2/
3-1-79

LA-7499-M
Manual

MASTER

DEMOS Primer

University of California

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

LOS ALAMOS SCIENTIFIC LABORATORY
Post Office Box 1663 Los Alamos, New Mexico 87545

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

An Affirmative Action/Equal Opportunity Employer

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

UNITED STATES
DEPARTMENT OF ENERGY
CONTRACT W-7405-ENG. 36

LA-7499-M
Manual

Special Distribution
Issued: January 1979

DEMOS PRIMER

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

SOFTWARE DOCUMENTATION
GROUP C-2
(505) 667-2432

LOS ALAMOS SCIENTIFIC LABORATORY

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

PREFACE

Don't plan to read this primer at home some balmy summer evening and then, the next day or next week or next month, when you have worked yourself up to it, try to run a job on DEMOS. Work yourself up first, then read this sitting at your terminal. This is supposed to be a do-as-you-read book--convert your Fortran code while you are reading Chapter 2, compile it while you are reading Chapter 3, etc.--so that you can get simple jobs running on DEMOS with a minimum of preliminary time invested. Why read a 180-page reference manual when all you want to do is compile and execute a routine program? If you want to know more than the primer tells you, look in PIM-7; but, for right now, if you are a new user with a simple job, ignore it. There is plenty of time for system reference manuals later, when you have something more complicated to do and feel comfortable running jobs on DEMOS.

CONTENTS

1 BEFORE YOU DO ANYTHING ELSE

The first thing you have to do is get validated--that is, you must have permission to run jobs on the CRAY-1 computer. If you were validated to run jobs on the CRAY-1 computer when it was running the Benchmark Operating System (BOS), you are also validated for DEMOS. But if you have the usual user luck (that is, little or none), you were not validated for BOS, in which case you must contact the C-7 group secretary to get validated on DEMOS. If you are not validated for LTSS, first call the C-Division Computer Security Office to get validated for LTSS, then call C-7 to get validated for DEMOS.

Once you are validated for DEMOS, sign on to LTSS, since jobs for the CRAY-1 are submitted through LTSS. If you are not familiar with LTSS, call the C-Division Program Library (667-6992) and ask for a copy of PIM-6, which describes the basic use of LTSS. Learn the rudiments (especially how to use TRIX), then come back and pick up here.

One thing about LTSS that I must explicitly mention is the tag

```
/ t p
```

that you will see attached to some LTSS execute lines in this Primer. The t is a symbol for time and the p for priority; for these symbols you should type some appropriate numbers. See page 1-10 of PIM-6 for information on time and priority.

At this point, I assume that you are validated for LTSS and DEMOS and that you have an LTSS file containing a Fortran program you want to execute on DEMOS. If so, start reading Chapter 2.

2 PREPARING YOUR FORTRAN CODE

Because DEMOS does not currently support a Fortran compiler, you will compile your program on LTSS using XFC. XFC accepts an input file of FTN statements and generates an output file of Cray Assembly Language (CAL) statements. The CAL file generated by XFC can then be sent to DEMOS to be assembled, loaded, and executed. (If your Fortran program was written for FUN/RUN, convert it to FTN with the SIFT utility run under SLOPE2 on LTSS.)

Before your code can be compiled, however, there are some things you must do to it so that it will be accepted by XFC.

- Remove or modify machine-dependent or word-size-dependent code. Note that character constants (A, H, L, and R formats) are stored 8 characters per word on the CRAY-1 computer. Note also that the intrinsic SHIFT function is left circular and right end-off with no sign extension.

- Remove references to LCM and ECS.

- Remove double-precision references.

- Remove calls to library routines not in MATHLIB, DEMLIB, or CGSFTN. The contents of MATHLIB and DEMLIB are listed in Appendix A of PIM-7. See PIM-2 for information on CGSFTN.

- Remove OVERLAY and CALL OVERLAY statements.

- Remove calls to the following FTN utility subprograms:

CHECKPTX	DISPLA	ERRSET
IOCHEC	LABEL	LEGVAR
RANF	RANGET	RANSET
SLITE	SLITET	STRACE
SYSTEM	SYSTEMC	

  ECS/LCM subprograms
  terminal interface subprograms

- Remove all calls to the Cyber Record Manager interface and the Sort/Merge interface.

