(ROOTL)

#           COMPUTE THE RANGE IN THE NEIGHBORHOOD
#

HIGH=AMAX1(VLE,VRE,VS,VL)
LOW =AMIN1(VLE,VRE,VS,VL)
RANGE=EPS+HIGH-LOW

#           COMPUTE THE DEPTH RATIO
#

IF ROOTS < ROOTL, THEN
  -SIZE < ROOTS < XINFLE

IF ROOTS > ROOTL, THEN
  XINFLE < ROOTS < SIZE

IF(ROOTS<ROOTL)
  IF(HDIZER==VALLEY)
    DEPTH=(AMIN1(VLE,VI)-VS)/RANGE
  ELSE
    DEPTH=(VS-AMAX1(VLE,VI))/RANGE

ELSE
  IF(HDIZER==VALLEY)
    DEPTH=(AMIN1(VRE,VI)-VS)/RANGE
  ELSE
    DEPTH=(VS-AMAX1(VRE,VI))/RANGE

```

```
#           IF( DEPTHA < DEPTHR) HDIZER=FLAT
           $)
           ELSE DEPTHA = 999999.
           $)
$)
GO TO 20

#
10  CONTINUE
#
           ROOTS=-SIZE
           ROOTL=SIZE
20  CONTINUE
#
#  CALL PPOP
           RETURN
           END
```

L APPENDIX L: FACET2 COMMAND

What follows are the files associated with the GIPSY command, FACET2. This includes the following files:

1. GIPSY_DOC:FACET2.DOC - on-line documentation,
2. GIPSY_RUN:FACET2.RUN - a run-file to test the command,
3. GIPSY_LNK:FACET2.LNK - a link file which instructs the system on how to build the command,
4. GIPSY_SRC:RFACT2.RAT - the command driver,
5. GIPSY_SRC:FACET2.RAT - main work routine for the host,
6. GIPSY_SRC:SRESER.RAT - compute mean of square of residual error,
7. GIPSY_SRC:APLBAS.RAT - apply row and column basis vectors,

8. GIPSY_SRC:APBASR.RAT - apply row basis vector to every buffer column,
9. GIPSY_SRC:APBASC.RAT - apply column basis vector to every buffer row,
10. GIPSY_SRC:CONVCO.RAT - convolution for zero-degree basis column vector,
11. GIPSY_SRC:CANTRN.RAT - select transformation matrix elements for canonical form coefficients,
12. GIPSY_SRC:OUTBCT.RAT - output scaled convolution results,
13. GIPSY_SRC:INTSQS.RAT - initialize buffer with row sums,
14. GIPSY_SRC:SQCNV1.RAT - get mean square sums of contiguous pixel sets,

15. GIPSY_SRC:DOP.RAT - generate discrete orthogonal polynomials,
16. GIPSY_MAPLNK:FACET2.MLK - link file for the mini-MAP module,
17. GIPSY_MAPSRC:FACET2.MMR - MAP-driver for the FACET2 command,
18. GIPSY_MAPSRC:MSKSCL.MMR - setup MAP-subroutine for the mini-MAP,
19. GIPSY_MAPSRC:MSKSCL.MMR - MAP-subroutine called by MSKSCL,
20. GIPSY_MAPSRC:APLBAS.MMR - main work MAP-subroutine,
21. GIPSY_MAPSRC:XFACET2. - include file for the FACET2 MAP-driver.

L.1 FACET2 Documentation

*FACET2 Generate central neighborhood facet of any degree

VERSION: A.01 DATE: 11-5-83 CONTACT: Jong Soo Lee

ACTION: This command performs the least square fit over every window whose size is specified by the user by convolving the pixel values in the window with the column basis vectors first and then the row basis vectors next. The entries in the basis vector are the normalized values of the discrete real orthogonal polynomial basis. If I flag is set, the unnormalized integer basis vectors are convolved with. The output is a integer multi-band SIF of fitting parameters scaled. The scale factors are also stored in the descriptor of the output file. The last band of the output is the total square of residual error divided by the number of degrees of freedom. With W flag set, the command performs the least square fit using binomially or Gaussian weighted discrete orthogonal polynomials.

SOURCE: Disk, input SIF

DESTINATION: Disk, output SIF

FLAGS: (B) If set, the fitting parameters are the expansion of the 2-D basis series (tensor product of row and column basis)

(E) If set, the last band is the actual error just scaled. Otherwise, it is LOG2 of the actual error.

(I) If set, the fitting is done by convolving with integer integer orthogonal polynomial basis, otherwise, with normalized discrete real orthogonal polynomial basis.

(K) If set, convolve with the binomially weighted DOP's

(T) If set, the fitting parameters are those of all possible tensor products of the row and column basis vectors.

(W) If set, performs the least square fitting with weighted discrete orthogonal polynomials by Gaussian weight standard deviation as

$$X (c) = \text{EXP} (-c**2 / (2.*a**2))$$

where c is the distance to the center pixel and the user inputs the value a for the standard deviation.

- QUESTIONS: (1) Neighborhood size.
(2) Degree of surface fit.
(3) Coefficient of Gaussian form curves for weighted DOP if W flag is set.
(4) Number of bits for output file.

COMMAND STRING EXAMPLE:

```
FACET2 TEST.SIF > TEST.FC2
Enter odd window of rows (D=5, 1 to 50)--5
Enter odd window of columns (D=5, 1 to 50)--5
Enter maximum degree of surface
    fit over the window(d=3,0 to 4)-3
Enter number of bits for output (D=15, 10 to 20)--15
```

The command performs the least square fit over every 5x5 window to get parameters of surface fit. The surface is cubic. And the the output consists of the bands of parameters l , r , c , $r \times r$, $r \times c$, $c \times c$, $r \times r \times r$, $r \times r \times c$, $r \times c \times c$, $c \times c \times c$ in the order. The number of bits for output is 15, and the last band is LOG2 of the mean square of residual error.

L.2 FACET2 Run-file

```
$ ! TESTING THE COMMAND FACET2.
$ ! FACET2: GENERATES CENTRAL NEIGHBORHOOD FACET
$ !           OF ANY DEGREE.
$ ! CREATE A DIAGONAL BAR SIF
$ !
$           MKBAR FACETIN.SIF
$           100
$           D
$           L
$ !
$ ! RUN FACETIC
$ !
$ !
$           FACET2 FACETOUT.SIF < FACETIN.SIF
$           5 5 3 13
$ !
$ ! EXTRACT FIRST BAND AND CHANGE MODE TO BINARY
$ ! INTEGR FILE
$ !
$           SBIMG FACETFIT.SIF < FACETOUT.SIF
$           1 50 1 50 1 1
$           MODCH FACETFIT.SIF < FACETFIT.SIF(B)
$ !
$ ! RMS BETWEEN INPUT AND OUTPUT SHOULD BE ZERO
$ !
$           RMS TT < FACETFIT.SIF,FACETIN.SIF
$ !
$ ! DELETE TEMPORARY FILES
$ !
$           DELETE
$           FACETIN.SIF
$           FACETOUT.SIF
$           FACETFIT.SIF
$
$           EXIT
```

L.3 FACET2 Host Link File

```
$ GENCMD FACET2 DFACT2,RFACT2,FACET2,PBDFC2,-  
MSKSCL,BASCAN,COMSCL,SRESER,-  
APLBAS,APBASR,APBASC,CONVCO,-  
CANTRN,OUTBCT,INTSQS,SQCNV1,DOP,-  
ROTBAS,WDOP,IDFCT2,IFACT2,-  
IMSKCL,IBASCN,IRESER,IAPBAS,-  
IAPBSR,IAPBSC,ICONCO,IOUTBC,-  
IINSQS,ISQCV1,IDOP,GCDN,-  
IROTBS GIPSY_LIB:LINPACK
```

L.4 RFACT2 Routine

#--RFACT2 DRIVER FOR SURFACE FIT WITH REAL BASIS VECTORS

IDENTIFICATION

TITLE RFACT2
AUTHOR JONG SOO LEE
VERSION A.01
DATE 17-OCT-1983
LANGUAGE RATFOR
SYSTEM VAX 11/780

UPDATE

UPDATE # 1
AUTHOR JONG SOO LEE
DATE 19-APR-1984
VERSION A.01
PURPOSE TO GENERALIZE THE DOP'S
AND GET SCALE FACTORS
OF COEFFICIENTS FOR
ROTATION INVARIANT MOMENT

UPDATE # 2
AUTHOR JONG SOO LEE
DATE 18-JUN-1984
VERSION A.01
PURPOSE TO CONVOLVE WITH THE BINOMIALLY
WEIGHTED DOP'S WITH K FLAG SET

UPDATE # 3
AUTHOR JONG SOO LEE
DATE 29-AUG-1984
VERSION A.01
PURPOSE TO CLEAR THE GAUSSIAN
WEIGHT COEFFICIENT
AS THE STANDARD DEVIATION

UPDATE # 4
AUTHOR ROBERT M. HARALICK
DATE 09-MAR-1985
VERSION A.01
PURPOSE REMOVED RFLAG AND CORRECTED
THE MAXIMUM
VALUE ON IMAGE CALCULATION.

PURPOSE
#

```

# DRIVER FOR SURFACE FITTING OF
# ANY DEGREE USING SEPARATE ORTHO-
# GONAL BASIS POLYNOMIALS OVER
# EACH PIXEL COORDINATE IN ROW AND
# COLUMN. THE OUTPUT IMAGE CONSISTS
# OF BANDS OF COEFFICIENTS
# CORRESPONDING TO 1,R,C,R**2,RC,C**2,R**3...
# IN CANONICAL FORM
# OF POLYNOMIAL. IF BFLAG IS SET,
# THE COEFFICIENTS CORRESPOND TO
# THOSE OF 2-D DISCRETE ORTHOGONAL
# POLYNOMIALS IN THE SAME ORDER.
# IF TFLAG IS SET, COEFFICIENTS OF
# ALL TENSOR PRODUCTS OF ROW BASIS
# COLUMN BASIS VECTORS ARE OBTAINED,
# EITHER IN CANONICAL FORM OR
# SERIES OF 2-D DOP'S. THE LAST
# BAND IS MEAN OF SQUARE OF RESIDUAL
# ERROR, JUST SCALED IF EFLAG IS
# SET, OR SCALED WITH LOG2 OTHERWISE.
#
# ENTRY POINT
#
# RFACT2 ( WORK, IEV, * )
#
# ARGUMENT LISTING
#
# WORK      INT ARRAY      GIPSY WORK ARRAY SPACE
# IEV       INTEGER       INTEGER EVENT VARIABLE
#                               FOR ERROR
# *         ALTRET        ALTERNATE RETURN
#
# INCLUDE FILES/COMMONS
#
# MACA1     INCLUDE      GIPSY GENERAL SYMBOL DEF'NS
# GIPCOM    COMMON       GIPSY COMMAND LINE COMMON DATA
# ERROR     COMMON       ERROR TRACE STACK COMMON DATA
# TTCOM    COMMON       FD'S FOR TERMINAL & RUNFILE IO
#
# LANGUAGE FEATURES REQUIRED
#
# ALTERNATE RETURN IS REQUIRED
#
# ROUTINES CALLED
#
# PPU      PUSH PROGRAM NAME ONTO ERROR STACK
#          (GIPSY PRIMITIVE)
# RDKIN    INITIALIZE AND ACCESS A STANDARD IMAGE
#          FILE. (GIPSY PRIMITIVE)
# CLOSE    CLOSE A FILE. (GIPSY PRIMITIVE)
# WNDWSZ   GET DIMENSIONS OF RECTANGULAR WINDOW

```

```

#           (GIPSY PRIMITIVE)
#       WHBAND      SELECT ONE BAND FROM SIF FILE
#           (GIPSY PRIMITIVE)
#       COMTIN      HAS USER INPUT COMMENT DESCRIPTOR
#           RECORDS (GIPSY PRIMITIVE)
#       RNGETI      GET AN INTEGER WITHIN A RANGE (WITH
#           PROMPT). (GIPSY PRIMITIVE)
#       FACET2      NUMBER CRUNCHER FOR SURFACE FITTING
#           OF ANY DEGREE (USER ROUTINE)
#       PPOP        POP PROGRAM NAME FROM THE ERROR STACK
#           (GIPSY PRIMITIVE)
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
#       INCLUDE MACA1
#       SUBROUTINE RFACT2 ( WORK, IEV, * )
#
#       IMPLICIT INTEGER (A-Z)
#
#       INCLUDE GIPCOM
#       INCLUDE TTCOM
#
#       INTEGER WORK ( .ARB )
#       INTEGER IDENT (.IDLENGTH)
#
#       REAL ALPHA, ALPHDV, ALPHMN
#
#       LOGICAL BFLAG, EFLAG, KFLAG, TFLAG, WFLAG
#
#       EQUIVALENCE (NBITS, IDENT(.IDNBITS))
#       EQUIVALENCE (NUMPPL, IDENT(.IDNPPL))
#       EQUIVALENCE (NLINS, IDENT(.IDNLINS))
#       EQUIVALENCE (NCOLS, IDENT(.IDNCOLS))
#       EQUIVALENCE (NROWS, IDENT(.IDNROWS))
#       EQUIVALENCE (NTBND, IDENT(.IDNBNDS))
#       EQUIVALENCE (NSBND, IDENT(.IDNSBND))
#       EQUIVALENCE (MODE, IDENT(.IDMODE))
#
#       EQUIVALENCE (BFLAG, UFLAG(.B)), (EFLAG, UFLAG(.E))
#       EQUIVALENCE (KFLAG, UFLAG(.K))
#       EQUIVALENCE (TFLAG, UFLAG(.T))
#       EQUIVALENCE (WFLAG, UFLAG(.W))
#
#
#           PUT PROGRAM NAME ON ERROR STACK
#
#
#       CALL PPUSH ("RFACT2")
#       CALL MMINIT( 0 )
#
#
#           GET IDENTIFICATION ARRAY FROM INPUT FILE

```

```

#
CALL RDKINL (FDI1, IDENT, .READONLY, IEV, %9998)
#
CALL CLOSE ( FDI1 )
#
#           CHECK THE INPUT FILE
#
IF ( MODE == 1 & MODE == 0 )
    GO TO 9000
IF ( NUMPPL == NCOLS ! NROWS == 1)
    GO TO 9010
IF ( NTBND - NSBND <= 0)
    GO TO 9020
#
#           GET THE WINDOW SIZE ( NUMBER OF ROWS
#           AND NUMBER OF COLUMNS )
#
REPEAT
    CALL RNGETI ( "Enter odd window number of rows.",
                 3, NLINS, 5, NR, IEV, %9998 )
UNTIL ( NR/2 == (NR-1)/2 )
#
REPEAT
    CALL RNGETI ( "Enter odd window number of columns.",
                 3, NUMPPL, 5, NC, IEV, %9998 )
UNTIL ( NC/2 == (NC-1)/2 )
#
#           GET THE BAND TO USE
#
CALL WHBAND ( FDI1, IDENT, -1, BND, IEV, %9998 )
#
#           ASK USER TO WRITE THE COMMENTS TO THE
#           DESCRIPTOR RECORDS
#
CALL COMTIN ( %9998 )
#
#           DEDUCE MAXIMUM POSSIBLE DEGREE OF
#           FITTING SURFACE FROM CONVOLUTION SIZE
#
MAXDEG = JMIN0 ( NR, NC ) - 1
DDEG = 3
IF ( MAXDEG < DDEG )
    DDEG = MAXDEG
#
#           ASK USER MAXIMUM DEGREE OF FITTING
#           SURFACE WITH DOP'S
#
CALL RNGETI ( "Enter maximum degree of
              surface fit over the window.", 1,
              MAXDEG, DDEG, MAXDEG, IEV, %9998 )
#

```