3 COMPILING YOUR PROGRAM

You should now be ready to compile your program. First, switch the name of the file containing the Fortran source code to PGM with the statement

```
SWITCH program PGM
```

where for program you type the name of your file. (After the compilation is completed, switch the name of the file from PGM back to its original name so that you don't lose track of it.)

Now get the procedure file named XFCCOMP from Hydra by typing

```
XPORT C4USER GET XFCCOMP
```

XFCCOMP consists of commands to compile a program named PGM with XFC.

To execute XFCCOMP, enter

```
SLOPE2 I=XFCCOMP / t p
```

As a result of executing this line, your program will be compiled by XFC, and the output, in the form of Cray Assembly Language (CAL) statements, will be written to a file named CINPUT. CINPUT is ready to be sent to DEMOS for assembly and execution. The source listing and dayfile from executing SLOPE2 will be written to a file called PSLOPEx, where x is the LTSS suffix you are executing SLOPE2 under.

Because you will undoubtedly be compiling another program sometime, it would be wise to switch the name of the XFCCOMP output file from the default name CINPUT to some unique name so that the next compilation does not destroy your output from this one.

4 CREATING A DEMOS JOB FILE

Now it is time to write your DEMOS job file (the file of DEMOS job control statements required to run your job). You will create it on LTSS using TRIX and then submit it, along with your program and data files, to DEMOS.

You could, of course, read Chapter 4 of PIM-7 to learn the job control language, but that would take a lot of time. The simplest thing to do is find a job file that does the same thing that you want to do and copy it. To that end, this chapter provides a basic set of sample files.

To simplify explanations of the examples, I will not mention all the possible arguments for each of the job control statements. Refer to Chapter 8 of PIM-7 for detailed information if the examples won't quite do.

After you look at the examples, you will realize that basic DEMOS job control is pretty conventional, so at the end of this section I have added a list of the most commonly used utilities, with their arguments and a brief description of how they work. If you want to do something a little different from what the examples illustrate, you should be able to patch it together and make it work.

Before you plunge into writing your job file, you should know that there are two kinds of DEMOS files, temporary files (on some systems called "local files") and resident files (on some systems called "permanent files"). The examples in this Primer assume that you have no pressing need to save files on DEMOS disk (resident files), so that all files needed by and created by your job can be temporary files. The purpose of this is to simplify your use of the system: there is no need to say anything about the DEMOS directory file structure if you only work with temporary files. So, if you want to save some files on resident file space, or if you are just curious, look at Chapter 2 of the DEMOS User's Guide (PIM-7) for a description of the file system. But for now, I will assume the simplest case, in which all are your files are temporary files.

Example 1

Assemble a program and execute it.

```
job z=12345 jn=sample1 ch=8005x003 cl=u t=19 m=180000
cal i=calin e
load map=ldmap arg=[amphib=frog fowl=duck]
dispose map
stop
```

The first statement in any job file is the JOB statement, in which you must list your Z-number (leading zeros are optional), the name of the job (sample1 in this example), the cost center and project codes, and the classification (u is unclassified). In addition, this JOB statement specifies a time limit of 19 seconds (default is 10) and a memory size of 180,000 decimal words (default is 98304).

The next statement calls the CAL assembler, with CALIN as the source code. (Assume that you changed the name of the file CINPUT to CALIN after compiling your program by means of the XFCCOMP procedure file.) By specifying E, any error listing goes to a file called OUT. The relocatable binary code is placed in the default output file BLD, which is also the default input file to the loader.

LOAD loads BLD and executes it (load-and-go is the default mode of the loader), with the load map sent to LDMAP. If you want to specify some relocatable binary file other than BLD as the file to be loaded, use the REL=file argument in the LOAD statement (see Example 6). The ARG argument specifies files to be used in place of files named in the program: the file FROG will be used wherever AMPHIB is specified; the file DUCK will be used wherever FOWL is specified.

The DISPOSE statement sends the load map (LDMAP) back to the LTSS machine.

STOP terminates your job. The dayfile LOG and the output file OUT are both automatically returned to LTSS.

But what happens to the output resulting from execution? If your Fortran program wrote its output to the file OUTPUT, it will appear on the file OUT. If the output was written to some other file, you must DISPOSE that file; otherwise, it will be destroyed when the job terminates.

Example 2

Assemble a program, load it, and return the executable file (but do not execute it).

```
job z=087524 jn=sample2 ch=8005x003 cl=u
cal i=calj e
load ex=exec map=ldmap g=off
dispose ldmap exec
stop
```

This JOB statement is quite standard, making use of the default time limit (10 seconds) and memory size (98304 decimal words).

The CAL statement assembles the temporary file CALJ (you shipped it to DEMOS after changing its name from CINPUT to CALJ) and writes the error file to OUT, as in Example 1. The assembled code is written to the default file BLD.

The LOAD statement loads the default relocatable binary output file from CAL (BLD). The executable code is written to the file EXEC. A load map is written to LDMAP. The argument G=OFF suppresses execution.

The DISPOSE statement sends the load map and the executable file to the LTSS machine. Presumably the executable file EXEC will be saved on Hydra.

STOP terminates the job. The dayfile LOG and the default output file OUT are both automatically returned to LTSS.

Example 3

Execute an executable file. This example will execute the file EXEC that was created in Example 2.

```
job z=087524 jn=sample3 ch=8005x003 cl=u t=12
exec kiwi=fig coca=nut
stop
```

As always, JOB must be the first statement in the job file.

The second statement executes the file EXEC, which was assembled, loaded, and saved on Hydra in the preceding example. The arguments following EXEC specify the files to be used instead of files named on the program card (FIG in place of KIWI, NUT in place of COCA). If EXEC alone is specified, the files named in the program will be used.

This job assumes that output from the executing program is written to the file OUTPUT and thus will be on the file OUT, which is automatically returned to LTSS at job termination. If output was written to any other file, that file must be DISPOSEd.

Example 4

Create a library of executable files.

```
job z=087524 jn=sample4 ch=8005x003 cl=u
cal i=cinput e
maklib newlib=mylib add=bld map=out
dispose mylib
stop
```

The JOB, CAL, and STOP statements are as in the preceding examples. The important new utility here is MAKLIB, which creates and edits libraries.

Let's assume you have compiled (with XFC) a Fortran program with which you want to start a library. This sample job file will assemble the program and then put the output in a library. The assembled program, in the default file BLD, is used as an "addfile" to MAKLIB, which means it is added to the library (since no OLDLIB=file argument is specified in this statement, the library is initially empty). Output (the library) is written to the file MYLIB, which is DISPOSEd to LTSS so that it can be saved on Hydra. The MAP argument assigns to OUT the record of MAKLIB's activities in creating this library.

Example 5

Edit a library of executable files that was shipped from LTSS to DEMOS.

```
job z=087524 jn=sample5 ch=8005x003 cl=u
cal i=cinput e
maklib newlib=goodlib oldlib=mylib add=bld map=out
dispose goodlib
stop
```

The MAKLIB statement in this example will add the relocatable binary file BLD (the default output from CAL) to the library MYLIB, which was created in the previous example. The resulting new library is placed in the file GOODLIB, which is DISPOSEd to LTSS so that it can be saved on Hydra.

Note that this sample job is not substantially different from the preceding one, where a library was created. The only difference is that this MAKLIB statement specifies an OLDLIB argument, meaning that an existing library is being modified.

Example 6

Assemble, load, and execute a program using the library GOODLIB (produced in Example 5), which you shipped over with the job file from LTSS.

```
job z=087524 jn=sample6 ch=8005x003 cl=u
cal i=caldem b=calcode e
load rel=calcode lib=goodlib lib=/lib/demlib #
     lib=/lib/mathlib map=map arg=[output=result]
dispose map calcode result
stop
```

There are three new things in this example. First, instead of using the default output file from CAL (BLD), the job writes the relocatable binary code to the file CALCODE. CALCODE is the file that is executed (load-and-go is default for the loader).

The second new thing is the specification of libraries for LOAD. DEMLIB and MATHLIB are default libraries; note that the complete names /LIB/DEMLIB and /LIB/MATHLIB must be used to identify them. If you specify another library to be used (as in LIB=GOODLIB), you must specify all libraries, including DEMLIB and MATHLIB. The routines available in DEMLIB and MATHLIB are listed in Appendix A of PIM-7.

The third new thing is the line continuation in the LOAD statement. To continue a line, just type a # between arguments.