```

#           GET WEIGHT COEFFICIENT TO DETERMINE
#           WEIGHT VECTOR
#
IF ( WFLAG & -KFLAG )
$(
  MMNRC = JMINO ( NR, NC ) / 2
  ALPHMN = SQRT ( MMNRC * MMNRC / ( 38. * ALOG ( 10. ) ) )
  ALPHDV = 1.5
  IF ( ALPHMN > ALPHDV )
    ALPHDV = ALPHMN
  CALL RNGETR("Enter Gaussian weight standard deviation",
    ALPHMN, .LARGEREAL, ALPHDV, ALPHA, IEV, %9998 )
  $)
#
#           COMPUTE NUMBER OF BANDS FOR OUTPUT
#
MXDG1 = MAXDEG + 1
MXDG11 = MXDG1 + 1
IF ( TFLAG )
  NBBND = MXDG1 * MXDG1
ELSE
$(
  NBBND = 0
  FOR ( N=0 ; N<=MAXDEG ; N=N+1 )
    NBBND = NBBND + N + 1
  $)
NBNDP1 = NBBND + 1
NBNDP3 = NBBND + 3
#
#           DIMENSION OF BUFFER 'PTRNCN' FOR CANONICAL
#           TRANSFORMATION FROM CONVOLUTION WITH BASIS
#
NDIM5 = 1
IF ( BFLAG )
$(
  NDIM5 = 0
#
#           IF TFLAG IS SET, DETERMINE NUMBER OF ENTRIES
#           IN THE BUFFER FOR CANONICAL TRANSFORMATION
#
IF ( TFLAG )
  DO DGR = 1, MXDG1
  DO DGC = 1, MXDG1
  DO I = DGR, MXDG1, 2
  DO J = DGC, MXDG1, 2
  NDIM5 = NDIM5 + 1
#
#           DETERMINE SIZE OF BUFFER FOR CANONICAL
#           TRANSFORMATION
#
ELSE

```

```

DO DGR = 1, MXDG1
$(
  MXDGC = MXDG11 - DGR
  DO DGC = 1, MXDGC
  $(
    MXDGR = MXDG11 - DGC
    DO I = DGR, MXDGR, 2
    $(
      MXDC = MXDG11 - I
      DO J = DGC, MXDC, 2
        NDIM5 = NDIM5 + 1
      $(
    $(
  $(
$(
#
#           GET BUFFER SIZES
#
MAXPPL = NUMPPL + NC
MAXPPL = MAXPPL - 1
NDIM1 = NR * MAXPPL
NDIM2 = MAXPPL * MXDG1
NDIM3 = NUMPPL * NBBND
NDIM4 = NBBND * NBBND
NDIM6 = NR * NUMPPL
NRNC = NR * NC
#
#           ALLOCATE WORK SPACE FOR INDIVIDUAL ARRAY
#
WRKSIZ = 1
#
OPNT   = GETWP ( WRKSIZ, .INTMODE, NBBND )
IOBUF  = GETWPM( WRKSIZ, .INTMODE, MAXPPL, "IOBUF" )
IPT    = GETWPM( WRKSIZ, .INTMODE, NR, "IPT" )
RBASIS = GETWPM( WRKSIZ, .REALMODE, NR*MXDG1, "RBASIS" )
CBASIS = GETWPM( WRKSIZ, .REALMODE, NC*MXDG1, "CBASIS" )
BUFFER = GETWPM( WRKSIZ, .REALMODE, NDIM1, "BUFFER" )
RCONV  = GETWPM( WRKSIZ, .REALMODE, NDIM2, "RCONV" )
RCCONV = GETWPM( WRKSIZ, .REALMODE, NDIM3, "RCCONV" )
SCALE  = GETWPM( WRKSIZ, .REALMODE, NBBND, "SCALE" )
SQSBUF = GETWPM( WRKSIZ, .REALMODE, NDIM6, "SQSBUF" )
SQSSUM = GETWPM( WRKSIZ, .REALMODE, NUMPPL, "SQSSUM" )
MTRXBC = GETWPM( WRKSIZ, .REALMODE, NDIM4, "MTRXBC" )
DMTXBC = GETWP ( WRKSIZ, .DBLPRCMODE, NDIM4 )
MTRXZZ = GETWP ( WRKSIZ, .DBLPRCMODE, NBBND )
CTINDX = GETWPM( WRKSIZ, .INTMODE, NBNDP1, "CTINDX" )
PTRNCN = GETWPM( WRKSIZ, .INTMODE, NDIM5, "PTRNCN" )
FCTDSC = GETWP ( WRKSIZ, .REALMODE, NBNDP3 )
#
IF ( .OK .EQ OSALOC (WRKSIZ) )
  GO TO 9997

```



```

IF ( - BFLAG & - EFLAG & - KFLAG & - TFLAG & - WFLAG )
  CALL MMALOC( WRKSIZ )
#
#           GET NUMBER OF BITS FOR OUTPUT IMAGE
#
MXNBTS = ALOG ( FLOAT ( NR*NC ) ) / ALOG ( 2. ) + 2*NBITS
MXNBTS = MINO ( MXNBTS, .NUMBITPERWORD - 2 )
DFNBTS = MINO ( NBITS + 6 , MXNBTS )
MNNBTS = MINO ( NBITS + 2 , DFNBTS )
CALL RNGETI ( "Enter number of bits for output image.",
             MNNBTS, MXNBTS, DFNBTS, NBITS0, IEV, %9997 )
#
MAXO = 2 ** ( NBITS0 - 1 ) - 1
#
#           CALL NUMBER CRUNCH FOR SURFACE FITTING
#
CALL FACET2 ( FDI1, FD01, IDENT,
             WORK ( OPNT ), WORK ( IOBUF ),
             WORK ( IPT ), WORK ( RBASIS ),
             WORK ( CBASIS ), WORK ( BUFFER ),
             WORK ( RCONV ), WORK ( RCONV ),
             WORK ( SCALE ), WORK ( SQSBUF ),
             WORK ( SQSSUM ), WORK ( MTRXBC ),
             WORK ( DMTXBC ), WORK ( MTRXZZ ),
             WORK ( CTINDX ), WORK ( PTRNCN ),
             WORK ( FCTDSC ), NR, NC, NLINS,
             NUMPPL, MAXPPL, MAXPP1, NRNC, MAXDEG,
             MXDG1, MXDG11, MAXO, BND, NBBND,
             NBNDP1, NBNDP3, NDIM1, NDIM2, NDIM3,
             NDIM4, NDIM5, NDIM6, ALPHA, BFLAG,
             EFLAG, KFLAG, TFLAG, WFLAG, IEV, %9998 )
#
#           NORMAL RETURN
#
CALL MMCLOS( 0 )
CALL PPOP
RETURN
#
#           ABNORMAL RETURN
#
9000  CONTINUE
#           NOT AN INTEGER FILE
IEV = -2012
GO TO 9998
#
#
#
9010  CONTINUE
#           NOT IN LINE FORMAT
IEV = -5001
GO TO 9998

```

```
#
#
#
9020  CONTINUE
#      NO NUMERIC BANDS
IEV = -5018
GO TO 9998
#
#
#
9030  CONTINUE
#      FLAGS ILLEGAL
IEV = -5016
GO TO 9998
#
#
#
9997  CONTINUE
#      READ/WRITE ERROR
IEV = OSGIEV ( IEV )
GO TO 9998
#
9998  CONTINUE
#      ERROR IN SUBPROGRAM
#
CALL MMCLOS( 0 )
CALL CLOSE ( FDI1 )
CALL CLOSE ( FDI1 )
#
RETURN 1
END
```

L.5 FACET2 Routine

```
#--FACET2          NUMBER CRUNCH FOR SURFACE FITTING
#
# IDENTIFICATION
#
#       TITLE          FACET2
#       AUTHOR         JONG SOO LEE
#       VERSION        A.01
#       DATE           17-OCT-1983
#       LANGUAGE       RATFOR
#       SYSTEM         VAX 11/780
#
#
# UPDATE
#
#       UPDATE         # 1
#       AUTHOR         JONG SOO LEE
#       DATE           19-APR-1984
#       VERSION        A.01
#       PURPOSE        TO CONVOLVE WITH GENERALIZED DOP'S
#                     AND GET SCALES OF COEFFICIENTS FOR
#                     ROTATION INVARIANT MOMENT
#
#       UPDATE         # 2
#       AUTHOR         JONG SOO LEE
#       DATE           18-JUN-1984
#       VERSION        A.01
#       PURPOSE        TO CONVOLVE WITH THE BINOMIALLY
#                     WEIGHTED DOP'S WITH K FLAG SET
#
#       UPDATE         # 3
#       AUTHOR         JONG SOO LEE
#       DATE           31-JUL-1984
#       VERSION        A.01
#       PURPOSE        DELETE USELESS COMMENT AND LINES
#
#       UPDATE         # 4
#       AUTHOR         ROBERT M. HARALICK
#       DATE           09-MAR-1985
#       VERSION        A.01
#       PURPOSE        REMOVED THE ARGUMENT RFLAG,
#                     CORRECTED THE ERROR
#                     SCALE CALCULATION AND PUT IN A
#                     SECTION OF CODE WHICH
#                     ALWAYS MAKES THE SCALE FACTORS
#                     COME OUT SO THAT THE EUCLIDEAN
#                     NORM OF THE TERMS OF THE SAME
#                     ORDER IS ROTATIONALLY INVARIANT.
#
```

```

# PURPOSE
#
#     NUMBER CRUNCHER FOR SURFACE
#     FITTING OF ANY DEGREE OVER THE
#     SPECIFIED WINDOW OF EACH PIXEL
#     IN THE INPUT IMAGE. THE OUTPUT
#     CONSISTS OF FITTING COEFFICIENTS
#     WHICH ARE SCALED CONVOLUTION
#     OF PIXEL VALUES IN THE WINDOW
#     WITH SEPARATE ROW AND COLUMN
#     BASIS VECTORS WHICH ARE DISCRETE
#     ORTHOGONAL POLYNOMIAL VALUES
#     AT THE ROW AND COLUMN COORDINATES
#     OF THOSE PIXELS.
#
# ENTRY POINT
#
#     FACET2 ( FDI, FDO, IDENT, OPNT, IOBUF,
#             IPT, RBASIS, CBASIS, BUFFER,
#             RCONV, RCONV, SCALE, SQSBUF,
#             SQSSUM, MTRXBC, DMTXBC,
#             MTRXZZ, CTINDX, PTRNCN, FCTDSC,
#             NR, NC, NLINS, NUMPPL,
#             MAXPPL, MAXPP1, NRNC, MAXDEG,
#             MXDG1, MXDG11, MAXO, BND,
#             NBBND, NBNDP1, NBNDP3, NDIM1,
#             NDIM2, NDIM3, NDIM4, NDIM5,
#             NDIM6, ALPHA, BFLAG, EFLAG,
#             TFLAG, WFLAG, IEV, * )
#
# ARGUMENT LISTING
#
#     FDI      CHAR ARRAY      INPUT FILE DESCRIPTOR ARRAY
#     FDO      CHAR ARRAY      OUTPUT FILE DESCRIPTOR ARRAY
#     IDENT    INT ARRAY        IDENTIFICATION ARRAY
#                                     FOR INPUT FILE
#     OPNT     INT ARRAY        POINTER OF CONVOLUTION
#                                     TO OUTPUT BAND
#     IOBUF    INT ARRAY        BUFFER FOR OUTPUT OF
#                                     EACH LINE
#     IPT      INT ARRAY        POINTER OF INPUT LINES
#                                     IN BUFFER
#     RBASIS   REAL ARRAY       ROW BASIS VECTORS FOR
#                                     CONVOLUTIONS (NR, MXDG1)
#     CBASIS   REAL ARRAY       COLUMN BASIS VECTORS FOR
#                                     CONVOLUTIONS (NC, MXDG1)
#     BUFFER   REAL ARRAY       BUFFER HOLDING INPUT LINES
#                                     (NR, NUMPPL)
#     RCONV    REAL ARRAY       BUFFER HOLDING ROW CONVOLVED
#                                     RESULT OF THE BUFFER 'BUFFER'
#                                     ( MAXPPL, MXDG1 )
#

```

#	RCCONV	REAL ARRAY	BUFEER HOLDING COLUMN
#			CONVOLVED RESULT OF ROW
#			CONV'D RESULT (NUMPPL,NBBND)
#	SCALE	REAL ARRAY	SCALE FACTORS OF CONVOLUTIONS
#			FOR OUTPUT
#	SQSBUF	REAL ARRAY	BUFEER FOR SQUARE SUMS OF
#			ENTRIES IN WINDOW
#	SQSSUM	REAL ARRAY	SQUARE SUM OF ENTRIES
#			IN AN INPUT LINE
#	MTRXBC	REAL ARRAY	TRANSFORM MATRIX TO CANONICAL
#			FORM (NBBND,NBBND)
#	DMTXBC	DBLPRC ARRAY	WORK ARRAY TO GET THE
#			TRANSFORM MATRIX 'MTRXBC'
#	MTRXZZ	DBLPRC ARRAY	WORK ARRAY TO GET 'MTRXBC'
#			IN DGECCO OF LINPACK
#	CTINDX	INTEGER	INDEX ARRAY TO TRANSFORM TO
#			CANONICAL FORM
#	PTRNCN	INTEGER	POINTERS TO NONZERO ENTRIES
#			IN TRANSFORM MATRIX
#	FCTDSC	REAL ARRAY	AUXILIARY INFORMATION TO
#			FILE DESCRIPTOR
#	NR	INTEGER	NUMBER OF ENTRIES IN ROW
#			BASIS VECTOR
#	NC	INTEGER	NUMBER OF ENTRIES IN COLUMN
#			BASIS VECTOR
#	NLINS	INTEGER	NUMBER OF LINES IN THE
#			INPUT IMAGE
#	NUMPPL	INTEGER	NUMBER OF POINTS PER
#			INPUT LINE
#	MAXPPL	INTEGER	AUGMENTED NUMBER OF POINTS
#			PER LINE FOR
#			REMOVING EDGE EFFECT
#	MAXPP1	INTEGER	MAXPPL + 1
#	NRNC	INTEGER	NUMBER OF ENTRIES CONVOLVED
#	MAXDEG	INTEGER	MAXIMUM DEGREE OF
#			FITTING SURFACE
#	MXDG1	INTEGER	MAXDEG + 1
#	MXDG11	INTEGER	MXDG1 + 1
#	MAXO	INTEGER	MAXIMUM VALUE ALLOWED
#			IN OUTPUT IMAGE
#	BND	INTEGER	BAND NUMBER OF INPUT
#			IMAGE TO CONVOLVE
#	NBBND	INTEGER	NUMBER OF CONVOLUTIONS
#			FOR SURFACE FITTING
#	NBNDP1	INTEGER	NBBND + 1
#	NBNDP3	INTEGER	NBBND + 3
#	NDIM1	INTEGER	DIMENSION OF BUFFER 'BUFFER'
#	NDIM2	INTEGER	DIMENSION OF BUFFER 'ROCNV'
#	NDIM3	INTEGER	DIMENSION OF BUFFER 'RCCONV'
#	NDIM4	INTEGER	DIMENSION OF BUFFER 'MTRXBC'
#	NDIM5	INTEGER	DIMENSION OF BUFFER 'PTRNCN'

```

#         NDIM6   INTEGER          DIMENSION OF BUFFER 'SQSBUF'
#         ALPHA   REAL             COEFFICIENT DETERMINING
#                                     GAUSSIAN FORM CURVE
#         BFLAG   LOGICAL          IF SET, OUTPUT BASIS
#                                     SERIES EXPANSION
#         EFLAG   LOGICAL          IF SET, ACTUAL MEAN SQUARE
#                                     OF RESIDUAL ERROR
#                                     IS ON THE LAST BAND
#         TFLAG   LOGICAL          IF SET, ALL POSSIBLE TENSOR
#                                     PRODUCTS OF ROW, COLUMN
#                                     BASIS VECTORS ARE
#                                     CONSIDERED
#         WFLAG   LOGICAL          IF SET, FIT WITH
#                                     WEIGHTED DOP'S
#         IEV     INTEGER          INTEGER EVENT VARIABLE FOR
#                                     ERROR
#         *       ALTRET           ALTERNATE RETURN
#
# INCLUDE FILES/COMMONS
#
#         MACA1   INCLUDE          GIPSY GENERAL SYMBOL DEF'NS
#
# LANGUAGE FEATURES REQUIRED
#
#         ALTERNATE RETURN IS REQUIRED
#
# INTERNAL VARIABLES
#
#         JDENT   INT ARRAY        IDENTIFICATION ARRAY FOR
#                                     OUTPUT FILE
#
# ROUTINES CALLED
#
#         PPUISH  PUSH PROGRAM NAME ONTO ERROR STACK
#                                     (GIPSY PRIMITIVE)
#         CPYIDR  COPY A RANDOM FILE INTO A TEMPORARY
#                                     SEQUENTIAL FILE. (GIPSY PRIMITIVE)
#         DSCNAM  WRITE DESCRIPTOR RECORD -- NAME RECORD.
#                                     (GIPSY PRIMITIVE)
#         PDSCI   PUT DESCRIPTOR RECORD - INTEGER.
#                                     (GIPSY PRIMITIVE)
#         PDSCAX  WRITE AUXILIARY INFORMATION ON FILE
#                                     DESCRIPTOR (GIPSY PRIMITIVE)
#         COPYDS  COPY DESCRIPTOR RECORDS FROM TEMP FILE.
#                                     (GIPSY PRIMITIVE)
#         PBDFC2  COMPUTE POINTER OF EACH CONVOLUTION
#                                     FOR OUTPUT BAND (USER ROUTINE)
#         BASCAN  GET TRANSFORM MATRIX TO CANONICAL FORM
#                                     (USER ROUTINE)
#         CANTRN  GET POINTERS OF NON ZERO ENTRIES IN
#                                     TRANSFORM MATRIX (USER ROUTINE)
#

```



```

#
REAL RBASIS ( NR, MXDGI ), CBASIS ( NC, MXDGI )
REAL BUFFER ( NDIM1 ), RCONV ( NDIM2 ), RCCONV ( NDIM3 )
REAL SCALE ( NBBND ), SQSBUF ( NDIM6 ), SQSSUM ( NUMPPL )
REAL MTRXBC ( NDIM4 ), FCTDSC ( NBNDP3 ), MINSCL(10)
#
DBLPRECISION MTRXZZ ( NBBND ), DMTXBC ( NDIM4 )
MAPINTEGER MAPIS( 8 )
#
REAL ALPHA, ALOG, TMPCON, ERRSCL, FNRNC, FLOAT
REAL WSCLC, WSCLR
#
INTEGER TOTDEG
#
CHARACTER FDI ( .FDLENGTH ), FDO ( .FDLENGTH )
#
LOGICAL BFLAG, EFLAG, KFLAG, TFLAG, WFLAG
#
#           PUT PROGRAM NAME ON ERROR STACK
#
CALL PPUSH ( "FACET2" )
#
#           SET UP IDENTIFICATION ARRAY FOR OUTPUT
#
DO I = 1, .IDLENGTH
  JDENT ( I ) = 0
  JDENT ( .IDMIN ) = - MAXO
  JDENT ( .IDMAX ) = MAXO
  JDENT ( .IDNPPL ) = NUMPPL
  JDENT ( .IDNLINS ) = NLINS
  JDENT ( .IDNCOLS ) = NUMPPL
  JDENT ( .IDNROWS ) = 1
  JDENT ( .IDNBND ) = NBNDP1
  JDENT ( .IDMODE ) = 1
#
#           SET CONSTANTS FOR FAST COMPUTATION OF ERROR
#
FNRNC = FLOAT ( NR * NC )
NUMNBN = NUMPPL * NBBND
#
#           GET ROW AND COLUMN BASIS VECTORS
#           AND DETERMINE SCALE FACTORS FOR THEM
#
MAXI = IDENT ( .IDMAX )
MAXI=MAXO(MAXI,IABS(IDENT(.IDMIN)))
IF ( MMSKIP( .TRUE. ) )
  CALL MSKSCL ( RBASIS, CBASIS, SCALE, BUFFER(1),
              MAXI, MAXO, NR, NC,
              FNRNC, MXDGI, NBBND, ERRSCL,
              BUFFER, NDIM1, ALPHA,
              WSCLC, WSCLR, KFLAG, TFLAG, WFLAG )

```



```

ELSE
  $(
    MAPIS( 1 ) = NR
    MAPIS( 2 ) = NC
    MAPIS( 3 ) = NUMPPL
    MAPIS( 4 ) = NBBND
    MAPIS( 5 ) = MXDG1
    MAPIS( 6 ) = MAXO
    MAPIS( 7 ) = MAXI
    MAPIS( 8 ) = NDIM5
    CALL MMSEND( MAPIS, 8, .MAPINTMODE, .MAPIS, IEV, %9998 )
    CALL MMEXEC( .WAIT, IEV, %9998 )
    CALL MMRCVA( "RBASIS", IEV, %9998 )
    CALL MMRCVA( "CBASIS", IEV, %9998 )
    CALL MMRCVA( "SCALE", IEV, %9998 )
    CALL MMRCVS( ERRSCL, -20, IEV, %9998 )
  $)

#
#           GET TRANSFORM MATRIX FROM BASIS SERIES
#           TO CANONICAL FORM AND RESCALE, AND FIND
#           ENTRIES INVOLVING THE TRANSFORMATION
#
IF ( BFLAG )
  $(
    CALL BASCAN ( OPNT, BUFFER(1), RBASIS, CBASIS,
                  SCALE, MTRXBC, DMTXBC,
                  MTRXZZ, NR, NC, MXDG1,
                  MXDG11, NBBND, NRNC, TFLAG )
    IF ( MMSKIP( .TRUE. ) )
      $(
        CALL CANTRN ( CTINDX, PTRNCN, NBNDP1, MXDG1,
                     MXDG11, NDIM5, TFLAG )
      $)

#
#           NORMALIZE SCALE SO THAT TERMS OF
#           THE SAME DEGREE HAVE THE SAME SCALE
#
#           INITIALIZE THE MINSCL(I) TO HIGH VALUE
#
DO I=1,10
  MINSCL(I)=1.E9

#
#           PUT THE MINIMUM SCALE IN MINSCL
#           FOR ALL TERMS OF THE SAME DEGREE
#
NB=0
DO DGR=1,MXDG1
  $(
    IF(TFLAG) MXDGC=MXDG1
    ELSE      MXDGC=MXDG1-DGR+1
  $)
#

```

```

DO DGC=1,MXDGC
  $(
    NB=NB+1
    TOTDEG=DGR+DGC-2
    IF(TOTDEG <=1 ) NEXT
    MINSCL(TOTDEG)=AMIN1(MINSCL(TOTDEG),SCALE(NB))
  $)
$)

#
#           RESCALE SCALE SO THAT ALL TERMS
#           OF THE SAME DEGREE HAVE THE SAME
#           SCALE
#

NB=0
DO DGR=1,MXDG1
  $(
    IF(TFLAG) MXDGC=MXDG1
    ELSE      MXDGC=MXDG1-DGR+1
    DO DGC=1,MXDGC
      $(
        TOTDEG=DGR+DGC-2
        NB=NB+1
        IF(TOTDEG<=1)NEXT
        SCALE(NB)=MINSCL(TOTDEG)
      $)
    $)

#
#           SCALE FACTOR IS ALWAYS ADJUSTED FOR
#           ROTATION INVARIANT NORMS EXCEPT WHEN
#           TENSOR PRODUCT IS DONE
#

NB = 0
DO DGR = 1, MXDG1
  $(
    IF ( TFLAG )
      MXDGC = MXDG1
    ELSE
      MXDGC = MXDG1 - DGR + 1
    DO DGC = 1, MXDGC
      $(
        NB = NB + 1
        IF ( DGR > 1 & DGC > 1 )
          $(
            MAXNUM = DGC
            MINNUM = DGR
            IF ( DGR > DGC )
              $(
                MAXNUM = DGR
                MINNUM = DGC
              $)
          $)
      $)
    IFCT1 = 1
  $)

```

```

        DO N = MAXNUM, DGR+DGC-2
            IFCT1 = IFCT1 * N
        IFCT2 = 1
        DO N = 2, MINNUM-1
            IFCT2 = IFCT2 * N
        SCALE ( NB ) = SQRT(FLOAT(IFCT2)/FLOAT(IFCT1))
            * SCALE ( NB )
        $)
    $)
$)
ELSE
    $(
        CALL MMSNDA( "SCALE", IEV, %9998 )
        CALL MMEEXEC( .NOWAIT, IEV, %9998 )
        CALL MMSNDA( "MTRXBC", IEV, %9998 )
    $)
$)
#
#         IF EFLAG IS NOT SET, ERROR SCALE IS
#         MODIFIED FOR LOG OF TOTAL RESIDUAL ERROR
#
#
IF( EFLAG)
    $(
        TMPCON = MAXO / ERRSCL + 1.
        ERRSCL = MAXO / ALOG ( TMPCON )
    $)
SCALE ( 1 ) = 1.0
#
#         GET POINTERS TO OUTPUT BANDS IN THE
#         ORDER OF CONVOLUTION
#
CALL PBDFC2 ( OPNT, MXDG1, NBBND, TFLAG )
#
#         OPEN AND COPY INPUT FILE DESCRIPTOR
#
CALL CPYIDR ( FDI, IDENT, .OPNTMP, IEV, %9998 )
#
#         PUT PROGRAM NAME ON FILE DESCRIPTOR
#
CALL DSCNAM ( "FACET2", IEV, %9998 )
#
#
#         PUT PARAMETERS IN FILE DESCRIPTOR
#
CALL PDSCI ( "Number of rows in the window  ",
            NR, IEV, %9998 )
CALL PDSCI ( "Number of columns in the window",
            NC, IEV, %9998 )
CALL PDSCI ( "Maximum degree of surface fit  ",

```

```

                MAXDEG, IEV, %9998 )
#
#           EACH BAND IS AS FOLLOWS
#
IF ( BFLAG )
    CALL PDSCPS ( "Bands in linear combination of basis as",
                IEV, %9998 )
ELSE
    CALL PDSCPS ( "Bands in the canonical form as below",
                IEV, %9998 )
MDEG = MXDG1 + 1
VDEG = 1
NNB = 0
WHILE ( VDEG < MDEG )
    $(
        DO I = 1, VDEG
            $(
                NNB = NNB + 1
                IOBUF ( 1 ) = NNB
                IOBUF ( 2 ) = VDEG - I
                IOBUF ( 3 ) = I - 1
                CALL PDSCIA ( "   *** Band      Row Deg      Col Deg",
                            IOBUF(1), 3, IEV, %9998 )
                DO J = 1, NBBND
                    IF ( OPNT(J) == NNB )
                        BREAK
                CALL PDSCR ( "                          The scale factor is",
                            SCALE(J), IEV, %9998 )
            $)
        VDEG = VDEG + 1
    $)
IF ( TFLAG )
    $(
        VDEG = 1
        VD = MXDG1
        WHILE ( VDEG < MXDG1 )
            $(
                DO I = VDEG, MAXDEG
                    $(
                        NNB = NNB + 1
                        IOBUF ( 1 ) = NNB
                        IOBUF ( 2 ) = VD - I
                        IOBUF ( 3 ) = I
                        CALL PDSCIA ( "   *** Band      Row Deg      Col Deg",
                                    IOBUF(1), 3, IEV, %9998 )
                        DO J = 1, NBBND
                            IF ( OPNT(J) == NNB )
                                BREAK
                        CALL PDSCR ( "                          Its scale factor is",
                                    SCALE(J), IEV, %9998 )
                    $)
            $)
    $)

```

```

        VDEG = VDEG + 1
        VD = VD + 1
        $)
    $)
IF ( EFLAG )
    CALL PDSCI ( "Total residual error /Degrees of Freedom",
                NBNDP1, IEV, %9998 )
ELSE
    CALL PDSCI ( "LOG10 of total residual error band",
                NBNDP1, IEV, %9998 )
CALL PDSCR ( " Its scale factor is",
            ERRSCL*(FNRNC-FLOAT(NBBND)), IEV, %9998 )
#
#           PUT SCALE FACTORS ON DSCRIPTOR RECORDS
#
FCTDSC ( 1 ) = NR
FCTDSC ( 2 ) = NC
FCTDSC ( 3 ) = NBBND
FCTDSC ( 3 ) = NBBND
DO I = 2, NBBND
    FCTDSC ( OPNT(I)+2 ) = SCALE ( I )
FCTDSC ( NBNDP3 ) = ERRSCL*(FNRNC-FLOAT(NBBND))
CALL PDSCAX ( .CAXSLOPEFAC, FCTDSC, NBNDP3, IEV, %9998 )
#
#           CREATE OUTPUT FILE ( FDO ) AND PUT
#           ALL INFORMATION AS FILE DESCRIPTOR
#
CALL COPYDS ( FDO, JDENT, IEV, %9998 )
#
#           SET VARIABLES TO INITIALIZE BUFFER
#
LDIST = NC / 2
RDIST = LDIST
UDIST = NR / 2
DDIST = UDIST
NRM1 = NR - 1
PNT = 1 - MAXPPI
HNC = LDIST
HNCP1 = HNC + 1
NCP1 = NC + 1
HNR = UDIST
HNRP1 = HNR + 1
NRP1 = NR + 1
#
#           INITIALIZE BUFFER
#
DO I = 1, NRM1
    $(
        PNT = PNT + MAXPPI
        CALL AREAD ( FDI, IOBUF, MAXPPL, LDIST, RDIST, UDIST,
                    DDIST, BND, I, IDENT, .WAIT, IEV, %9998 )

```

```

#
#           PUT THE INTEGER LINE INTO REAL BUFFER
#           AND FIRST ONE IS SET ZERO FOR FAST CONVOLUTION
#
      BUFFER ( PNT ) = 0.
      DO J = 1, MAXPPL
        BUFFER ( PNT+J ) = FLOAT ( IOBUF(J) )
      $)

#
#           SET UDIST FOR NEXT LINES TO READ
#
      UDIST = 0

#
#           INITIALIZE SQSBUF FOR SQUARE OF RESIDUAL ERROR
#
      IF ( MMSKIP( .TRUE. ) )
        $(
          CALL INTSQS ( BUFFER, SQSBUF, SQSSUM, NC, FNRNC,
                     NRML, NUMPPL, MAXPPL, NDIM1, NDIM6 )
        $)
      ELSE
        $(
          CALL MMWAIT( 2, IEV, %9998 )
          CALL MMRCVS( ERRSCL, -20, IEV, %9998 )
          CALL MMSNDA( "BUFFER", IEV, %9998 )
          CALL MMEXEC( .WAIT, IEV, %9998 )
          CALL MMRCVA( "SQSBUF", IEV, %9998 )
          CALL MMRCVA( "SQSSUM", IEV, %9998 )
        $)