A Miscellany of DEMOS Utilities

Before describing some of the more commonly used utilities, I should tell you the two syntax rules you need to know to use these utilities:

- Arguments must be separated from each other and from the name of the utility by at least one space.
- To continue a line, type a # followed by a carriage return at a point where a space could occur (between arguments).

In each case below, where I give the general form of the statement, upper-case letters represent literally what you must type; lower-case letters indicate a symbol for which you supply the exact text (a file name, for example). As a matter of normal practice, use lower-case letters to enter your job control statements.

CAL

See Chapter 8 of PIM-7 for details.

DISPOSE filelist

Sends the files in filelist from DEMOS to the source machine. Separate the file names in filelist by at least one space. Note that any file that is DISPOSEd is no longer available on DEMOS.

Example: dispose this that thother

JOB

This must be the first statement of any DEMOS job file. See Chapter 8 of PIM-7 for all the arguments and defaults.

LOAD

See Chapter 8 of PIM-7 for all the arguments and defaults of the LOAD loader.

MAKLIB

See Chapter 8 of PIM-7 for all the arguments and defaults for MAKLIB.

NOTE charstring

Appends the string charstring to the LOG file. Restrict the characters you use to letters, numbers, spaces, and normal punctuation marks. Charstring can be only one line.

Example: note I sure hope this works.

STOP

Terminates execution of your job.

5 SUBMITTING A JOB TO DEMOS

Now, at last, you should be ready to submit your job to DEMOS. To do so, you need to use two utilities on LTSS--one to convert your LTSS files to DEMOS format, and one to actually submit your job.

Conversion is necessary because the CDC 7600s that run LTSS have 60-bit words (ten 6-bit characters per word) and the CRAY-1 has 64-bit words (eight 8-bit characters per word). You must convert all files you are shipping to DEMOS except the output from the XFCCOMP procedure file (which, you will remember, is your program translated into CAL); the output from XFCCOMP is already converted into DEMOS format. The job file and all data files must be converted.

Conversion entails no great effort. In the simplest form, for each file you need only write

```
CRACON oldfile newfile
```

and you are done. The file oldfile will be converted to DEMOS format and written to file newfile. There are a number of options that can be used with CRACON, but for straightforward data conversion they can be ignored. If you want to look at them, they are described in the section on CRACON in Chapter 7 of PIM-7.

Now the actual submittal, for which you use the LTSS utility called CRALINK. First, enter the execute line

```
CRALINK
```

LTSS will respond with the prompt

```
ok:
```

Now you can tell CRALINK what you want to submit with the line

```
SUBMIT L=jobfile DATA=filelist
```

where for jobfile you type the name of the job file (see the preceding chapter) and for filelist you specify any programs and data files you want shipped to DEMOS. All of the files will be interpreted as

temporary files when they arrive at DEMOS.

For example, if your job file is called JOBIN, your program is in HOPE, and the data files are CURSE and SWEAR, then you would type

```
submit l=jobin data=hope curse swear
```

Note that the file names in filelist are separated by at least one space. You can have as many names in filelist as will fit on the line.

After you complete the SUBMIT command, CRALINK will inform you of a three-digit number that has been assigned to your job. This number should be used to communicate with CRALINK about the job (see below). The number will also be appended to the names of any files returned to LTSS by the job.

There are three other CRALINK commands you may have occasion to use: STATUS, TERMINATE, and END. They can be typed in response to the ok: prompt.

The STATUS command allows you to get information about the status of the CRAY-1 computer and about your jobs. For example, if you are planning to submit a job and are wondering how many jobs are already waiting, just type

```
STATUS
```

In return, you will be told the number of jobs in the DEMOS queue and the total time for the queue. If you have already submitted a job and want to know how far along it is, type

```
STATUS jobno
```

where jobno is the number of the job returned after the SUBMIT command. You can specify more than one job number by leaving a blank between them. The information you get consists of the job number, the CRAY ID, submit time, job type, and state (waiting or executing). If you can't remember the job number assigned to your job, you can get the same information about all the jobs you have running by typing

```
STATUS ALL
```

The TERMINATE command allows you to stop execution of a job you have submitted. Just type

```
TERMINATE jobno
```

where jobno is the DEMOS job number of the job you want to stop. The job will terminate and the &LOG and &OUT files will be converted and returned to LTSS.