#
#           INITIALIZE POINTER TO BUFFER
#
      DO I = 1, NR
        IPT ( I ) = I

#
#           READ IN REST OF LINES AND CONVOLVE
#
      DO LN = DDIST+1, NLINS+DDIST
        $(
#
#           POINTER TO 'BUFFER' WHERE INPUT LINE GOES
#           AND OUTPUT LINE NUMBER WHERE CONVOLVED
#           OUTPUT LINE GOES IN OUTPUT IMAGE
#
          LL = IPT ( NR )
          OL = LN - DDIST

#
#           POINTER TO 'BUFFER' WHERE THE FIRST ENTRY OF
#           INPUT LINE GOES
#
          PNT = LL * MAXPPL - MAXPPL

```

```

#
#       READ IN AUGMENTED LINE OF INPUT IMAGE
#
CALL AREAD ( FDI, IOBUF, MAXPPL, LDIST, RDIST, UDIST,
            DDIST, BND, LN, IDENT, .WAIT, IEV, %9998 )
#
#       PUT INPUT LINE INTO REAL BUFFER
#
IF ( MMSKIP( .TRUE. ) )
  $(
  BUFFER ( PNT ) = 0.
  DO J = 1, MAXPPL
    BUFFER ( PNT+J ) = FLOAT ( IOBUF(J) )
#
#       CONVOLVE WITH ROW AND COLUMN BASIS VECTOR
#
CALL APLBAS ( BUFFER, RCONV, RCCONV, RBASIS,
            CBASIS, NR, NC,
            NUMPPL, MAXPPL, MAXPP1, MXDG1, NBBND,
            NDIM1, NDIM2, NDIM3, TFLAG, WFLAG )
  $)
ELSE
  $(
  CALL MMSNDA( "IOBUF", IEV, %9998 )
  CALL MMEXEC( .WAIT, IEV, %9998 )
  CALL MMRCVA( "RCONV", IEV, %9998 )
  CALL MMRCVA( "RCCONV", IEV, %9998 )
  $)
#
#       OUTPUT AFTER TRANSFORMING CONVOLVED RESULTS
#       AND MULTIPLYING WITH SCALE FACTOR
#
CALL OUTBCT ( FDO, JDENT, OPNT, IOBUF(1),
            RCCONV, SCALE, MTRXBC,
            CTINDX, PTRNCN, OL, NUMPPL, NBBND, NBNPD1,
            NDIM3, NDIM4, NDIM5, BFLAG, IEV, %9998 )
#
#       COMPUTE SQUARE SUMS OF NEW INPUT LINE
#       AND ADD THEM TO SQSBUF
#
IF ( MMSKIP( .TRUE. ) )
  $(
  CALL SQCNV1 ( BUFFER(PNT), SQSBUF, SQSSUM,
            NC, NCP1, HNC, HNC1, LL,
            FNRNC, NUMPPL, MAXPP1, NDIM6,
            CBASIS(1,1), WSCLC, WFLAG )
#
#       COMPUTE RESIDUAL ERROR SUM
#
CALL SRESER ( SQSSUM, SQSBUF, RCCONV, RCONV(1),
            NUMPPL, NBBND, FNRNC,

```

```

                                NUMBN, NDIM3, NDIM6, RBASIS(1,1),
                                NR, NRP1, HNR, HNRP1,
                                IPT, WSCLR, WFLAG )
#
    IF ( EFLAG )
        DO COL = 1, NUMPPL
            IOBUF ( COL ) = RCONV ( COL ) * ERRSCL + .5
        ELSE
            DO COL = 1, NUMPPL
                IOBUF ( COL ) = ALOG(RCONV(COL)+1.) *
                    ERRSCL + .5
            $)
        ELSE
            $(
                CALL MMWAIT( 2, IEV, %9998 )
                CALL MMRCVA( "IOBUF", IEV, %9998 )
            $)
        CALL RWRITE ( FDO, IOBUF, NBNDP1, OL,
                    JDENT, .WAIT, IEV, %9998 )
#
#           ROTATE POINTER TO 'BUFFER'
#
    IF ( MMSKIP( .TRUE. ) )
        $(
            CALL ROTPTU ( IPT, NR )
#
#           ROTATE ROW BASIS VECTORS SUCH THAT THEY
#           ARE CONSISTENT WITH POINTER OF 'BUFFER'
#           FOR FAST COMPUTATION
#
            CALL ROTBAS ( RBASIS, NR, MXDG1 )
            $)
#
#           $)
#
#           CLOSE INPUT AND OUTPUT FILE
#
        CALL CLOSE ( FDI )
        CALL CLOSE ( FDO )
#
#           NORMAL RETURN
#
        CALL PPOP
        RETURN
#
#           ABNORMAL RETURN
#
        9998 CONTINUE
        RETURN1
        END

```


L.6 SRESER Routine

```
#--SRESER          COMPUTE MEAN OF SQUARE OF RESIDUAL ERROR
#
# IDENTIFICATION
#
# TITLE            SRESER
# AUTHOR           JONG SOO LEE
# VERSION          A.01
# DATE             04-NOV-1983
# LANGUAGE         RATFOR
# SYSTEM           VAX 11/780
#
#
# UPDATE
#
# UPDATE           # 1
# AUTHOR           JONG SOO LEE
# DATE             19-APR-1984
# VERSION          A.01
# PURPOSE          TO COMPUTE RESIDUAL ERROR OF FITTING
#                  BY WEIGHTED DOP'S.
#
# UPDATE           # 2
# AUTHOR           JONG SOO LEE
# DATE             31-JUL-1984
# VERSION          A.01
# PURPOSE          DELETE USELESS COMMENT LINES
#
# UPDATE           # 3
# AUTHOR           ROBERT M. HARALICK
# DATE             09-MAR-1985
# VERSION          A.01
# PURPOSE          CHANGED THE ERROR CALCULATION TO
#                  TOTAL RESIDUAL ERROR FROM
#                  AVERAGE RESIDUAL ERROR.
#
# UPDATE           # 4
# AUTHOR           ROBERT M. HARALICK
# DATE             24-OCT-1985
# VERSION          A.01
# PURPOSE          DECLARE ABS AS REAL
#
# PURPOSE
#
# COMPUTE MEAN SQUARE OF RESIDUAL
# ERROR FROM SQUARE SUM OF PIXEL
# VALUES CONVOLVED AND RESULTS
# CONVOLVED WITH ROW AND COLUMN
# BASIS VECTORS FOR SURFACE FIT.
```

```

#
# ENTRY POINT
#
#       SRESER ( SQSSUM, SQSBUF, RCCONV, ERRBUF,
#               NUMPPL, NBBND, FNRNC,
#               NUMNBN, NDIM3, NDIM6, WGHTR,
#               NR, NRP1, HNR, HNRP1, IPT,
#               WSCLR, WFLAG )
#
# ARGUMENT LISTING
#
#       SQSSUM  REAL ARRAY      SQUARE SUM OF DATA
#                               VALUES IN WINDOW
#       SQSBUF  REAL ARRAY      COLUMN SQUARE SUMS OF
#                               PIXELS VALUES CONVOLVED
#       RCCONV  REAL ARRAY      CONVOLVED RESULTS WITH
#                               BASIS VECTORS
#       ERRBUF  REAL ARRAY      MEAN SQUARE OF RESIDUAL ERROR
#       NUMPPL  INTEGER         NUMBER OF PIXELS IN A LINE
#       NBBND   INTEGER         NUMBER OF CONVOLUTIONS
#                               FOR SURFACE FIT
#       FNRNC   REAL           NUMBER OF ENTRIES
#                               CONVOLVED IN WINDOW
#       NUMNBN  INTEGER         NUMPPL*NBBND FOR
#                               FAST COMPUTATION
#       NDIM3   INTEGER         DIMENSION OF BUFFER "RCCONV"
#       NDIM6   INTEGER         DIMENSION OF BUFFER 'SQSBUF'
#       WGHTR   REAL ARRAY      SCALED WEIGHT VECTOR
#                               DOWN THE ROW
#       NR      INTEGER         NUMBER OF ROWS IN WINDOW
#       NRP1    INTEGER         NR + 1
#       HNR     INTEGER         NR / 2
#       HNRP1   INTEGER         HNR + 1
#       IPT     INT ARRAY       POINTER ARRAY INDICATING
#                               ORDER OF LINES
#       WSCLR   REAL ARRAY      SCALE OF WEIGHT VECTOR
#       WFLAG   LOGICAL         IF SET, SQUARE SUM
#                               IS WEIGHTED
#
# ROUTINES CALLED
#
#       PPUSH          PUSH PROGRAM NAME ONTO ERROR STACK
#                     (GIPSY PRIMITIVE)
#       PPOP           POP PROGRAM NAME FROM THE ERROR STACK
#                     (GIPSY PRIMITIVE)
#
# *****
#
# SUBROUTINE SRESER ( SQSSUM, SQSBUF, RCCONV, ERRBUF,

```

```

                                NUMPPL, NBBND, FNRNC,
                                NUMNBN, NDIM3, NDIM6, WGHTR,
                                NR, NRP1, HNR, HNRP1,
                                IPT, WSCLR, WFLAG )
#
IMPLICIT INTEGER ( A - Z )
#
INTEGER IPT ( NR )
#
REAL SQSSUM ( NUMPPL ), RCONV ( NDIM3 ), ERRBUF ( NUMPPL )
REAL QSBUF ( NDIM6 ), WGHTR ( NR )
#
REAL SQUERR, COEF, FNRNC
REAL WHTV, WSCLR, ABS
#
LOGICAL WFLAG
#
#           PUT PROGRAM NAME ON ERROR STACK
#
CALL PPUSE ( "SRESER" )
#
#           FROM EACH CONVOLVED RESULT, COMPUTE SQUARE
#           ERROR SUM
#
IF ( WFLAG )
  $(
  DO C = 1, NUMPPL
    SQSSUM ( C ) = 0.
  DO R = 1, HNR
    $(
    WHTV = WGTR ( R )
    CP = IPT ( R ) * NUMPPL - NUMPPL
    DO C = 1, NUMPPL
      $(
      CP = CP + 1
      SQSSUM ( C ) = SQSSUM ( C ) +
                    QSBUF ( CP ) * WHTV
      $)
    CP = IPT ( NRP1-R ) * NUMPPL - NUMPPL
    DO C = 1, NUMPPL
      $(
      CP = CP + 1
      SQSSUM ( C ) = SQSSUM ( C ) +
                    QSBUF ( CP ) * WHTV
      $)
    $)
  CP = IPT ( HNRP1 ) * NUMPPL - NUMPPL
  WHTV = WGTR ( HNRP1 )
  DO C = 1, NUMPPL
    $(
    CP = CP + 1

```

```

SQSSUM ( C ) = SQSSUM ( C ) +
                    SQSBUF ( CP ) * WGH TV
$)
#
#           COMPUTE AVERAGE RESIDUAL ERROR
#
DO COL = 1, NUMPPL
$(
  K = COL + NUMNBN
  SQUERR = 0.
  DO NB = 1, NBBND
    $(
      K = K - NUMPPL
      COEF = RCCONV ( K )
      SQUERR = SQUERR + COEF * COEF
    $)
    ERRBUF ( COL ) = ABS ( SQSSUM ( COL ) * WSCLR
                        - SQUERR )
  $)
$)
ELSE
$(
  DO COL = 1, NUMPPL
    $(
#
#           SET POINTER AND SQUARE ERROR SUM
#
      K = COL + NUMNBN
      SQUERR = 0.
#
#           ERROR SUM IS SQUARE SUM OF CONVOLVED RESULTS
#
      DO NB = 1, NBBND
        $(
          K = K - NUMPPL
          COEF = RCCONV ( K )
          SQUERR = SQUERR + COEF * COEF
        $)
        ERRBUF ( COL ) = ABS ( SQSSUM ( COL ) - SQUERR )
      $)
    $)
#
#           POP UP PROGRAM NAME FROM ERROR STACK
#
CALL PPOP
RETURN
END

```

L.7 APLBAS Routine

#--APLBAS APPLY ROW AND COLUMN BASIS VECTOR TO INPUT BUFFER

#

IDENTIFICATION

#

#	TITLE	APLBAS
#	AUTHOR	JONG SOO LEE
#	VERSION	A.01
#	DATE	15-OCT-1983
#	LANGUAGE	RATFOR
#	SYSTEM	VAX 11/780

#

#

UPDATE

#

#	UPDATE	# 1
#	AUTHOR	JONG SOO LEE
#	DATE	23-MAR-1984
#	VERSION	A.01
#	PURPOSE	FOR CONVOLUTION WITH WEIGHTED BASIS VECTOR

#

#

#	UPDATE	# 2
#	AUTHOR	JONG SOO LEE
#	DATE	19-APR-1984
#	VERSION	A.01
#	PURPOSE	TO CONVOLVE WITH GENERALIZED DOP'S, I.E., WEIGHTED DOP'S.

#

PURPOSE

#

INPUT BUFFER IS, FIRST, CONVOLVED
DOWN THE COLUMN WITH THE ROW
BASIS VECTOR AND THEN THE RESULT
IS CONVOLVED ALONG THE ROW WITH
COLUMN BASIS VECTOR TO GET
SURFACE FITTING COEFFICIENTS.

#

ENTRY POINT

#

APLBAS (BUFFER, RCONV, RCONV, RBASIS,
CBASIS, NR, NC, NUMPPL,
MAXPPL, MAXPP1, MXDG1, NBBND,
NDIM1, NDIM2, NDIM3, TFLAG,
WFLAG)

#

ARGUMENT LISTING

#

BUFFER REAL ARRAY INPUT BUFFER (MAXPPL, NR)

```

#          RCONV   REAL ARRAY      OUTPUT RESULTS CONVOLVED
#
#          RCONV   REAL ARRAY      DOWN THE COLUMN
#                                     ( MAXPPL, MXDG1 )
#          RCONV   REAL ARRAY      CONVOLVED RESULT OF 'RCONV'
#                                     ALONG THE ROW WITH COLUMN
#                                     BASIS ( NUMPPL, NBBND )
#          RBASIS  REAL ARRAY      ROW BASIS VECTORS UP TO
#                                     DEGREE NXDG1 ( NR, MXDG1 )
#          CBASIS  REAL ARRAY      COLUMN BASIS VECTORS UP TO
#                                     DEGREE MXDG1 ( NC, MXDG1 )
#          NR      INTEGER          NUMBER OF ROWS CONVOLVED
#          NC      INTEGER          NUMBER OF COLUMNS CONVOLVED
#          NUMPPL  INTEGER          NUMBER OF POINT PER LINE
#                                     IN INPUT IMAGE
#          MAXPPL  INTEGER          INCREASED NUMBER OF POINTS
#                                     BY REPEATING AT THE
#                                     FIRST AND LAST COLUMNS
#          MAXPPL  INTEGER          MAXPPL + 1
#          MXDG1   INTEGER          MAXIMUM DEGREE OF
#                                     FITTING SURFACE
#          NBBND   INTEGER          NUMBER OF BANDS FOR THE
#                                     COEFFICIENTS OF FITTING
#                                     SURFACE
#          NDIM1   INTEGER          DIMENSION OF BUFFER
#                                     AS LINEAR ARRAY
#          NDIM2   INTEGER          DIMENSION OF RCONV
#                                     AS LINEAR ARRAY
#          NDIM3   INTEGER          DIMENSION OF RCONV
#                                     AS LINEAR ARRAY
#          TFLAG   LOGICAL          IF SET, ALL POSSIBLE
#                                     TENSOR PRODUCTS ARE
#                                     CONVOLVED FOR SURFACE FIT
#          WFLAG   LOGICAL          IF SET, FIT WITH WEIGHTED
#                                     BASIS VECTORS
#
# INTERNAL VARIABLES
#
#          COLST   INTEGER          POINTER TO BUFFER 'BUFFER'
#                                     AND 'RCONV'
#          RCOLST  INTEGER          POINTER TO BUFFER 'RCONV'
#          CCOLST  INTEGER          POINTER TO BUFFER 'RCONV'
#
# ROUTINES CALLED
#
#          PPUSH   PUSH PROGRAM NAME ON THE ERROR STACK
#                                     (GIPSY PRIMITIVE)
#          APBASR  CONVOLVE BUFFER DOWN THE COLUMN WITH
#                                     ROW BASIS VECTOR (USER ROUTINE)
#          APBASC  CONVOLVE RCONV ALONG THE ROW WITH COLUMN
#                                     BASIS VECTOR (USER ROUTINE)
#          CONVCO  CONVOLVE CONSTANT BASIS VECTOR

```

```

#           (USER ROUTINE)
#           PPOP           POP PROGRAM NAME FROM THE ERROR STACK
#                           (GIPSY PRIMITIVE)
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
SUBROUTINE APLBAS ( BUFFER, RCONV, RCCONV, RBASIS,
                  CBASIS, NR, NC,
                  NUMPPL, MAXPPL, MAXPP1, MXDG1, NBBND,
                  NDIM1, NDIM2, NDIM3, TFLAG, WFLAG )
#
IMPLICIT INTEGER ( A - Z )
#
REAL BUFFER ( NDIM1 )
REAL RCONV ( NDIM2 ), RCCONV ( NDIM3 )
REAL RBASIS ( NR, MXDG1 ), CBASIS ( NC, MXDG1 )
#
LOGICAL TFLAG, WFLAG
#
#           PUT PROGRAM NAME ON ERROR STACK
#
CALL PPUSH ( "APLBAS" )
#
#           INITIALIZE POINTER OF BUFFER AND RCONV
#
COLST = 1 - MAXPP1
#
#           FOR EACH DEGREE OF ROW BASIS VECTOR, CONVOLVE
#
DO DEG = 1, MXDG1
  $(
    COLST = COLST + MAXPP1
#
#           CONVOLVE THE INPUT BUFFER DOWN THE COLUMN
#           WITH ROW BASIS VECTOR
#
CALL APBASR ( BUFFER, RCONV(COLST), RBASIS(1,DEG),
             NR, MAXPP1, NDIM1 )
  $)
#
#           INITIALIZE POINTER 'RCOLST' OF 'RCONV'
#           AND POINTER 'CCOLST' OF 'RCCONV'
#
RCOLST = 1 - MAXPP1
CCOLST = 1 - NUMPPL
#
#           FOR EACH DEGREE OF ROW AND COLUMN BASIS
#           VECTORS, CONVOLVE
#

```

```

DO DEGR = 1, MXDG1
  $(
    RCOLST = RCOLST + MAXPPI
  #
  #           FAST CONVOLUTION FOR 0 COLUMN DEGREE
  #
    CCOLST = CCOLST + NUMPPL
    IF ( WFLAG )
      CALL APBASC ( RCONV(RCOLST), RCONV(CCOLST),
                  CBASIS(1,1), NC, NUMPPL, MAXPPI )
    ELSE
      CALL CONVC0 ( RCONV(RCOLST), RCONV(CCOLST),
                  CBASIS(1,1), NC, NUMPPL, MAXPPI )
  #
  #           MAXIMUM DEGREE OF COLUMN COORDINATE
  #
    IF ( TFLAG )
      MXDEGC = MXDG1
    ELSE
      MXDEGC = MXDG1 - DEGR + 1
    DO DEGC = 2, MXDEGC
      $(
        CCOLST = CCOLST + NUMPPL
      #
      #           CONVOLVE ALONG THE ROW WHICH WAS CONVOLVED
      #           ALREADY DOWN THE COLUMN OF INPUT BUFFER
      #
        CALL APBASC ( RCONV(RCOLST), RCONV(CCOLST),
                    CBASIS(1,DEGC), NC, NUMPPL, MAXPPI )
      $)
    $)
  #
  #           POP UP PROGRAM NAME OFF THE ERROR STACK
  #
  CALL PPOP
  #
  RETURN
  END

```


L.8 APBASR Routine

```
#--APBASR          APPLY ROW BASIS VECTOR TO EACH ROW IN BUFFER
#
# IDENTIFICATION
#
#     TITLE          APBASR
#     AUTHOR         JONG SOO LEE
#     VERSION        A.01
#     DATE           15-OCT-1983
#     LANGUAGE       RATFOR
#     SYSTEM         VAX 11/780
#
#
# PURPOSE
#
#     EACH COLUMN IN THE BIFFER
#     'BUFFIN' IS CONVOLVED DOWN THE
#     COLUMN WITH ROW BASIS VECTORS
#     ROTATED CORRESPONDING TO
#     THE ORDER OF LINES IN 'BUFFIN'.
#
# ENTRY POINT
#
#     APBASR ( BUFFIN, BUFOUT, BASIS, NR, MAXPP1, NDIM1 )
#
# ARGUMENT LISTING
#
#     BUFFIN  REAL ARRAY      BUFFER TO BE CONVOLVED
#                                     WITH BASIS VECTOR
#     BUFOUT  REAL ARRAY      OUTPUT OF BUFFIN
#                                     CONVOLVED WITH BASIS
#     BASIS   INT ARRAY       BASIS VECTOR TO
#                                     CONVOLVE WITH BUFFER
#     NR      INTEGER         NUMBER OF ROWS IN THE BUFFER
#     MAXPP1  INTEGER         INCREASED NUMBER OF
#                                     COLUMNS FOR 'BUFOUT'
#     NDIM1   INTEGER         DIMENSION OF BUFFER "BUFFIN"
#
# INTERNAL VARIABLES
#
#     SUM      REAL           INTERMEDIATELY CONVOLVED
#                                     VALUE
#
# ROUTINES CALLED
#
#     PPUSH    PUSH PROGRAM NAME ONTO ERROR STACK
#               (GIPSY PRIMITIVE)
#     PPOP     POP PROGRAM NAME FROM THE ERROR STACK
#               (GIPSY PRIMITIVE)
```

```

#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
SUBROUTINE APBASR ( BUFFIN, BUFOUT, BASIS,
                  NR, MAXPP1, NDIM1 )
#
IMPLICIT INTEGER ( A - Z )
#
REAL BUFFIN ( NDIM1 )
REAL BASIS ( NR ), BUFOUT ( MAXPP1 )
REAL SUM
#
#           PUT PROGRAM NAME ON ERROR STACK
#
CALL PPUSH ( "APBASR" )
#
#           FIRST ENTRY OF 'BUFOUT' IS SET 0 FOR
#           FAST COMPUTATION OF CONVOLUTION WHOSE
#           DEGREE OF COLUMN BASIS VVECTOR IS ZERO.
#
BUFOUT ( 1 ) = 0.
#
#           CONVOLVE EACH ROW IN THE BUFFER 'BUFFIN'
#           WITH ROW BASIS VECTOR
#
DO I = 2, MAXPP1
  $(
  K = I - MAXPP1
  SUM = 0.
  DO J = 1, NR
    $(
    K = K + MAXPP1
    SUM = SUM + BASIS ( J ) * BUFFIN ( K )
    $)
  BUFOUT ( I ) = SUM
  $)
#
#           POP UP ROUTINE NAME FROM ERROR STACK
#
CALL PPOP
#
RETURN
END

```

L.9 APBASC Routine

```
#--APBASC          APPLY COLUMN BASIS VECTOR TO A BUFFER
#
# IDENTIFICATION
#
#     TITLE          APBASC
#     AUTHOR         JONG SOO LEE
#     VERSION        A.01
#     DATE           15-OCT-1983
#     LANGUAGE       RATFOR
#     SYSTEM         VAX 11/780
#
#
# PURPOSE
#
#     THE BUFFER 'BUFFIN' IS CONVOLVED
#     ALONG THE ROW WITH COLUMN
#     BASIS VECTOR CONTIGUOUSLY TO GET
#     SURFACE FITTING COEFFICIENT.
#
# ENTRY POINT
#
#     APBASC ( BUFFIN, BUFOUT, BASIS, NC, NUMPPL, MAXPPI )
#
# ARGUMENT LISTING
#
#     BUFFIN  REAL ARRAY    BUFFER TO BE CONVOLVED
#                                     WITH BASIS VECTOR
#     BUFOUT  REAL ARRAY    OUTPUT OF BUFFIN
#                                     CONVOLVED WITH BASIS
#     BASIS   REAL ARRAY    COLUMN BASIS VECTOR
#                                     TO CONVOLVE WITH
#     NC      INTEGER       NUMBER OF ENTRIES IN
#                                     COLUMN BASIS VECTOR
#     NUMPPL  INTEGER       NUMBER OF ENTRIES IN
#                                     THE BUFFER 'BUFOUT'
#     MAXPPI  INTEGER       NUMBER OF ENTRIES IN
#                                     THE BUFFER 'BUFFIN'
#
# INTERNAL VARIABLES
#
#     SUM     REAL          INTERMEDIATELY CONVOLVED
#                                     VALUE
#
# ROUTINES CALLED
#
#     PPOP          POP PROGRAM NAME FROM THE ERROR STACK
#                   (GIPSY PRIMITIVE)
#     PPUSH         PUSH PROGRAM NAME ONTO ERROR STACK
```

```

#           (GIPSY PRIMITIVE)
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
SUBROUTINE APBASC ( BUFFIN, BUFOUT, BASIS,
                  NC, NUMPPL, MAXPPI )
#
IMPLICIT INTEGER ( A - Z )
#
REAL BUFFIN ( MAXPPI ), BUFOUT ( NUMPPL )
REAL BASIS ( NC )
REAL SUM
#
#           PUT PROGRAM NAME ON ERROR STACK
#
CALL PPUSH ( "APBASC" )
#
#           CONVOLVE EACH POSSIBLE NC COLUMNS IN
#           THE BUFFER 'BUFFIN' WITH COLUMN BASIS
#
DO I = 1, NUMPPL
$(
SUM = 0.
  DO J = 1, NC
    SUM = SUM + BASIS ( J ) * BUFFIN ( I+J )
  BUFOUT ( I ) = SUM
$(
#
#           POP UP ROUTINE NAME FROM ERROR STACK
#
CALL PPOP
#
RETURN
END

```

L.10 CONVCO Routine

#--CONVCO FAST CONVOLUTION WITH 0 DEGREE BASIS COL VECTOR

#

IDENTIFICATION

#

#	TITLE	CONVCO
#	AUTHOR	JONG SOO LEE
#	VERSION	A.01
#	DATE	21-OCT-1983
#	LANGUAGE	RATFOR
#	SYSTEM	VAX 11/780

#

#

PURPOSE

#

TO REDUCE THE COMPLEXITY OF
CONVOLUTION WHEN CONVOLVED WITH ZERO
DEGREE COLUMN BASIS VECTOR BY
ADDING AND SUBTRACTING THE ENTRIES

#

ENTRY POINT

#

CONVCO (BUFFIN, BUFOUT, FVAL, NC, NUMPPL, MAXPPI)

#

ARGUMENT LISTING

#

#	BUFFIN	REAL ARRAY	ROW CONVOLVED RESULTS TO BE CONVOLVED WITH 0 DEGREE COLUMN BASIS VECTOR
#	BUFOUT	REAL ARRAY	COLUMN CONVOLVED RESULT OF 'BUFFIN'
#	FVAL	REAL	COMMON ENTRY OF ZERO DEGREE COLUMN BASIS
#	NC	INTEGER	NUMBER ENTRIES OF COLUMN BASIS VECTOR
#	NUMPPL	INTEGER	DIMENSION OF BUFFER 'BUFOUT'
#	MAXPPI	INTEGER	DIMENSION OF BUFFER 'BUFFIN'

#

INTERNAL VARIABLES

#

#	SUM	REAL	INTERMEDIATELY CONVOLVED VALUE
---	-----	------	-----------------------------------

#

ROUTINES CALLED

#

#	PPUSH	PUSH PROGRAM NAME ONTO ERROR STACK (GIPSY PRIMITIVE)
#	PPOP	POP PROGRAM NAME FROM THE ERROR STACK (GIPSY PRIMITIVE)

#

```

#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
SUBROUTINE CONVCO ( BUFFIN, BUFOUT, FVAL,
                  NC, NUMPPL, MAXPPL )
#
IMPLICIT INTEGER ( A - Z )
#
REAL BUFFIN ( MAXPPL ), BUFOUT ( NUMPPL )
REAL FVAL, SUM
#
#           PUT PROGRAM NAME ON ERROR STACK
#
CALL PPUSH ( "CONVCO" )
#
#           INITIALIZE SUM FOR FAST COMPUTATION
#
SUM = 0.
DO COL = 1, NC
    SUM = SUM + BUFFIN ( COL )
#
#           CONVOLVE EACH POSSIBLE NC COLUMNS IN
#           THE BUFFER 'BUFFIN' BY ADDING NEW ENTRY
#           AND SUBTRACTING LAST ENTRY IN PREVIOUS
#           COLUMN FROM 'SUM'
#
DO COL = 1, NUMPPL
    $(
    PNTNEW = COL + NC
    SUM = SUM + BUFFIN ( PNTNEW ) - BUFFIN ( COL )
    BUFOUT ( COL ) = SUM * FVAL
    $)
#
#           POP UP ROUTINE NAME FROM ERROR STACK
#
CALL PPOP
#
RETURN
END

```

L.11 CANTRN Routine

```
#--CANTRN  GET ENTRIES OF TRANSFORM IN EFFECT FOR CANONICAL
#
# IDENTIFICATION
#
#       TITLE           CANTRN
#       AUTHOR          JONG SOO LEE
#       VERSION         A.01
#       DATE            04-NOV-1983
#       LANGUAGE        RATFOR
#       SYSTEM          VAX 11/780
#
#
# PURPOSE
#
#       SELECT ENTRIES IN TRANSFORMATION
#       MATRIX WHICH ACTUALLY AFFECT
#       TO COMPUTE COEFFICIENTS IN CANONICAL
#       FORM. INDEX ARRAY INDICATE
#       THE FIRST ENTRY IN THE BUFFER
#       WHICH POINTS TO THE FIRST NONZERO
#       ENTRY OF CORRESPONDING ROW IN
#       THE TRANSFORMATION MATRIX.
#
# ENTRY POINT
#
#       CANTRN ( CTINDX, PTRNCN, NBNDP1,
#               MXDG1, MXDG11, NDIM5, TFLAG )
#
# ARGUMENT LISTING
#
#       CTINDX  INT ARRAY      INDEX ARRAY FOR EACH
#                               COEFFICIENT IN CAN-
#                               ONICAL FROM POINTING
#                               TO BUFFER "PTRNCN"
#
#       PTRNCN  INT ARRAY      BUFFER POINTING NON ZERO
#                               ENTRIES IN
#                               TRANSFORMATION MATRIX
#
#       NBNDP1  INTEGER        DIMENSION OF INDEX
#                               ARRAY "CTINDX"
#
#       MXDG1   INTEGER        MAXIMUM DEGREE OF SURFACE FIT
#
#       MXDG11  INTEGER        MXDG1 + 1
#
#       NDIM5   INTEGER        DIMENSION OF BUFFER "PTRNCN"
#
#       TFLAG   LOGICAL        IF SET, ALL POSSIBLE TENSOR
#                               PRODUCTS OF ROW, COLUMN
#                               BASIS VECTORS ARE CONSIDERED
#
#
# ROUTINES CALLED
#
```