And, finally, to get yourself out of the CRALINK session, just type

```
END
```

and you are done. This will not affect your job execution on the CRAY-1. For example, you can SUBMIT a job, END the session, and an hour later execute CRALINK again and check on the status of your job.

6 PUTTING YOUR JOB TO BED

When your job completes execution under DEMOS, the dayfile (LOG) and the default output file (OUT) are automatically returned to the source machine of the job, where you can find them under the names LOGnnn and OUTnnn (nnn is the DEMOS job number). All files you have DISPOSEd will also have the three-digit job number appended to the file name when they arrive at LTSS.

Because any files returned will still be in DEMOS format, you must again use the CRACON routine to convert them, this time back to LTSS format, if you want to output them. Fortunately, CRACON is designed so that you can use it exactly the same way, whichever direction you are converting: if the input file is not in DEMOS format, it is converted to DEMOS format; if it is in DEMOS format, it is converted to LTSS packed-ASCII.

Now, check a little bit of each file created by your job to make sure that they are not just garbage. If all is in order, you can use ALLOUT to print a copy of the complete file, or you can save it on Hydra.

7 VECTORIZING YOUR FORTRAN PROGRAM

Now that you can run jobs on DEMOS, it would be worth your while to consider vectorizing your Fortran program if it is to be used repeatedly. After all, one of the reasons for having the CRAY-1 computer is its vector processing capability. Furthermore, because the vectorization is done by a procedure file (VECEZ), very little programmer effort is required.

To vectorize your program, execute the procedure file VECEZ (available under SLOPE2) using your program as the input file. VECEZ will access a vectorizing program (the vectorizer) that will, wherever possible, substitute vector instructions for your conventional Fortran statements. You will then compile this vectorized code with XFC.

As a result of executing VECEZ, you will be provided with the following:

- vectorized code in a form suitable for compilation by XFC;
- a listing of the source code;
- a cross-reference listing;
- information on ways to revise the Fortran source code so that more advantage can be taken of the vector processing capabilities.

Although this last information is useful if you want to see how much of the code was vectorized, it can be ignored. However, in the best of all possible worlds you would use this information to revise your source code and then resubmit it to the vectorizer (never resubmit the vectorized code-- the vectorizer doesn't know what to do with its own output).

But before you get your terminal fired up to send off for VECEZ, consider two things:

- Make sure your Fortran source code already runs on the CRAY-1. Debugging a program that is peppered with vector instructions inserted by the vectorizer is a task reserved for wizards (or at least for people who speak in tongues).

- Check the size of your Fortran source code. The vectorizer gobbles up CPU time (translation: bank account points), so don't feed a large program to it in one big chunk and expect to have any friends left. Instead, vectorize large programs a routine at a time, first doing the the most heavily used routine, then another, then another, etc. XFC doesn't care whether or not the entire program has been through the vectorizer.

Once you are ready to use VECEZ, enter the following command sequence. The XPORT, LIX, and DESTROY routines will print ALL DONE as they complete. Wait for this response before you type the next line.

```
XPORT C4USER GET VCLOVL
LIX VCLOVL!GR. ALL.
END
DESTROY VCLOVL
SLOPE2 I=VECEZ
```

The system will provide the following response:

```
SLOPE2 I=VECEZ
◆ COPYCF(,VR)
   END OF INFORMATION ENCOUNTERED.
◆ ◆ VECTORIZER PROCEDURE
◆ ◆ CALL(VR(IF=INFILE,OF=OUTFILE))
◆ ◆       IF - INPUT SOURCE FILE IN ASCII.
◆ ◆       OF - ASCII OUTPUT VECTORIZED SOURCE FILE.
◆ ◆           OPTIONS ARE CJP  .
◆ DIE.
```

After the system output is completed, in response to the slash prompt from SLOPE2 enter the call line

```
CALL(VR(IF=infile,OF=outfile))
```

where infile is the name of the packed-ASCII file containing your Fortran code and outfile is the name of the file to which the vectorized code (which will also be in packed-ASCII) should be written. This is the file you will send to the XFC compiler. For example, if your source file is named SOL, type

```
call(vr(if=sol,of=pgm))
```

and your vectorized code is ready for XFC since the output file is already named PGM.