```

#          PPUSH          PUSH PROGRAM NAME ONTO ERROR STACK
#                          (GIPSY PRIMITIVE)
#          PPOP           POP PROGRAM NAME FROM THE ERROR STACK
#                          (GIPSY PRIMITIVE)
#
#
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#
#
SUBROUTINE CANTRN ( CTINDX, PTRNCN, NBNDP1, MXDG1,
                  MXDG11, NDIM5, TFLAG )
#
IMPLICIT INTEGER ( A - Z )
#
INTEGER CTINDX ( NBNDP1 ), PTRNCN ( NDIM5 )
#
LOGICAL TFLAG
#
#          PUT PROGRAM NAME ON ERROR STACK
#
CALL PPUSH ( "CANTRN" )
#
#          IN CASE OF TFLAG, ENTRIES IN TRANSFORM MATRIX
#          FOR CANONICAL FORM ARE SELECTED
#
IF ( TFLAG )
  $(
  NB = 0
  NP = 1
  MXDG2 = MXDG1 + MXDG1
  M2 = - MXDG1
  DO DGR = 1, MXDG1
    $(
    M2 = M2 + MXDG1
    DO DGC = 1, MXDG1
      $(
      NB = NB + 1
      CTINDX ( NB ) = NP
      MM = - MXDG2 + M2
      DO I = DGR, MXDG1, 2
        $(
        MM = MM + MXDG2
        DO J = DGC, MXDG1, 2
          $(
          LL = MM + J
          PTRNCN ( NP ) = LL
          NP = NP + 1
          $(
        $(
      $(
    $(
  $(
  $(

```



```

CTINDX ( NBNDP1 ) = NP
$(
#
#           IF TFLAG IS NOT SET, ENTRIES IN TRANSFORM MATRIX
#           ARE SELECTED FOR CANONICAL FORM OF OUTPUT
#
ELSE
$(
NB = 0
NP = 1
DCR1 = MXDG11
PNT1 = - DCR1
DO DGR = 1, MXDG1
$(
PNT1 = PNT1 + DCR1
DCR1 = DCR1 - 1
MXDGC = MXDG11 - DGR
DO DGC = 1, MXDGC
$(
NB = NB + 1
CTINDX ( NB ) = NP
PNT2 = PNT1
DCR2 = DCR1
MXDGR = MXDG11 - DGC
DO I = DGR, MXDGR, 2
$(
MXDC = MXDG11 - I
DO J = DGC, MXDC, 2
$(
LL = PNT2 + J
PTRNCN ( NP ) = LL
NP = NP + 1
$(
PNT2 = PNT2 + DCR2 + DCR2 - 1
DCR2 = DCR2 - 2
$(
$(
$(
CTINDX ( NBNDP1 ) = NP
$(
#
#           NORMAL RETURN
#
CALL PPOP
RETURN
END

```

L.12 OUTBCT Routine

```
#--OUTBCT  OUTPUT CONVOLVED RESULTS SCALED TO INTEGERS
#
#  IDENTIFICATION
#
#      TITLE           OUTBCT
#      AUTHOR          JONG SOO LEE
#      VERSION         A.01
#      DATE            04-NOV-1983
#      LANGUAGE        RATFOR
#      SYSTEM          VAX 11/780
#
#
#  UPDATE
#
#      UPDATE          # 1
#      AUTHOR          ROBERT M. HARALICK
#      DATE            09-MAR-1985
#      VERSION         A.01
#      PURPOSE         MAKE THE CODE A LITTLE MORE READABLE
#
#  PURPOSE
#
#      IF BFLAG IS SET, CONVOLVED RESULTS
#      ARE SCALED, ROUNDED, AND THEN
#      WRITTEN ON OUTPUT FILE. OTHERWISE,
#      THEY ARE MULTIPLIED WITH
#      NONZERO ENTRIES OF TRANSFORMATION
#      MATRIX INDICATED BY THE INDEX
#      BUFFER AND POINTER BUFFER, AND
#      SCALED, ROUNDED, AND WRITTEN ON
#      OUTPUT FILE.
#
#  ENTRY POINT
#
#      OUTBCT ( FDO, JDENT, OPNT, IOBUF,
#              RCCONV, SCALE, MTRXBC, CTINDX,
#              PTRNCN, OL, NUMPPL, NBBND,
#              NBNDP1, NDIM3, NDIM4, NDIM5,
#              BFLAG, IEV, * )
#
#  ARGUMENT LISTING
#
#      FDO      CHAR ARRAY      OUTPUT FILE DESCRIPTOR ARRAY
#      JDENT    INT ARRAY       IDENTIFICATION ARRAY FOR
#                               OUTPUT FILE
#      OPNT     INT ARRAY       POINTER OF CONVOLUTION TO
#                               OUTPUT BAND
#      IOBUF    INT ARRAY       BUFFER FOR OUTPUT ON THE FILE
```

```

#          RCCONV  REAL ARRAY      CONVOLVED RESULTS
#          SCALE   REAL ARRAY      SCALE FCATOR OF CONVOLVED
#                                     RESULTS
#          MTRXBC  REAL ARRAY      TRANSFORMATION MATRIX TO
#                                     CANONICAL FORM (NBBND, NBBND)
#          CTINDX  INT ARRAY       INDEX OF EACH COEFFICIENT
#                                     IN CANONICAL FORM TO POINTER
#                                     BUFFER "PTRNCN" WHERE POINTS
#                                     THE FIRST NONZERO ENTRY
#          PTRNCN  INT ARRAY       POINTERS TO NON ZERO ENTRIES
#                                     IN THE TRANSFORMATION MATRIX
#                                     IN COLUMN ORDER
#          OL      INTEGER         OUTPUT LINE NUMBER
#          NUMPPL  INTEGER         NUMBER OF PIXEL PER A LINE
#          NBBND   INTEGER         NUMBER OF CONVOLUTIONS FOR
#                                     SURFACE FIT
#          NBNDP1  INTEGER         NBBND + 1
#          NDIM3   INTEGER         DIMENSION OF BUFFER "RCCONV"
#          NDIM4   INTEGER         DIMENSION OF BUFFER "MTRXBC"
#          NDIM5   INTEGER         DIMENSION OF BUFFER "PTRNCN"
#          IEV     INTEGER         INTEGER EVENT VARIABLE
#                                     FOR ERROR
#          *       ALTRET          ALTERNATE RETURN
#
#  INCLUDE FILES/COMMONS
#
#          MACA1    INCLUDE        GIPSY GENERAL SYMBOL DEF'NS
#
#  LANGUAGE FEATURES REQUIRED
#
#          ALTERNATE RETURN IS REQUIRED
#
#  ROUTINES CALLED
#
#          PPUSH    PUSH PROGRAM NAME ONTO ERROR STACK
#                  (GIPSY PRIMITIVE)
#          RWRITE   WRITE A BLOCK OF A STANDARD IMAGE FILE
#                  (SIF). (GIPSY PRIMITIVE)
#          PPOP     POP PROGRAM NAME FROM THE ERROR STACK
#                  (GIPSY PRIMITIVE)
#
#
#*****
#
#  INCLUDE MACA1
#  SUBROUTINE OUTBCT ( FDO, JDENT, OPNT, IOBUF,
#                    RCCONV, SCALE, MTRXBC,
#                    CTINDX, PTRNCN, OL,
#                    NUMPPL, NBBND, NBNDP1,
#                    NDIM3, NDIM4, NDIM5, BFLAG, IEV, * )

```

```

#
  IMPLICIT INTEGER ( A - Z )
#
  INTEGER OPNT ( NBBND ), IOBUF ( NUMPPL )
  INTEGER CTINDX ( NBNDP1 ), PTRNCN ( NDIM5 )
  INTEGER JDENT ( .IDLLENGTH )
#
  REAL RCCONV ( NDIM3 ), MTRXBC ( NDIM4 )
  REAL SCALE ( NBBND )
#
  REAL SCL, TMPCON
#
  CHARACTER FDO ( .FDLENGTH )
#
  LOGICAL BFLAG
#
#           PUT PROGRAM NAME ON ERROR STACK
#
  CALL PPUSH ( "OUTBCT" )
#
#           IF BFLAG IS SET, MULTIPLY CONVOLVED RESULT
#           WITH SCALE FACTOR FOR OUTPUT
#
  IF ( BFLAG )
    $(
      BSTART = 0
      DO NB = 1, NBBND
        $(
          SCL = SCALE ( NB )
          DO COL = 1, NUMPPL
            $(
              BSTART = BSTART + 1
              TMPCON = RCCONV ( BSTART ) * SCL
              IF ( TMPCON > 0. )
                IOBUF ( COL ) = TMPCON + .5
              ELSE
                IOBUF ( COL ) = TMPCON - .5
            $)
          OBND = OPNT ( NB )
          CALL RWRITE ( FDO, IOBUF, OBND, OL,
                      JDENT, .WAIT, IEV, %9998 )
        $)
      $)
#
#           FOR OUTPUT OF CANONICAL FORM, CONVOLVED
#           RESULTS ARE MULTIPLIED WITH PROPER ENTRIES
#           OF CORRESPONDING ROW OF TRANSFORM MATRIX
#
  ELSE
    $(
      TSTART = 0

```

```

DO NB = 1, NBBND
  $(
  IF ( MMSKIP( .TRUE. ) )
  $(
  FP = CTINDX ( NB )
  LP = CTINDX ( NB+1 )-1
  SCL = SCALE ( NB )
  DO COL = 1, NUMPPL
    $(
    TMPCON = 0.
    DO CT = FP, LP
      $(
      CP = PTRNCN ( CT )
      TMPCON = TMPCON + MTRXBC(TSTART+CP) *
        RCCONV(COL+(CP-1)*NUMPPL)
      $)
    TMPCON = TMPCON * SCL
    IF ( TMPCON > 0. )
      IOBUF ( COL ) = TMPCON + .5
    ELSE
      IOBUF ( COL ) = TMPCON - .5
    $)
  $)
  ELSE
    $(
    CALL MMWAIT( 2, IEV, %9998 )
    CALL MMRCVA( "IOBUF", IEV, %9998 )
    CALL MMEEXEC( .NOWAIT, IEV, %9998 )
    $)
  OBND = OPNT ( NB )
  CALL RWRITE ( FDO, IOBUF, OBND, OL,
    JDENT, .WAIT, IEV, %9998 )
  TSTART = TSTART + NBBND
  $)
  $)
#
#          NORMAL RETURN
#
CALL PPOP
RETURN
#
#          ABNORMAL RETURN
#
9998 CONTINUE
RETURN1
END

```

L.13 INTSQS Routine

```
#--INTSQS INITIALIZE BUFFER FOR SQUARE SUMS OF INPUT BUFFER
#
# IDENTIFICATION
#
# TITLE INTSQS
# AUTHOR JONG SOO LEE
# VERSION A.01
# DATE 23-OCT-1983
# LANGUAGE RATFOR
# SYSTEM VAX 11/780
#
#
# UPDATE
#
# UPDATE # 1
# AUTHOR JONG SOO LEE
# DATE 19-APR-1984
# VERSION A.01
# PURPOSE TO INITIALIZE BUFFER FOR
# RESIDUAL ERROR OF
# FITTING WITH WEIGHTED DOP'S.
#
# UPDATE # 2
# AUTHOR ROBERT M. HARALICK
# DATE 24-OCT-1985
# VERSION A.01
# PURPOSE TO PRIME THE LAST ROW
# OF SQSBUF WITH ZERO
#
# PURPOSE
#
# TO INITIALIZE BUFFER WITH ROW SUMS FOR FAST
# COMPUTATION OF SQUARE SUMS OF INPUT BUFFER.
#
# ENTRY POINT
#
# INTSQS ( BUFFER, SQSBUF, SQSSUM, NC, FNRNC,
# NRMI, NUMPPL, MAXPPL, NDIM1, NDIM6 )
#
# ARGUMENT LISTING
#
# BUFFER REAL ARRAY INPUT BUFFER FOR SQUARE
# SUMS (MAXPPL,NR)
# SQSBUF REAL ARRAY ROW SQUARE SUMS OF 'BUFFER'
# SQSSUM REAL ARRAY SQUARE SUM OF DATA VALUES
# IN THE WINDOW
# NC INTEGER NUMBER OF COLUMNS IN
# 'BUFFER' TO BE SQUARED
```

```

#          FNRNC   REAL           NUMBER OF ENTRIES CONVOLVED
#          NRM1    INTEGER        NUMBER OF LINES IN 'BUFFER'
#                                     SQUARED - 1
#          NUMPPL  INTEGER        DIMENSION FOR BUFFER 'SQSBUF'
#          MAXPP1  INTEGER        INCREASE BUFFER SIZE TO
#                                     READ IN A LINE
#          NDIM1   INTEGER        DIMENSION FOR BUFFER 'BUFFER'
#          NDIM6   INTEGER        DIMENSION OF BUFFR 'SQSBUF'
#
# INTERNAL VARIABLES
#
#          SQSUM   REAL           INTERMEDIATE SQUARE SUM
#          FVAL    REAL           AN ENTRY OF 'BUFFER' SQUARED
#
# ROUTINES CALLED
#
#          PPUSH   PUSH PROGRAM NAME ONTO ERROR STACK
#                  (GIPSY PRIMITIVE)
#          PPOP    POP PROGRAM NAME FROM THE ERROR STACK
#                  (GIPSY PRIMITIVE)
#
#
#*****
#
#          SUBROUTINE INTSQS ( BUFFER, SQSBUF, SQSSUM, NC,
#                             FNRNC, NRM1, NUMPPL, MAXPP1,
#                             NDIM1, NDIM6 )
#
#          IMPLICIT INTEGER ( A - Z )
#
#          REAL BUFFER ( NDIM1 ), SQSBUF ( NDIM6 )
#          REAL SQSSUM ( NUMPPL )
#
#          REAL SQSUM, FVAL, FNRNC
#
#          PUT PROGRAM NAME ON ERROR STACK
#
#          CALL PPUSH ( "INTSQS" )
#
#          SET POINTER OF ROW SUM OUTPUT
#
#          NCM1 = NC - 1
#          PNT1 = 1 - MAXPP1
#          PNT2 = - NUMPPL
#
#          DO I = 1, NRM1
#             $(
#                PNT1 = PNT1 + MAXPP1
#                PNT2 = PNT2 + NUMPPL
#                DO J = 1, NUMPPL

```

```

        $(
        SQSUM = 0.
        DO K = J, J+NCM1
            $(
            FVAL = BUFFER ( PNT1+K )
            SQSUM = SQSUM + FVAL * FVAL
            $)
        SQSBUF ( PNT2+J ) = SQSUM
        $)
    $)
#
#                                     PRIME THE LAST ROW OF SQSBUF
WITH ZERO
#
        PNT2=PNT2+NUMPPL
        DO J=1,NUMPPL
            SQSBUF(PNT2+J)=0
#
#             INITIALIZE SQSSUM
#
        DO I = 1, NUMPPL
            $(
            PNT1 = I - NUMPPL
            SQSUM = 0.
            DO J = 1, NRM1
                $(
                PNT1 = PNT1 + NUMPPL
                SQSUM = SQSUM + SQSBUF ( PNT1 )
                $)
            SQSSUM ( I ) = SQSUM
            $)
#
#             POP UP PRGRAM NAME FROM ERROR STACK
#
        CALL PPOP
#
        RETURN
        END

```


L.14 SQCNV1 Routine

#--SQCNV1 GET MEAN SQUARE SUM OF FIXED SIZE FROM INPUT LINE

IDENTIFICATION

TITLE SQCNV1
AUTHOR JONG SOO LEE
VERSION A.01
DATE 04-NOV-1983
LANGUAGE RATFOR
SYSTEM VAX 11/780

UPDATE

UPDATE # 1
AUTHOR JONG SOO LEE
DATE 19-APR-1984
VERSION A.01
PURPOSE TO GET SQUARE SUM OF
DATA VALUES WEIGHTED FOR
FITTING ERROR BY WEIGHTED DOP'S.

UPDATE # 2
AUTHOR JONG SOO LEE
DATE 31-JUL-1984
VERSION A.01
PURPOSE SWITCH TWO LINES WHICH
CAUSED TROUBLE

UPDATE # 3
AUTHOR JONG SOO LEE
DATE 31-AUG-1984
VERSION A.01
PURPOSE CORRECTION OF A MISTAKE

PURPOSE

GET MEAN SQUARE SUM OF EVERY
POSSIBLE CONTIGUOUS ENTRIES OF
FIXED NUMBER FROM AN INPUT LINE.
FOR FAST COMPUTATION, NEW ENTRY
IS ADDED TO AND LAST ENTRY IS
SUBTRACTED FROM THE PREVIOUS SUM.

ENTRY POINT

SQCNV1 (BUFFIN, SQSBUF, SQSSUM, NC,
NCPI, HNC, HNCPI, LL,

```

#           FNRNC, NUMPPL, MAXPPL, NDIM6,
#           WGHTC, WSCLC, WFLAG )
#
# ARGUMENT LISTING
#
#       BUFFIN REAL ARRAY      INPUT LINE BUFFER
#       SQSBUF REAL ARRAY      COLUMN SQUARE SUM OF DATA
#                               VALUES IN WINDOW
#       SQSSUM REAL ARRAY      SQUARE SUM OF DATA
#                               VALUES IN WINDOW
#       NC      INTEGER         NUMBER OF CONTIGUOUS
#                               ENTRIES TO BE SQUARED
#       NCP1    INTEGER         NC + 1
#       HNC     INTEGER         NC / 2
#       HNCPL   INTEGER         HNC + 1
#       LL      INTEGER         POINTER INDICATING
#                               CURRENT LINE
#       FNRNC   INTEGER         NUMBER OF ENTRIES CONVOLVED
#       NUMPPL  INTEGER         NUMBER OF PIXELS IN A LINE
#       MAXPPL  INTEGER         DIMENSION OF BUFFER "BUFFIN"
#       NDIM6   INTEGER         DIMENSION OF BUFFER 'SQSBUF'
#       WGHTC   REAL ARRAY      COLUMN WEIGHT VECTOR SCALED
#       WSCLC   REAL           SCALE OF WEIGHT
#                               VECTOR 'WGHTC'
#       WFLAG   LOGICAL        IF SET, SUMMING THE
#                               SQUARES WEIGHTED
#
# ROUTINES CALLED
#
#       PPUSH      PUSH PROGRAM NAME ONTO ERROR STACK
#                 (GIPSY PRIMITIVE)
#       PPOP       POP PROGRAM NAME FROM THE ERROR STACK
#                 (GIPSY PRIMITIVE)
#
# *****
#
#       SUBROUTINE SQCNV1 ( BUFFIN, SQSBUF, SQSSUM, NC,
#                           NCP1, HNC, HNCPL, LL,
#                           FNRNC, NUMPPL, MAXPPL, NDIM6,
#                           WGHTC, WSCLC, WFLAG )
#
#       IMPLICIT INTEGER ( A - Z )
#
#       REAL BUFFIN ( MAXPPL ), SQSBUF ( NDIM6 ), SQSSUM ( NUMPPL )
#       REAL WGHTC ( NC )
#       REAL SUM, FVAL, FVAL2, FNRNC
#       REAL WSCLC
#
#       LOGICAL WFLAG

```

```

#
#           PUT PROGRAM NAME ON ERROR STACK
#
CALL PPUSH ( "SQCNV1" )
#
#           INITIALIZE SUM FOR FAST COMPUTATION
#
PNT = ( LL - 1 ) * NUMPPL
DO COL = 1, NUMPPL
  $(
  SUM = 0.
  IF ( WFLAG )
    $(
    DO C = 1, HNC
      $(
      FVAL = BUFFIN ( COL+C )
      FVAL2 = BUFFIN ( COL+NCPI-C )
      SUM = SUM + ( FVAL * FVAL + FVAL2 * FVAL2 )
                * WGHTC ( C )
      $)
    FVAL = BUFFIN ( COL+HNCPI )
    SUM = SUM + FVAL * FVAL * WGHTC ( HNCPI ) * WSCLC
    $)
  ELSE
    DO C = COL+1, COL+NC
      $(
      FVAL = BUFFIN ( C )
      SUM = SUM + FVAL * FVAL
      $)
    PNTC = PNT + COL
    SQSSUM ( COL ) = SQSSUM ( COL ) - SQSBUF ( PNTC ) + SUM
    SQSBUF ( PNTC ) = SUM
    $)
#
#           POP UP ROUTINE NAME FROM ERROR STACK
#
CALL PPOP
#
RETURN
END

```

L.15 DOP Routine

```
#--DOP          DISCRETE ORTHOGONAL POLYNOMIAL GENERATION
#
# IDENTIFICATION
#
#     TITLE          DOP
#     AUTHOR         ROBERT M. HARALICK
#     VERSION        A.01
#     DATE           26-MAR-1983
#     LANGUAGE       FORTRAN
#     SYSTEM         VAX 11/780
#
#
# UPDATE
#
#     UPDATE         # 1
#     AUTHOR         JONG SOO LEE
#     DATE           19-JAN-1984
#     VERSION        A.01
#     PURPOSE        THE INTERNAL VARIABLE D1 IS
#                   NOT USED FOR GENERATING THE
#                   DISCRETE ORTHOGONAL BASIS
#                   VECTORS. SO, IT IS DELETED.
#
# PURPOSE
#
#     THIS SUBROUTINE INPUTS THE
#     NUMBER OF DATAPOINTS AND THE
#     NUMBER OF POLYNOMIALS TO
#     GENERATE. IT RETURNS THE DISCRETELY
#     ORTHOGONAL POLYNOMIALS AS COLUMN
#     VECTORS IN A MATRIX.
#
# ENTRY POINT
#
#     DOP(POLY,N,MORDER)
#
# ARGUMENT LISTING
#
#     POLY    REAL          MATRIX WHOSE COLUMNS
#                   CONTAIN THE DISCRETELY
#                   ORTHOGONAL POLYNOMIALS
#
#     N       INTEGER      DIMENSION OF THE DATA VECTORS
#
#     MORDER  INTEGER      NUMBER OF POLYNOMIALS TO
#                   GENERATE
#
# ALGORITHM
#
#     FORSYTHE, G.E. "GENERATION AND
```

```

#       USE OF ORTHOGONAL POLYNOMIALS
#       FOR DATA FITTING WITH A DIGITAL
#       COMPUTER" JOURNAL SOCIETY
#       OF INDUSTRIAL AND APPLIED
#       MATHEMATICS, VOL 5, P74-88, 1957.
#
#     ROUTINES CALLED
#
#       NONE
#
#*****
#
#     SUBROUTINE DOP(POLY,N,NORDER)
#
#     REAL POLY(N,NORDER)
#
#         PRODUCE POLYNOMIAL OF DEGREE 0
#
#     A=1./SQRT(FLOAT(N))
#     DO I=1,N
#       POLY(I,1)=A
#
#         PRODUCE POLYNOMIAL OF DEGREE 1
#
#     SS=0.
#     DO I=1,N
#       $(
#         POLY(I,2)=(FLOAT(I)-.5*FLOAT(N+1))
#         SS=SS+POLY(I,2)*POLY(I,2)
#       $)
#
#     SS=1./SQRT(SS)
#     DO I=1,N
#       POLY(I,2)=POLY(I,2)*SS
#
#         PRODUCE REMAINING POLYNOMIALS
#
#     IF(NORDER.LT. 3) RETURN
#     DO K=3,NORDER
#       $(
#         D2=0.
#         DO I=1,N
#           $(
#             A=FLOAT(I)-.5*FLOAT(N+1)
#             SS=A*POLY(I,K-1)
#             D2=D2+SS*POLY(I,K-2)
#           $)
#
#         SS=0.

```

```

DO I=1,N
  $(
    A=FLOAT(I)-.5*FLOAT(N+1)
    POLY(I,K)=-D2*POLY(I,K-2)+A*POLY(I,K-1)
    SS=SS+POLY(I,K)*POLY(I,K)
  $)
#
  SS=1./SQRT(SS)
  DO I=1,N
    POLY(I,K)=POLY(I,K)*SS
#
  $)
#
RETURN
END

```

L.16 ROTBAS Routine

```

#--ROTBAS          ROTATE ROW BASIS VECTORS
#
# IDENTIFICATION
#
#       TITLE          ROTBAS
#       AUTHOR         JONG SOO LEE
#       VERSION        A.01
#       DATE           04-NOV-1983
#       LANGUAGE       RATFOR
#       SYSTEM         VAX 11/780
#
#
# PURPOSE
#
#       ROTATE BASIS VECTOR SUCH THAT
#       THEY ARE IN THE ORDER OF INPUT
#       LINE BUFFER FOR FAST COMPUTATION.
#
# ENTRY POINT
#
#       ROTBAS ( BASIS, NR, MXDG1 )
#
# ARGUMENT LISTING
#
#       BASIS   REAL ARRAY   BASIS VECTORS (NR,MXDG1)
#       NR      INTEGER      NUMBER OF ENTRIES IN A VECTOR
#       MXDG1   INTEGER      NUMBER OF VECTORS FOR
#                               CONVOLUTION
#
# ROUTINES CALLED
#
#       PPU      PUSH PROGRAM NAME ONTO ERROR STACK
#               (GIPSY PRIMITIVE)
#       PPOP     POP PROGRAM NAME FROM THE ERROR STACK
#               (GIPSY PRIMITIVE)
#
#
#*****
#
# SUBROUTINE ROTBAS ( BASIS, NR, MXDG1 )
#
# REAL BASIS ( NR, MXDG1 )
# REAL TEMP
#
#       PUT PROGRAM NAME ON ERROR STACK
#
# CALL PPU ( "ROTBAS" )

```

```

#
#           IF NLines IS LESS THAN TWO NO ROTATION CAN
#           BE DONE.
#
IF ( NR < 2 ) RETURN
#
#           ROTATE EACH BASIS VECTORS
#
DO DG = 1, MXDG1
  $(
    TEMP = BASIS ( NR, DG )
    DO I = NR, 2, -1
      BASIS ( I, DG ) = BASIS ( I-1, DG )
    BASIS ( 1, DG ) = TEMP
  $)
#
#           POP UP PROGRAM NAME FROM ERROR STACK
#
CALL PPOP
RETURN
#
END

```


L.17 FACET2 Mini-MAP Link File

\$ GENMAP FACET2 FACET2,MSKSCL,DOP,APLBAS

L.18 FACET2 MAP-driver

```

INCLUDE MAPDRV
DEFINE(NR,I1)
DEFINE(NC,I2)
DEFINE(NUMPPL,I3)
DEFINE(NBBND,I4)
DEFINE(MXDG1,I5)
DEFINE(MAX0,I6)
DEFINE(MAXI,I7)
DEFINE(NDIM5,I8)
INTEGER NDIM1, NDIM6, NDIM2, PNT, LL
REAL FLOAT
INCLUDE XFACET2
I20 = NR * MXDG1
I19 = NC * MXDG1
I18 = NR * NC
I17 = NBBND + 1
I16 = MXDG1 + 1
NDIM1 = NR * ( NUMPPL + NC )
NDIM6 = NR * NUMPPL
I15 = NR - 1
I14 = NUMPPL + NC
I13 = NR * 2
R19 = FLOAT( I18 )
CALL MSKSCL( RBASIS, CBASIS, BUFFER, SCALE, I20, I19, I18,
            NBBND, NR, MXDG1, NC, MAXI, MAX0, R20,
            CTINDX, PTRNCN, I17, I16, NDIM5,
            SQSBUF, SQSSUM, R19, I15, NUMPPL, I14, NDIM1,
            NDIM6, IPT, I13 )
NDIM2 = MXDG1 * I14
I12 = NUMPPL * 2
I11 = NC - 1
I10 = NUMPPL * NBBND
I9 = NBBND * NBBND
PAUSE
REPEAT
  $(
    CALL APLBAS( BUFFER, RCONV, RCONV, RBASIS,
                CBASIS, NR, NC, NUMPPL, PNT, LL,
                I14 - 1, I14, MXDG1, NBBND, NDIM1,
                NDIM2, I20, I19, IOBUF, IPT, I13,
                SCALE, MTRXBC, CTINDX, PTRNCN,
                I17, I10, I9, NDIM5,
                SQSBUF, SQSSUM,
                I10, NDIM6, R20, I12, I11 )
    PAUSE
  $)
RETURN
END

```

L.19 MSKSCL MAP-subroutine

```

INCLUDE MAPSUB
SUBROUTINE MSKSCL( RBASIS, CBASIS, BUFFER, SCALE,
                  NRMDG1, NCMDG1, NRXNC, NBBND,
                  NR, MXDG1, NC, MAXI, MAXO, ERRSCL,
                  CTINDX, PTRNCN, NBNDP1, MXDG11, NDIM5,
                  SQSBUF, SQSSUM, FNRNC, NRM1, NUPPPL,
                  MAXPP1, NDIM1, NDIM6, IPT, NRX2 )
#
  IMPLICIT INTEGER ( A - Z )
#
  REAL RBASIS( NRMDG1 ), CBASIS( NCMDG1 ), SCALE( NBBND )
  REAL ERRSCL, P, SUM
  INTEGER CTINDX ( NBNDP1 ), PTRNCN ( NDIM5 )
  REAL MINSCL( 10 ), TMPCON
  REAL SQRT, FLOAT, ALOG
  INTEGER IPT( NRX2 )
  REAL BUFFER ( NDIM1 ), SQSBUF ( NDIM6 )
  REAL SQSSUM ( NUPPPL )
#
  REAL SQSUM, FVAL, FNRNC
#
#       GET ROW BASIS VECTORS
#
  CALL DOP ( RBASIS, NRMDG1, NR, MXDG1 )
#
#       GET COLUMN BASIS VECTORS
#
  CALL DOP ( CBASIS, NCMDG1, NC, MXDG1 )
#
#       INITIALIZE VARIABLE FOR ERROR SCALE
#
#
#       GET SCALE FACTOR BY FINDING MAXIMUM
#       AND MINIMUM PRODUCT OF ROW AND COLUMN
#       BASIS VECTORS
#
  NB = 0
  DO DGR = 1, MXDG1
    $(
      MXDGC = MXDG1 - DGR + 1
      DO DGC = 1, MXDGC
        $(
          NB = NB + 1
#
#       PRODUCT OF ROW BASIS WITH COLUMN BASIS
#
          SUM = 0.0
          DO R = 1, NR

```