The following example illustrates a complete session with VECEZ; it includes all system responses.

```
SLOPE2 I=VECEZ
◆ COPYCF(,VR)
   END OF INFORMATION ENCOUNTERED.
◆ ◆ VECTORIZER PROCEDURE
◆ ◆ CALL(VR(IF=INFILE,OF=OUTFILE))
◆ ◆       IF - INPUT SOURCE FILE IN ASCII.
◆ ◆       OF - ASCII OUTPUT VECTORIZED SOURCE FILE.
◆ ◆            OPTIONS ARE CJ@  .
◆ DIS.
/ CALL(VR(IF=PGM,OF=VECPGM))
◆ DFILE,VEC/CM.
◆ DFILE,PGM/PA.
◆ CFILE,VECPGM/PA.
◆ RFL(150000,250000)
◆ VEC(I=PGM,L=OUTPUT,C=VECPGM,S=CJ@)
  MAIN
  CJAC
      STOP
      .738 CP SECONDS EXECUTION TIME
◆ EXIT.
/ END
$%CPU TIME        0.923 SEC
$%SYS TIME        0.236 SEC
$%I/O TIME       25.916 SEC
$%TOTAL  =        0.451 MINUTES

 ALL DONE
```

8 SHOULD I HAVE TOLD YOU THIS BEFORE?

It turns out that submitting a job to DEMOS can be done using a BCON controller named DEMON, which will take a simple Fortran source program and associated files as input and do the appropriate things to execute the program on DEMOS. Specifically, DEMON will perform the following tasks:

- compile a Fortran program with XFC;
- convert specified BCD data files to DEMOS text file format;
- create a simple job file that will assemble, load, and execute the program;
- submit the job file and associated files to DEMOS;
- return output files to LTSS and convert them, if required.

Clearly, what DEMON can do for you is limited, but if this is all you need, it is very handy.

Get DEMON from Hydra with the line

```
XPORT C4USER GET DEMON
```

After LTSS has responded with ALL DONE, enter a line with the form

```
BCON DEMON filename type costcode box memory comp char
```

where the arguments have the following meaning:

filename	name of the file containing the program to be executed.
type	S if filename specifies a Fortran source code on LTSS, or L if filename specifies a library file stored in DEMOS resident file space.
costcode	four-digit cost center code.

box	box number for output as specified for ALLOUT. If box is specified and the job does not terminate until after 4 P.M., DEMON will ALLOUT your files to that box.
memory	required field length. Default: 150000 decimal words.
comp	Possible values are ONLYXFC and GO. If ONLYXFC is specified, DEMOS will compile the program but will not submit the job for execution. If GO is specified instead, DEMON will call XFC to compile the program and then will submit it to DEMOS for execution.
char	character c used with the EXCL. option in CRACON (used only with CAL code intermixed with Fortran).

The last four arguments are optional. However, if you specify one of them, you may not omit any arguments that precede it. For example, if you want to specify ONLYXFC, you must also specify box and memory. On the other hand, you can specify box without specifying the others.

If the second argument is S and the ONLYXFC argument is not used, when the job terminates a file named EXEfilename will remain in your DEMOS resident file space. This file contains the executable code for the program you submitted. For example, if filename is BIGPIG, after your job terminates the file EXEBIGPIG on DEMOS resident file space will contain the executable code. As a result, any subsequent time you want to use DEMON to execute that same program, you should enter EXEBIGPIG as filename and then specify L as the second argument. That way, your program will not be re-compiled, re-assembled, and re-loaded every time you want to execute it.

After you have called DEMON with the BCON DEMON line, you will be queried about such things as job time limit, input files, and output files. DEMON will then proceed to do your work for you.

9 HOW TO AVOID NIGHTMARES

(1) If your job fails, keep a copy of the job file, the LOG file, and any other files related to the job. This will increase the probability that the consultants will be able to help you.

(2) Make a habit of getting a load map (use the MAP argument in the LOAD statement). That way, if you get into trouble with your program and seek the advice of the consultants, they will not have to tell you to rerun your job in order to get a load map.

(3) Don't forget to revise your FTN code to be acceptable to XFC.

(4) Get your Fortran code running on the CRAY-1 before you vectorize it.

(5) Make sure your files are in DEMOS format before you submit them. After they come back, convert them back to LTSS format if you want to examine them.