```

DO C = 1, NC
  $(
    P = RBASIS ( R + NR * ( DGR - 1 ) ) *
      CBASIS ( C + NC * ( DGC - 1 ) )
    BUFFER ( R + NR * ( C - 1 ) ) = P
    IF ( P >= 0.0 )
      SUM = SUM + P
    ELSE
      SUM = SUM - P
    $)
  SCALE ( NB ) = FLOAT( MAXO ) /
    ( SUM * FLOAT( MAXI ) )
#
#       SUM FOR ERROR SCALE
#
#       $)
#
#
#       COMPUTE ERROR SCALE
#
ERRSCL = FLOAT( MAXO ) / FLOAT( NRXNC * MAXI * MAXI )
#
PAUSE
#
NB = 0
NP = 1
DCR1 = MXDG11
PNT1 = - DCR1
DO DGR = 1, MXDG1
  $(
    PNT1 = PNT1 + DCR1
    DCR1 = DCR1 - 1
    MXDGC = MXDG11 - DGR
    DO DGC = 1, MXDGC
      $(
        NB = NB + 1
        CTINDX ( NB ) = NP
        PNT2 = PNT1
        DCR2 = DCR1
        MXDGR = MXDG11 - DGC
        DO I = DGR, MXDGR, 2
          $(
            MXDC = MXDG11 - I
            DO J = DGC, MXDC, 2
              $(
                PTRNCN ( NP ) = PNT2 + J
                NP = NP + 1
              $)
            PNT2 = PNT2 + DCR2 + DCR2 - 1
            DCR2 = DCR2 - 2
          $)
        $)
      $)
    $)
  $)

```

```

        $)
    $)
CTINDX ( NBNDP1 ) = NP
#
#           NORMALIZE SCALE SO THAT TERMS OF
#           THE SAME DEGREE HAVE THE SAME SCALE
#
#           INITIALIZE THE MINSCL(I) TO HIGH VALUE
#
DO I=1,10
MINSCL(I)=1.E9
#
#           PUT THE MINIMUM SCALE IN MINSCL
#           FOR ALL TERMS OF THE SAME DEGREE
#
NB=0
DO DGR=1,MXDG1
    $(
    MXDGC=MXDG1-DGR+1
#
    DO DGC=1,MXDGC
        $(
        NB=NB+1
        TOTDEG=DGR+DGC-2
        IF(TOTDEG <=1 ) NEXT
        IF ( MINSCL( TOTDEG ) > SCALE( NB ) )
            MINSCL(TOTDEG)=SCALE(NB)
        $(
    $(
#
#           RESCALE SCALE SO THAT ALL TERMS
#           OF THE SAME DEGREE HAVE THE SAME
#           SCALE
#
NB=0
DO DGR=1,MXDG1
    $(
    MXDGC=MXDG1-DGR+1
    DO DGC=1,MXDGC
        $(
        TOTDEG=DGR+DGC-2
        NB=NB+1
        IF(TOTDEG<=1)NEXT
        SCALE(NB)=MINSCL(TOTDEG)
    $(
    $(
#
#           SCALE FACTOR IS ALWAYS ADJUSTED FOR
#           ROTATION INVARIANT NORMS EXCEPT WHEN
#           TENSOR PRODUCT IS DONE

```

```

#
NB = 0
DO DGR = 1, MXDG1
  $(
  MXDGC = MXDG1 - DGR + 1
  DO DGC = 1, MXDGC
    $(
    NB = NB + 1
    IF ( DGR > 1 & DGC > 1 )
      $(
      MAXNUM = DGC
      MINNUM = DGR
      IF ( DGR > DGC )
        $(
        MAXNUM = DGR
        MINNUM = DGC
        $)
      IFCT1 = 1
      DO N = MAXNUM, DGR+DGC-2
        IFCT1 = IFCT1 * N
      IFCT2 = 1
      DO N = 2, MINNUM-1
        IFCT2 = IFCT2 * N
      SCALE( NB ) = SQRT(FLOAT(IFCT2)/FLOAT(IFCT1))
                    * SCALE ( NB )
      $)
    $)
  $)
#
#           ERROR SCALE IS
#           MODIFIED FOR LOG OF TOTAL RESIDUAL ERROR
#
#
#           TMPCON = MAXO / ERRSCL + 1.
#           ERRSCL = MAXO / ALOG ( TMPCON )
#           SCALE ( 1 ) = 1.0
#
#           PAUSE
#
#           SET POINTER OF ROW SUM OUTPUT
#
#           NCM1 = NC - 1
#           PNT1 = 1 - MAXPP1
#           PNT2 = - NUMPPL
#
#           DO I = 1, NRM1
#             $(
#               PNT1 = PNT1 + MAXPP1
#               PNT2 = PNT2 + NUMPPL
#               DO J = 1, NUMPPL
#                 $(

```

```

        SQSUM = 0.
        DO K = J, J+NCM1
            $(
                FVAL = BUFFER ( PNT1+K )
                SQSUM = SQSUM + FVAL * FVAL
            $)
        SQSBUF ( PNT2+J ) = SQSUM
    $)
$)
PNT2 = PNT2 + NUMPPL
DO J = 1, NUMPPL
    SQSBUF( PNT2 + J ) = 0.0
#
#           INITIALIZE SQSSUM
#
DO I = 1, NUMPPL
    $(
        PNT1 = I - NUMPPL
        SQSUM = 0.
        DO J = 1, NRM1
            $(
                PNT1 = PNT1 + NUMPPL
                SQSUM = SQSUM + SQSBUF ( PNT1 )
            $)
        SQSSUM ( I ) = SQSUM
    $)
#
DO I = 1, NR
    $(
        IPT( I * 2 - 1 ) = I
        IPT( I * 2 ) = 0
    $)
#
RETURN
END

```

L.20 DOP MAP-subroutine

```

SUBROUTINE DOP(POLY,SIZE,N,NORDER)
#
INTEGER N,NORDER,SIZE
REAL POLY(SIZE)
#
INTEGER I1,I2,I3,I4,I5,I6,K
REAL A,SS,D2
REAL SQRT, FLOAT
#
        PRODUCE POLYNOMIAL OF DEGREE 0
#
A=1./SQRT(FLOAT(N))
DO I1=1,N
    POLY(I1)=A
#
        PRODUCE POLYNOMIAL OF DEGREE 1
#
SS=0.
DO I2=1,N
    $(
        POLY(I2+N)=(FLOAT(I2)-.5*FLOAT(N+1))
        SS=SS+POLY(I2+N)*POLY(I2+N)
    $)
#
SS=1./SQRT(SS)
DO I3=1,N
    POLY(I3+N)=POLY(I3+N)*SS
#
        PRODUCE REMAINING POLYNOMIALS
#
IF(NORDER < 3) RETURN
DO K=3,NORDER
    $(
        D2=0.
        DO I4=1,N
            $(
                A=FLOAT(I4)-.5*FLOAT(N+1)
                SS=A*POLY(I4+N*(K-2))
                D2=D2+SS*POLY(I4+N*(K-3))
            $)
#
SS=0.
DO I5=1,N
    $(
        A=FLOAT(I5)-.5*FLOAT(N+1)
        POLY(I5+N*(K-1))=-D2*POLY(I5+N*(K-3))+
            A*POLY(I5+N*(K-2))
        SS=SS+POLY(I5+N*(K-1))*POLY(I5+N*(K-1))
    $)

```



```
#
      SS=1./SQRT(SS)
      DO I6=1,N
        POLY(I6+N*(K-1))=POLY(I6+N*(K-1))*SS
#
      $)
#
      RETURN
      END
```

L.21 APLBAS MAP-subroutine

```

SUBROUTINE APLBAS ( BUFFER, RCONV, RCONV, RBASIS,
                   CBASIS, NR, NC, NUMPPL, PNT, LL,
                   MAXPPL, MAXPP1, MXDG1, NBBND,
                   NDIM1, NDIM2, NRMDG1, NCMDG1,
                   IOBUF, IPT, NRX2, SCALE, MTRXBC,
                   CTINDX, PTRNCN, NBNDP1, NDIM3,
                   NDIM4, NDIM5, SQSBUF, SQSSUM,
                   NUMNBN, NDIM6, ERRSCL, NMPPL2, NCM1 )
#
  IMPLICIT INTEGER ( A - Z )
#
  REAL RBASIS ( NRMDG1 ), CBASIS ( NCMDG1 ), SUM
  INTEGER CTINDX ( NBNDP1 ), PTRNCN ( NDIM5 )
  REAL MTRXBC ( NDIM4 ), SCALE ( NBBND ), SCL, TMPCON
  INTEGER IPT ( NRX2 ), IOBUF( NMPPL2 )
  REAL SQSSUM ( NUMPPL ), RCONV ( NDIM3 ), RCONV ( NDIM2 )
  REAL BUFFER( NDIM1 ), SQSBUF( NDIM6 ), SQUERR, COEF, ERRSCL
  REAL ALOG, FLOAT
#
#           PUT INPUT LINES INTO REAL BUFFER
#
  LL = IPT( NRX2 - 1 )
  PNT = LL * MAXPP1 - MAXPPL
  BUFFER( PNT ) = 0.0
  DO J = 1, MAXPPL
    BUFFER( PNT + J ) = FLOAT( IOBUF( J ) )
#
#           INITIALIZE POINTER OF BUFFER AND RCONV
#
  COLST = 1 - MAXPP1
#
#           FOR EACH DEGREE OF ROW BASIS VECTOR, CONVOLVE
#
  DO DEG = 1, MXDG1
    $(
      COLST = COLST + MAXPP1
#
#           CONVOLVE THE INPUT BUFFER DOWN THE COLUMN
#           WITH ROW BASIS VECTOR
#
      RCONV( COLST ) = 0.0
      DO I = 2, MAXPP1
        $(
          K = I - MAXPP1
          SUM = 0.0
          DO J = 1, NR
            $(
              K = K + MAXPP1

```

```

                SUM = SUM + RBASIS( J + NR * ( DEG - 1 ) )
                        * BUFFER( K )
        $)
    RCONV( I + COLST - 1 ) = SUM
    $)
$)
#
#           INITIALIZE POINTER 'RCOLST' OF 'RCONV'
#           AND POINTER 'CCOLST' OF 'RCONV'
#
    RCOLST = 1 - MAXPPL
    CCOLST = 1 - NUMPPL
#
#           FOR EACH DEGREE OF ROW AND COLUMN BASIS
#           VECTORS, CONVOLVE
#
    DO DEGR = 1, MXDGI
        $(
            RCOLST = RCOLST + MAXPPL
#
#           FAST CONVOLUTION FOR 0 COLUMN DEGREE
#
            CCOLST = CCOLST + NUMPPL
            SUM = 0.0
            DO COL = 1, NC
                SUM = SUM + RCONV( COL + RCOLST - 1 )
            DO COL2 = 1, NUMPPL
                $(
                    COL = COL2 + NC + RCOLST - 1
                    SUM = SUM + RCONV( COL )
                        - RCONV( COL2 + RCOLST - 1 )
                    RCCONV( COL2 + CCOLST - 1 ) = SUM * CBASIS( 1 )
                $)
#
#           MAXIMUM DEGREE OF COLUMN COORDINATE
#
            MXDEGC = MXDGI - DEGR + 1
            DO DEGC = 2, MXDEGC
                $(
                    CCOLST = CCOLST + NUMPPL
#
#           CONVOLVE ALONG THE ROW WHICH WAS CONVOLVED
#           ALREADY DOWN THE COLUMN OF INPUT BUFFER
#
                DO I = 0, NUMPPL - 1
                    $(
                        SUM = 0.0
                        DO J = 1, NC
                            SUM = SUM + CBASIS( J + NC * ( DEGC - 1 ) )
                                * RCONV( I + J + RCOLST )
                        RCCONV( I + CCOLST ) = SUM
                    $)
                $)
            $)
        $)

```

```

        $)
    $)
$)
#
#           FOR OUTPUT OF CANONICAL FORM, CONVOLVED
#           RESULTS ARE MULTIPLIED WITH PROPER ENTRIES
#           OF CORRESPONDING ROW OF TRANSFORM MATRIX
#
TSTART = 0
DO NB = 1, NBBND
    $(
    FP = CTINDX ( NB )
    LP = CTINDX ( NB+1 )-1
    SCL = SCALE ( NB )
    DO COL = 1, NUMPPL
        $(
        TMPCON = 0.
        DO CT = FP, LP
            $(
            CP = PTRNCN ( CT )
            TMPCON = TMPCON + MTRXBC(TSTART+CP) *
                RCCONV(COL+(CP-1)*NUMPPL)
            $)
        TMPCON = TMPCON * SCL
        IOBUF( COL * 2 ) = 0
        IF ( TMPCON > 0. )
            IOBUF ( COL * 2 - 1 ) = TMPCON + .5
        ELSE
            IOBUF ( COL * 2 - 1 ) = TMPCON - .5
        $)
    PAUSE
    TSTART = TSTART + NBBND
    $)
#
P = ( LL - 1 ) * NUMPPL
DO COL = 1, NUMPPL
    $(
#
#           INITIALIZE SUM FOR FAST COMPUTATION
#
SUM = 0.0
COL2 = COL + PNT
DO C = COL2, COL2 + NCM1
    SUM = SUM + BUFFER( C ) * BUFFER( C )
SQSSUM( COL ) = SQSSUM( COL ) - SQSBUF( COL2 ) + SUM
SQSBUF( COL2 ) = SUM
#
#           SET POINTER AND SQUARE ERROR SUM
#
K = COL + NUMNBN
SQUERR = 0.

```

```

#
#           ERROR SUM IS SQUARE SUM OF CONVOLVED RESULTS
#
      DO NB = 1, NBBND
        $(
          K = K - NUMPPL
          COEF = RCCONV ( K )
          SQUERR = SQUERR + COEF * COEF
        $)
      RCONV ( COL ) = SQSSUM ( COL ) - SQUERR
      IF ( RCONV( COL ) < 0.0 )
        RCONV( COL ) = - RCONV( COL )
      $)
#
      DO COL2 = 1, NUMPPL
        $(
          IOBUF( 2 * COL - 1 ) = ALOG( RCONV( COL ) + 1.0 )
                                * ERRSCL + 0.5
          IOBUF( 2 * COL ) = 0
        $)
#
      K = IPT( 1 )
      DO COL3 = 1, NRX2 - 3, 2
        IPT( COL3 ) = IPT( COL3 + 2 )
      IPT( NRX2 - 1 ) = K
#
      DO DG = 1, MXDG1
        $(
          K = NR * ( DG - 1 )
          TEMP = RBASIS( NR + K )
          DO DG2 = NR, 2, -1
            RBASIS( DG2 + K ) = RBASIS( DG2 - 1 + K )
          RBASIS( 1 + K ) = TEMP
        $)
#
      RETURN
      END

```

L.22 XFACET2 Include File

```
NAME6(.F,.A,.C,.E,.T,.D2)
GETARRAYS
IDESC1( 1 )
DEFINE(IOBUF,IAR1)
IDESC1( 2 )
DEFINE(IPT,IAR2)
RDESC1( 3 )
DEFINE(RBASIS,RAR3)
RDESC1( 4 )
DEFINE(CBASIS,RAR4)
RDESC1( 5 )
DEFINE(BUFFER,RAR5)
RDESC1( 6 )
DEFINE(RCONV,RAR6)
RDESC1( 7 )
DEFINE(RCCONV,RAR7)
RDESC1( 8 )
DEFINE(SCALE,RAR8)
RDESC1( 9 )
DEFINE(SQSBUF,RAR9)
RDESC1( 10 )
DEFINE(SQSSUM,RAR10)
RDESC1( 11 )
DEFINE(MTRXBC,RAR11)
IDESC1( 12 )
DEFINE(CTINDX,IAR12)
IDESC1( 13 )
DEFINE(PTRNCN,IAR13)
```

**The vita has been removed from
the scanned document**